CHAPTER

11

Method of Least Squares

In this chapter, we use a model-dependent procedure known as the *method of least squares* to solve the linear filtering problem, without invoking assumptions on the statistics of the inputs applied to the filter. To illustrate the basic idea of least squares, suppose we have a set of real-valued measurements $u(1), u(2), \ldots, u(N)$, made at times $t_1, t_2, \ldots, t_N$, respectively, and the requirement is to construct a curve that is used to *fit* these points in some optimum fashion. Let the time dependence of this curve be denoted by $f(t_i)$. According to the method of least squares, the "best" fit is obtained by *minimizing the sum of squares of difference* between $f(t_i)$ and $u(i)$ for $i = 1, 2, \ldots, N$, hence the name of the method.

The method of least squares may be viewed as an alternative to Wiener filter theory. Basically, Wiener filters are derived from *ensemble averages* with the result that one filter (optimum in a probabilistic sense) is obtained for all realizations of the operational environment, assumed to be wide-sense stationary. On the other hand, the method of least squares is *deterministic* in approach. Specifically, it involves the use of time averages, with the result that the filter depends on the number of samples used in the computation. We begin our study in the next section by outlining the essence of the linear least-squares estimation problem.

11.1 STATEMENT OF THE LINEAR LEAST-SQUARES ESTIMATION PROBLEM

Consider a physical phenomenon that is characterized by two sets of variables, $d(i)$ and $u(i)$. The variable $d(i)$ is observed at time i in *response* to the subset of variables $u(i)$, $u(i-1), \ldots, u(i-M+1)$ applied as *inputs*. That is, $d(i)$ is a function of the inputs $u(i)$,

$u(i-1), \ldots, u(i-M+1)$. This functional relationship is hypothesized to be *linear*. In particular, the response $d(i)$ is modeled as

$$d(i) = \sum_{k=0}^{M-1} w_{ok}^* u(i-k) + e_o(i) \tag{11.1}$$

where the w_{ok} are *unknown parameters* of the *model*, and $e_o(i)$ represents the *measurement error* to which the statistical nature of the phenomenon is ascribed; each term in the summation in Eq. (11.1) represents a scalar inner product. In effect, the model of Eq. (11.1) says that the variable $d(i)$ may be determined as a linear combination of the input variables $u(i), u(i-1), \ldots, u(i-M+1)$, except for the error $e_o(i)$. This model, represented by the signal-flow graph shown in Fig. 11.1, is called a *multiple linear regression model*.

The *measurement error* $e_o(i)$ is an *unobservable* random variable that is introduced into the model to account for its inaccuracy. It is customary to assume that the measurement error process $e_o(i)$ is white with zero mean and variance σ^2. That is,

$$E[e_o(i)] = 0 \qquad \text{for all } i$$

and

$$E[e_o(i)e_o^*(k)] = \begin{cases} \sigma^2, & i = k \\ 0, & i \neq k \end{cases}$$

The implication of this assumption is that we may rewrite Eq. (11.1) in the ensemble-averaged form

$$E[d(i)] = \sum_{k=0}^{M-1} w_{ok}^* u(i-k)$$

where the values of $u(i), u(i-1), \ldots, u(i-M+1)$ are known. Hence, the mean of the response $d(i)$, in theory, is uniquely determined by the model.

The problem we have to solve is to *estimate* the unknown parameters of the multiple linear regression model of Fig. 11.1, the w_{ok}, given the two *observable* sets of variables: $u(i)$ and $d(i)$, $i = 1, 2, \ldots, N$. To do this, we postulate the linear transversal filter of Fig. 11.2 as the model of interest. By forming inner scalar products of the *tap inputs* $u(i), u(i-1), \ldots, u(i-M+1)$ and the corresponding *tap weights* $w_0, w_1, \ldots, w_{M-1}$, and by utilizing $d(i)$ as the *desired response*, we define the *estimation error* or *residual* $e(i)$ as the difference between the desired response $d(i)$ and the *filter output* $y(i)$, as shown by

$$e(i) = d(i) - y(i) \tag{11.2}$$

where

$$y(i) = \sum_{k=0}^{M-1} w_k^* u(i-k) \tag{11.3}$$

That is,

$$e(i) = d(i) - \sum_{k=0}^{M-1} w_k^* u(i-k) \tag{11.4}$$

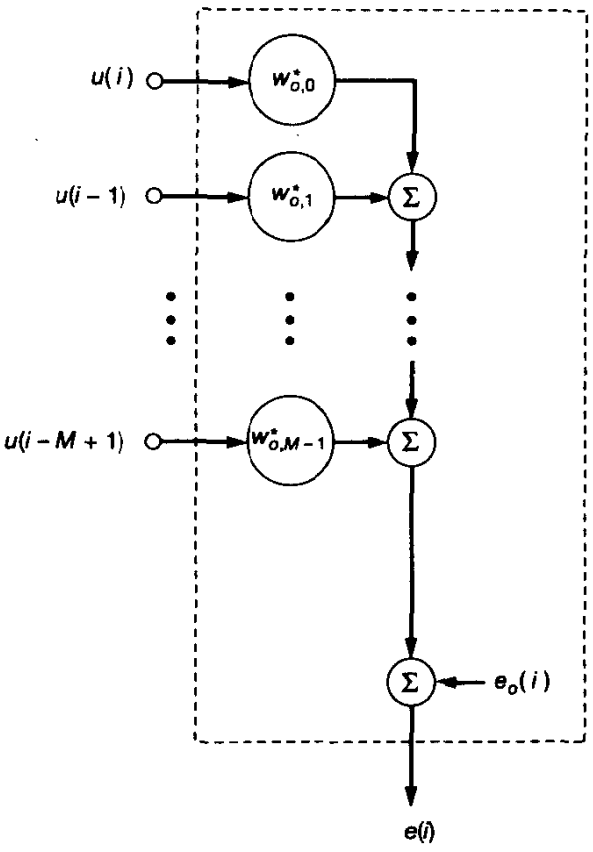


Figure 11.1 Multiple linear regression model.

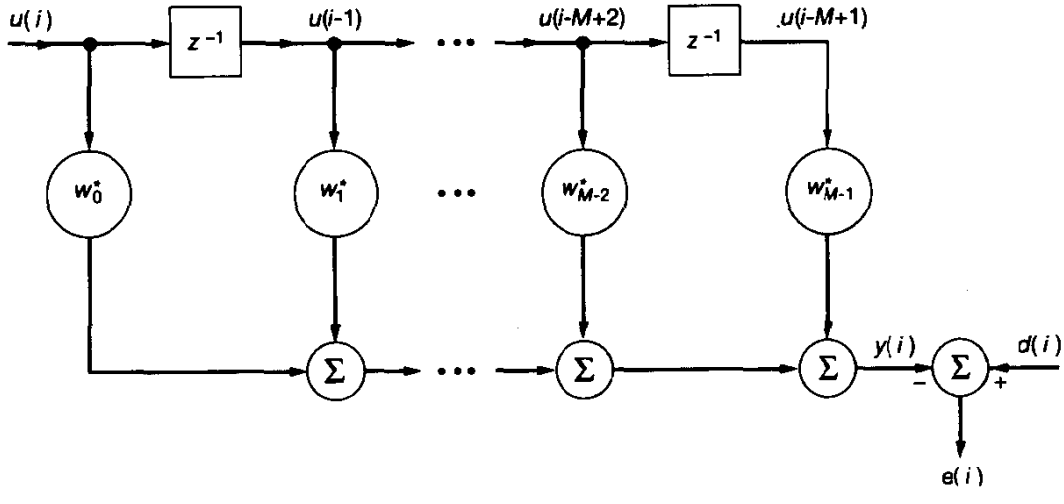


Figure 11.2 Linear transversal filter model.

In the method of least squares, we choose the tap weights of the transversal filter, the w_k, so as to minimize a cost function that consists of the *sum of error squares*:

$$\mathscr{E}(w_0, \ldots, w_{M-1}) = \sum_{i=i_1}^{i_2} |e(i)|^2 \tag{11.5}$$

where i_1 and i_2 define the index limits at which the error minimization occurs; this sum may also be viewed as an *error energy*. The values assigned to these limits depend on the type of *data windowing* employed, as discussed in Section 11.2. Basically, the problem we have to solve is to substitute Eq. (11.4) into (11.5) and then minimize the cost function $\mathscr{E}(w_0, \ldots, w_{M-1})$ with respect to the tap weights of the transversal filter in Fig. 11.2. For this minimization, the tap weights of the filter $w_0, w_1, \ldots, w_{M-1}$ are held *constant* during the interval $i_1 \leq i \leq i_2$. The filter resulting from the minimization is termed a *linear least-squares filter*.

11.2 DATA WINDOWING

Given M as the number of tap weights used in the transversal filter model of Fig. 11.2, the rectangular matrix constructed from the input data, $u(1), u(2), \ldots, u(N)$, may assume different forms, depending on the values assigned to the limits i_1 and i_2 in Eq. (11.5). In particular, we may distinguish four different methods of *windowing* the input data:

1. *Covariance method*, which makes no assumptions about the data outside the interval $[1, N]$. Thus, by defining the limits of interest as $i_1 = M$ and $i_2 = N$, the input data may be arranged in the matrix form

$$\begin{bmatrix} u(M) & u(M+1) & \cdots & u(N) \\ u(M-1) & u(M) & \cdots & u(N-1) \\ \vdots & \vdots & \vdots & \vdots \\ u(1) & u(2) & \cdots & u(N-M+1) \end{bmatrix}$$

2. *Autocorrelation method*, which makes the assumption that the data prior to time $i = 1$ and the data after $i = N$ are zero. Thus, by using $i_1 = 1$ and $i_2 = N + M - 1$, the matrix of input data takes on the form

$$\begin{bmatrix} u(1) & u(2) & \cdots & u(M) & u(M+1) & \cdots & u(N) & 0 & \cdots & 0 \\ 0 & u(1) & \cdots & u(M-1) & u(M) & \cdots & u(N-1) & u(N) & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & u(1) & u(2) & \cdots & u(N-M+1) & u(N-M) & \cdots & u(N) \end{bmatrix}$$

3. *Prewindowing method*, which makes the assumption that the input data prior to $i = 1$ are zero, but makes no assumption about the data after $i = N$. Thus, by using $i_1 = 1$ and $i_2 = N$, the matrix of the input data assumes the form

$$\begin{bmatrix} u(1) & u(2) & \cdots & u(M) & u(M+1) & \cdots & u(N) \\ 0 & u(1) & \cdots & u(M-1) & u(M) & \cdots & u(N-1) \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdots & u(1) & u(2) & \cdots & u(N-M+1) \end{bmatrix}$$

4. *Postwindowing method*, which makes no assumption about the data prior to time $i = 1$, but makes the assumption that the data after $i = N$ are zero. Thus, by using $i_1 = M$ and $i_2 = N + M - 1$, the matrix of input data takes on the form

$$\begin{bmatrix} u(M) & u(M+1) & \cdots & u(N) & 0 & \cdots & 0 \\ u(M-1) & u(M) & \cdots & u(N-1) & u(N) & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ u(1) & u(2) & \cdots & u(N-M+1) & u(N-M) & \cdots & u(N) \end{bmatrix}$$

The terms "covariance method" and "autocorrelation method" are commonly used in speech-processing literature (Makhoul, 1975; Markel and Gray, 1976). It should, however, be emphasized that the use of these two terms is *not* based on the standard definition of the covariance function as the correlation function with the means removed. Rather, these two terms derive their names from the way we interpret the meaning of the *known parameters* contained in the system of equations that result from minimizing the index of performance of Eq. (11.5). The covariance method derives its name from control theory literature where, with zero-mean tap inputs, these known parameters represent the elements of a *covariance matrix*, hence the name of the method. The autocorrelation method, on the other hand, derives its name from the fact that, for the conditions stated, these known parameters represent the *short-term autocorrelation function* of the tap inputs, hence the name of the second method. It is of interest to note that, among the four windowing methods described above, the autocorrelation method is the only one that yields a *Toeplitz* correlation matrix for the input data.

In the remainder of this chapter, except for Problem 4, which deals with the autocorrelation method, we will be exclusively concerned with the covariance method. The prewindowing method is considered in subsequent chapters.

11.3 PRINCIPLE OF ORTHOGONALITY (REVISITED)

When we developed the Wiener filter theory in Chapter 5, we proceeded by first deriving the principle of orthogonality (in the ensemble sense) for wide-sense stationary discrete-

time stochastic processes, which were then used to derive the Wiener–Hopf equations that provide the mathematical basis of Wiener filters. In this chapter we proceed in a similar fashion by first deriving the principle of orthogonality based on time averages, and then use it to derive a system of equations known as the normal equations that provides the mathematical basis of linear least-squares filters. The development of this theory will be done for the covariance method.

The cost function or the sum of the error squares in the covariance method is defined by

$$\mathscr{E}(w_0, \ldots, w_{M-1}) = \sum_{i=M}^{N} |e(i)|^2 \tag{11.6}$$

By choosing the limits on the time index i in this way, in effect, we make sure that for each value of i, all the M tap inputs of the transversal filter in Fig. 11.2 have nonzero values. As mentioned previously, the problem we have to solve is to determine the tap weights of the transversal filter of Fig. 11.2 for which the sum of error squares is minimum.

We first rewrite Eq. (11.6) as

$$\mathscr{E}(w_0, \ldots, w_{M-1}) = \sum_{i=M}^{N} e(i)e^*(i) \tag{11.7}$$

where the estimation error $e(i)$ is defined in Eq. (11.4). Let the kth tap-weight w_k be expressed in terms of its real and imaginary parts as follows:

$$w_k = a_k + jb_k, \qquad k = 0, 1, \ldots, M-1 \tag{11.8}$$

Thus, substituting Eq. (11.8) in (11.4), we get

$$e(i) = d(i) - \sum_{k=0}^{M-1} (a_k - jb_k)u(i-k) \tag{11.9}$$

We define the kth component of the gradient vector $\nabla\mathscr{E}$ as the derivative of the cost function $\mathscr{E}(w_0, \ldots, w_{M-1})$ with respect to the real and imaginary parts of tap-weight w_k, as shown by

$$\nabla_k\mathscr{E} = \frac{\partial\mathscr{E}}{\partial a_k} + j\frac{\partial\mathscr{E}}{\partial b_k} \tag{11.10}$$

Hence, substituting Eq. (11.7) in (11.10), and recognizing that the estimation error $e(i)$ is complex valued, in general, we get

$$\nabla_k\mathscr{E} = -\sum_{i=M}^{N}\left[e(i)\frac{\partial e^*(i)}{\partial a_k} + e^*(i)\frac{\partial e(i)}{\partial a_k} + je(i)\frac{\partial e^*(i)}{\partial b_k} + je(i)\frac{\partial e(i)}{\partial b_k}\right] \tag{11.11}$$

Next, differentiating $e(i)$ in Eq. (11.9) with respect to the real and imaginary parts of w_k, we get the following four partial derivatives:

$$\begin{aligned}
\frac{\partial e(i)}{\partial a_k} &= -u(i-k) \\
\frac{\partial e^*(i)}{\partial a_k} &= -u^*(i-k) \\
\frac{\partial e(i)}{\partial b_k} &= ju(i-k) \\
\frac{\partial e^*(i)}{\partial b_k} &= -ju^*(i-k)
\end{aligned} \tag{11.12}$$

Thus, the substitution of these four partial derivatives in Eq. (11.11) yields the result:

$$\nabla_k \mathscr{E} = -2 \sum_{i=M}^{N} u(i-k)e^*(i) \tag{11.13}$$

For the minimization of the cost function $\mathscr{E}(w_0, \ldots, w_{M-1})$ with respect to the tap weights $w_0, \ldots, w_{M-1}$ of the transversal filter in Fig. 11.2, we require that the following conditions be satisfied simultaneously:

$$\nabla_k \mathscr{E} = 0, \qquad k = 0, 1, \ldots, M-1 \tag{11.14}$$

Let $e_{\min}(i)$ denote the special value of the estimation error $e(i)$ that results when the cost function $\mathscr{E}(w_0, \ldots, W_{M-1})$ is minimized (i.e., the transversal filter is optimized) in accordance with Eq. (11.14). From Eq. (11.13) we then readily see that the set of conditions (11.14) is equivalent to the following:

$$\sum_{i=M}^{N} u(i-k)e^*_{\min}(i) = 0, \qquad k = 0, 1, \ldots, M-1 \tag{11.15}$$

Equation (11.15) is the mathematical description of the temporal version of the *principle of orthogonality*. The *time average*[1] on the left-hand side of Eq. (11.15) represents the cross-correlation between the tap input $u(i-k)$ and the minimum estimation error $e_{\min}(i)$ over the values of time i in the interval $[M, N]$, for a fixed value of k. Accordingly, we may state the *principle of orthogonality* as follows:

The minimum error time series $e_{\min}(i)$ is orthogonal to the time series $u(i-k)$ applied to tap k of a transversal filter of length M for $k = 0, 1, \ldots, M-1$, when the filter is operating in its least-squares condition.

This principle provides the basis of a simple *test* that we can carry out in practice to check whether or not the transversal filter is operating in its *least-square condition*. We

[1]To be precise in the use of the term "time average," we should divide the sum on the left-hand side of Eq. (11.15) by the number of terms $(N - M + 1)$ used in the summation. Clearly, such an operation has no effect on Eq. (11.15). We have chosen to ignore the inclusion of this scaling factor merely for convenience of presentation.

merely have to determine the time-averaged cross-correlation between the estimation error and the time series applied to *each* tap input of the filter. It is *only* when *all* these M cross-correlation functions are identically zero that we find the cost function $\mathscr{E}(w_0, \ldots, w_{M-1})$ is minimum.

Corollary

Let $\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_{M-1}$ denote the special values of the tap weights $w_0, w_1, \ldots, w_{M-1}$ that result when the transversal filter of Fig. 11.2 is optimized to operate in its least-squares condition. The filter output, denoted by $y_{\min}(i)$, is obtained from Eq. (11.3) to be

$$y_{\min}(i) = \sum_{k=0}^{M-1} \hat{w}_k^* u(i-k) \tag{11.16}$$

This filter output provides a *least-squares estimate* of the desired response $d(i)$; the estimate is said to be *linear* because it is a linear combination of the tap inputs $u(i), u(i-1), \ldots, u(i-M+1)$. Let $\mathscr{U}_i$ denote the space spanned by the tap inputs $u(i), \ldots, u(i-M+1)$. Let $\hat{d}(i|\mathscr{U}_i)$ denote the least-squares estimate of the desired response $d(i)$, given the tap inputs spanned by the space $\mathscr{U}_i$. We may thus write

$$\hat{d}(i|\mathscr{U}_i) = y_{\min}(i) \tag{11.17}$$

or, equivalently,

$$\hat{d}(i|\mathscr{U}_i) = \sum_{k=0}^{M-1} \hat{w}_k^* u(i-k) \tag{11.18}$$

Returning to Eq. (11.15), suppose we multiply both sides of this equation by $\hat{w}_k^*$ and then sum the result over the values of k in the interval $[0, M-1]$. We then get (after interchanging the order of summation):

$$\sum_{i=M}^{N} \left[\sum_{k=0}^{M-1} \hat{w}_k^* u(i-k) \right] e_{\min}^*(i) = 0 \tag{11.19}$$

The summation term inside the parentheses on the left-hand side of Eq. (11.19) is recognized to be the least-squares estimate $\hat{d}(i|\mathscr{U}_i)$ of Eq. (11.18). Accordingly, we may simplify Eq. (11.19) to

$$\sum_{i=M}^{N} \hat{d}(i|\mathscr{U}_i) e_{\min}^*(i) = 0 \tag{11.20}$$

Equation (11.20) is a mathematical description of the *corollary to the principle of orthogonality*. We recognize the time average on the left-hand side of Eq. (11.20) is the cross-correlation of the two time series $\hat{d}(i|\mathscr{U}_i)$ and $e_{\min}(i)$. Accordingly, we may state the corollary to the principle of orthogonality as follows:

When a transversal filter operates in its least-squares condition, the least-squares estimate of the desired response, produced at the filter output and represented by the time series

$\hat{d}(i|\mathcal{U}_i)$, and the minimum estimation error time series $e_{\min}(i)$ are orthogonal to each other over time i.

A geometric illustration of this corollary to the principle of orthogonality is deferred to Section 11.6.

11.4 MINIMUM SUM OF ERROR SQUARES

The principle of orthogonality, given in Eq. (11.15), describes the least-squares condition of the transversal filter in Fig. 11.2 when the cost function $\mathscr{E}(w_0, \ldots, w_{M-1})$ is minimized with respect to the tap weights $w_0, \ldots, w_{M-1}$ in the filter. To find the minimum value of this cost function, that is, the *minimum sum of error squares* $\mathscr{E}_{\min}$, it is obvious that we may write

$$\underbrace{d(i)}_{\substack{\text{desired}\\ \text{response}}} = \underbrace{\hat{d}(i|\mathcal{U}_i)}_{\substack{\text{estimate of}\\ \text{desired}\\ \text{response}}} + \underbrace{e_{\min}(i)}_{\substack{\text{estimation}\\ \text{error}}} \tag{11.21}$$

Hence, evaluating the energy of the time series $d(i)$ for values of time i in the interval $[M, N]$, and using the corollary to the principle of orthogonality [i.e., Eq. (11.20)], we get the simple result

$$\mathscr{E}_d = \mathscr{E}_{\text{est}} + \mathscr{E}_{\min} \tag{11.22}$$

where

$$\mathscr{E}_d = \sum_{i=M}^{N} |d(i)|^2 \tag{11.23}$$

$$\mathscr{E}_{\text{est}} = \sum_{i=M}^{N} |\hat{d}(i|\mathcal{U}_i)|^2 \tag{11.24}$$

$$\mathscr{E}_{\text{est}} = \sum_{i=M}^{N} |e_{\min}(i)|^2 \tag{11.25}$$

Rearranging Eq. (11.22), we may express the minimum sum of error squares $\mathscr{E}_{\min}$ in terms of the energy $\mathscr{E}_d$ and the energy $\mathscr{E}_{\text{est}}$, contained in the time series $d(i)$ and $\hat{d}(i|\mathcal{U}_i)$, respectively, as follows:

$$\mathscr{E}_{\min} = \mathscr{E}_d - \mathscr{E}_{\text{est}} \tag{11.26}$$

Clearly, given the specification of the desired response $d(i)$ for varying i, we may use Eq. (11.23) to evaluate the energy $\mathscr{E}_d$. As for the energy $\mathscr{E}_{\text{est}}$ contained in the time series $\hat{d}(i|\mathcal{U}_i)$ representing the estimate of the desired response, we are going to defer its evaluation to the next section.

Since $\mathscr{E}_{\min}$ is nonnegative, it follows that the second term on the right-hand side of Eq. (11.26) can never exceed $\mathscr{E}_d$. Indeed, it reaches the value of $\mathscr{E}_d$ when the measurement error $e_o(i)$ in the multiple linear regression model of Fig. 11.1 is zero for all i, which is a practical impossibility.

Another case for which $\mathscr{E}_{\min}$ equals $\mathscr{E}_d$ occurs when the least-squares problem is *underdetermined*. Such a situation arises when there are fewer data points than parameters, in which case the estimation error and therefore $\mathscr{E}_{\text{est}}$ is zero. Note, however, that when the least-squares problem is underdetermined, there is no unique solution to the problem. Discussion of this issue is deferred to the latter part of the chapter.

11.5 NORMAL EQUATIONS AND LINEAR LEAST-SQUARES FILTERS

There are two different, and yet basically equivalent, methods of describing the least-squares condition of the linear transversal filter in Fig. 11.1. The principle of orthogonality, described in Eq. (11.15), represents one method. The system of *normal equations* represents the other method; interestingly enough, the system of normal equations derives its name from the corollary to the principle of orthogonality. Naturally, we may derive this system of equations in its own independent way by formulating the gradient vector $\nabla\mathscr{E}$ in terms of the tap weights of the filter, and then solving for the tap-weight vector $\hat{\mathbf{w}}$ for which $\nabla\mathscr{E}$ is zero. Alternatively, we may derive the system of normal equations from the principle of orthogonality. We are going to pursue the latter (indirect) approach in this section, and leave the former (direct) approach to the interested reader as Problem 7.

The principle of orthogonality in Eq. (11.15) is formulated in terms of a set of tap inputs and the minimum estimation error $e_{\min}(i)$. Setting the tap weights in Eq. (11.4) to their least-squares values, we get

$$e_{\min}(i) = d(i) - \sum_{t=0}^{M-1} \hat{w}_t^* u(i-t) \tag{11.27}$$

where on the right-hand side we have purposely used t as the dummy summation index. Hence, substituting Eq. (11.27) in (11.15), and then rearranging terms, we get a system of M simultaneous equations:

$$\sum_{t=0}^{M-1} \hat{w}_t \sum_{i=M}^{N} u(i-k)u^*(i-t) = \sum_{i=M}^{N} u(i-k)d^*(i), \qquad k = 0, \ldots, M-1 \tag{11.28}$$

The two summations in Eq. (11.28) involving the index i represent time-averages, except for a scaling factor. They have the following interpretations:

1. The time average (over i) on the left-hand side of Eq. (11.28) represents the *time averaged autocorrelation function* of the tap inputs in the transversal filter of Fig. 11.2. In particular, we may write

$$\phi(t,k) = \sum_{i=M}^{N} u(i-k)u^*(i-t), \qquad 0 \le (t,k) \le M-1 \tag{11.29}$$

2. The time average (also over i) on the right-hand side of Eq. (11.28) represents the *cross-correlation* between the tap inputs and the desired response. In particular, we may write

$$z(-k) = \sum_{i=M}^{N} u(i-k)d^*(i), \qquad 0 \le k \le M-1 \tag{11.30}$$

Accordingly, we may rewrite the system of simultaneous equations (11.28) as follows:

$$\sum_{i=0}^{M-1} \hat{w}_i \phi(t, k) = z(-k), \qquad k = 0, 1, \ldots, M-1 \tag{11.31}$$

The system of equations (11.31) represents *the expanded system of the normal equations* for a linear least-squares filter.

Matrix Formulation of the Normal Equations

We may recast this system of equations in matrix form by first introducing the following definitions:

1. The M-by-M *time-averaged correlation matrix* of the tap inputs $u(i)$, $u(i-1)$, $\ldots$, $u(i-M+1)$:

$$\mathbf{\Phi} = \begin{bmatrix} \phi(0,0) & \phi(1,0) & \cdots & \phi(M-1,0) \\ \phi(0,1) & \phi(1,1) & \cdots & \phi(M-1,1) \\ \vdots & \vdots & \vdots & \vdots \\ \phi(0,M-1) & \phi(1,M-1) & \cdots & \phi(M-1,M-1) \end{bmatrix} \tag{11.32}$$

2. The M-by-1 *time-averaged cross-correlation vector* between the tap inputs $u(i)$, $u(i-1), \ldots, u(i-M+1)$ and the desired response $d(i)$:

$$\mathbf{z} = [z(0), z(-1), \ldots, z(-M+1)]^T \tag{11.33}$$

3. The M-by-1 tap-weight vector of the least-squares filter:

$$\hat{\mathbf{w}} = [\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_{M-1}]^T \tag{11.34}$$

Hence, in terms of these matrix definitions, we may now rewrite the system of M simultaneous equations (11.31) simply as

$$\mathbf{\Phi}\hat{\mathbf{w}} = \mathbf{z} \tag{11.35}$$

Equation (11.35) is *the matrix form of the normal equations for linear least-squares filters.*

Assuming that $\mathbf{\Phi}$ is nonsingular and therefore the inverse matrix $\mathbf{\Phi}^{-1}$ exists, we may solve Eq. (11.35) for the tap-weight vector of the linear least-squares filter:

$$\hat{\mathbf{w}} = \mathbf{\Phi}^{-1}\mathbf{z} \tag{11.36}$$

The condition for the existence of the inverse matrix $\mathbf{\Phi}^{-1}$ is discussed in Section 11.6.

Equation (11.36) is a very important result. In particular, it is the linear least-squares counterpart to the solution of the matrix form of the Wiener–Hopf equations (5.36). Basically, Eq. (11.36) states that the tap-weight vector $\hat{\mathbf{w}}$ of a linear least-squares filter is uniquely defined by the product of the inverse of the time-averaged correlation matrix $\mathbf{\Phi}$ of the tap inputs of the filter and the time-averaged cross-correlation vector $\mathbf{z}$ between the tap inputs and the desired response. Indeed, this equation is fundamental to the development of all recursive formulations of the linear least-squares filter, as pursued in subsequent chapters of the book.

Minimum Sum of Error Squares

Equation (11.26), derived in the preceding section, defines the minimum sum of error squares $\mathscr{E}_{\min}$. We now complete the evaluation of $\mathscr{E}_{\min}$, expressed as the difference between the energy $\mathscr{E}_d$ of the desired response and the energy $\mathscr{E}_{\text{est}}$ of its estimate. Usually, $\mathscr{E}_d$ is determined from the time series representing the desired response. To evaluate $\mathscr{E}_{\text{est}}$, we write

$$\begin{aligned}\mathscr{E}_{\text{est}} &= \sum_{i=M}^{N} |\hat{d}(i|\mathcal{U}_i)|^2 \\ &= \sum_{i=M}^{N} \sum_{t=0}^{M-1} \sum_{k=0}^{M-1} \hat{w}_t \hat{w}_k^* u(i-k)u^*(i-t) \\ &= \sum_{t=0}^{M-1} \sum_{k=0}^{M-1} \hat{w}_t \hat{w}_k^* \sum_{i=M}^{N} u(i-k)u^*(i-t)\end{aligned} \tag{11.37}$$

where, in the second line, we have made use of Eq. (11.18). The inner summation over time i in the final line of Eq. (11.37) represents the time-averaged autocorrelation function $\phi(t, k)$ [see Eq. (11.29)]. Hence, we may rewrite Eq. (11.37) as

$$\begin{aligned}\mathscr{E}_{\text{est}} &= \sum_{t=0}^{M-1} \sum_{k=0}^{M-1} \hat{w}_t^* \phi(t, k) \hat{w}_t \\ &= \hat{\mathbf{w}}^H \mathbf{\Phi} \hat{\mathbf{w}}\end{aligned} \tag{11.38}$$

where $\hat{\mathbf{w}}$ is the least-squares tap-weight vector and $\mathbf{\Phi}$ is the time-averaged correlation matrix of the tap inputs. We may further simplify the formula for $\mathscr{E}_{\text{est}}$ by noting that from the normal equations (11.35), the matrix product $\mathbf{\Phi}\hat{\mathbf{w}}$ equals the cross-correlation vector $\mathbf{z}$. Accordingly, we have

$$\begin{aligned}\mathscr{E}_{\text{est}} &= \hat{\mathbf{w}}^H \mathbf{z} \\ &= \mathbf{z}^H \hat{\mathbf{w}}\end{aligned} \tag{11.39}$$

Finally, substituting Eq. (11.39) in (11.26), and then using Eq. (11.36) for $\hat{\mathbf{w}}$, we get

$$\begin{aligned}\mathscr{E}_{\min} &= \mathscr{E}_d - \mathbf{z}^H\hat{\mathbf{w}} \\ &= \mathscr{E}_d - \mathbf{z}^H\boldsymbol{\Phi}^{-1}\mathbf{z}\end{aligned} \tag{11.40}$$

Equations (11.40) is the formula for the minimum sum of error squares, expressed in terms of three known quantities: the energy $\mathscr{E}_d$ of the desired response, the time-averaged correlation matrix $\boldsymbol{\Phi}$ of the tap inputs, and the time-averaged cross-correlation vector $\mathbf{z}$ between the tap inputs and the desired response.

11.6 TIME-AVERAGED CORRELATION MATRIX Φ

The time-averaged correlation matrix or simply the correlation matrix $\boldsymbol{\Phi}$ of the tap inputs is shown in its expanded form in Eq. (11.32), with the element $\phi(t, k)$ defined in Eq. (11.29). The index k in $\phi(t, k)$ refers to the row number in the matrix $\boldsymbol{\Phi}$, and t refers to the column number. Let the M-by-1 tap-input vector $\mathbf{u}(i)$ be defined by

$$\mathbf{u}(i) = [u(i), u(i-1), \ldots, u(i-M+1)]^T \tag{11.41}$$

Hence, we may use Eqs. (11.29) and (11.41) to redefine the correlation matrix $\boldsymbol{\Phi}$ as the time average of the outer product $\mathbf{u}(i)\mathbf{u}^H(i)$ over i as follows:

$$\boldsymbol{\Phi} = \sum_{i=M}^{N} \mathbf{u}(i)\mathbf{u}^H(i) \tag{11.42}$$

To restate what we said earlier under footnote 1, the summation in Eq. (11.42) should be divided by the scaling factor $(N - M + 1)$ for the correlation matrix $\boldsymbol{\Phi}$ to be a time average in precise terms. In the statistics literature, this scaled form of $\boldsymbol{\Phi}$ is referred to as the *sample correlation matrix*. In any event, on the basis of the definition given in Eq. (11.42), we may readily establish the following properties of the correlation matrix:

Property 1. *The correlation matrix $\boldsymbol{\Phi}$ is Hermitian; that is*

$$\boldsymbol{\Phi}^H = \boldsymbol{\Phi}$$

This property follows directly from Eq. (11.42).

Property 2. *The correlation matrix $\boldsymbol{\Phi}$ is nonnegative definite; that is,*

$$\mathbf{x}^H\boldsymbol{\Phi}\mathbf{x} \geq 0$$

for any M-by-1 vector $\mathbf{x}$.

Using the definition of Eq. (11.42), we may write

$$\begin{aligned}\mathbf{x}^H\mathbf{\Phi}\mathbf{x} &= \sum_{i=M}^{N} \mathbf{x}^H\mathbf{u}(i)\mathbf{u}^H(i)\mathbf{x} \\ &= \sum_{i=M}^{N} [\mathbf{x}^H\mathbf{u}(i)][\mathbf{x}^H\mathbf{u}(i)]^H \\ &= \sum_{i=M}^{N} |\mathbf{x}^H\mathbf{u}(i)|^2 \geq 0\end{aligned}$$

which proves Property 2. The fact that the correlation matrix $\mathbf{\Phi}$ is nonnegative definite means that its determinant and all principal minors are nonnegative. When the above condition is satisfied with the inequality sign, the determinant of $\mathbf{\Phi}$ and its principal minors are likewise nonzero. In the latter case, $\mathbf{\Phi}$ is nonsingular and the inverse $\mathbf{\Phi}^{-1}$ exists.

Property 3. *The eigenvalues of the correlation matrix $\mathbf{\Phi}$ are all real and nonnegative.*

The real requirement on the eigenvalues of $\mathbf{\Phi}$ follows from Property 1. The fact that all these eigenvalues are also nonnegative follows from Property 2.

Property 4. *The correlation matrix is the product of two rectangular Toeplitz matrices that are the Hermitian transpose of each other.*

The correlation matrix $\mathbf{\Phi}$ is, in general, non-Toeplitz, which is clearly seen by examining the expanded form of the correlation matrix given in Eq. (11.32). The elements on the main diagonal, $\phi(0, 0), \phi(1, 1), \ldots, \phi(M-1, M-1)$, have different values; this also applies to secondary diagonal above or below the main diagonal. However, the matrix $\mathbf{\Phi}$ has a special structure in the sense that it is the product of two Toeplitz rectangular matrices. To prove this property, we first use Eq. (11.42) to express the matrix $\mathbf{\Phi}$ as follows:

$$\mathbf{\Phi} = [\mathbf{u}(M), \mathbf{u}(M+1), \ldots, \mathbf{u}(N)] \begin{bmatrix} \mathbf{u}^H(M) \\ \mathbf{u}^H(M-1) \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{u}^H(N) \end{bmatrix} \tag{11.43}$$

Next, for convenience of presentation, we introduce a *data matrix* $\mathbf{A}$, whose Hermitian transpose is defined by

$$\begin{aligned}\mathbf{A}^H = &\quad [\mathbf{u}(M), \quad \mathbf{u}(M+1), \quad \cdots, \mathbf{u}(N)] \\ &= \begin{bmatrix} u(M) & u(M+1) & \cdots & u(N) \\ u(M-1) & u(M) & \cdots & u(N-1) \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ u(1) & u(2) & \cdots & u(N-M+1) \end{bmatrix}\end{aligned} \tag{11.44}$$

The expanded matrix on the right-hand side of Eq. (11.44) is recognized to be the matrix of input data for the covariance method of data windowing (see point 1 of Section 11.2). Thus, using the definition of Eq. (11.44), we may rewrite Eq. (11.43) in the compact form

$$\boldsymbol{\Phi} = \mathbf{A}^H\mathbf{A} \tag{11.45}$$

From the expanded form of the matrix given in the second line of Eq. (11.44), we see that $\mathbf{A}^H$ consists of an M- by-$(N - M + 1)$ *rectangular Toeplitz matrix*. The data matrix $\mathbf{A}$ itself is likewise an $(N - M + 1)$-by-M rectangular Toeplitz matrix. According to Eq. (11.45), therefore, the correlation matrix $\boldsymbol{\Phi}$ is the product of two rectangular Toeplitz matrices that are the Hermitian transpose of each other: this completes the proof of Property 4.

11.7 REFORMULATION OF THE NORMAL EQUATIONS IN TERMS OF DATA MATRICES

The system of normal equations for a least-squares transversal filter is given by Eq. (11.35) in terms of the correlation matrix $\boldsymbol{\Phi}$ and the cross-correlation vector $\mathbf{z}$. We may reformulate the normal equations in terms of data matrices by using Eq. (11.45) for the correlation matrix $\boldsymbol{\Phi}$ of the tap inputs, and a corresponding relation for the cross-correlation vector $\mathbf{z}$ between the tap inputs and the desired response. To do this, we introduce a *desired data vector* $\mathbf{d}$, consisting of the *desired response* $d(i)$ for values of i in the interval $[M, N]$; in particular, we define

$$\mathbf{d}^H = [d(M), d(M+1), \ldots, d(N)] \tag{11.46}$$

Note that we have purposely used Hermitian transposition rather than ordinary transposition in the definition of vector $\mathbf{d}$ to be consistent with the definition of the data matrix $\mathbf{A}$ in Eq. (11.44). With the definitions of Eqs. (11.44) and (11.46) at hand, we may now use Eqs. (11.30) and (11.33) to express the cross-correlation vector $\mathbf{z}$ as

$$\mathbf{z} = \mathbf{A}^H\mathbf{d} \tag{11.47}$$

Furthermore, we may use Eqs. (11.45) and (11.47) in (11.35), and so express the system of normal equations in terms of the data matrix $\mathbf{A}$ and the desired data vector $\mathbf{d}$ as

$$\mathbf{A}^H\mathbf{A}\mathbf{w} = \mathbf{A}^H\mathbf{d}$$

Hence, the system of equations used in the minimization of the cost function $\mathscr{E}$ may be represented by $\mathbf{A}\mathbf{w} = \mathbf{d}$. Furthermore, assuming that the inverse matrix $(\mathbf{A}^H\mathbf{A})^{-1}$ exists, we may solve this system of equations by expressing the tap-weight vector $\hat{\mathbf{w}}$ as

$$\hat{\mathbf{w}} = (\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H\mathbf{d} \tag{11.48}$$

We may complete the reformulation of our results for the linear least-squares problem in terms of the data matrices $\mathbf{A}$ and $\mathbf{d}$ by using (1) the definitions of Eqs. (11.45) and (11.47) in (11.40), and (2) the definitions of Eq. (11.46) in (11.23). By so doing, we may rewrite the formula for the minimum sum of error squares as

$$\mathscr{E}_{\min} = \mathbf{d}^H\mathbf{d} - \mathbf{d}^H\mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H\mathbf{d} \tag{11.49}$$

Although this formula looks somewhat cumbersome, its nice feature is that it is expressed explicitly in terms of the data matrix $\mathbf{A}$ and the desired data vector $\mathbf{d}$.

Projection Operator

Equation (11.48) defines the least-squares tap-weight vector $\hat{\mathbf{w}}$ in terms of the data matrix $\mathbf{A}$ and the desired data vector $\mathbf{d}$. The least-squares estimate of $\mathbf{d}$ is therefore given by

$$\begin{aligned}\hat{\mathbf{d}} &= \mathbf{A}\hat{\mathbf{w}} \\ &= \mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H\mathbf{d}\end{aligned} \tag{11.50}$$

Accordingly, we may view the multiple matrix product $\mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H$ as a *projection operator* onto the linear space spanned by the columns of the data matrix $\mathbf{A}$, which is the same space $\mathscr{U}_i$ mentioned previously for $i = N$. Denoting this projection operator by $\mathbf{P}$, we may thus write

$$\mathbf{P} = \mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H \tag{11.51}$$

The matrix difference

$$\mathbf{I} - \mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H = \mathbf{I} - \mathbf{P}$$

is the *orthogonal complement projector*. Note that both the projection operator and its complement are uniquely determined by the data matrix A. The projection operator, $\mathbf{P}$, applied to the desired data vector $\mathbf{d}$, yields the corresponding estimate $\hat{\mathbf{d}}$. On the other hand, the orthogonal complement projector, $\mathbf{I} - \mathbf{P}$, applied to the desired data vector $\mathbf{d}$, yields the estimation error vector $\mathbf{e}_{min} = \mathbf{d} - \hat{\mathbf{d}}$. Figure 11.3 illustrates the functions of the projection operator and the orthogonal complement projector as described herein.

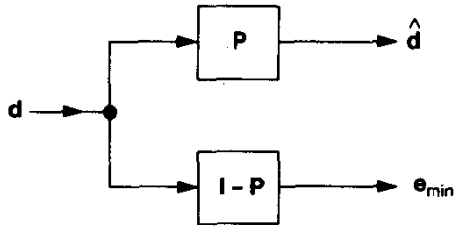


Figure 11.3 Projection operator **P** and orthogonal complement projector **I** − **P**.

Example 1

Consider the example of a linear least-squares filter with two taps (i.e., $M = 2$) and a *real-valued* input time series consisting of four samples (i.e., $N = 4$), hence $N - M + 1 = 3$. The input data matrix **A** and the desired data vector **d** have the following values:

$$\mathbf{A} = \begin{bmatrix} u(2) & u(1) \\ u(3) & u(2) \\ u(4) & u(3) \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 3 \\ 1 & 2 \\ -1 & 1 \end{bmatrix}$$

$$\mathbf{d} = \begin{bmatrix} d(2) \\ d(3) \\ d(4) \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ 1 \\ 1/34 \end{bmatrix}$$

The purpose of this example is to evaluate the projection operator and the orthogonal complement projector, and use them to illustrate the principle of orthogonality.

The use of Eq. (11.51), reformulated for real data, yields the value of the projection operator **P** as

$$\mathbf{P} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$$

$$= \frac{1}{35}\begin{bmatrix} 26 & 15 & -2 \\ 15 & 10 & 5 \\ -3 & 5 & 34 \end{bmatrix}$$

The corresponding value of the orthogonal complement projector is

$$\mathbf{I} - \mathbf{P} = \frac{1}{35}\begin{bmatrix} 9 & -15 & 3 \\ -15 & 25 & -5 \\ -3 & -5 & 1 \end{bmatrix}$$

Accordingly, the estimate of the desired data vector and the estimation error vector have the following values, respectively:

$$\hat{\mathbf{d}} = \mathbf{P}\mathbf{d} = \begin{bmatrix} 1.91 \\ 1.15 \\ 0 \end{bmatrix}$$

$$\mathbf{e}_{\min} = (\mathbf{I} - \mathbf{P})\mathbf{d} = \begin{bmatrix} 0.09 \\ -0.15 \\ 0.03 \end{bmatrix}$$

Figure 11.4 depicts three-dimensional geometric representations of the vectors $\hat{\mathbf{d}}$ and $\mathbf{e}_{\min}$. This figure clearly shows that these two vectors are *normal* (i.e., *perpendicular*) to each other in accordance with the corollary to the principle of orthogonality, hence the terminology "normal" equations. This condition is the geometric portrayal of the fact that in a linear least-squares filter the inner product $\mathbf{e}_{\min}^H\mathbf{d}$ is zero. Figure 11.4 also depicts the desired data vector $\mathbf{d}$ as the "vector sum" of the estimate $\hat{\mathbf{d}}$ and the error $\mathbf{e}_{\min}$. Note also that the vector $\mathbf{e}_{\min}$ is orthogonal to span($\mathbf{A}$), defined as the set of all linear combinations of the column vectors of the data matrix $\mathbf{A}$. The estimate $\hat{\mathbf{d}}$ is just one vector in span($\mathbf{A}$).

Uniqueness Theorem

The linear least-squares problem of minimizing the sum of error squares, $\mathscr{E}(n)$, always has a solution. That is, for given values of the data matrix $\mathbf{A}$ and the desired data vector $\hat{\mathbf{d}}$, we can always find a vector $\hat{\mathbf{w}}$ that satisfies the normal equations. It is therefore important that we know *if* and *when* the solution is *unique*. This requirement is covered by the following *uniqueness theorem* (Stewart, 1973):

The least-squares estimate $\hat{\mathbf{w}}$ is unique if and only if the nullity of the data matrix A equals zero.

Let $\mathbf{A}$ be a K-by-M matrix; in the case of the data matrix $\mathbf{A}$ defined in Eq. (11.44), we have $K = N - M + 1$. We define the *null space* of matrix $\mathbf{A}$, written as $\mathcal{N}(\mathbf{A})$, as the space of all vectors $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} = \mathbf{0}$. We define the *nullity* of matrix $\mathbf{A}$, written as null($\mathbf{A}$), as the dimension of the null space $\mathcal{N}(\mathbf{A})$. In general, we find that

$$\text{null}(\mathbf{A}) \neq \text{null}(\mathbf{A}^H).$$

In light of the uniqueness theorem, which is intuitively satisfying, we may expect a unique solution to the linear least-squares problem *only* when the data matrix $\mathbf{A}$ has *linearly independent columns;* that is, when the data matrix $\mathbf{A}$ is of *full column rank*. This implies that the matrix $\mathbf{A}$ has at least as many rows as columns; that is, $(N - M + 1) \geq M$. This latter condition means that the system of equations represented by $\mathbf{A}\mathbf{w} = \mathbf{d}$ used in the minimization is *overdetermined*, in that it has more equations than unknowns. Thus,

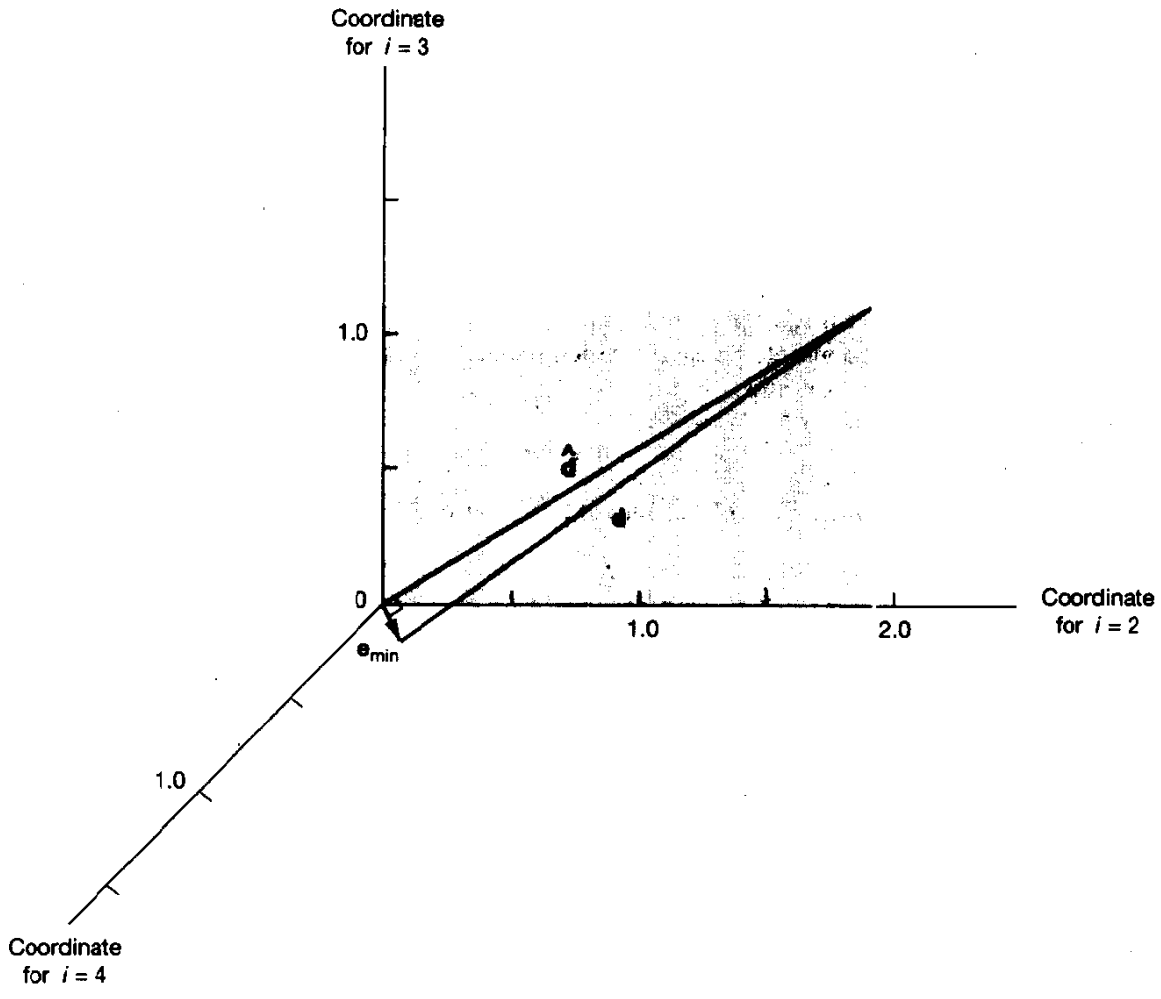


Figure 11.4 Three-dimensional geometric interpretations of vectors $\mathbf{d}$, $\hat{\mathbf{d}}$, and $\mathbf{e}_{min}$.

provided that the data matrix $\mathbf{A}$ is of full column rank, the M-by-M matrix $\mathbf{A}^H\mathbf{A}$ is *non-singular*, and the least-squares estimate has the unique value given in Eq. (11.48).

When, however, the matrix $\mathbf{A}$ has *linearly dependent columns*, that is, it is *rank deficient*, the nullity of the matrix $\mathbf{A}$ is nonzero, and the result is that an infinite number of solutions can be found for minimizing the sum of error squares. In such a situation, the linear least-squares problem becomes quite involved, in that we now have the new problem of deciding which particular solution to adopt. We defer discussion of this issue to the latter part of the chapter. In the meantime, we assume that the data matrix $\mathbf{A}$ is of full column rank, so that the least-squares estimate $\hat{\mathbf{w}}$ has the unique value defined by Eq. (11.48).

11.8 PROPERTIES OF LEAST-SQUARES ESTIMATES

The method of least squares has a strong intuitive feel that is reinforced by several outstanding properties of the method, assuming that the data matrix **A** is known with *no* uncertainty. These properties, four in number, are described next (Miller, 1974; Goodwin and Payne, 1977).

Property 1. *The least-squares estimate* $\hat{\mathbf{w}}$ is unbiased, provided that the measurement error process $e_o(i)$ *has zero mean.*

From the multiple linear regression model of Fig. 11.1, we have [using the definitions of Eqs. (11.44) and (11.46)]

$$\mathbf{d} = \mathbf{A}\mathbf{w}_o + \boldsymbol{\epsilon}_o \tag{11.52}$$

Hence, substituting Eq. (11.52) into (11.48), we may express the least-squares estimate $\hat{\mathbf{w}}$ as

$$\begin{aligned}\hat{\mathbf{w}} &= (\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H\mathbf{A}\mathbf{w}_o + (\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H\boldsymbol{\epsilon}_o \\ &= \mathbf{w}_o + (\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H\boldsymbol{\epsilon}_o\end{aligned} \tag{11.53}$$

The matrix product $(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H$ is a known quantity, since the data matrix **A** is completely defined by the set of given observations $u(1), u(2), \ldots, u(N)$; see Eq. (11.44). Hence, if the measurement error process $e_o(i)$ or, equivalently, the measurement error vector $\boldsymbol{\epsilon}_o$ has zero mean, we find by taking the expectation of both sides of Eq. (11.53) that the estimate $\hat{\mathbf{w}}$ is *unbiased*; that is,

$$E[\hat{\mathbf{w}}] = \mathbf{w}_o \tag{11.54}$$

Property 2. *When the measurement error process* $e_o(i)$ *is white with zero mean and variance* σ^2, *the covariance matrix of the least-squares estimate* $\hat{\mathbf{w}}$ equals $\sigma^2\boldsymbol{\Phi}^{-1}$.

Using the relation of Eq. (11.53), we find that the covariance matrix of the least-squares estimate $\hat{\mathbf{w}}$ equals

$$\begin{aligned}\text{cov}[\hat{\mathbf{w}}] &= E[(\hat{\mathbf{w}} - \mathbf{w}_o)(\hat{\mathbf{w}} - \mathbf{w}_o)^H] \\ &= E[(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H\boldsymbol{\epsilon}_o\boldsymbol{\epsilon}_o^H\mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1}] \\ &= (\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H E[\boldsymbol{\epsilon}_o\boldsymbol{\epsilon}_o^H]\mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1}\end{aligned} \tag{11.55}$$

With the measurement error process $e_o(i)$ assumed to be white with zero mean and variance σ^2, we have

$$E[\boldsymbol{\epsilon}_o\boldsymbol{\epsilon}_o^H] = \sigma^2\mathbf{I} \tag{11.56}$$

where $\mathbf{I}$ is the identity matrix. Hence, Eq. (11.55) reduces to

$$\begin{aligned}\text{cov}[\hat{\mathbf{w}}] &= \sigma^2(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H\mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1} \\ &= \sigma^2(\mathbf{A}^H\mathbf{A})^{-1} \\ &= \sigma^2\boldsymbol{\Phi}^{-1}\end{aligned} \tag{11.57}$$

which proves Property 2.

Property 3. *When the measurement error process $e_o(i)$ is white with zero mean, the least-squares estimate $\hat{\mathbf{w}}$ is the best linear unbiased estimate.*

Consider any linear unbiased estimator $\tilde{\mathbf{w}}$ that is defined by

$$\tilde{\mathbf{w}} = \mathbf{B}\mathbf{d} \tag{11.58}$$

where $\mathbf{B}$ is an M-by-$(N - M + 1)$ matrix. Substituting Eq. (11.52) into (11.58), we get

$$\tilde{\mathbf{w}} = \mathbf{B}\mathbf{A}\mathbf{w}_o + \mathbf{B}\boldsymbol{\epsilon}_o \tag{11.59}$$

With the measurement error vector $\boldsymbol{\epsilon}_o$ having zero mean in accordance with Property 1, we find that the expected value of $\tilde{\mathbf{w}}$ equals

$$E[\tilde{\mathbf{w}}] = \mathbf{B}\mathbf{A}\mathbf{w}_o$$

For the linear estimator $\tilde{\mathbf{w}}$ to be unbiased, we therefore require that the matrix $\mathbf{B}$ satisfy the condition

$$\mathbf{B}\mathbf{A} = \mathbf{I}$$

Accordingly, we may rewrite Eq. (11.59) as follows:

$$\tilde{\mathbf{w}} = \mathbf{w}_o + \mathbf{B}\boldsymbol{\epsilon}_o$$

The covariance matrix of $\tilde{\mathbf{w}}$ equals

$$\begin{aligned}\text{cov}[\tilde{\mathbf{w}}] &= E[(\tilde{\mathbf{w}} - \mathbf{w}_o)(\tilde{\mathbf{w}} - \mathbf{w}_o)^H] \\ &= E[\mathbf{B}\boldsymbol{\epsilon}_o\boldsymbol{\epsilon}_o^H\mathbf{B}^H] \\ &= \sigma^2\mathbf{B}\mathbf{B}^H\end{aligned} \tag{11.60}$$

Here, we have made use of Eq. (11.56), which describes the assumption that the elements of the measurement error vector $\boldsymbol{\epsilon}_o$ are uncorrelated and have a common variance σ^2; that is, the measurement error process $e_o(i)$ is white. We next define a new matrix $\boldsymbol{\Psi}$ in terms of $\mathbf{B}$ as

$$\boldsymbol{\Psi} = \mathbf{B} - (\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H \tag{11.61}$$

Now we form the matrix product $\Psi\Psi^H$ and note that $\mathbf{BA} = \mathbf{I}$:

$$\begin{aligned}\Psi\Psi^H &= [\mathbf{B} - (\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H][\mathbf{B}^H - \mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1}] \\ &= \mathbf{B}\mathbf{B}^H - \mathbf{B}\mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1} - (\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H\mathbf{B}^H + (\mathbf{A}^H\mathbf{A})^{-1} \\ &= \mathbf{B}\mathbf{B}^H - (\mathbf{A}^H\mathbf{A})^{-1}\end{aligned}$$

Since the diagonal elements of $\Psi\Psi^H$ are always nonnegative, we may use this relation to write

$$\sigma^2 \operatorname{diag}[\mathbf{B}\mathbf{B}^H] \geq \sigma^2 \operatorname{diag}[(\mathbf{A}^H\mathbf{A})^{-1}] \tag{11.62}$$

The term $\sigma^2\mathbf{B}\mathbf{B}^H$ equals the covariance matrix of the linear estimate $\tilde{\mathbf{w}}$, as in Eq. (11.60). From Property 2, we also know that the term $\sigma^2(\mathbf{A}^H\mathbf{A})^{-1}$ equals the covariance matrix of the least-squares estimate $\hat{\mathbf{w}}$. Thus, Eq. (11.62) shows that within the class of linear unbiased estimates the least-squares estimate $\hat{\mathbf{w}}$ is the "best" estimate of the unknown parameter vector $\mathbf{w}_o$ of the multiple linear regression model, in the sense that each element of $\hat{\mathbf{w}}$ has the smallest possible variance. Accordingly, when the measurement error process e_o contained in this model is white with zero mean, the least-squares estimate $\hat{\mathbf{w}}$ is the *b*est *l*inear *u*nbiased *e*stimate (BLUE).

Thus far we have not made any assumption about the statistical distribution of the measurement error process $e_o(i)$ other than that it is a zero mean white-noise process. By making the further assumption that the process $e_o(i)$ is *Gaussian* distributed, we obtain a stronger result on the optimality of the linear least-squares estimate, as discussed next.

Property 4. *When the measurement error process $e_o(i)$ is white and Gaussian, with zero mean, the least-squares estimate $\hat{\mathbf{w}}$ achieves the Cramér–Rao lower bound for unbiased estimates.*

Let $f_E(\boldsymbol{\epsilon}_o)$ denote the joint probability density function of the measurement error vector $\boldsymbol{\epsilon}_o$. Let $\hat{\mathbf{w}}$ denote any unbiased estimate of the unknown parameter vector $\hat{\mathbf{w}}_o$ of the multiple linear regression model. Then the covariance matrix of $\hat{\mathbf{w}}$ satisfies the inequality

$$\operatorname{cov}[\hat{\mathbf{w}}] \geq \mathbf{J}^{-1} \tag{11.63}$$

where

$$\operatorname{cov}[\hat{\mathbf{w}}] = E[(\hat{\mathbf{w}} - \mathbf{w}_o)(\hat{\mathbf{w}} - \mathbf{w}_o)^H] \tag{11.64}$$

The matrix $\mathbf{J}$ is called *Fisher's information matrix*; it is defined by[2]

$$\mathbf{J} = E\left[\left(\frac{\partial l}{\partial \mathbf{w}_o^*}\right)\left(\frac{\partial l}{\partial \mathbf{w}_o^T}\right)\right] \tag{11.65}$$

where l is the *log-likelihood function*, that is, the natural logarithm of the joint probability density of $\boldsymbol{\epsilon}_o$, as shown by

$$l = \ln f_E(\boldsymbol{\epsilon}_o) \tag{11.66}$$

[2]Fisher's information matrix is discussed in Appendix D for the case of real parameters.

Since the measurement error process $e_o(n)$ is white, the elements of the vector $\boldsymbol{\epsilon}_o$ are uncorrelated. Furthermore, since the process $e_o(n)$ is Gaussian, the elements of $\boldsymbol{\epsilon}_o$ are statistically independent. With $e_o(i)$ assumed to be complex with a mean of zero and variance σ^2, we have (see Section 2.11)

$$f_E(\boldsymbol{\epsilon}_o) = \frac{1}{(\pi\sigma^2)^{(N-M+1)}} \exp\left[-\frac{1}{\sigma^2}\sum_{i=M}^{N} |e_o(i)|^2\right] \tag{11.67}$$

The log-likelihood function is therefore

$$\begin{aligned} l &= F - \frac{1}{\sigma^2}\sum_{i=M}^{N} |e_o(i)|^2 \\ &= F - \frac{1}{\sigma^2}\boldsymbol{\epsilon}_o^H\boldsymbol{\epsilon}_o \end{aligned} \tag{11.68}$$

where F is a constant defined by

$$F = -(N - M + 1)\ln(\pi\sigma^2)$$

From Eq. (11.52), we have

$$\boldsymbol{\epsilon}_o = \mathbf{d} - \mathbf{A}\mathbf{w}_o$$

Using this relation in Eq. (11.68), we may rewrite l in terms of $\mathbf{w}_o$ as

$$l = F - \frac{1}{\sigma^2}\mathbf{d}^H\mathbf{d} + \frac{1}{\sigma^2}\mathbf{w}_o^H\mathbf{A}^H\mathbf{d} + \frac{1}{\sigma^2}\mathbf{d}^H\mathbf{A}\mathbf{w}_o - \frac{1}{\sigma^2}\mathbf{w}_o^H\mathbf{A}^H\mathbf{A}\mathbf{w}_o \tag{11.69}$$

Differentiating the real-valued log-likelihood function l with respect to the complex-valued unknown parameter vector $\mathbf{w}_o$ in accordance with the notation described in Appendix B, we get

$$\begin{aligned} \frac{\partial l}{\partial \mathbf{w}_o^*} &= \frac{1}{\sigma^2}\mathbf{A}^H(\mathbf{d} - \mathbf{A}\mathbf{w}_o) \\ &= \frac{1}{\sigma^2}\mathbf{A}^H\boldsymbol{\epsilon}_o \end{aligned} \tag{11.70}$$

Thus, substituting Eq. (11.70) into (11.65) yields Fisher's information matrix for the problem at hand as

$$\begin{aligned} \mathbf{J} &= \frac{1}{\sigma^4}E[\mathbf{A}^H\boldsymbol{\epsilon}_o\boldsymbol{\epsilon}_o^H\mathbf{A}] \\ &= \frac{1}{\sigma^4}\mathbf{A}^H E[\boldsymbol{\epsilon}_o\boldsymbol{\epsilon}_o^H]\mathbf{A} \\ &= \frac{1}{\sigma^2}\mathbf{A}^H\mathbf{A} \\ &= \frac{1}{\sigma^2}\boldsymbol{\Phi} \end{aligned} \tag{11.71}$$

where, in the third line, we have made use of Eq. (11.56) describing the assumption that the measurement error process $e_o(i)$ is white with zero mean and variance σ^2. Accordingly, the use of Eq. (11.63) shows that the covariance matrix of the unbiased estimate $\tilde{\mathbf{w}}$ satisfies the inequality

$$\text{cov}[\tilde{\mathbf{w}}] \geq \sigma^2 \boldsymbol{\Phi}^{-1} \tag{11.72}$$

However, from Property 2, we know that $\sigma^2\boldsymbol{\Phi}^{-1}$ equals the covariance matrix of the least-squares estimate $\hat{\mathbf{w}}$. Accordingly, $\hat{\mathbf{w}}$ achieves the Cramér–Rao lower bound. Moreover, using Property 1, we conclude that when the measurement error process $e_o(i)$ is a zero-mean white Gaussian noise process, the least-squares estimate $\hat{\mathbf{w}}$ is a *m*inimum *v*ariance *u*nbiased *e*stimate (MVUE).

11.9 PARAMETRIC SPECTRUM ESTIMATION

The method of least squares is particularly well suited for solving *parametric spectrum estimation* problems. In this section we study this important application of the method of least squares. We first consider the case of *autoregressive (AR) spectrum estimation*, assuming the use of an AR model of *known order*. From the discussion of linear prediction presented in Chapter 6, we know that there is a one-to-one correspondence between the coefficients of a prediction-error filter and those of an AR model of similar order. Next, we consider the case of *minimum variance distortionless response (MVDR) spectrum estimation*. In this second case, we have a constrained optimization problem to solve.

AR Spectrum Estimation

The specific estimation procedure described herein relies on the combined use of *forward and backward linear prediction (FBLP)*.[3] Since the method of least squares is basically a *block* estimation procedure, we may therefore view the FBLP algorithm as an alternative to the Burg algorithm (described in Section 6.15) for solving AR modeling problems. There are, however, three basic differences between the FBLP and the Burg algorithms:

1. The FBLP algorithm estimates the coefficients of a *transversal-equivalent model* for the input data, whereas the Burg algorithm estimates the reflection coefficients of a *lattice-equivalent model*.
2. In the method of least squares, and therefore the FBLP algorithm, no assumptions are made concerning the statistics of the input data. The Burg algorithm, on the other hand, exploits the decoupling property of a multistage lattice predictor, which, in turn, assumes wide-sense stationarity of the input data. Accordingly, the

[3]The first application of the FBLP method to the design of a linear predictor that has a transversal filter structure, in accordance with the method of least squares, was developed independently by Ulrych and Clayton (1976) and Nuttall (1976).

FBLP algorithm does not suffer from some of the anomalies that are known to arise in the application of the Burg algorithm.[4]

3. The Burg algorithm yields a minimum-phase solution in the sense that the reflection coefficients of the equivalent lattice predictor have a magnitude less than or equal to unity. The FBLP algorithm, on the other hand, does *not* guarantee such a solution. In spectrum estimation, however, the lack of a minimum-phase solution is of no particular concern.

Consider then the *forward linear predictor*, shown in Fig. 11.5(a). The tap weights of the predictor are denoted by $\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_M$ and the tap inputs by $u(i-1), u(i-2), \ldots, u(i-M)$, respectively. The forward prediction error, denoted by $f_M(i)$, equals

$$f_M(i) = u(i) - \sum_{k=1}^{M} \hat{w}_k^* u(i-k) \tag{11.73}$$

The first term, $u(i)$, represents the desired response. The convolution sum, constituting the second term, represents the predictor output; it consists of the sum of scalar inner products. Using matrix notation, we may also express the forward prediction error as

$$f_M(i) = u(i) - \mathbf{w}^H \mathbf{u}(i-1) \tag{11.74}$$

where $\hat{\mathbf{w}}$ is the M-by-1 tap-weight vector of the predictor:

$$\mathbf{w} = [\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_M]^T$$

and $\mathbf{u}(i-1)$ is the corresponding tap-input vector:

$$\mathbf{u}(i-1) = [u(i-1), u(i-2), \ldots, u(i-M)]^T$$

Consider next Fig. 11.5(b), which depicts the reconfiguration of the predictor so that it performs backward linear prediction. We have *purposely* retained $\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_M$ as the tap weights of the predictor. The change in the format of the tap inputs is inspired by the discussion presented in Section 6.2 on backward linear prediction and its relation to forward linear prediction for the case of wide-sense stationary inputs. In particular, the tap inputs in the predictor of Fig. 11.5(b) differ from those of the forward linear predictor of Fig. 11.5(a) in two respects:

1. The tap inputs in Fig. 11.5(b) are *time reversed*, in that they appear from right to left whereas in Fig. 11.5(a) they appear from left to right.

[4]For example, when the Burg algorithm is used to estimate the frequency of an unknown sine wave in additive noise, under certain conditions a phenomenon commonly referred to as *spectral line splitting* may occur. This phenomenon refers to the occurrence of two (or more) closely spaced spectral peaks where there should only be a single peak; for a discussion of spectral line splitting, see Marple (1987), Kay (1988), and Haykin (1989); the original reference is Fougere et al. (1976). This anomaly, however, does not arise in the application of the FBLP algorithm.

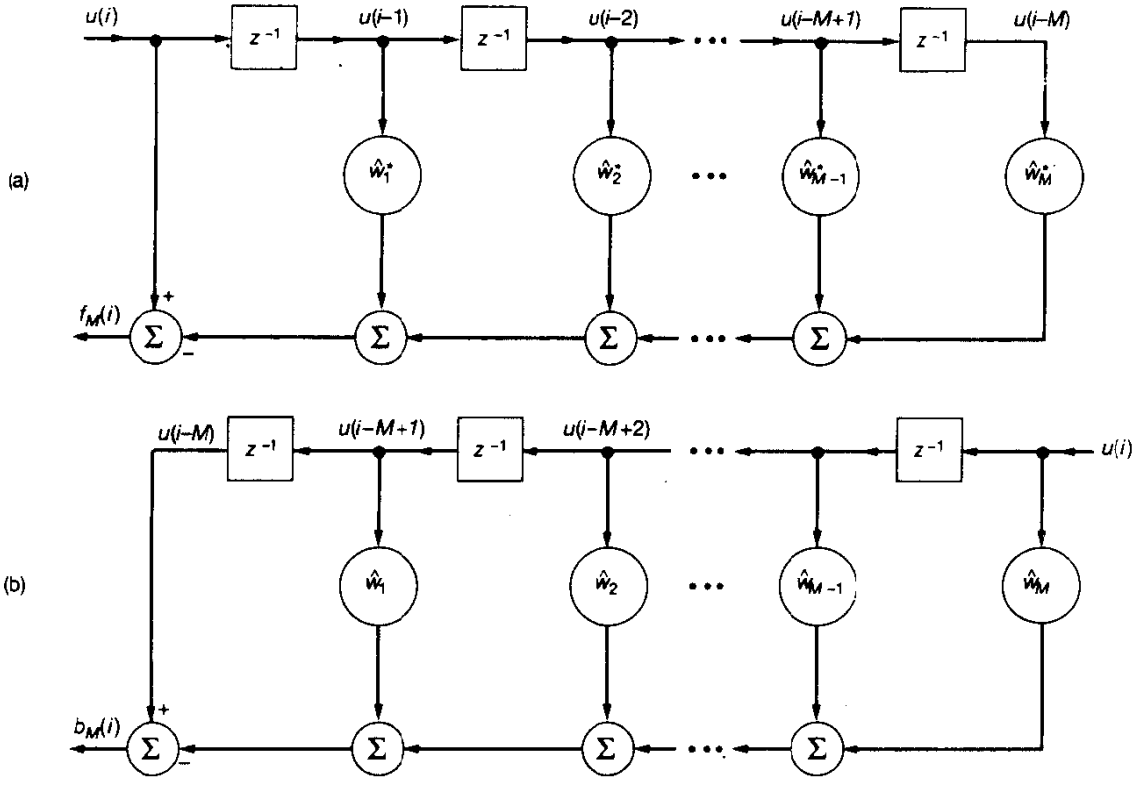


Figure 11.5 (a) Forward linear predictor; (b) reconfiguration of the predictor so as to perform backward linear prediction.

2. With $u(i)$, $u(i-1)$, . . . , $u(i-M+1)$ used as tap inputs, the structure of Fig. 11.5(b) produces a linear prediction of $u(i-M)$. In other words, it performs backward linear prediction. Denoting the backward prediction error by $b_M(i)$, we may thus express it as

$$b_M(i) = u(i-M) - \sum_{k=1}^{M} \hat{w}_k u(i-M+k) \tag{11.75}$$

where the first term represents the desired response and the second term is the predictor output. Equivalently, in terms of matrix notation, we may write

$$b_M(i) = u(i-M) - \mathbf{u}^{BT}(i)\hat{\mathbf{w}} \tag{11.76}$$

where $\mathbf{u}^B(i)$ is the *time-reversed tap-input vector*:

$$\mathbf{u}^{BT}(i) = [u(i-M+1), \ldots, u(i-1), u(i)]$$

Let $\mathscr{E}_M$ denote the *minimum value of the forward-backward prediction-error energy*. In accordance with the method of least squares, we may therefore write

$$\mathscr{E}_M = \sum_{i=M+1}^{N} [|f_M(i)|^2 + |b_M(i)|^2] \tag{11.77}$$

where the subscript M signifies the order of the predictor or that of the AR model. The lower limit on the time index i equals $M + 1$ so as to ensure that the forward and backward prediction errors are formed only when all the tap inputs of interest assume nonzero values. In particular, we may make two observations:

1. The variable $u(i - M)$, representing the last tap input in the forward prediction of Fig. 11.5(a), assumes a nonzero value for the first time when $i = M + 1$.
2. The variable $u(i - M)$, playing the role of desired response in the backward predictor of Fig. 11.5(b), also assumes a nonzero value for the first time when $i = M + 1$.

Thus, by choosing $(M + 1)$ as the lower limit on i and N as the upper limit, as in Eq. (11.77), we make no assumptions about the data outside the interval $[1, N]$, as required by the covariance method.

Let $\mathbf{A}$ denote the $2(N - M)$-by-M *data matrix*, whose Hermitian transpose is defined by

$$\mathbf{A}^H = \begin{bmatrix} u(M) & \cdots & u(N-1) & u^*(2) & \cdots & u^*(N-M+1) \\ u(M-1) & \cdots & u(N-2) & u^*(3) & \cdots & u^*(N-M+2) \\ \vdots & & \vdots & \vdots & & \vdots \\ u(1) & \cdots & u(N-M) & u^*(M+1) & \cdots & u^*(N) \end{bmatrix}$$

forward half | backward half (11.78)

The elements constituting the left half of matrix $\mathbf{A}^H$ represent the various sets of tap inputs used to make a total of $(N - M)$ *forward* linear predictions. The complex-conjugated elements constituting the right half of matrix $\mathbf{A}^H$ represent the corresponding sets of tap inputs used to make a total of $(N - M)$ *backward* linear predictions. Note that as we move from one column to the next in the forward or backward half in Eq. (11.78), we drop a sample, add a new one, and reorder the samples.

Let $\mathbf{d}$ denote the $2(N - M)$-by-1 *desired data vector*, defined in a manner corresponding to that shown in Eq. (11.78):

$$\mathbf{d}^H = \underbrace{[u(M+1), \ldots, u(N)}_{\text{forward half}}, \underbrace{u^*(1), \ldots, u^*(N-M)]}_{\text{backward half}} \tag{11.79}$$

Each element in the left half of the vector $\mathbf{d}^H$ represents a desired response for forward linear prediction. Each complex-conjugated element in the right half represents a desired response for backward linear prediction.

The FBLP method is a product of the method of least squares; it is therefore described by the system of normal equations [see Eq. (11.48)]

$$\mathbf{A}^H\mathbf{A}\hat{\mathbf{w}} = \mathbf{A}^H\mathbf{d} \tag{11.80}$$

The resulting minimum value of the forward-backward prediction error energy equals [see Eq. (11.49)]

$$\mathscr{E}_{\min} = \mathbf{d}^H\mathbf{d} - \mathbf{d}^H\mathbf{A}(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H\mathbf{d} \tag{11.81}$$

The data matrix $\mathbf{A}$ and the desired data vector $\mathbf{d}$ are defined by Eqs. (11.78) and (11.79), respectively.

We may combine Eqs. (11.80) and (11.81) into a single matrix relation, as shown by

$$\begin{bmatrix} \mathbf{d}^H\mathbf{d} & \mathbf{d}^H\mathbf{A} \\ \mathbf{A}^H\mathbf{d} & \mathbf{A}^H\mathbf{A} \end{bmatrix} \begin{bmatrix} 1 \\ -\hat{\mathbf{w}} \end{bmatrix} = \begin{bmatrix} \mathscr{E}_{\min} \\ \mathbf{0} \end{bmatrix} \tag{11.82}$$

where $\mathbf{0}$ is the M-by-1 null vector. Equation (11.82) is the matrix form of the *augmented normal equations for FBLP*. Define the $(M + 1)$-by-$(M + 1)$ *augmented correlation matrix*:

$$\mathbf{\Phi} = \begin{bmatrix} \mathbf{d}^H\mathbf{d} & \mathbf{d}^H\mathbf{A} \\ \mathbf{A}^H\mathbf{d} & \mathbf{A}^H\mathbf{A} \end{bmatrix} \tag{11.83}$$

The $\mathbf{\Phi}$ in Eq. (11.83) is an $(M + 1)$-by-$(M + 1)$ matrix; it is *not* to be confused with the $\mathbf{\Phi}$ in Eq. (11.45) that is an M-by-M matrix. Define the $(M + 1)$-by-1 tap-weight vector of the *prediction-error filter of order M*:

$$\hat{\mathbf{a}} = \begin{bmatrix} 1 \\ -\hat{\mathbf{w}} \end{bmatrix} \tag{11.84}$$

Figure 11.6 shows the transversal structure of the prediction-error filter, where $a_0, a_1, \ldots, a_M$ denote the tap weights[5] and $a_0 = 1$. Then

$$\mathbf{\Phi}\hat{\mathbf{a}} = \begin{bmatrix} \mathscr{E}_{\min} \\ \mathbf{0} \end{bmatrix} \tag{11.85}$$

The augmented correlation matrix $\mathbf{\Phi}$ is *Hermitian persymmetric*; that is, the individual elements of the matrix $\mathbf{\Phi}$ satisfy two conditions:

$$\phi(k, i) = \phi^*(i, k) \qquad 0 \le (i, k) \le M \tag{11.86}$$

$$\phi(M - k, M - i) = \phi^*(k, i), \qquad 0 \le (i, k) \le M \tag{11.87}$$

[5]The subscripts assigned to the tap weights in the prediction-error filter of Fig. 11.6 do not include a direct reference to the prediction order M, unlike the terminology used in Chapter 6. The reason for this simplification is that, in the material presented here, there is no order update to be considered.

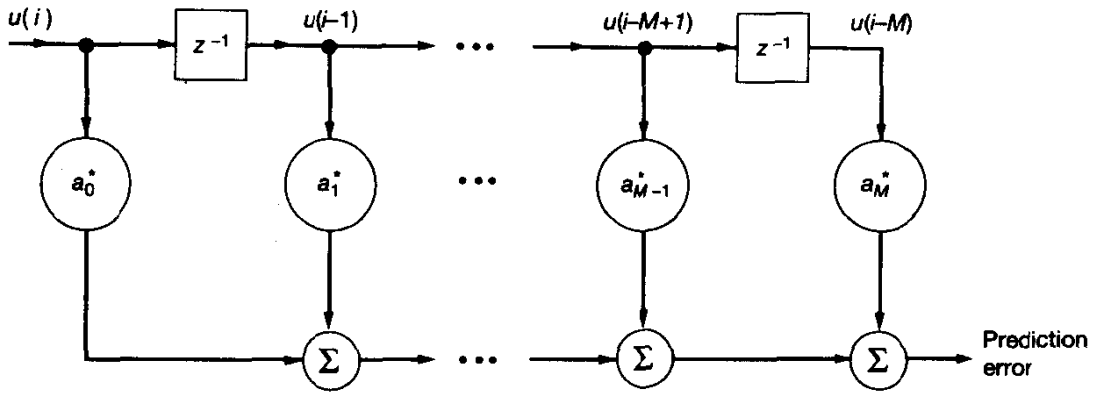


Figure 11.6 Forward prediction-error filter.

The property described in Eq. (11.87) is unique to a correlation matrix that is obtained by time averaging of the input data in the forward as well as backward direction; see the data matrix $\mathbf{A}$ and the desired data vector $\mathbf{d}$ defined in Eqs. (11.78) and (11.79), respectively. The matrix $\mathbf{\Phi}$ has another property: it is composed of the sum of two Toeplitz matrix products. The special Toeplitz structure of the matrix $\mathbf{\Phi}$ has been exploited in the development of fast recursive algorithms[6] for the efficient solution of the augmented normal equations (11.85).

Starting with the time series $u(i)$, $1 \leq i \leq N$, the FBLP algorithm is used to compute the tap-weight vector $\hat{\mathbf{w}}$ of a forward linear predictor or, equivalently, the tap-weight vector $\hat{\mathbf{a}}$ of the corresponding prediction-error filter. The vector $\hat{\mathbf{a}}$ represents an estimate of the coefficient vector of an *autoregressive (AR) model* used to fit the time series $u(i)$. Similarly, the minimum mean-squared error $\mathscr{E}_{\min}$, except for a scaling factor, represents an estimate of the white-noise variance σ^2 in the AR model. We may thus use Eq. (6.101) to formulate an *estimate of the AR spectrum* as follows:

$$\hat{S}_{\mathrm{AR}}(\omega) = \frac{\mathscr{E}_{\min}}{\left|1 + \sum_{k=1}^{M} \hat{a}_k^* e^{-j\omega k}\right|^2} \tag{11.88}$$

where the $\hat{a}_k$ are the elements of the vector $\hat{\mathbf{a}}$; the leading element $\hat{a}_0$ of the vector $\hat{\mathbf{a}}$ is equal to unity, by definition. We may also express $\hat{S}_{\mathrm{AR}}(\omega)$ as

$$\hat{S}_{\mathrm{AR}}(\omega) = \frac{\mathscr{E}_{\min}}{|\hat{\mathbf{a}}^H \mathbf{s}(\omega)|^2} \tag{11.89}$$

[6]The correlation matrix $\mathbf{\Phi}$ of Eq. (11.83) does *not* possess a Toeplitz structure. Accordingly, we cannot use the Levinson recursion to develop a fast solution of the augmented normal equations (11.85), as was the case with the augmented Wiener–Hopf equations for stationary inputs. However, Marple (1980, 1981) describes fast recursive algorithms for the efficient solution of the augmented normal equations (11.85). Marple exploits the special Toeplitz structure of the correlation matrix $\mathbf{\Phi}$. The computational complexity of Marple's fast algorithm is proportional to M^2. When the predictor order M is large, the use of Marple's algorithm results in significant savings in computation.

where $\mathbf{s}(\omega)$ is a *variable-frequency vector* or *frequency scanning vector*:

$$\mathbf{s}(\omega) = [1, e^{-j\omega}, \ldots, e^{-j\omega M}]^T, \qquad -\pi < \omega \leq \pi \tag{11.90}$$

Intuitively, the model order M should be as large as possible in order to have a large aperture for the predictor. However, in applying the FBLP algorithm the use of large values of M gives rise to spurious spectral peaks in the AR spectrum. For best performance of the FBLP algorithm, Lang and McClellan (1980) suggest the value

$$M \approx \frac{N}{3} \tag{11.91}$$

where N is the data length.

MVDR Spectrum Estimation

In the method of least squares, as described up to this point in our discussion, there are no *constraints* imposed on the solution. In certain applications, however, the use of such an approach may be unsatisfactory, in which case we may resort to a *constrained* version of the method of least squares. For example, in *adaptive beamforming* that involves spatial processing, we may wish to *minimize the variance (i.e., average power) of the beamformer output while a distortionless response is maintained along the direction of a target signal of interest*. Correspondingly, in the temporal counterpart to this problem, we may be required to *minimize the average power of the spectrum estimator, while a distortionless response is maintained at a particular frequency*. In such applications, the resulting solution is referred to as a *minimum-variance distortionless response (MVDR) estimator* for obvious reasons. To be consistent with the material presented heretofore, we will formulate the temporal version of the MVDR algorithm.

Consider then a linear transversal filter, as depicted in Fig. 11.7. Let the filter output be denoted by $y(i)$. This output is in response to the tap inputs $u(i)$, $u(i-1)$, . . . , $u(i-M)$. Specifically, we have

$$y(i) = \sum_{t=0}^{M} a_t^* u(i-t) \tag{11.92}$$

where $a_0, a_1, \ldots, a_m$ are the transversal filter coefficients. Note, however, that unlike the prediction-error filter of Fig. 11.6, there is no restriction on the filter coefficient a_0; the only reason for using the same terminology as in Fig. 11.6 is because of a desire to be consistent. The requirement is to minimize the *output energy* (assuming the use of the covariance method of data windowing):

$$\mathscr{E}_{\text{out}} = \sum_{i=M+1}^{N} |y(i)|^2$$

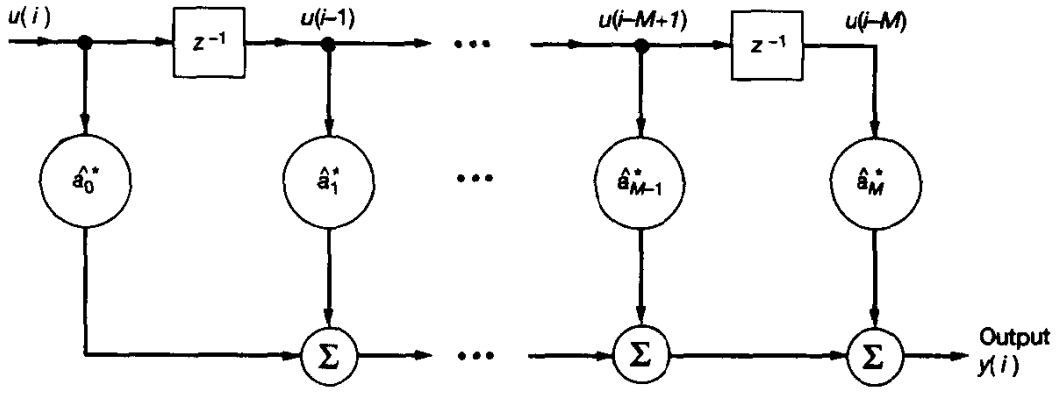


Figure 11.7 Transversal filter.

subject to the *constraint*

$$\sum_{k=0}^{M} a_k^* e^{-jk\omega_0} = 1 \tag{11.93}$$

where ω_0 is an angular frequency of special interest. As in the conventional method of least squares, the filter coefficients $a_0, a_1, \ldots, a_M$ are held constant for the observation interval $1 \leq i \leq N$, where N is the total data length.

To solve this *constrained minimization* problem, we use the *method of Lagrange multipliers*.[7] Specifically, we define the *constrained cost function*

$$\mathscr{C} = \underbrace{\sum_{i=M+1}^{N} |y(i)|^2}_{\text{output energy}} + \underbrace{\lambda\left(\sum_{k=0}^{M} a_k^* e^{-jk\omega_0} - 1\right)}_{\text{linear constraints}} \tag{11.94}$$

where λ is a *complex Lagrange multiplier*. Note that in the constrained approach described herein, there is *no* desired response; in place of it, however, we have a set of linear constraints. Note also that in the absence of a desired response and therefore no frame of reference, the principle of orthogonality loses its meaning in this new setting.

To solve for the optimum values of the filter coefficients, we first determine the gradient vector $\nabla\mathscr{C}$ and then set it equal to zero. Thus, proceeding in a manner similar to that described in Section 11.3, we find that the kth element of the gradient vector for the constrained cost function of Eq. (11.94) is

$$\nabla_k\mathscr{C} = 2\sum_{i=M+1}^{N} u(i-k)y^*(i) + \lambda^* e^{-jk\omega_0} \tag{11.95}$$

[7]The method of Lagrange multipliers is described in Appendix C.

Next, substituting Eq. (11.92) in (11.95), and rearranging terms, we get

$$\nabla_k \mathscr{E} = 2 \sum_{t=0}^{M} a_t \sum_{i=M+1}^{N} u(i-k)u^*(i-t) + \lambda^* e^{-jk\omega_0}$$

$$= 2 \sum_{t=0}^{M} a_t \phi(t, k) + \lambda^* e^{-jk\omega_0} \tag{11.96}$$

where, in the first term of the second line, we have made use of the definition of Eq. (11.29) for the time-averaged autocorrelation function $\phi(t, k)$ of the tap inputs. To minimize the constrained cost function $\mathscr{E}$, we set

$$\nabla_k \mathscr{E} = 0, \qquad k = 0, 1, \ldots, M \tag{11.97}$$

Accordingly, we find from Eq. (11.96) that the tap-weights of the *optimized* transversal filter satisfy the following system of $M + 1$ simultaneous equations:

$$\sum_{t=0}^{M} \hat{a}_t \phi(t, k) = -\frac{1}{2} \lambda^* e^{-jk\omega_0}, \qquad k = 0, 1, \ldots, M \tag{11.98}$$

Using matrix notation, we may rewrite this system of equations in the compact form

$$\mathbf{\Phi}\hat{\mathbf{a}} = -\frac{1}{2} \lambda^* \mathbf{s}(\omega_0) \tag{11.99}$$

where $\mathbf{\Phi}$ is the $(M + 1)$-by-$(M + 1)$ time-averaged correlation matrix of the tap inputs; $\hat{\mathbf{a}}$ is the $(M\ 1)$-by-1 vector of optimum tap weights; and $\mathbf{s}(\omega_0)$ is the $(M + 1)$-by-1 *fixed frequency vector*:

$$\mathbf{s}(\omega_0) = [1, e^{-j\omega_0}, \ldots, e^{-jM\omega_0}]^T \tag{11.100}$$

Assuming $\mathbf{\Phi}$ is nonsingular and therefore its inverse $\mathbf{\Phi}^{-1}$ exists, we may solve Eq. (11.99) for the optimum tap-weight vector:

$$\hat{\mathbf{a}} = -\frac{1}{2} \lambda^* \mathbf{\Phi}^{-1} \mathbf{s}(\omega_0) \tag{11.101}$$

There only remains the problem of evaluating the Lagrange multiplier λ. To solve for λ, we use the linear constraint in Eq. (11.93) for the optimized transversal filter, written in matrix form as

$$\hat{\mathbf{a}}^H \mathbf{s}(\omega_0) = 1 \tag{11.102}$$

Hence, evaluating the *inner product* of the vector $\mathbf{s}_0$ and the vector $\hat{\mathbf{a}}$ in Eq. (11.101), setting the result equal to 1 and solving for λ, we get

$$\lambda^* = -\frac{2}{\mathbf{s}^H(\omega_0)\mathbf{\Phi}^{-1}\mathbf{s}(\omega_0)} \tag{11.103}$$

Finally, substituting this value of λ in Eq. (11.101), we get the MVDR solution:[8]

$$\hat{\mathbf{a}} = \frac{\boldsymbol{\Phi}^{-1}\mathbf{s}(\omega_0)}{\mathbf{s}^H(\omega_0)\boldsymbol{\Phi}^{-1}\mathbf{s}(\omega_0)} \tag{11.104}$$

Thus, given the time-averaged correlation matrix $\boldsymbol{\Phi}$ of the tap inputs and the frequency vector $\mathbf{s}(\omega_0)$, we may use the *MVDR formula* of (11.104) to compute the optimum tap-weight vector $\hat{\mathbf{a}}$ of the transversal filter in Fig. 11.7.

Let $\hat{S}_{\text{MVDR}}(\omega_0)$ denote the minimum value of the output energy $\mathscr{E}_{\text{out}}$, which results when the MVDR solution $\hat{\mathbf{a}}$ of Eq. (11.104) is used for the tap-weight vector under the condition that the response is tuned to the angular frequency ω_0. We may then write

$$\hat{S}_{\text{MVDR}}(\omega_0) = \hat{\mathbf{a}}^H\boldsymbol{\Phi}\hat{\mathbf{a}} \tag{11.105}$$

Substituting Eq. (11.104) in (11.105), and then simplifying the result, we finally get

$$\hat{S}_{\text{MVDR}}(\omega_0) = \frac{1}{\mathbf{s}^H(\omega_0)\boldsymbol{\Phi}^{-1}\mathbf{s}(\omega_0)} \tag{11.106}$$

Equation (11.106) may be given a more general interpretation. Suppose that we define a frequency-scanning vector $\mathbf{s}(\omega)$ as in Eq. (11.90), where the angular frequency ω is now variable in the interval $(-\pi, \pi]$. For each ω, let the tap-weight vector of the transversal filter be assigned a corresponding MVDR estimate. The output energy of the optimized filter then becomes a function of ω. Let $\hat{S}_{\text{MVDR}}(\omega)$ describe this functional dependence, and so we may write[9]

$$\hat{S}_{\text{MVDR}}(\omega) = \frac{1}{\mathbf{s}^H(\omega)\boldsymbol{\Phi}^{-1}\mathbf{s}(\omega)} \tag{11.107}$$

We refer to Eq. (11.107) as the *MVDR spectrum estimate*, and the solution given in Eq. (11.104) as the *MVDR estimate* of the tap-weight vector. Note that at any ω, power due to other frequencies is minimized. Hence, the MVDR spectrum computed in accordance with Eq. (11.107) exhibits relatively sharp peaks.

The MVDR spectrum and AR spectrum are commonly referred to as *super-resolution* or *high-resolution spectra*, in the sense that they both exhibit sub-Rayleigh resolution as power spectrum estimators. For the numerical computation of these spectra, and linear least-squares solutions in general, the recommended procedure is to use a technique known as singular value decomposition, which is considered next.

[8]Equation (11.104) is of the same form as that of Eq. (5.97), except for the use of the time-averaged correlation matrix $\boldsymbol{\Phi}$ in place of the ensemble-averaged correlation matrix $\mathbf{R}$, and the use of symbol $\mathbf{a}$ in place of $\mathbf{w}_o$ for the tap-weight vector.

[9]The method for computing the spectrum in Eq. (11.107) is also referred to in the literature as *Capon's method* (Capon, 1969). The term "minimum-variance distortionless response" owes its origin to Owsley (1984).

11.10 SINGULAR VALUE DECOMPOSITION

The analytic power of *singular-value decomposition* lies in the fact that it applies to square as well as rectangular matrices, be they real or complex. As such, it is extremely well suited for the numerical solution of linear least-squares problems in the sense that *it can be applied directly to the data matrix.*

In Sections 11.5 and 11.7 we described two different forms of the normal equations for computing the linear least-squares solution:

1. The form given in Eq. (11.36), namely,

$$\hat{\mathbf{w}} = \boldsymbol{\Phi}^{-1}\mathbf{z}$$

where $\hat{\mathbf{w}}$ is the least-squares estimate of the tap-weight vector of a transversal filter model, $\boldsymbol{\Phi}$ is the time-averaged correlation matrix of the tap inputs, and $\mathbf{z}$ is the time-averaged cross-correlation vector between the tap inputs and some desired response.

2. The form given in Eq. (11.48) *directly in terms of data matrices*, namely,

$$\hat{\mathbf{w}} = (\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H\mathbf{d}$$

where $\mathbf{A}$ is the data matrix representing the time evolution of the tap input vectors, and $\mathbf{d}$ is the desired data vector representing the time evolution of the desired response.

These two forms are indeed mathematically equivalent. Yet they point to different computational procedures for evaluating the least-squares solution $\hat{\mathbf{w}}$. Equation (11.36) requires knowledge of the time-averaged correlation matrix $\boldsymbol{\Phi}$ that involves computing the product of $\mathbf{A}^H$ and $\mathbf{A}$. On the other hand, in Eq. (11.48) the entire term $(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}$ can be interpreted, in terms of the singular-value decomposition applied directly to the data matrix $\mathbf{A}$, in such a way that the solution computed for $\hat{\mathbf{w}}$ has *twice the number of correct digits* as the solution computed by means of Eq. (11.36) for the same numerical precision. To be specific, define the matrix

$$\mathbf{A}^+ = (\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H \tag{11.108}$$

Then we may rewrite Eq. (11.36) simply as

$$\hat{\mathbf{w}} = \mathbf{A}^+\mathbf{d} \tag{11.109}$$

The matrix $\mathbf{A}^+$ is called the *pseudoinverse* or the *Moore-Penrose generalized inverse* of the matrix $\mathbf{A}$ (Stewart, 1973; Golub and Van Loan 1989). Equation (11.109) represents a convenient way of saying that "the vector $\hat{\mathbf{w}}$ solves the linear least-squares problem." Indeed, it was with the simple format of Eq. (11.109) in mind and also the desire to be consistent with definitions of the time-averaged correlation matrix $\boldsymbol{\Phi}$ and the cross-correlation vector $\mathbf{z}$ used in Section 11.5 that we defined the data matrix $\mathbf{A}$ and the desired data vector $\mathbf{d}$ in the manner shown in Eqs. (11.44) and (11.46), respectively.

In practice, we often find that the data matrix $\mathbf{A}$ contains linearly dependent columns. Consequently, we are faced with a new situation where we now have to decide on which of an infinite number of possible solutions to the least-squares problem to work with. This issue can indeed be resolved by using the singular-value decomposition technique as described in Section 11.12, even when null($\mathbf{A}$) $\neq \varnothing$, where $\varnothing$ denotes the *null set*.

The Singular-Value Decomposition Theorem

The *singular-value decomposition (SVD)* of a matrix is one of the most elegant algorithms in numerical algebra for providing quantitative information about the structure of a system of linear equations (Klema and Laub, 1980). The system of linear equations that is of specific interest to us is described by

$$\mathbf{A}\hat{\mathbf{w}} = \mathbf{d} \tag{11.110}$$

in which $\mathbf{A}$ is a K-by-M matrix, $\mathbf{d}$ is a K-by-1 vector, and $\hat{\mathbf{w}}$ (representing an estimate of the unknown parameter vector) is an M-by-1 vector. Equation (11.110) represents a simplified matrix form of the normal equations. In particular, premultiplication of both sides of the equation by the vector $\mathbf{A}^H$ yields the normal equations for the least-squares weight vector $\hat{\mathbf{w}}$.

Given the data matrix $\mathbf{A}$, there are two unitary matrices $\mathbf{V}$ and $\mathbf{U}$, such that we may write

$$\mathbf{U}^H\mathbf{A}\mathbf{V} = \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{11.111}$$

where $\mathbf{\Sigma}$ is a diagonal matrix:

$$\mathbf{\Sigma} = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_W) \tag{11.112}$$

The σ's are ordered as $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_W > 0$. Equation (11.111) is a mathematical statement of the *singular-value decomposition theorem*. This theorem is also referred to as the *Autonne–Eckart–Young theorem* in recognition of its originators.[10]

Figure 11.8 presents a diagrammatic interpretation of the singular value decomposition theorem, as described in Eq. (11.111). In this diagram we have assumed that the number of rows K contained in the data matrix $\mathbf{A}$ is larger than the number of columns M, and that the number of nonzero singular values W is less than M. We may of course restructure the diagrammatic interpretation of the singular value decomposition theorem by expressing the data matrix in terms of the unitary matrices $\mathbf{U}$ and $\mathbf{V}$, and the diagonal matrix $\mathbf{\Sigma}$; this is left as an exercise for the reader.

[10]According to DeMoor and Golub (1989), the singular-value decomposition was introduced in its general form by Autonne in 1902, and an important characterization of it was described by Eckart and Young (1936). For additional notes on the history of the singular-value decomposition, see Klema and Laub (1980).

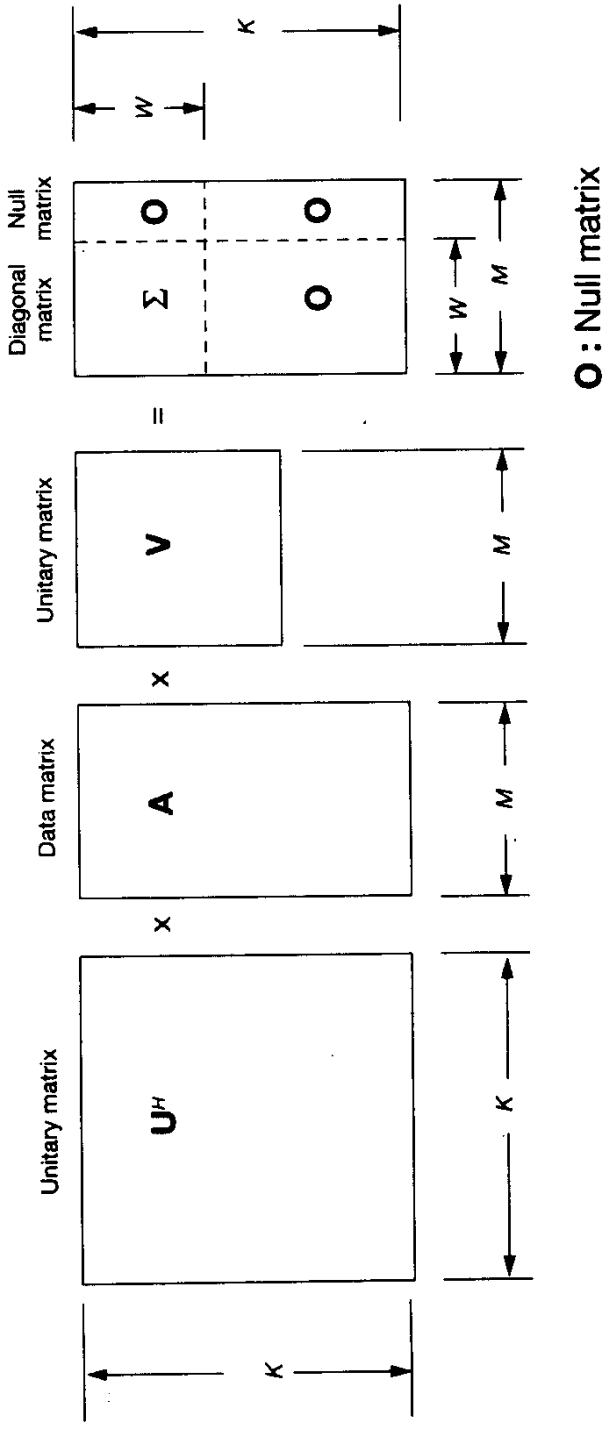


Figure 11.8 Diagrammatic interpretation of the singular value decomposition theorem.

The subscript W in Eq. (11.112) is the *rank* of matrix $\mathbf{A}$, written as rank($\mathbf{A}$); it is defined as the number of linearly independent columns in the matrix $\mathbf{A}$. Note that we always have rank($\mathbf{A}^H$) = rank($\mathbf{A}$).

Since it is possible to have $K > M$ or $K < M$, there are two distinct cases to be considered. We prove the singular-value decomposition theorem by considering both cases, independently of each other. For the case when $K > M$, we have an *overdetermined system* in that we have more equations than unknowns. On the other hand, when $K < M$, we have an *underdetermined system* in that we have more unknowns than equations. In the sequel, we consider these two cases in turn.

Case 1: Overdetermined System. For the case when $K > M$, we form the M-by-M matrix $\mathbf{A}^H\mathbf{A}$ by premultiplying the matrix $\mathbf{A}$ by its Hermitian transpose $\mathbf{A}^H$. Since the matrix $\mathbf{A}^H\mathbf{A}$ is Hermitian and nonnegative definite, its eigenvalues are all real nonnegative numbers. Let these eigenvalues be denoted by $\sigma_1^2, \sigma_2^2, \ldots, \sigma_M^2$, where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_W > 0$, and $\sigma_{W+1}, \sigma_{W+2}, \ldots$ are all zero, where $1 \leq W \leq M$. The matrix $\mathbf{A}^H\mathbf{A}$ has the same rank as $\mathbf{A}$; hence, there are W nonzero eigenvalues. Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M$ denote a set of orthonormal eigenvectors of $\mathbf{A}^H\mathbf{A}$ that are associated with the eigenvalues $\sigma_1^2, \sigma_2^2, \ldots, \sigma_M^2$, respectively. Also, let $\mathbf{V}$ denote the M-by-M unitary matrix whose columns are made up of the eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M$. Thus, using the eigendecomposition of the matrix $\mathbf{A}^H\mathbf{A}$, we may write

$$\mathbf{V}^H\mathbf{A}^H\mathbf{A}\mathbf{V} = \begin{bmatrix} \mathbf{\Sigma}^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{11.113}$$

Let the unitary matrix $\mathbf{V}$ be partitioned as

$$\mathbf{V} = [\mathbf{V}_1, \mathbf{V}_2] \tag{11.114}$$

where $\mathbf{V}_1$ is an M-by-W matrix,

$$\mathbf{V}_1 = [\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_W] \tag{11.115}$$

and $\mathbf{V}_2$ is an M-by-$(M - W)$ matrix,

$$\mathbf{V}_2 = [\mathbf{v}_{W+1}, \mathbf{v}_{W+2}, \ldots, \mathbf{v}_M] \tag{11.116}$$

with

$$\mathbf{V}_1^H\mathbf{V}_2 = \mathbf{0} \tag{11.117}$$

We may therefore make two deductions from Eq. (11.113):

1. For matrix $\mathbf{V}_1$, we have

$$\mathbf{V}_1^H\mathbf{A}^H\mathbf{A}\mathbf{V}_1 = \mathbf{\Sigma}^2$$

Consequently,

$$\mathbf{\Sigma}^{-1}\mathbf{V}_1^H\mathbf{A}^H\mathbf{A}\mathbf{V}_1\mathbf{\Sigma}^{-1} = \mathbf{I} \tag{11.118}$$

2. For matrix $\mathbf{V}_2$, we have

$$\mathbf{V}_2^H \mathbf{A}^H \mathbf{A} \mathbf{V}_2 = \mathbf{0}$$

Consequently,

$$\mathbf{A}\mathbf{V}_2 = \mathbf{0} \tag{11.119}$$

We now define a new K-by-W matrix

$$\mathbf{U}_1 = \mathbf{A}\mathbf{V}_1\boldsymbol{\Sigma}^{-1} \tag{11.120}$$

Then, from Eq. (11.118) it follows that

$$\mathbf{U}_1^H \mathbf{U}_1 = \mathbf{I} \tag{11.121}$$

which means that the columns of the matrix $\mathbf{U}_1$ are orthonormal with respect to each other. Next, we choose another K-by-$(K - W)$ matrix $\mathbf{U}_2$ such that the K-by-K matrix formed from $\mathbf{U}_1$ and $\mathbf{U}_2$, namely,

$$\mathbf{U} = [\mathbf{U}_1, \mathbf{U}_2] \tag{11.122}$$

is a unitary matrix. This means that

$$\mathbf{U}_1^H \mathbf{U}_2 = \mathbf{0} \tag{11.123}$$

Accordingly, we may use Eqs. (11.114), (11.122), (11.119), (11.120), and (11.123), in that order, and so write

$$\begin{aligned}
\mathbf{U}^H \mathbf{A} \mathbf{V} &= \begin{bmatrix} \mathbf{U}_1^H \\ \mathbf{U}_2^H \end{bmatrix} \mathbf{A}[\mathbf{V}_1, \mathbf{V}_2] \\
&= \begin{bmatrix} \mathbf{U}_1^H \mathbf{A}\mathbf{V}_1 & \mathbf{U}_1^H \mathbf{A}\mathbf{V}_2 \\ \mathbf{U}_2^H \mathbf{A}\mathbf{V}_1 & \mathbf{U}_2^H \mathbf{A}\mathbf{V}_2 \end{bmatrix} \\
&= \begin{bmatrix} (\boldsymbol{\Sigma}^{-1}\mathbf{V}_1^H \mathbf{A}^H)\mathbf{A}\mathbf{V}_1 & \mathbf{U}_1^H(\mathbf{0}) \\ \mathbf{U}_2^H(\mathbf{U}_1\boldsymbol{\Sigma}) & \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{\Sigma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}
\end{aligned}$$

which proves Eq. (11.111) for the overdetermined case.

Case 2: Underdetermined System. Consider next the case when $K < M$. This time we form the K-by-K matrix $\mathbf{A}\mathbf{A}^H$ by postmultiplying the matrix $\mathbf{A}$ by its Hermitian transpose $\mathbf{A}^H$. The matrix $\mathbf{A}\mathbf{A}^H$ is also Hermitian and nonnegative definite, so its eigenvalues are likewise real nonnegative numbers. The nonzero eigenvalues of $\mathbf{A}\mathbf{A}^H$ are the *same* as those of $\mathbf{A}^H\mathbf{A}$. We may therefore denote the eigenvalues of $\mathbf{A}\mathbf{A}^H$ as σ_1^2, $\sigma_2^2, \ldots, \sigma_K^2$, where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_W > 0$, and $\sigma_{W+1}, \sigma_{W+2}, \ldots$ are all zero, where

$1 \leq W \leq K$. Let $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_K$ denote a set of orthonormal eigenvctors of the matrix $\mathbf{A}\mathbf{A}^H$ that are associated with the eigenvalues $\sigma_1^2, \sigma_2^2, \ldots, \sigma_K^2$, respectively. Also, let $\mathbf{U}$ denote the unitary matrix whose columns are made up of the eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_K$. Thus, using the eigendecomposition of $\mathbf{A}\mathbf{A}^H$, we may write

$$\mathbf{U}^H\mathbf{A}\mathbf{A}^H\mathbf{U} = \begin{bmatrix} \mathbf{\Sigma}^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{11.124}$$

Let the unitary matrix $\mathbf{U}$ be partitioned as

$$\mathbf{U} = [\mathbf{U}_1, \mathbf{U}_2] \tag{11.125}$$

where

$$\mathbf{U}_1 = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_W] \tag{11.126}$$

$$\mathbf{U}_2 = [\mathbf{u}_{W+1}, \mathbf{u}_{W+2}, \ldots, \mathbf{u}_K] \tag{11.127}$$

and

$$\mathbf{U}_1^H\mathbf{U}_2 = \mathbf{0} \tag{11.128}$$

We may therefore make two deductions from Eq. (11.124):

1. For matrix $\mathbf{U}_1$, we have

$$\mathbf{U}_1^H\mathbf{A}\mathbf{A}^H\mathbf{U}_1 = \mathbf{\Sigma}^2$$

Consequently,

$$\mathbf{\Sigma}^{-1}\mathbf{U}_1^H\mathbf{A}\mathbf{A}^H\mathbf{U}_1\mathbf{\Sigma}^{-1} = \mathbf{I} \tag{11.129}$$

2. For matrix $\mathbf{U}_2$, we have

$$\mathbf{U}_2^H\mathbf{A}\mathbf{A}^H\mathbf{U}_2 = \mathbf{0}$$

Consequently,

$$\mathbf{A}^H\mathbf{U}_2 = \mathbf{0} \tag{11.130}$$

We now define an M-by-W matrix

$$\mathbf{V}_1 = \mathbf{A}^H\mathbf{U}_1\mathbf{\Sigma}^{-1} \tag{11.131}$$

Then from Eq. (11.129), it follows that

$$\mathbf{V}_1^H\mathbf{V}_1 = \mathbf{I} \tag{11.132}$$

which means that the columns of the matrix $\mathbf{V}_1$ are orthonormal with respect to each other. Next, we choose another M-by-$(M - W)$ matrix $\mathbf{V}_2$ such that the M-by-M matrix formed from $\mathbf{V}_1$ and $\mathbf{V}_2$, namely,

$$\mathbf{V} = [\mathbf{V}_1, \mathbf{V}_2] \tag{11.133}$$

is a unitary matrix. This means that

$$\mathbf{V}_2^H\mathbf{V}_1 = \mathbf{0} \tag{11.134}$$

Accordingly, we may use Eqs. (11.125), (11.133), (11.130), (11.131), and (11.134), in that order, and so write

$$\begin{aligned}\mathbf{U}^H\mathbf{A}\mathbf{V} &= \begin{bmatrix}\mathbf{U}_1^H\\ \mathbf{U}_2^H\end{bmatrix}\mathbf{A}[\mathbf{V}_1, \mathbf{V}_2]\\ &= \begin{bmatrix}\mathbf{U}_1^H\mathbf{A}\mathbf{V}_1 & \mathbf{U}_1^H\mathbf{A}\mathbf{V}_2\\ \mathbf{U}_2^H\mathbf{A}\mathbf{V}_1 & \mathbf{U}_2^H\mathbf{A}\mathbf{V}_2\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{U}_1^H\mathbf{A}(\mathbf{A}^H\mathbf{U}_1\boldsymbol{\Sigma}^{-1}) & (\boldsymbol{\Sigma}\mathbf{V}_1^H)\mathbf{V}_2\\ \end{bmatrix}\\ &= \begin{bmatrix}\boldsymbol{\Sigma} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\end{aligned}$$

This proves Eq. (11.111) for the underdetermined case, and with it the proof of the singular-value decomposition (SVD) theorem is completed.

Terminology and Relation to Eigenanalysis

The numbers $\sigma_1, \sigma_2, \ldots, \sigma_W$, constituting the diagonal matrix $\boldsymbol{\Sigma}$, are called the *singular values* of the matrix $\mathbf{A}$. The columns of the unitary matrix $\mathbf{V}$, that is, $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M$, are the *right singular vectors* of $\mathbf{A}$, and the columns of the second unitary matrix $\mathbf{U}$, that is, $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_K$ are the *left singular vectors* of $\mathbf{A}$. We note from the preceding discussion that the right singular vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M$ are eigenvectors of $\mathbf{A}^H\mathbf{A}$, whereas the left singular vectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_K$ are eigenvectors of $\mathbf{A}\mathbf{A}^H$. Note that the number of positive singular values is equal to the rank of the data matrix $\mathbf{A}$. The singular-value decomposition therefore provides the basis of a practical method for determining the rank of a matrix.

Since $\mathbf{U}\mathbf{U}^H$ equals the identity matrix, we find from Eq. (11.111) that

$$\mathbf{A}\mathbf{V} = \mathbf{U}\begin{bmatrix}\boldsymbol{\Sigma} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}$$

It follows therefore that

$$\begin{aligned}\mathbf{A}\mathbf{v}_i &= \sigma_i\mathbf{u}_i, \qquad i = 1, 2, \ldots, W\\ \mathbf{A}\mathbf{v}_i &= \mathbf{0}, \qquad i = W+1, \ldots, K\end{aligned} \tag{11.135}$$

Correspondingly, we may express the data matrix $\mathbf{A}$ in the expanded form

$$\mathbf{A} = \sum_{i=1}^{W}\sigma_i\mathbf{u}_i\mathbf{v}_i^H \tag{11.136}$$

Since $\mathbf{V}\mathbf{V}^H$ equals the identity matrix, we also find from Eq. (11.111) that

$$\mathbf{U}^H\mathbf{A} = \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\mathbf{V}^H$$

or, equivalently,

$$\mathbf{A}^H\mathbf{U} = \mathbf{V}\begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

It follows therefore that

$$\begin{aligned} \mathbf{A}^H\mathbf{u}_i &= \sigma_i\mathbf{v}_i, \qquad i = 1, 2, \ldots, W \\ \mathbf{A}^H\mathbf{u}_i &= \mathbf{0}, \qquad i = W + 1, \ldots, M \end{aligned} \tag{11.137}$$

In this case, we may express the Hermitian transpose of the data matrix $\mathbf{A}$ in the expanded form

$$\mathbf{A}^H = \sum_{i=1}^{W} \sigma_i\mathbf{v}_i\mathbf{u}_i^H \tag{11.138}$$

which checks exactly with Eq. (11.136), and so it should.

Example 2

In this example, we use the SVD to deal with the different facets of *matrix rank*. To be specific, let $\mathbf{A}$ be a K-by-M data matrix with rank W. The matrix $\mathbf{A}$ is said to be of *full rank* if

$$W = \min(K, M)$$

Otherwise, the matrix $\mathbf{A}$ is *rank deficient*. As mentioned previously, the rank W is simply the number of nonzero singular values of matrix $\mathbf{A}$.

Consider next a computational environment that yields a numerical value for each element of the matrix $\mathbf{A}$ that is accurate to within $\pm\epsilon$. Let $\mathbf{B}$ denote the approximate value of matrix $\mathbf{A}$ so obtained. We define the ϵ-*rank* of matrix $\mathbf{A}$ as follows (Golub and Van Loan, 1989):

$$\text{rank}(\mathbf{A}, \epsilon) = \min_{\|\mathbf{A}-\mathbf{B}\|<\epsilon} \text{rank}(\mathbf{B}) \tag{11.139}$$

where $\|\mathbf{A} - \mathbf{B}\|$ is the *spectral norm* of the error matrix $\mathbf{A} - \mathbf{B}$ that results from the use of inaccurate computations. Extending the definition of spectral norm of the matrix introduced in Chapter 4 to the situation at hand, the spectral norm $\|\mathbf{A} - \mathbf{B}\|$ equals the largest singular value of the difference $\mathbf{A} - \mathbf{B}$. In any event, the K-by-M matrix $\mathbf{A}$ is said to be *numerically rank deficient* if

$$\text{rank}(\mathbf{A}, \epsilon) < \min(K, M)$$

The SVD provides a sensible method for characterizing the ϵ-rank and the numerical rank deficiency of the matrix, because the singular values resulting from its use indicate how close a given matrix **A** is to another matrix **B** of lower rank in a simple fashion.

11.11 PSEUDOINVERSE

Our interest in the singular-value decomposition is to formulate a general definition of pseudoinverse. Let **A** denote a K-by-M matrix that has the singular-value decomposition described in Eq. (11.111). We define the pseudoinverse of the matrix **A** as (Stewart, 1973; Golub and Van Loan, 1989):

$$\mathbf{A}^+ = \mathbf{V}\begin{bmatrix} \mathbf{\Sigma}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\mathbf{U}^H \tag{11.140}$$

where

$$\mathbf{\Sigma}^{-1} = \text{diag}(\sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_W^{-1})$$

and W is the rank of the data matrix **A**. The pseudoinverse $\mathbf{A}^+$ may be expressed in the expanded form:

$$\mathbf{A}^+ = \sum_{i=1}^{W} \frac{1}{\sigma_i}\mathbf{v}_i\mathbf{u}_i^H \tag{11.141}$$

We may identify two special cases that can arise as described next.

Case 1: Overdetermined System. In this case, we have $K > M$, and we assume that the rank W equals M so that the inverse matrix $(\mathbf{A}^H\mathbf{A})^{-1}$ exists. The pseudoinverse of the data matrix **A** is defined by

$$\mathbf{A}^+ = (\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H \tag{11.142}$$

To show the validity of this special formula, we note from Eqs. (11.118) and (11.120) that

$$(\mathbf{A}^H\mathbf{A})^{-1} = \mathbf{V}_1\mathbf{\Sigma}^{-2}\mathbf{V}_1^H$$

and

$$\mathbf{A}^H = \mathbf{V}_1\mathbf{\Sigma}\mathbf{U}_1^H$$

Therefore, using this pair of relations, we may express the right-hand side of Eq. (11.142) as follows:

$$\begin{aligned}(\mathbf{A}^H\mathbf{A})^{-1}\mathbf{A}^H &= (\mathbf{V}_1\mathbf{\Sigma}^{-2}\mathbf{V}_1^H)(\mathbf{V}_1\mathbf{\Sigma}\mathbf{U}_1^H) \\ &= \mathbf{V}_1\mathbf{\Sigma}^{-1}\mathbf{U}_1^H \\ &= \mathbf{V}\begin{bmatrix} \mathbf{\Sigma}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\mathbf{U}^H \\ &= \mathbf{A}^+\end{aligned}$$

Case 2: Underdetermined System. In this second case, we have $M > K$, and we assume that the rank W equals K so that the inverse matrix $(\mathbf{A}\mathbf{A}^H)^{-1}$ exists. The pseudoinverse of the data matrix $\mathbf{A}$ is now defined by

$$\mathbf{A}^+ = \mathbf{A}^H(\mathbf{A}\mathbf{A}^H)^{-1} \tag{11.143}$$

To show the validity of this second special formula, we note from Eqs. (11.129) and (11.131) that

$$(\mathbf{A}\mathbf{A}^H)^{-1} = \mathbf{U}_1\mathbf{\Sigma}^{-2}\mathbf{U}_1^H$$

and

$$\mathbf{A}^H = \mathbf{V}_1\mathbf{\Sigma}\mathbf{U}_1^H$$

Therefore, using this pair of relations in the right-hand side of Eq. (11.143), we get

$$\begin{aligned}\mathbf{A}^H(\mathbf{A}\mathbf{A}^H)^{-1} &= (\mathbf{V}_1\mathbf{\Sigma}\mathbf{U}_1^H)(\mathbf{U}_1\mathbf{\Sigma}^{-2}\mathbf{U}_1^H)\\ &= \mathbf{V}_1\mathbf{\Sigma}^{-1}\mathbf{U}_1^H\\ &= \mathbf{V}\begin{bmatrix}\mathbf{\Sigma}^{-1} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{U}^H\\ &= \mathbf{A}^+\end{aligned}$$

Note, however, the pseudoinverse $\mathbf{A}^+$ as described in Eq. (11.140) or equivalently, Eq. (11.141) is of general application, in that it applies whether the data matrix $\mathbf{A}$ refers to an overdetermined or an underdetermined system and regardless of what the rank W is. Most importantly, it is numerically stable.

11.12 INTERPRETATION OF SINGULAR VALUES AND SINGULAR VECTORS

Consider a K-by-M data matrix $\mathbf{A}$, for which the singular-value decomposition is given in Eq. (11.111) and the pseudoinverse is correspondingly given in Eq. (11.140). We assume that the system is overdetermined. Define a K-by-1 vector $\mathbf{y}$ and an M-by-1 vector $\mathbf{x}$ that are related to each other by the transformation matrix $\mathbf{A}$, as shown by

$$\mathbf{y} = \mathbf{A}\mathbf{x} \tag{11.144}$$

The vector $\mathbf{x}$ is constrained to have a Euclidean norm of unity; that is,

$$\|\mathbf{x}\| = 1 \tag{11.145}$$

Given the transformation of Eq. (11.144) and the constraint of Eq. (11.145), we wish to find the resulting locus of the points defined by the vector $\mathbf{y}$ in a K-dimensional space.

Solving Eq. (11.144) for $\mathbf{x}$, we get

$$\mathbf{x} = \mathbf{A}^+\mathbf{y} \tag{11.146}$$

where $\mathbf{A}^+$ is the pseudoinverse of $\mathbf{A}$. Substituting Eq. (11.142) in (11.146), we get

$$\begin{aligned}\mathbf{x} &= \sum_{i=1}^{W} \frac{1}{\sigma_i}\mathbf{v}_i\mathbf{u}_i^H\mathbf{y} \\ &= \sum_{i=1}^{W} \frac{(\mathbf{u}_i^H\mathbf{y})}{\sigma_i}\mathbf{v}_i\end{aligned} \tag{11.147}$$

where W is the rank of matrix $\mathbf{A}$, and the inner product $\mathbf{u}_i^H\mathbf{y}$ is a scalar. Imposing the constraint of Eq. (11.145) on (11.147), and recognizing that the right singular vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_W$ form an orthonormal set, we get

$$\sum_{i=1}^{W} \frac{|\mathbf{y}^H\mathbf{u}_i|^2}{\sigma_i^2} = 1 \tag{11.148}$$

Equation (11.148) defines the locus traced out by the tip of vector $\mathbf{y}$ in a K-dimensional space. Indeed, this is the equation of a *hyperellipsoid* (Golub and Van Loan, 1989).

To see this interpretation in a better way, define the complex scalar

$$\begin{aligned}\zeta_i &= \mathbf{y}^H\mathbf{u}_i \\ &= \sum_{k=1}^{K} y_k^* u_{ik}, \qquad i = 1, \ldots, W\end{aligned} \tag{11.149}$$

In other words, the complex scalar ζ_i is a linear combination of all possible values of the elements of the left singular vector $\mathbf{u}_i$, so ζ_i is referred to as the "span" of $\mathbf{u}_i$. We may thus rewrite Eq. (11.148) as

$$\sum_{i=1}^{W} \frac{|\zeta_i|^2}{\sigma_i^2} = 1 \tag{11.150}$$

This is the equation of a hyperellipsoid with coordinates $|\zeta_1|, \ldots, |\zeta_W|$ and semi-axis whose lengths are the singular values $\sigma_1, \ldots, \sigma_W$, respectively. Figure 11.9 illustrates the locus traced out by Eq. (11.148) for the case of $W = 2$ and $\sigma_1 > \sigma_2$, assuming that the data matrix A is real.

11.13 MINIMUM NORM SOLUTION TO THE LINEAR LEAST-SQUARES PROBLEM

Having equipped ourselves with the general definition of the pseudoinverse of a matrix $\mathbf{A}$ in terms of its singular-value decomposition, we are now ready to tackle the solution to the linear least-squares problem even when null(A) $\neq \varnothing$. In particular, we define the solution to the least-squares problem as in Eq. (11.109), reproduced here for convenience:

$$\hat{\mathbf{w}} = \mathbf{A}^+\mathbf{d} \tag{11.151}$$

The pseudoinverse matrix $\mathbf{A}^+$ is itself defined by Eq. (11.140). We thus find that, out of the many vectors that solve the least-squares problem when null $(\mathbf{A}) \neq \varnothing$, the one defined

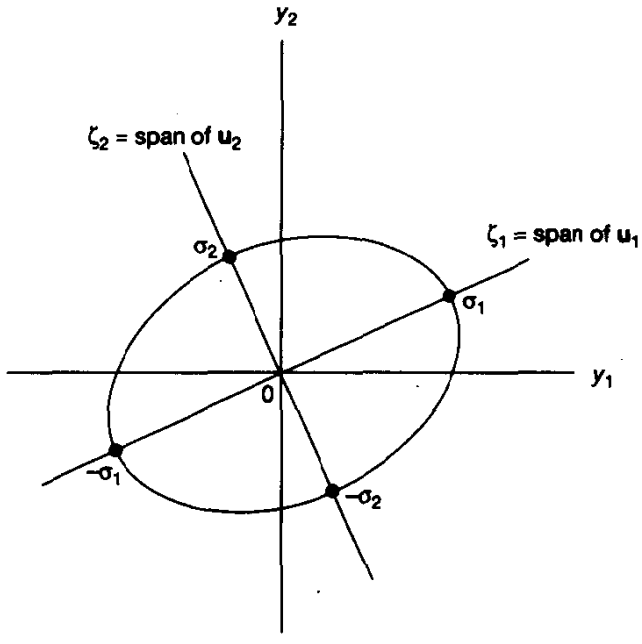


Figure 11.9 Locus of Eq. (11.150) for real data with $W = 2$ and $\sigma_1 > \sigma_2$.

by Eq. (11.151) is *unique* in that it has the *shortest length possible in the Euclidean sense* (Stewart, 1973).

We prove this important result by manipulating the equation that defines the minimum value of the sum of error squares produced in the method of least squares. We note that both matrix products $\mathbf{V}\mathbf{V}^H$ and $\mathbf{U}\mathbf{U}^H$ equal identity matrices. Hence, we may start with Eq. (11.49) and combine it with Eq. (11.48), and then write

$$\begin{aligned}\mathscr{E}_{\min} &= \mathbf{d}^H\mathbf{d} - \mathbf{d}^H\mathbf{A}\hat{\mathbf{w}} \\ &= \mathbf{d}^H(\mathbf{d} - \mathbf{A}\hat{\mathbf{w}}) \\ &= \mathbf{d}^H\mathbf{U}\mathbf{U}^H(\mathbf{d} - \mathbf{A}\mathbf{V}\mathbf{V}^H\hat{\mathbf{w}}) \\ &= \mathbf{d}^H\mathbf{U}(\mathbf{U}^H\mathbf{d} - \mathbf{U}^H\mathbf{A}\mathbf{V}\mathbf{V}^H\hat{\mathbf{w}})\end{aligned} \tag{11.152}$$

Let

$$\begin{aligned}\mathbf{V}^H\hat{\mathbf{w}} &= \mathbf{b} \\ &= \begin{bmatrix}\mathbf{b}_1 \\ \mathbf{b}_2\end{bmatrix}\end{aligned} \tag{11.153}$$

and

$$\begin{aligned}\mathbf{U}^H\mathbf{d} &= \mathbf{c} \\ &= \begin{bmatrix}\mathbf{c}_1 \\ \mathbf{c}_2\end{bmatrix}\end{aligned} \tag{11.154}$$

where $\mathbf{b}_1$ and $\mathbf{c}_1$ are W-by-1 vectors, and $\mathbf{b}_2$ and $\mathbf{c}_2$ are two other vectors. Thus, substituting Eqs. (11.111), (11.153), and (11.154) in (11.152), we get

$$\begin{aligned}\mathscr{E}_{\min} &= \mathbf{d}^H\mathbf{U}\left(\begin{bmatrix}\mathbf{c}_1\\ \mathbf{c}_2\end{bmatrix} - \begin{bmatrix}\boldsymbol{\Sigma} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{b}_1\\ \mathbf{b}_2\end{bmatrix}\right)\\ &= \mathbf{d}^H\mathbf{U}\begin{bmatrix}\mathbf{c}_1 - \boldsymbol{\Sigma}\mathbf{b}_1\\ \mathbf{c}_2\end{bmatrix}\end{aligned} \tag{11.155}$$

For $\mathscr{E}_{\min}$ to be minimum, we require that

$$\mathbf{c}_1 = \boldsymbol{\Sigma}\mathbf{b}_1 \tag{11.156}$$

or, equivalently,

$$\mathbf{b}_1 = \boldsymbol{\Sigma}^{-1}\mathbf{c}_1 \tag{11.57}$$

We observe that $\mathscr{E}_{\min}$ is independent of $\mathbf{b}_2$. Hence, the value of $\mathbf{b}_2$ is arbitrary. However, if we let $\mathbf{b}_2 = \mathbf{0}$, we get the special result

$$\begin{aligned}\hat{\mathbf{w}} &= \mathbf{V}\mathbf{b}\\ &= \mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma}^{-1}\mathbf{c}_1\\ \mathbf{0}\end{bmatrix}\end{aligned} \tag{11.158}$$

We may also express $\hat{\mathbf{w}}$ in the equivalent form:

$$\begin{aligned}\hat{\mathbf{w}} &= \mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma}^{-1} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{c}_1\\ \mathbf{c}_2\end{bmatrix}\\ &= \mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma}^{-1} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{U}^H\mathbf{d}\\ &= \mathbf{A}^+\mathbf{d}\end{aligned}$$

This coincides exactly with the value defined by Eq. (11.151), where the pseudoinverse $\mathbf{A}^+$ is defined by Eq. (11.140). In effect, we have shown that this value of $\hat{\mathbf{w}}$ does indeed solve the linear least-squares problem.

Moreover, the vector $\hat{\mathbf{w}}$ so defined is *unique*, in that it has the minimum Euclidean norm possible. In particular, since $\mathbf{V}\mathbf{V}^H = \mathbf{I}$, we find from Eq. (11.158) that the squared Euclidean norm of $\hat{\mathbf{w}}$ equals

$$\|\hat{\mathbf{w}}\|^2 = \|\boldsymbol{\Sigma}^{-1}\mathbf{c}_1\|^2$$

Consider now another possible solution to the linear least-squares problem that is defined by

$$\mathbf{w}' = \mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma}^{-1}\mathbf{c}_1\\ \mathbf{b}_2\end{bmatrix}, \qquad \mathbf{b}_2 \neq \mathbf{0}$$

The squared Euclidean norm of $\mathbf{w}'$ equals

$$\|\mathbf{w}'\|^2 = \|\boldsymbol{\Sigma}^{-1}\mathbf{c}_1\|^2 + \|\mathbf{b}_2\|^2$$

For any $\mathbf{b}_2 \neq \mathbf{0}$, we see therefore that

$$\|\mathbf{w}\| < \|\mathbf{w}'\| \tag{11.159}$$

In summary, the tap-weight $\hat{\mathbf{w}}$ of a linear transversal filter defined in by Eq. (11.151) is a unique solution to the linear least-squares problem, even when null(A) $\neq \emptyset$. *The vector $\hat{\mathbf{w}}$ is unique in the sense that it is the only tap-weight vector that simultaneously satisfies two requirements:* (1) *it produces the minimum sum of error squares, and* (2) *it has the smallest Euclidean norm possible. This special value of the tap-weight vector $\hat{\mathbf{w}}$ is called the minimum-norm solution.*

Another Formulation of the Minimum-Norm Solution

We may develop an expanded formulation of the minimum-norm solution, depending on whether we are dealing with the overdetermined or underdetermined case. These two cases are considered in turn.

Case 1: Overdetermined. For this case, the number of equations K is greater than the number of unknown parameters M. To proceed then, we substitute Eq. (11.140) in (11.151), and then use the partitioned forms of the unitary matrices $\mathbf{V}$ and $\mathbf{U}$. We may thus write

$$\begin{aligned}\hat{\mathbf{w}} &= (\mathbf{V}_1\boldsymbol{\Sigma}^{-1})(\mathbf{A}\mathbf{V}_1\boldsymbol{\Sigma}^{-1})^H\mathbf{d} \\ &= \mathbf{V}_1\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}^{-1}\mathbf{V}_1^H\mathbf{A}^H\mathbf{d} \\ &= \mathbf{V}_1\boldsymbol{\Sigma}^{-2}\mathbf{V}_1\mathbf{A}^H\mathbf{d}\end{aligned} \tag{11.160}$$

Hence, using the definition [see Eq. (11.115)]

$$\mathbf{V}_1 = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_W]$$

in Eq. (11.160), we get the following expanded formulation for $\hat{\mathbf{w}}$ for the overdetermined case:

$$\hat{\mathbf{w}} = \sum_{i=1}^{W} \frac{\mathbf{v}_i}{\sigma_i^2}\mathbf{v}_i^H\mathbf{A}^H\mathbf{d} \tag{11.161}$$

Case 2: Underdetermined. For this second case, the number of equations K is smaller than the number of unknowns M. This time we find it appropriate to use the representation given in Eq. (11.131) for the submatrix $\mathbf{V}_1$ in terms of the data matrix $\mathbf{A}$. Thus, substituting Eq. (11.131) in (11.151), we get

$$\begin{aligned}\hat{\mathbf{w}} &= (\mathbf{A}^H\mathbf{U}_1\boldsymbol{\Sigma}^{-1})(\boldsymbol{\Sigma}^{-1}\mathbf{U}_1\mathbf{d}) \\ &= \mathbf{A}^H\mathbf{U}_1\boldsymbol{\Sigma}^{-2}\mathbf{U}_1^H\mathbf{d}\end{aligned} \tag{11.162}$$

Substituting the definition [see Eq. (11.126)]

$$\mathbf{U}_1 = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_W]$$

in Eq. (11.162), we get the following expanded formulation for $\hat{\mathbf{w}}$ for the underdetermined case:

$$\hat{\mathbf{w}} = \sum_{i=1}^{W} \frac{(\mathbf{u}_i^H \mathbf{d})}{\sigma_i^2} \mathbf{A}^H \mathbf{u}_i \tag{11.163}$$

which is different from that of Eq. (11.161) for the overdetermined case.

The important point to note is that the expanded solutions of $\hat{\mathbf{w}}$ given in Eqs. (11.161) and (11.163) for the overdetermined and underdetermined systems, respectively, are both contained in the compact formula of Eq. (11.151). Indeed, from a numerical computation point of view, the use of Eq. (11.151) is the preferred method for computing the least-squares estimatee $\hat{\mathbf{w}}$.

11.14 NORMALIZED LMS ALGORITHM VIEWED AS THE MINIMUM-NORM SOLUTION TO AN UNDERDETERMINED LEAST-SQUARES ESTIMATION PROBLEM

In Chapter 9 we derived the normalized least-mean-square (LMS) algorithm as the solution to a constrained minimization problem. In this section we revisit this algorithm in light of the theory developed on singular-value decomposition. In particular, we show that the normalized LMS algorithm is indeed the minimum-norm solution to an underdetermined linear least-squares problem involving a single error equation with M unknowns, where M is the dimension of the tap-weight vector in the algorithm.

Consider the error equation

$$\epsilon(n) = d(n) - \hat{\mathbf{w}}^H(n+1)\mathbf{u}(n) \tag{11.164}$$

where $d(n)$ is a desired response and $\mathbf{u}(n)$ is a tap-input vector, both measured at time n. The requirement is to find the tap-weight vector $\hat{\mathbf{w}}(n+1)$, measured at time $n+1$, such that the change in the tap-weight vector given by

$$\delta\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n+1) - \hat{\mathbf{w}}(n) \tag{11.165}$$

is minimized, subject to the constraint

$$\epsilon(n) = 0 \tag{11.166}$$

Using Eq. (11.165) in (11.164), we may reformulate the error $\epsilon(n)$ as

$$\epsilon(n) = d(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n) - \delta\hat{\mathbf{w}}^H(n+1)\mathbf{u}(n) \tag{11.167}$$

We now recognize the customary definition of the estimation error, namely,

$$e(n) = d(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n) \tag{11.168}$$

Hence, we may simplify Eq. (11.168) as

$$\epsilon(n) = e(n) - \delta\hat{\mathbf{w}}^H(n+1)\mathbf{u}(n) \tag{11.169}$$

TABLE 11.1 SUMMARY OF CORRESPONDENCES BETWEEN LINEAR LEAST-SQUARES ESTIMATION AND NORMALIZED LMS ALGORITHM

	Linear least-squares estimation (underdetermined)	Normalized LMS algorithm
Data matrix	$\mathbf{A}$	$\mathbf{u}^H(n)$
Desired data vector	$\mathbf{d}$	$e^*(n)$
Parameter vector	$\hat{\mathbf{w}}$	$\delta\hat{\mathbf{w}}(n+1)$
Rank	W	1
Eigenvalue	$\sigma_i^2, i = 1, \ldots, W$	$\|\mathbf{u}(n)\|^2$
Eigenvector	$\mathbf{u}_i, i = 1, \ldots, W$	1

Thus, complex conjugating both sides of Eq. (11.169), we note that the constraint of Eq. (11.166) is equivalent to

$$\mathbf{u}^H(n)\delta\hat{\mathbf{w}}(n+1) = e^*(n) \tag{11.170}$$

Accordingly, we may restate our constrained minimization problem as follows:

Find the minimum-norm solution for the change $\delta\hat{\mathbf{w}}(n+1)$ in the tap-weight vector at time $n+1$, which satisfies the constraint

$$\mathbf{u}^H(n)\delta\hat{\mathbf{w}}(n+1) = e^*(n)$$

This problem is one of linear least-squares estimation that is underdetermined. To solve it, we may use the method of singular-value decomposition described in Eq. (11.163). To help us in the application of this method, we use Eq. (11.170) to make the identifications listed in Table 11.1 between the normalized LMS algorithm and linear least-squares estimation. In particular, we note that the normalized LMS algorithm has only one nonzero singular value equal to the squared norm of the tap-input vector $\mathbf{u}(n)$; that is, the rank $W = 1$. The corresponding left-singular vector is therefore simply equal to one. Hence, with the aid of Table 11.1, the application of Eq. (11.163) yields

$$\delta\hat{\mathbf{w}}(n+1) = \frac{1}{\|\mathbf{u}(n)\|^2}\,\mathbf{u}(n)e^*(n) \tag{11.171}$$

This is precisely the result that we derived previously in Chapter 9; see Eq. (9.139).

We may next follow a reasoning similar to that described in Section 9.10 and redefine the change $\delta\hat{\mathbf{w}}(n+1)$ by introducing a scaling factor $\tilde{\mu}$ as shown by [see Eq. (9.140)]

$$\delta\hat{\mathbf{w}}(n+1) = \frac{\tilde{\mu}}{\|\mathbf{u}(n)\|^2}\,\mathbf{u}(n)e^*(n)$$

or, equivalently, we may write

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \frac{\tilde{\mu}}{\|\mathbf{u}(n)\|^2}\,\mathbf{u}(n)e^*(n) \tag{11.172}$$

By so doing, we are able to exercise control over the change in the tap-weight vector from one iteration to the next without changing its direction. Equation (11.172) is the tap-weight vector update for the normalized LMS algorithm.

The important point to note from the discussion presented in this section is that the singular-value decomposition provides an insightful link between the underdetermined form of linear least-squares estimation and LMS theory. In particular, we have shown that the weight update in the normalized LMS algorithm may indeed be viewed as the minimum norm solution to an underdetermined form of the linear least-squares problem. The problem involves a single error equation with a number of unknowns equal to the dimension of the tap-weight vector in the algorithm.

11.15 SUMMARY AND DISCUSSION

In this chapter we presented a detailed discussion of the method of least-squares for solving the linear adaptive filtering problem. The distinguishing features of this approach include the following:

- It is a model-dependent procedure that operates on the input data on a block-by-block basis.
- It yields a solution for the tap-weight vector of an adaptive transversal filter that is the best linear unbiased estimate (BLUE), assuming that the measurement error process in the underlying model is white with zero mean.

The method of least squares is well suited for solving super-resolution spectrum estimation/beamforming problems, such as those based on autoregressive (AR) and minimum-variance distortionless response (MVDR) models. For the efficient computation of these spectra, and linear least-squares solution in general, the recommended procedure is to use singular value decomposition (SVD) that operates on the input data directly. The SVD is defined by the following parameters:

- A set of left singular vectors that form a unitary matrix
- A set of right singular vectors that form another unitary matrix
- A corresponding set of nonzero singular values

The important advantage of using the SVD to solve a linear least-squares problem is that the solution, defined in terms of the pseudoinverse of the input data matrix, is numerically stable. An algorithm is said to be *numerically stable* if it does not introduce any more sensitivity to perturbation than that which is inherently present in the problem under study (Klema and Laub, 1980).

Another useful application of the SVD is in *rank determination*. The *column rank* of a matrix is defined by the number of linearly independent columns of the matrix. Specifi-

cally, we say that an M-by-K matrix, with $M \geq K$, has *full* column rank if and only if it has K independent columns. In theory, the issue of full rank determination is a yes–no type of proposition in the sense that either the matrix in question has full rank or it does not. In practice, however, the fuzzy nature of a data matrix and the use of inexact (finite-precision) arithmetic complicate the rank determination problem. The SVD provides a practical method for determining the rank of a matrix, given fuzzy data and roundoff errors due to finite-precision computations.

PROBLEMS

1. Consider a linear array consisting of M uniformly spaced sensors. The output of sensor k observed at time i is denoted by $u(k, i)$ where $k = 1, 2, \ldots, M$ and $i = 1, 2, \ldots, n$. In effect, the observations $u(1, i), u(2, i), \ldots, u(M, i)$ define snapshot i. Let $\mathbf{A}$ denote the n-by-M data matrix, whose Hermitian transpose is defined by

$$\mathbf{A}^H = \begin{bmatrix} u(1,1) & u(1,2) & \cdots & u(1,n) \\ u(2,1) & u(2,2) & \cdots & u(2,n) \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ u(M,1) & u(M,2) & \cdots & u(M,n) \end{bmatrix}$$

   where the number of columns equals the number of snapshots, and the number of rows equals the number of sensors in the array. Demonstrate the following interpretations:
   (a) The M-by-M matrix $\mathbf{A}^H\mathbf{A}$ is the *spatial* correlation matrix with temporal averaging. This form of averaging assumes that the environment is temporally stationary.
   (b) The n-by-n matrix $\mathbf{A}\mathbf{A}^H$ is the *temporal* correlation matrix with spatial averaging. This form of averaging assumes that the environment is spatially stationary.

2. We say that the least-squares estimate $\hat{\mathbf{w}}$ is *consistent* if, in the long run, the difference between $\hat{\mathbf{w}}$ and the unknown parameter vector $\mathbf{w}_o$ of the multiple linear regression model becomes negligibly small in the mean-square sense. Hence, show that the least-squares estimate $\hat{\mathbf{w}}$ is consistent if the error vector $\boldsymbol{\epsilon}_o$ has zero mean and its elements are uncorrelated and if the trace of the inverse matrix $\boldsymbol{\Phi}^{-1}$ approaches zero as the number of observations, N, approaches infinity.

3. In Example 1 in Section 11.6, we used a 3-by-2 input data matrix and 3-by-1 desired data vector to illustrate the corollary to the principle of orthogonality. Use the data given in that example to calculate the two tap-weights of the linear least-squares filter.

4. In the autocorrelation method of linear prediction, we choose the tap-weight vector of a transversal predictor to minimize the error energy

$$\mathscr{E}_f = \sum_{n=1}^{\infty} |f(n)|^2$$

   where $f(n)$ is the prediction error. Show that the transfer function $H(z)$ of the (forward) prediction-error filter is minimum phase, in that its roots must lie strictly within the unit circle.

Hints: (1) Express the transfer function $H(z)$ of order M (say) as the product of a simple zero factor $(1 - z_i z^{-1})$ and a function $H'(z)$. Hence, minimize the prediction-error energy with respect to the magnitude of zero z_i.

(2) Use the Cauchy–Schwartz inequality:

$$\mathrm{Re}\left[\sum_{n=1}^{\infty} e^{j\theta} g(n-1)g^*(n)\right] \le \left[\sum_{n=1}^{\infty} |g(n)|^2\right]^{1/2}\left[\sum_{n=1}^{\infty} |e^{j\theta} g(n-1)|^2\right]^{1/2}$$

The equality holds if and only if $g(n) = e^{j\theta} g(n-1)$ for $n = 1, 2, \ldots, \infty$.

5. Figure 11.5(a) shows a *forward linear predictor* using a transversal structure, with the tap inputs $u(i-1), u(i-2), \ldots, u(i-M)$ used to make a linear prediction of $u(i)$. The problem is to find the tap-weight vector $\hat{\mathbf{w}}$ that minimizes the sum of forward prediction-error squares:

$$\mathscr{E}_f = \sum_{i=M+1}^{N} |f_M(i)|^2$$

where $f_M(i)$ is the forward prediction error. Find the following parameters:

(a) The M-by-M correlation matrix of the tap inputs of the predictor.

(b) The M-by-1 cross-correlation vector between the tap inputs of the predictor and the desired response $u(i)$.

(c) The minimum value of $\mathscr{E}_f$.

6. Figure 11.5(b) shows a *backward linear predictor* using a transversal structure, with the tap inputs $u(i-M+1), \ldots, u(i-1), u(i)$ used to make a linear prediction of the input $u(i-M)$. The problem is to find the tap-weight vector $\hat{\mathbf{w}}$ that minimizes the sum of backward prediction-error squares

$$\mathscr{E}_b = \sum_{i=M+1}^{N} |b_M(i)|^2$$

where $b_M(i)$ is the backward prediction error. Find the following parameters:

(a) The M-by-M correlation matrix of the tap inputs.

(b) The M-by-1 correlation vector between the tap inputs and the desired response $u(i-M)$.

(c) The minimum value of $\mathscr{E}_b$.

7. Use a direct approach to derive the system of normal equations given in expanded form in Eq. (11.31).

8. Calculate the singular values and singular vectors of the 2-by-2 real matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & -1 \\ 0.5 & 2 \end{bmatrix}$$

Do the calculation using two different methods:

(a) Eigendecomposition of the matrix product $\mathbf{A}^T\mathbf{A}$.

(b) Eigendecomposition of the matrix product $\mathbf{A}\mathbf{A}^T$.

Hence, find the pseudoinverse of matrix $\mathbf{A}$.

9. Consider the 2-by-2 complex matrix

$$\mathbf{A} = \begin{bmatrix} 1+j & 1+0.5j \\ 0.5-j & 1-j \end{bmatrix}$$

Calculate the singular values and singular vectors of the matrix $\mathbf{A}$ by proceeding as follows:

(a) Construct the matrix $\mathbf{A}^H\mathbf{A}$; hence, evaluate the eigenvalues and eigenvectors of $\mathbf{A}^H\mathbf{A}$.

(b) Construct the matrix $\mathbf{A}\mathbf{A}^H$; hence, evaluate the eigenvalues and eigenvectors of $\mathbf{A}\mathbf{A}^H$.

(c) Relate the eigenvalues and eigenvectors calculated in parts (a) and (b) to the singular values and singular vectors of $\mathbf{A}$.

10. Refer back to Example 1 in Section 11.7. For the sets of data given in that example, do the following:

(a) Calculate the pseudo-inverse of the 3-by-2 data matrix $\mathbf{A}$.

(b) Use this value of the pseudo-inverse $\mathbf{A}^+$ to calculate the two tap weights of the linear least-squares filter.

11. In this problem we explore the derivation of the weight update for the normalized LMS algorithm described in Eq. (9.144) using the idea of singular-value decomposition. This problem may be viewed as an extension of the discussion presented in Section 11.14. Find the minimum norm solution for the coefficient vector

$$\mathbf{c}(n+1) = \begin{bmatrix} \delta\hat{\mathbf{w}}(n+1) \\ 0 \end{bmatrix}$$

that satisfies the equation

$$\mathbf{x}^H(n)\mathbf{c}(n+1) = e^*(n)$$

where

$$\mathbf{x}(n) = \begin{bmatrix} \mathbf{u}(n) \\ \sqrt{a} \end{bmatrix}$$

Hence, show that

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \frac{\tilde{\mu}}{a + \|\mathbf{u}(n)\|^2}\,\mathbf{u}(n)e^*(n)$$

where $a > 0$, and $0 < \tilde{\mu} < 2$. [This is the weight update described in Eq. (9.144).]

12. You are given a processor that is designed to perform the singular-value composition of a K-by-M data matrix $\mathbf{A}$. Using such a processor, develop block diagrams for the following two super-resolution algorithms:

(a) The autoregressive (AR) algorithm

(b) The minimum-variance distortionless response (MVDR) algorithm

CHAPTER

12

Rotations and Reflections

In the previous chapter we emphasized the importance of singular value decomposition (SVD) as a tool for solving the linear least-squares problem. In this chapter we turn our attention to the practical issue of how to compute the SVD of a data matrix. With numerical stability as a primary design objective, the recommended procedure for SVD computation is to work directly with the data matrix. In this context, we may mention two different algorithms for SVD computation:

- *QR algorithm*, which proceeds by using a sequence of planar reflections known as Householder transformations
- *Cyclic Jacobi algorithm*, which employs a sequence of 2-by-2 plane rotations known as Jacobi rotations or Givens rotations

The cyclic Jacobi algorithm and QR algorithm are both *data adaptive* and *block-processing* oriented. They share a common goal, albeit in different ways:

- The diagonalization of the data matrix in a step-by-step fashion, and to within some prescribed numerical precision

It is important to note that plane rotations and reflections are wide ranging in their applications. In particular, they play a key role in the design of square-root Kalman filters

and related linear adaptive filters. We therefore have reason in later chapters of the book to refer back to some of the fundamental concepts presented herein. However, the main focus of attention in this chapter is on numerically stable algorithms for SVD computation using rotations and reflections. We begin the discussion by considering plane rotations; planar reflections are considered later in this chapter.

12.1 PLANE ROTATIONS

An algebraic tool that is fundamental to the cyclic Jacobi algorithm is the 2-by-2 *orthogonal matrix*:[1]

$$\mathbf{\Theta} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \tag{12.1}$$

where c and s are real parameters defined by

$$c = \cos\theta \tag{12.2}$$

and

$$s = \sin\theta \tag{12.3}$$

with the trigonometric *constraint*:

$$c^2 + s^2 = 1 \tag{12.4}$$

We refer to the transformation $\mathbf{\Theta}$ as a "plane rotation," because multiplication of a 2-by-1 data vector by $\mathbf{\Theta}$ amounts to a plane rotation of that vector. This property holds whether the data vector is premultiplied or postmultiplied by $\mathbf{\Theta}$.

The transformation of Eq. (12.1) is referred to as the *Jacobi rotation* in honor of Jacobi (1846), who proposed a method for reducing a symmetric matrix to diagonal form. It is also referred to as the *Givens rotation*. In this book we will use the latter terminology, or simply plane rotation.

To illustrate the nature of this plane rotation, consider the case of a real 2-by-1 vector:

$$\mathbf{a} = \begin{bmatrix} a_i \\ a_k \end{bmatrix}$$

Then premultiplication of the vector $\mathbf{a}$ by $\mathbf{\Theta}$ yields

$$\begin{aligned} \mathbf{x} &= \mathbf{\Theta a} \\ &= \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a_i \\ a_k \end{bmatrix} \\ &= \begin{bmatrix} ca_i + sa_k \\ -sa_i + ca_k \end{bmatrix} \end{aligned}$$

[1] In SVD terminology (and eigenanalysis for that matter), the term "orthogonal matrix" is used in the context of *real* data, whereas the term "unitary matrix" is used for complex data.

We may readily show, in view of the definitions of the rotation parameters c and s, that the vector $\mathbf{x}$ has the same Euclidean length as the vector $\mathbf{a}$. Moreover, given that the angle θ is positive, the transformation $\boldsymbol{\Theta}$ *rotates* the vector $\mathbf{a}$ in a clockwise direction into the new position defined by $\mathbf{x}$, as illustrated in Fig. 12.1. Note that the vectors $\mathbf{a}$ and $\mathbf{x}$ *remain in the same* (i, k) *plane*, hence the name "plane rotation."

12.2 TWO-SIDED JACOBI ALGORITHM

To pave the way for a development of the cyclic Jacobi algorithm, consider the simple case of a *real* 2-by-2 data matrix:

$$\mathbf{A} = \begin{bmatrix} a_{ii} & a_{ik} \\ a_{ki} & a_{kk} \end{bmatrix} \tag{12.5}$$

We assume that $\mathbf{A}$ is nonsymmetric; that is, $a_{ki} \neq a_{ik}$. The requirement is to *diagonalize* this 2-by-2 matrix. We do so by means of two plane rotations $\boldsymbol{\Theta}_1$ and $\boldsymbol{\Theta}_2$, as shown by

$$\underbrace{\begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}^T}_{\boldsymbol{\Theta}_1} \underbrace{\begin{bmatrix} a_{ii} & a_{ik} \\ a_{ki} & a_{kk} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix}}_{\boldsymbol{\Theta}_2} = \underbrace{\begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}}_{\text{diagonal matrix}} \tag{12.6}$$

To design the two plane rotations indicated in Eq. (12.5), we proceed in two stages. Stage I transforms the 2-by-2 data matrix $\mathbf{A}$ into a symmetric matrix; we refer to this stage as "symmetrization." Stage II diagonalizes the symmetric matrix resulting from stage I; we refer to this second stage as "diagonalization." Of course, if the data matrix is symmetric to begin with, we proceed to stage II directly.

Stage I: Symmetrization. To transform the 2-by-2 data matrix $\mathbf{A}$ into a symmetric matrix, we premultiply it by the transpose of a plane rotation $\boldsymbol{\Theta}$ and thus write

$$\underbrace{\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T}_{\boldsymbol{\Theta}^T} \underbrace{\begin{bmatrix} a_{ii} & a_{ik} \\ a_{ki} & a_{kk} \end{bmatrix}}_{\mathbf{A}} = \underbrace{\begin{bmatrix} y_{ii} & y_{ik} \\ y_{ki} & y_{kk} \end{bmatrix}}_{\mathbf{Y}} \tag{12.7}$$

Expanding the left-hand side of Eq. (12.7) and equating terms, we get

$$y_{ii} = ca_{ii} - sa_{ki} \tag{12.8}$$

$$y_{kk} = sa_{ik} + ca_{kk} \tag{12.9}$$

$$y_{ik} = ca_{ik} - sa_{kk} \tag{12.10}$$

$$y_{ki} = sa_{ii} + ca_{ki} \tag{12.11}$$

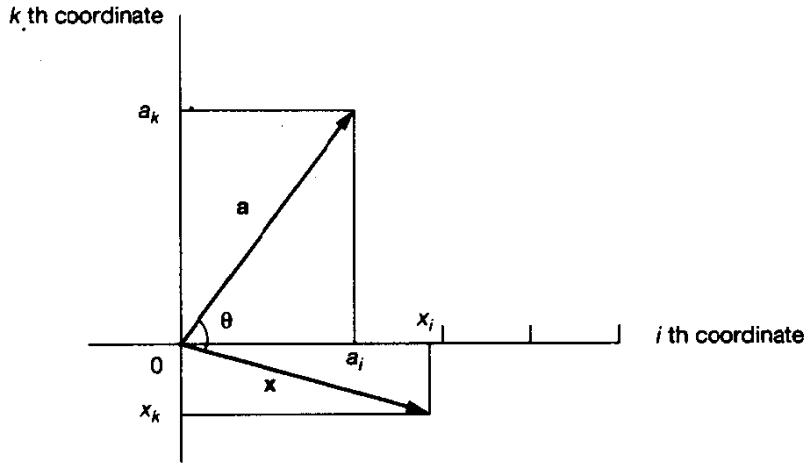


Figure 12.1 Plane rotation of a real 2-by-1 vector.

The purpose of stage I is to compute the cosine–sine pair (c, s) such that the 2-by-2 matrix **Y** produced by the plane rotation $\boldsymbol{\Theta}$ is symmetric. In other words, the elements y_{ik} and y_{ki} are to equal each other.

Define a parameter ρ as the ratio of c to s; that is,

$$\rho = \frac{c}{s} \tag{12.12}$$

We may relate ρ to the elements of the data matrix by setting $y_{ik} = y_{ki}$. Thus, using Eqs. (12.10) and (12.11), we obtain

$$\rho = \frac{a_{ii} + a_{kk}}{a_{ik} - a_{ki}}, \qquad a_{ki} \neq a_{ik} \tag{12.13}$$

Next, we determine the value of s by eliminating c between Eqs. (12.4) and (12.12); hence,

$$s = \frac{\text{sgn}(\rho)}{\sqrt{1 + \rho^2}} \tag{12.14}$$

The computation of c and s thus proceeds as follows:

- Use Eq. (12.13) to evaluate ρ.
- Use Eq. (12.14) to evaluate the sine parameter s.
- Use Eq. (12.12) to evaluate the cosine parameter c.

If **A** is symmetric to begin with, then $a_{ki} = a_{ik}$, in which case we have $s = 0$ and $c = 1$; that is, stage I is bypassed.

Stage II: Diagonalization. The purpose of stage II is to diagonalize the symmetric matrix $\mathbf{Y}$ produced in stage I. To do so, we premultiply and postmultiply it by $\mathbf{\Theta}_2^T$ and $\mathbf{\Theta}_2$, respectively, where $\mathbf{\Theta}_2$ is a second plane rotation to be determined. This operation is simply an orthogonal similarity transformation applied to a symmetric matrix. We may thus write

$$\underbrace{\begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix}^T}_{\mathbf{\Theta}_2^T} \underbrace{\begin{bmatrix} y_{ii} & y_{ki} \\ y_{ki} & y_{kk} \end{bmatrix}}_{\mathbf{Y}} \underbrace{\begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix}}_{\mathbf{\Theta}_2} = \underbrace{\begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}}_{\mathbf{D}} \tag{12.15}$$

Expanding the left-hand side of Eq. (12.15) and then equating the respective diagonal terms, we get

$$d_1 = c_2^2 y_{ii} - 2c_2 s_2 y_{ki} + s_2^2 y_{kk} \tag{12.16}$$

$$d_2 = s_2^2 y_{ii} - 2c_2 s_2 y_{ki} + s_2^2 y_{kk} \tag{12.17}$$

Let o_1 and o_2 denote the off-diagonal terms of the 2-by-2 matrix formed by carrying out the matrix multiplications indicated on the left-hand side of Eq. (12.15). From symmetry considerations, we have

$$o_1 = o_2 \tag{12.18}$$

Evaluating these off-diagonal terms and equating them to zero for diagonalization, we get

$$0 = (y_{ii} - y_{kk}) - \left(\frac{s_2}{c_2}\right) y_{ki} + \left(\frac{c_2}{s_2}\right) y_{ki} \tag{12.19}$$

Equation (12.19) suggests that we introduce the following two definitions:

$$t = \frac{s_2}{c_2} \tag{12.20}$$

and

$$\zeta = \frac{y_{kk} - y_{ii}}{2y_{ki}} \tag{12.21}$$

Hence, we may rewrite Eq. (12.19) as

$$t^2 + 2\zeta t - 1 = 0 \tag{12.22}$$

Equation (12.22) is a quadratic in t; it therefore has two possible solutions, yielding the following different plane rotations:

1. *Inner rotation,* for which we have the solution

$$t = \frac{\text{sign}(\zeta)}{|\zeta| + \sqrt{1 + \zeta^2}} \tag{12.23}$$

Having computed t, we may use Eqs. (12.4) and (12.20) to solve for c_2 and s_2, obtaining

$$c_2 = \frac{1}{\sqrt{1 + t^2}} \tag{12.24}$$

and

$$s_2 = tc_2 \tag{12.25}$$

We note from Eqs. (12.2), (12.3), and (12.20) that the rotation angle θ_2 is related to t as follows:

$$\theta_2 = \arctan t \tag{12.26}$$

Hence, adoption of the solution given in Eq. (12.23) produces a plane rotation Θ_2 for which $|\theta_2|$ lies in the interval $[0, \pi/4]$; this rotation is therefore called an *inner rotation*. The computation thus proceeds as follows:

(a) Use Eq. (12.21) to compute ζ.

(b) Use Eq. (12.23) to compute t.

(c) Use Eqs. (12.24) and (12.25) to compute c_2 and s_2, respectively.

If the original matrix $\mathbf{A}$ is diagonal, then $a_{ik} = a_{ki} = 0$, in which case the angle $\theta_2 = 0$, and so the matrix remains unchanged.

2. *Outer rotation*, for which we have the solution:

$$t = -\operatorname{sign}(\zeta)\,(|\zeta| + \sqrt{1 + \zeta^2}) \tag{12.27}$$

Having computed t, we may then evaluate c_2 and s_2 using the formulas (12.24) and (12.25), respectively; we may do so because the derivations of these two equations are independent of the quadratic equation (12.22). In this second case, however, the use of Eq. (12.27) in (12.26) yields a plane rotation for which $|\theta_2|$ lies in the interval $[\pi/4, \pi/2]$. The rotation associated with the second solution is therefore referred to as the *outer rotation*. Note that if the original matrix $\mathbf{A}$ is diagonal, then $a_{ik} = a_{ki} = 0$, in which case $\theta_2 = \pi/2$. In this special case, the diagonal elements of the matrix are interchanged, as shown by

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a_{ii} & 0 \\ 0 & a_{kk} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} a_{kk} & 0 \\ 0 & a_{ii} \end{bmatrix} \tag{12.28}$$

Fusion of Rotations Θ and Θ_2

Substituting the matrix $\mathbf{Y}$ of Eq. (12.7) in (12.15) and comparing the resulting equation with (12.6), we deduce the following definition for Θ_1 in terms of Θ (determined in the symmetrization stage) and Θ_2 (determined in the diagonalization stage):

$$\Theta_1^T = \Theta_2^T \Theta^T$$

or, equivalently,

$$\Theta_1 = \Theta\,\Theta_2 \tag{12.29}$$

In other words, in terms of the cosine–sine parameters, we have

$$\underbrace{\begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}}_{\Theta_1} = \underbrace{\begin{bmatrix} c & s \\ -s & c \end{bmatrix}}_{\Theta} \underbrace{\begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix}}_{\Theta_2} \tag{12.30}$$

Expanding and equating terms, we thus obtain

$$c_1 = cc_2 - ss_2 \tag{12.31}$$

and

$$s_1 = sc_2 + cs_2 \tag{12.32}$$

For real data, from Eqs. (12.31) and (12.32) we find that the angles θ and θ_2, associated with the plane rotations Θ and Θ_2, respectively, add to produce the angle θ_1 associated with Θ_1.

Two Special Cases

For reasons that will become apparent later in Section 12.3, the Jacobi algorithm for computing the singular-value decomposition has to be capable of handling two special cases:

Case 1: $a_{kk} = a_{ik} = 0$. In this case we need only to perform the symmetrization of **A**, as shown by

$$\begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}^T \begin{bmatrix} a_{ii} & 0 \\ a_{ki} & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} d_1 & 0 \\ 0 & 0 \end{bmatrix} \tag{12.33}$$

Case 2: $a_{kk} = a_{ki} = 0$. In this case, we have

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{ii} & a_{ik} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix} = \begin{bmatrix} d_1 & 0 \\ 0 & 0 \end{bmatrix} \tag{12.34}$$

Additional Operations for Complex Data

The plane rotation described in Eq. (12.6) applies to real data only, because (to begin with) the cosine and sine parameters defining the rotation were all chosen to be real. To extend its application to the more general case of complex data, we have to perform additional operations on the data. At first sight, it may appear that we merely have to modify stage I (symmetrization) of the two-sided Jacobi algorithm so as to accommodate a complex 2-by-2 matrix. In reality, however, the issue of dealing with complex data (in the context

of the Jacobi algorithm) is not so simple. The approach taken here is first to reduce the complex 2-by-2 matrix of Eq. (12.5) to a real form, and then proceed with the application of the two-sided Jacobi algorithm in the usual way.[2] The *complex-to-real data reduction* is performed by following a two-stage procedure, as described next.

Stage I: Triangularization. Consider a complex 2-by-2 data matrix **A** having the form given in Eq. (12.5). Without loss of generality, we assume that the leading element a_{ii} is a positive real number. This assumption may be justified (if need be) by factoring out the exponential term $e^{j\theta_{ii}}$, where θ_{ii} is the phase angle of a_{ii}. The factorization has the effect of leaving inside the 2-by-2 matrix a positive real term equal to the magnitude of a_{ii}, and subtracting θ_{ii} from the phase angle of each of the remaining three complex terms in the matrix.

Let the matrix **A** so described be premultiplied by a 2-by-2 plane rotation for the purpose of its triangularization, as shown by

$$\begin{bmatrix} c & s^* \\ -s & c \end{bmatrix} \begin{bmatrix} a_{ii} & a_{ik} \\ a_{ki} & a_{kk} \end{bmatrix} = \begin{bmatrix} \omega_{ii} & \omega_{ik} \\ 0 & \omega_{kk} \end{bmatrix} \tag{12.35}$$

The cosine parameter c is real, but the sine parameter s is now *complex*. To emphasize this point, we write

$$s = |s|e^{j\alpha} \tag{12.36}$$

where $|s|$ is the magnitude of s, and α is its phase angle. In addition, the (c, s) pair is required to satisfy the constraint

$$c^2 + |s|^2 = 1 \tag{12.37}$$

The objective is to choose the (c, s) pair so as to annihilate the kith (off-diagonal) term. To do this, we must satisfy the condition

$$-sa_{ii} + ca_{ki} = 0$$

or, equivalently,

$$s = \frac{a_{ki}}{a_{ii}} c \tag{12.38}$$

Substituting Eq. (12.38) in (12.37), and solving for the cosine parameter, we get

$$c = \frac{|a_{ii}|}{\sqrt{|a_{ii}|^2 + |a_{ki}|^2}} \tag{12.39}$$

Note that, in Eq. (12.39), we have chosen to work with the *positive* real root for the cosine parameter c. Also, if a_{ki} is zero, that is, the data matrix is upper triangular to begin with, then $c = 1$ and $s = 0$, in which case we may bypass stage I. If, by the same token, a_{ik} is zero, we apply transposition and proceed to stage II.

[2]F. T. Luk, private communication, 1990.

Having determined the values of c and s needed for the triangularization of the 2-by-2 matrix **A**, we may now determine the elements of the resulting upper triangular matrix shown on the right-hand side of Eq. (12.35) as follows:

$$\omega_{ii} = ca_{ii} + s^* a_{ki} \tag{12.40}$$

$$\omega_{ik} = ca_{ik} + s^* a_{kk} \tag{12.41}$$

$$\omega_{kk} = -sa_{ik} + ca_{kk} \tag{12.42}$$

Given that a_{ii} is positive real, by assumption, the use of Eqs. (12.38) and (12.39) in (12.40) reveals that the diagonal element ω_{ii} is real and nonnegative; that is,

$$\omega_{ii} \geq 0 \tag{12.43}$$

In general, however, the remaining two elements ω_{ik} and ω_{kk} of the upper triangular matrix on the right-hand side of Eq. (12.35) are complex valued.

Stage II: Phase Cancelation. As already mentioned, the elements ω_{ik} and ω_{kk} may be complex. To reduce them to real form, we premultiply and postmultiply the upper triangular matrix on the right-hand side of Eq. (12.35) by a pair of *phase-canceling diagonal matrices* as follows:

$$\begin{bmatrix} e^{-j\beta} & 0 \\ 0 & e^{-j\gamma} \end{bmatrix} \begin{bmatrix} \omega_{ii} & \omega_{ik} \\ 0 & \omega_{kk} \end{bmatrix} \begin{bmatrix} e^{j\beta} & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \omega_{ii} & |\omega_{ik}| \\ 0 & |\omega_{kk}| \end{bmatrix} \tag{12.44}$$

The *rotation angles* β and γ of the premultiplying matrix are chosen so as to cancel the phase angles of ω_{ik} and ω_{kk}, respectively, as shown by

$$\beta = \arg(\omega_{ik}) \tag{12.45}$$

$$\gamma = \arg(\omega_{kk}) \tag{12.46}$$

The postmultiplying matrix is included so as to *correct* for the phase change in the element ω_{ii} produced by the premultiplying matrix. In other words, the combined process of premultiplication and postmultiplication in Eq. (12.44) leaves the diagonal element ω_{ii} unchanged.

Stage II thus yields an upper triangular matrix whose three nonzero elements are all real and nonnegative. Note that the procedure described herein for reducing the complex 2-by-2 matrix **A** to a real upper triangular form requires four degrees of freedom, namely, the (c, s) pair, and the angles β and γ. The way is now paved for us to proceed with the application of the Jacobi method for a real 2-by-2 matrix, as described earlier in the section.

12.3 CYCLIC JACOBI ALGORITHM

We are now ready to describe the *cyclic Jacobi algorithm* or *generalized Jacobi algorithm* for a square data matrix by solving an appropriate sequence of 2-by-2 singular-value

decomposition problems. The description will be presented for real data. To deal with complex data, we incorporate the complex-to-real data reduction developed in the preceding section.

Let $\Theta_1(i, k)$ denote a plane rotation in the (i, k) plane, where $k > i$. The matrix $\Theta_1(i, k)$ is the same as the M-by-M identity matrix, except for the four strategic elements located on rows i, k and columns i, k, as shown by

$$\Theta_1(i,k) = \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 \\ \cdot & \cdot & & \cdot & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot \\ 0 & c_1 & \cdots & s_1 & \cdots & 0 \\ \cdot & \cdot & & \cdot & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot \\ 0 & -s_1 & \cdots & c_1 & \cdots & 0 \\ \cdot & \cdot & & \cdot & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot \\ 0 & 0 & \cdots & 0 & \cdots & 1 \\ & \uparrow & & \uparrow & & \\ & \text{column } i & & \text{column } k & & \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \longleftarrow \text{row } i \\ \\ \\ \\ \longleftarrow \text{row } k \\ \\ \\ \\ \\ \\ \end{matrix} \tag{12.47}$$

Let $\Theta_2(i, k)$ denote a second plane rotation in the (i, k) plane that is similarly defined; the dimension of this second transformation is also M. The Jacobi transformation of the data matrix $\mathbf{A}$ is thus described by

$$\mathbf{T}_{ik}\colon \mathbf{A} \leftarrow \Theta_1^T(i, k)\mathbf{A}\Theta_2(i, k) \tag{12.48}$$

The Jacobi rotations $\Theta_1(i, k)$ and $\Theta_2(i, k)$ are designed to annihilate the (i, k) and (k, i) elements of $\mathbf{A}$. Accordingly, the transformation $\mathbf{T}_{ik}$ produces a matrix $\mathbf{X}$ (equal to the updated value of $\mathbf{A}$) that is more diagonal than the original $\mathbf{A}$ in the sense that

$$\text{off}(\mathbf{X}) = \text{off}(\mathbf{A}) - a_{ik}^2 - a_{ki}^2 \tag{12.49}$$

where off(A) is the *norm of the off-diagonal elements*:

$$\text{off}(\mathbf{A}) = \sum_{i=1}^{M} \sum_{\substack{k=1 \\ k \neq i}}^{M} a_{ik}^2 \qquad \text{for } \mathbf{A} = \{a_{ik}\} \tag{12.50}$$

In the cyclic Jacobi algorithm the transformation (12.48) is applied for a total of $m = M(M - 1)/2$ different index pairs ("pivots") that are selected in some fixed order. Such a sequence of m transformations is called a *sweep*. The construction of a sweep may be *cyclic by rows* or *cyclic by columns*, as illustrated in Example 1 below. In either case, we obtain a new matrix $\mathbf{A}$ after each sweep, for which we compute off(A). If off(A) $\leq \delta$,

where δ is some small machine-dependent number, we stop the computation. If on the other hand, $\text{off}(\mathbf{A}) > \delta$, we repeat the computation. For typical values of δ [e.g., $\delta = 10^{-12}\,\text{off}(\mathbf{A}_0)$, where $\mathbf{A}_0$ is the original matrix], the algorithm converges in about 4 to 10 sweeps for values of M in the range of 4 to 2000.

As far as we know, the row ordering or the column ordering is the only ordering that guarantees convergence of the Jacobi cyclic algorithm.[3] By "convergence" we mean

$$\text{off}(\mathbf{A}^{(k)}) \longrightarrow 0 \qquad \text{as } k \longrightarrow \infty \tag{12.51}$$

where $\mathbf{A}^{(k)}$ is the M-by-M matrix computed after sweep number k.

Example 1

Consider a 4-by-4 *real* matrix $\mathbf{A}$. With the matrix dimension $M = 4$, we have a total of six orderings in each sweep. A sweep of orderings cyclic by rows is represented by

$$\mathbf{T}_R = \mathbf{T}_{34}\mathbf{T}_{24}\mathbf{T}_{23}\mathbf{T}_{14}\mathbf{T}_{13}\mathbf{T}_{12}$$

A sweep of orderings cyclic by columns is represented by

$$\mathbf{T}_C = \mathbf{T}_{34}\mathbf{T}_{24}\mathbf{T}_{14}\mathbf{T}_{23}\mathbf{T}_{13}\mathbf{T}_{12}$$

It is easily checked that the transformation $\mathbf{T}_{ik}$ and $\mathbf{T}_{pq}$ *commute* if two conditions hold:

1. The index i is neither p nor q.
2. The index k is neither p nor q.

Accordingly, we find that the transformations $\mathbf{T}_R$ and $\mathbf{T}_C$ are indeed equivalent, as they should be.

Consider next the application of the transformation $\mathbf{T}_R$ (obtained from the sweep of orderings cyclic by rows) to the data matrix $\mathbf{A}$. In particular, using the rotation of Eq. (12.48), we may write the following transformations:

$$\mathbf{T}_{12} : \mathbf{A} \leftarrow \boldsymbol{\Theta}_1^T(1, 2)\mathbf{A}\boldsymbol{\Theta}_2(1, 2)$$

$$\mathbf{T}_{13}\mathbf{T}_{12} : \mathbf{A} \leftarrow \boldsymbol{\Theta}_3^T(1, 3)\boldsymbol{\Theta}_1^T(1, 2)\mathbf{A}\boldsymbol{\Theta}_2(1, 2)\boldsymbol{\Theta}_4(1, 3)$$

$$\mathbf{T}_{14}\mathbf{T}_{13}\mathbf{T}_{12} : \mathbf{A} \leftarrow \boldsymbol{\Theta}_5^T(1, 4)\boldsymbol{\Theta}_3^T(1, 3)\boldsymbol{\Theta}_1^T(1, 2)\mathbf{A}\boldsymbol{\Theta}_2(1, 2)\boldsymbol{\Theta}_4(1, 3)\boldsymbol{\Theta}_6(1, 4)$$

and so on. The final step in this sequence of transformations may be written as

$$\mathbf{T}_R : \mathbf{A} \longleftarrow \mathbf{U}^T\mathbf{A}\mathbf{V}$$

[3] A proof of convergence of the Jacobi cyclic algorithm, based on row ordering or column ordering, is given in Forsythe and Henrici (1960). Subsequently, Luk and Park (1989) proved that many of the orderings used in parallel implementation of the algorithm are equivalent to the row ordering, and thus guarantee convergence as well.

which defines the singular value decomposition of the real data matrix **A**. The orthogonal matrices **U** and **V** are respectively defined by

$$\mathbf{U} = \boldsymbol{\Theta}_1(1, 2)\boldsymbol{\Theta}_3(1, 3)\boldsymbol{\Theta}_5(1, 4)\boldsymbol{\Theta}_7(2, 3)\boldsymbol{\Theta}_9(2, 4)\boldsymbol{\Theta}_{11}(3, 4)$$

and

$$\mathbf{V} = \boldsymbol{\Theta}_2(1, 2)\boldsymbol{\Theta}_4(1, 3)\boldsymbol{\Theta}_6(1, 4)\boldsymbol{\Theta}_8(2, 3)\boldsymbol{\Theta}_{10}(2, 4)\boldsymbol{\Theta}_{12}(3, 4)$$

Rectangular Data Matrix

Thus far we have focused attention on the cyclic Jacobi algorithm for computing the singular-value decomposition of a square matrix. To handle the more general case of a rectangular matrix, we may extend the use of this algorithm by proceeding as follows. Consider first the case of a K-by-M real data matrix **A**, for which K is greater than M. We generate a square matrix by appending $(K - M)$ columns of zeros to **A**. We may thus write

$$\tilde{\mathbf{A}} = [\mathbf{A}, \mathbf{O}] \tag{12.52}$$

We refer to $\tilde{\mathbf{A}}$ as the *augmented data matrix*. We then proceed as before by applying the cyclic Jacobi algorithm to the K-by-K matrix $\tilde{\mathbf{A}}$. In performing this computation, we require the use of special case 1 described in Eq. (12.33). In any event, we emerge with the factorization

$$\mathbf{U}^T[\mathbf{A}, \mathbf{O}]\begin{bmatrix} \mathbf{V} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{bmatrix} = \text{diag}(\sigma_1, \ldots, \sigma_M, 0, \ldots, 0) \tag{12.53}$$

The desired factorization of the original data matrix **A** is obtained by writing

$$\mathbf{U}^T\mathbf{A}\mathbf{V} = \text{diag}(\sigma_1, \ldots, \sigma_M) \tag{12.54}$$

If, on the other hand, the dimension M of matrix **A** is greater than K, we augment it by adding $(M - K)$ rows; we may thus write

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} \\ \mathbf{O} \end{bmatrix} \tag{12.55}$$

We then treat the square matrix $\tilde{\mathbf{A}}$ in the same way as before. In this second situation, we require the use of special case 2 described in Eq. (12.34).

In the case of a complex rectangular data matrix **A**, we may proceed in a fashion similar to that described above, except for a change in the characterization of matrices **U** and **V**. For a real data matrix, the matrices **U** and **V** are both orthogonal, whereas for a complex data matrix they are both unitary.

The strategy of matrix augmentation described herein represents a straightforward extension of the cyclic Jacobi algorithm for a square matrix. A drawback of this approach,

however, is that the algorithm becomes too inefficient if the dimension K of matrix $\mathbf{A}$ is much greater than the dimension M, or vice versa.[4]

12.4 HOUSEHOLDER TRANSFORMATION

We turn next to the *Householder transformation* or the *Householder matrix*, which is so named in recognition of its originator (Householder, 1958 a,b, 1964). To proceed with a discussion of this issue, let $\mathbf{u}$ be an M-by-1 vector whose Euclidean norm is

$$\|\mathbf{u}\| = (\mathbf{u}^H\mathbf{u})^{1/2}$$

Then, the Householder transformation, denoted by an M-by-M matrix $\mathbf{Q}$, is defined by

$$\mathbf{Q} = \mathbf{I} - \frac{2\mathbf{u}\mathbf{u}^H}{\|\mathbf{u}\|^2} \tag{12.56}$$

where $\mathbf{I}$ is the M-by-M identity matrix.

For a geometric interpretation of the Householder transformation, consider an M-by-1 vector $\mathbf{x}$ premultiplied by the matrix $\mathbf{Q}$, as shown by

$$\begin{aligned}\mathbf{Q}\mathbf{x} &= \left(\mathbf{I} - \frac{2\mathbf{u}\mathbf{u}^H}{\|\mathbf{u}\|^2}\right)\mathbf{x} \\ &= \mathbf{x} - \frac{2\mathbf{u}^H\mathbf{x}}{\|\mathbf{u}\|^2}\,\mathbf{u}\end{aligned} \tag{12.57}$$

By definition, the *projection* of $\mathbf{x}$ onto $\mathbf{u}$ is given by

$$\mathbf{P}_u(\mathbf{x}) = \frac{\mathbf{u}^H\mathbf{x}}{\|\mathbf{u}\|^2}\,\mathbf{u} \tag{12.58}$$

This projection is illustrated in Fig. 12.2. In this figure we have also included the vector representation of the product $\mathbf{Qx}$. We thus see that $\mathbf{Qx}$ is the mirror-image *reflection* of the vector $\mathbf{x}$ with respect to the hyperplane span $\{\mathbf{u}\}^\perp$ which is perpendicular to the vector $\mathbf{u}$. It is for this reason that the Householder transformation is also known as the *Householder reflection*.[5]

[4]An alternative approach that overcomes this difficulty is to proceed as follows (Luk, 1986):

1. *Triangularize* the K-by-M data matrix $\mathbf{A}$ by performing a QR-decomposition, defined by

$$\mathbf{A} = \mathbf{Q}\begin{bmatrix}\mathbf{R} \\ \mathbf{O}\end{bmatrix}$$

where $\mathbf{Q}$ is a K-by-K orthogonal matrix, and $\mathbf{R}$ is an M-by-M upper triangular matrix.

2. Diagonalize the matrix $\mathbf{R}$ using the cyclic Jacobi algorithm.

3. Combine the results of steps 1 and 2.

[5]For a tutorial review of the Householder transformation and its use in adaptive signal processing, see Steinhardt (1988).

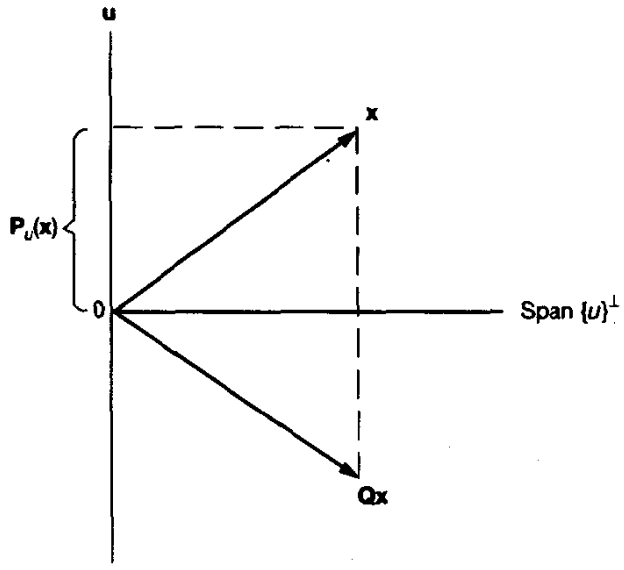


Figure 12.2 Geometric interpretation of the Householder transformation.

The Householder transformation, defined in Eq. (12.56), has the following properties:

Property 1. *The Householder transformation* $\mathbf{Q}$ *is Hermitian*; that is,

$$\mathbf{Q}^H = \mathbf{Q} \tag{12.59}$$

Property 2. *The Householder transformation* $\mathbf{Q}$ *is unitary*; that is,

$$\mathbf{Q}^{-1} = \mathbf{Q}^H \tag{12.60}$$

Property 3. *The Householder transformation is length preserving*; that is,

$$\|\mathbf{Qx}\| = \|\mathbf{x}\| \tag{12.61}$$

This property is illustrated in Fig. 12.2, where we see that the vector $\mathbf{x}$ and its reflection $\mathbf{Qx}$ have exactly the same length.

Property 4. *If two vectors undergo the same Householder transformation, their inner product remains unchanged.*

Consider any three vectors, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{u}$. Let the Householder matrix $\mathbf{Q}$ be defined in terms of the vector $\mathbf{u}$, as in Eq. (12.56). Let the remaining two vectors $\mathbf{x}$ and $\mathbf{y}$ be transformed by $\mathbf{Q}$, yielding $\mathbf{Qx}$ and $\mathbf{Qy}$, respectively. The inner product of these two transformed vectors is

$$\begin{aligned}(\mathbf{Qx})^H(\mathbf{Qy}) &= \mathbf{x}^H\mathbf{Q}^H\mathbf{Qy} \\ &= \mathbf{x}^H\mathbf{y}\end{aligned} \tag{12.62}$$

where we have made use of Property 2. Hence, the transformed vectors $\mathbf{Qx}$ and $\mathbf{Qy}$ have the same inner product as the original vectors $\mathbf{x}$ and $\mathbf{y}$.

Property 4 has important practical implications in the numerical solution of linear least-squares problems. Specifically, Householder transformations are used to reduce the given data matrix to a *sparse* matrix (i.e., one that consists mostly of zeros), but which is "equivalent" to the original data matrix in some mathematical sense. Needless to say, the particular form of matrix sparseness used depends on the application of interest. Whatever the application, however, the data reduction is used in order to simplify the numerical computations involved in solving the problem. In this context, a popular form of data reduction is that of *triangularization*, referring to the reduction of a full data matrix to an upper triangular one. Given this form of data reduction, we may then simply use *Gaussian elimination* to perform the matrix inversion and thereby compute the least-squares solution to the problem.

Properties 1 through 4 apply not only to the Householder transformations but also to the Givens rotations. It is the next two properties that distinguish Householder transformations from Givens rotations.

Property 5. *Given the Householder transformation* $\mathbf{Q}$, *the transformed vector* $\mathbf{Qx}$ *is a reflection of* $\mathbf{x}$ *above the hyperplane perpendicular to the vector* $\mathbf{u}$ *involved in the definition of* $\mathbf{Q}$.

This property is merely a restatement of Eq. (12.57). The following two limiting cases of Property 5 are especially noteworthy:

- The vector $\mathbf{x}$ is a scalar multiple of $\mathbf{u}$: In this case, Eq. (12.57) simplifies to
$$\mathbf{Qx} = -\mathbf{x}$$
- The vector $\mathbf{x}$ is orthogonal to $\mathbf{u}$; that is, their inner product is zero: In this second case, Eq. (12.57) reduces to
$$\mathbf{Qx} = \mathbf{x}$$

Property 6. *Let* $\mathbf{x}$ *be any nonzero* M*-by-*1 *vector with Euclidean norm* $\|\mathbf{x}\|$. *Let the* M*-by-*1 *vector* $\mathbf{1}$ *denote the first column of the identity matrix; that is,*

$$\mathbf{1} = [1, 0, \ldots, 0]^T \tag{12.63}$$

Then there exists a Householder transformation $\mathbf{Q}$ defined by the vector

$$\mathbf{u} = \mathbf{x} - \|\mathbf{x}\|\mathbf{1} \tag{12.64}$$

such that the transformed vector $\mathbf{Qx}$ *corresponding to* $\mathbf{u}$ *is a linear multiple of the vector* $\mathbf{1}$.

With the vector $\mathbf{u}$ assigned the value in Eq. (12.64), we have

$$\begin{aligned} \|\mathbf{u}\|^2 &= \mathbf{u}^H\mathbf{u} \\ &= (\mathbf{x} - \|\mathbf{x}\|\mathbf{1})^H(\mathbf{x} - \|\mathbf{x}\|\mathbf{1}) \\ &= 2\|\mathbf{x}\|^2 - 2\|\mathbf{x}\|x_1 \\ &= 2\|\mathbf{x}\|(\|\mathbf{x}\| - x_1) \end{aligned} \tag{12.65}$$

where x_1 is the first element of the vector $\mathbf{x}$. Similarly, we may write

$$\begin{aligned}\mathbf{u}^H\mathbf{x} &= (\mathbf{x} - \|\mathbf{x}\|\mathbf{1})^H\mathbf{x} \\ &= \|\mathbf{x}\|^2 - \|\mathbf{x}\|x_1 \\ &= \|\mathbf{x}\|(\|\mathbf{x}\| - x_1)\end{aligned} \tag{12.66}$$

Accordingly, substituting Eqs. (12.65) and (12.66) in Eq. (12.57), we find that the transformed vector $\mathbf{Qx}$ corresponding to the defining vector $\mathbf{u}$ of Eq. (12.64) is given by

$$\begin{aligned}\mathbf{Qx} &= \mathbf{x} - \mathbf{u} \\ &= \mathbf{x} - (\mathbf{x} - \|\mathbf{x}\|\mathbf{1}) \\ &= \|\mathbf{x}\|\mathbf{1}\end{aligned} \tag{12.67}$$

which proves Property 6.

From Eq. (12.65), we observe that the first element x_1 of the vector $\mathbf{x}$ has to be real, and the Euclidean norm of $\mathbf{x}$ has to satisfy the condition

$$\|\mathbf{x}\| > |x_1| \tag{12.68}$$

This condition merely says that not only the first element of $\mathbf{x}$ but also one other element must be nonzero. Then, the vector $\mathbf{u}$ defined by Eq. (12.64) is indeed effective.

Property 6 makes the Householder transformation a very powerful computational tool. Given a vector $\mathbf{x}$, we may use Eq. (12.64) to define the vector $\mathbf{u}$ such that the corresponding Householder transformation $\mathbf{Q}$ annihilates all the M elements of the vector $\mathbf{x}$ except for the first one. This result is equivalent to the application of $(M - 1)$ plane rotations, with a minor difference: The determinant of the Householder matrix $\mathbf{Q}$ defined in Eq. (12.56) is

$$\begin{aligned}\det(\mathbf{Q}) &= \det\left(\mathbf{I} - \frac{2\mathbf{u}\mathbf{u}^H}{\|\mathbf{u}\|^2}\right) \\ &= -1\end{aligned} \tag{12.69}$$

Hence, the Householder transformation reverses the orientation of the configuration.

Having familiarized ourselves with the Householder transformation, we are ready to resume our discussion of SVD computation by describing the QR algorithm, which we do in the next section.

12.5 THE QR ALGORITHM

The starting point in the development of the *QR algorithm* for SVD computation is that of finding a class of unitary matrices, which preserve the singular values of a data matrix $\mathbf{A}$. In this context, the matrix $\mathbf{A}$ is said to be *unitarily equivalent* to another matrix $\mathbf{B}$ if

$$\mathbf{B} = \mathbf{PAQ} \tag{12.70}$$

where $\mathbf{P}$ and $\mathbf{Q}$ are unitary matrices; that is,

$$\mathbf{P}^H\mathbf{P} = \mathbf{I}$$

$$\mathbf{Q}^H\mathbf{Q} = \mathbf{I}$$

Consequently, we have

$$\begin{aligned}\mathbf{B}^H\mathbf{B} &= \mathbf{Q}^H\mathbf{A}^H\mathbf{P}^H\mathbf{P}\mathbf{A}\mathbf{Q} \\ &= \mathbf{Q}^H\mathbf{A}^H\mathbf{A}\mathbf{Q}\end{aligned} \tag{12.71}$$

Postmultiplying the correlation matrix $\mathbf{A}^H\mathbf{A}$ by a unitary matrix $\mathbf{Q}$ and premultiplying it by the Hermitian transpose of the matrix $\mathbf{Q}$ leaves the eigenvalues of $\mathbf{A}^H\mathbf{A}$ unchanged. Accordingly, the correlation matrices $\mathbf{A}^H\mathbf{A}$ and $\mathbf{B}^H\mathbf{B}$, or more simply, the matrices $\mathbf{A}$ and $\mathbf{B}$ themselves are said to be *eigen-equivalent*.

The purpose of using the transformation defined in Eq. (12.70) is to reduce the data matrix $\mathbf{A}$ to *upper bidiagonal* form, with eigen-equivalence maintained, for which Householder transformations are well suited. The reduced data matrix $\mathbf{B}$ is said to be upper bidiagonal if all of its elements except for those on the main diagonal and the superdiagonal are zero; that is, the ijth element of $\mathbf{B}$ is

$$b_{ij} = 0 \qquad \text{whenever } i > j \text{ or } j > i + 1 \tag{12.72}$$

Having reduced the data matrix $\mathbf{A}$ to upper bidiagonal form, the next step is the application of the Golub–Kahan SVD algorithm. These two steps, in turn, are considered next.

Householder Bidiagonalization

Consider a K-by-M data matrix $\mathbf{A}$, where $K \geq M$. Let $\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_M$ denote a set of K-by-K Householder matrices, and let $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_{M-2}$ denote another set of M-by-M Householder matrices. In order to reduce the data matrix $\mathbf{A}$ to upper bidiagonal form, we determine the products of Householder matrices

$$\mathbf{Q}_B = \begin{cases} \mathbf{Q}_1\mathbf{Q}_2 \ldots \mathbf{Q}_{M-1}, & K = M \\ \mathbf{Q}_1\mathbf{Q}_2 \ldots \mathbf{Q}_M, & K > M \end{cases} \tag{12.73}$$

and

$$\mathbf{P}_B = \mathbf{P}_1\mathbf{P}_2 \ldots \mathbf{P}_{M-2} \tag{12.74}$$

such that

$$\mathbf{Q}_B^H\mathbf{A}\mathbf{P}_B = \mathbf{B} = \begin{bmatrix} d_1 & f_2 & & \mathbf{O} \\ & d_2 & \ddots & \\ & & \ddots & f_M \\ \mathbf{O} & & & d_M \\ & & \mathbf{O} & \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \} (K-M)\text{-by-}M \text{ null matrix} \end{matrix} \tag{12.75}$$

For $K > M$, premultiplication of the data matrix $\mathbf{A}$ by the Householder matrices $\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_M$ corresponds to reflecting the respective columns of $\mathbf{A}$, whereas postmultiplication by the Householder matrices $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_{M-2}$ corresponds to reflecting the respective rows of $\mathbf{A}$. The desired upper bidiagonal form is attained by "ping-ponging" column and row reflections. Note that for $K > M$ the number of Householder matrices constituting $\mathbf{Q}_B$ is M, whereas those constituting $\mathbf{P}_B$ number $M - 2$. Note also that, by construction, the matrix product $\mathbf{P}_1\mathbf{P}_2 \ldots \mathbf{P}_{M-2}$ does *not* alter the first column of any matrix that it postmultiplies.

We illustrate this data reduction process by way of an example.

Example 2

Consider a 5-by-4 data matrix $\mathbf{A}$ written in expanded form as follows:

$$\mathbf{A} = \begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{bmatrix}$$

where the x's denote nonzero matrix entries. The upper bidiagonalization of $\mathbf{A}$ proceeds as follows. First, $\mathbf{Q}_1$ is chosen so that $\mathbf{Q}_1^H\mathbf{A}$ has zeros in the positions distinguished below:

$$\begin{bmatrix} x & x & x & x \\ \otimes & x & x & x \\ \otimes & x & x & x \\ \otimes & x & x & x \\ \otimes & x & x & x \end{bmatrix}$$

Thus, $\mathbf{Q}_1^H\mathbf{A}$ may be written as

$$\mathbf{Q}_1^H\mathbf{A} = \begin{bmatrix} x & x & \otimes & \otimes \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix} \tag{12.76}$$

Next, $\mathbf{P}_1$ is chosen so that $\mathbf{Q}_1^H\mathbf{A}\mathbf{P}_1$ has zeros in the positions distinguished in the first row of $\mathbf{Q}_1^H\mathbf{A}$, as in Eq. (12.76). Hence, $\mathbf{Q}_1^H\mathbf{A}\mathbf{P}_1$ has the form

$$\mathbf{Q}_1^H\mathbf{A}\mathbf{P}_1 = \left[\begin{array}{c|ccc} x & x & 0 & 0 \\ \hline 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{array}\right] \tag{12.77}$$

Note that $\mathbf{P}_1$ does not affect the first column of the matrix.

The data reduction is continued by operating on the trailing 4-by-3 submatrix of $\mathbf{Q}^H\mathbf{A}\mathbf{P}_1$ that has nonzero entries. Specifically, we choose $\mathbf{Q}_2$ and $\mathbf{P}_2$ so that $\mathbf{Q}_2^H\mathbf{Q}_1^H\mathbf{A}\mathbf{P}_1\mathbf{P}_2$ has the form

$$\mathbf{Q}_2^H\mathbf{Q}_1^H\mathbf{A}\mathbf{P}_1\mathbf{P}_2 = \left[\begin{array}{cc:cc} x & x & 0 & 0 \\ 0 & x & x & 0 \\ \hdashline 0 & 0 & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{array}\right] \tag{12.78}$$

Next, we operate on the trailing 3-by-2 submatrix of $\mathbf{Q}_2^H\mathbf{Q}_1^H\mathbf{A}\mathbf{P}_1\mathbf{P}_2$ that has nonzero entries. Specifically, we choose $\mathbf{Q}_3$ such that $\mathbf{Q}_3^H\mathbf{Q}_2^H\mathbf{Q}_1^H\mathbf{A}\mathbf{P}_1\mathbf{P}_2$ has the form

$$\mathbf{Q}_3^H\mathbf{Q}_2^H\mathbf{Q}_1^H\mathbf{A}\mathbf{P}_1\mathbf{P}_2 = \left[\begin{array}{ccc:c} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ \hdashline 0 & 0 & 0 & x \\ 0 & 0 & 0 & x \end{array}\right] \tag{12.79}$$

Finally, we choose $\mathbf{Q}_4$ to operate on the trailing 2-by-1 submatrix of $\mathbf{Q}_3^H\mathbf{Q}_2^H\mathbf{Q}_1^H\mathbf{Q}\mathbf{P}_1\mathbf{P}_2$, such that we may write

$$\mathbf{B} = \mathbf{Q}_4^H\mathbf{Q}_3^H\mathbf{Q}_2^H\mathbf{Q}_1^H\mathbf{A}\mathbf{P}_1\mathbf{P}_2 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{12.80}$$

This completes the upper bidiagonalization of the data matrix $\mathbf{A}$.

The Golub–Kahan Step

The bidiagonalization of the data matrix $\mathbf{A}$ is followed by an *iterative process* that reduces it further to *diagonal* form. Referring to Eq. (12.75), we see that the matrix $\mathbf{B}$, resulting from the bidiagonalization of $\mathbf{A}$, is zero below the Mth row. Evidently, the last $K - M$ rows of zeros in the matrix $\mathbf{B}$ do *not* contribute to the singular values of the original data matrix $\mathbf{A}$. Accordingly, it is convenient to delete the last $K - M$ rows of matrix $\mathbf{B}$ and thus treat it as a square matrix with dimension M. The basis of the diagonalization of matrix $\mathbf{B}$ is the *Golub–Kahan algorithm* (Golub and Kahan, 1965), which is an adaptation of the QR *algorithm* developed originally for solving the symmetric eigenvalue problem.[6]

[6]The explicit form of the QR algorithm is a variant of the QL algorithm discussed in Chapter 4.

Let $\mathbf{B}$ denote an M-by-M upper bidiagonal matrix having no zeros on its main diagonal or superdiagonal. The first iteration of the Golub–Kahan algorithm proceeds as follows (Golub and Kahan, 1965; Golub and Van Loan, 1989):

1. Identify the trailing 2-by-2 submatrix of the product $\mathbf{T} = \mathbf{B}^H\mathbf{B}$, which has the form

$$\begin{bmatrix} |d_{M-1}|^2 + |f_{M-1}|^2 & d_{M-1}^* f_M \\ f_M^* d_{M-1} & |d_M|^2 + |f_M|^2 \end{bmatrix} \tag{12.81}$$

   where d_{M-1} and d_M are the trailing diagonal elements of matrix $\mathbf{B}$, and f_{M-1} and f_M are the trailing superdiagonal elements; see the right-hand side of Eq. (12.75). Compute the eigenvalue λ of this 2-by-2 submatrix, which is closer to $|d_M|^2 + |f_M|^2$; this particular eigenvalue λ is known as the *Wilkinson shift*.

2. Compute the Givens rotation parameters c_1 and s_1 such that

$$\begin{bmatrix} c_1 & s_1^* \\ -s_1 & c_1 \end{bmatrix} \begin{bmatrix} |d_1|^2 - \lambda \\ f_2^* d_1 \end{bmatrix} = \begin{bmatrix} \star \\ 0 \end{bmatrix} \tag{12.82}$$

   where d_1 and f_2 are the leading main diagonal and superdiagonal elements of matrix $\mathbf{B}$, respectively; see the right-hand side of Eq. (12.75). The element marked $\star$ on the right-hand side of Eq. (12.82) indicates a nonzero element. Set

$$\mathbf{\Theta}_1 = \left[\begin{array}{cc:c} c_1 & s_1^* & \\ -s_1 & c_1 & \mathbf{O} \\ \hdashline & & \\ \multicolumn{2}{c:}{\mathbf{O}} & \mathbf{I} \end{array}\right] \tag{12.83}$$

3. Apply the Givens rotation $\mathbf{\Theta}_1$ to matrix $\mathbf{B}$ directly. Since $\mathbf{B}$ is upper bidiagonal, and $\mathbf{\Theta}_1$ is a rotation in the (2, 1) plane, it follows that the matrix product $\mathbf{B}$ has the following form (illustrated for the case of $M = 4$):

$$\mathbf{B}\mathbf{\Theta}_1 = \begin{bmatrix} \mathrm{x} & \mathrm{x} & 0 & 0 \\ z^{(1)} & \mathrm{x} & \mathrm{x} & 0 \\ 0 & 0 & \mathrm{x} & \mathrm{x} \\ 0 & 0 & 0 & \mathrm{x} \end{bmatrix}$$

   where $z^{(1)}$ is a new element produced by the Givens rotation $\mathbf{\Theta}_1$.

4. Determine the sequence of Givens rotations $\mathbf{U}_1$, $\mathbf{V}_2$, $\mathbf{U}_2$, . . . , $\mathbf{V}_{M-1}$, and $\mathbf{U}_{M-1}$ operating on $\mathbf{B}\mathbf{\Theta}_1$ in a "ping-pong" fashion so as to chase the unwanted nonzero element $z^{(1)}$ down the bidiagonal. This sequence of operations is illustrated below, again for the case of $M = 4$:

$$\mathbf{U}_1^H\mathbf{B}\boldsymbol{\Theta}_1 = \begin{bmatrix} x & x & z^{(2)} & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{bmatrix}$$

$$\mathbf{U}_1^H\mathbf{V}\mathbf{B}\boldsymbol{\Theta}_1\mathbf{V}_2 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & z^{(3)} & x & x \\ 0 & 0 & 0 & x \end{bmatrix}$$

$$\mathbf{U}_2^H\mathbf{U}_1^H\mathbf{B}\boldsymbol{\Theta}_1\mathbf{V}_2 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & z^{(4)} \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{bmatrix}$$

$$\mathbf{U}_2^H\mathbf{U}_1^H\mathbf{B}\boldsymbol{\Theta}_1\mathbf{V}_2\mathbf{V}_3 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & z^{(5)} & x \end{bmatrix}$$

$$\mathbf{U}_3^H\mathbf{U}_2^H\mathbf{U}_1^H\mathbf{B}\boldsymbol{\Theta}_1\mathbf{V}_2\mathbf{V}_3 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{bmatrix}$$

The iteration thus terminates with a new bidiagonal matrix $\mathbf{B}$ that is related to the original bidiagonal matrix $\mathbf{B}$ as follows:

$$\mathbf{B} \leftarrow (\mathbf{U}_{M-1}^H \ldots \mathbf{U}_2^H\mathbf{U}_1^H)\mathbf{B}(\boldsymbol{\Theta}_1\mathbf{V}_2 \ldots \mathbf{V}_{M-1}) = \mathbf{U}^H\mathbf{B}\mathbf{V} \tag{12.84}$$

where

$$\mathbf{U} = \mathbf{U}_1\mathbf{U}_2 \ldots \mathbf{U}_{M-1} \tag{12.85}$$

and

$$\mathbf{V} = \boldsymbol{\Theta}_1\mathbf{V}_2 \ldots \mathbf{V}_{M-1} \tag{12.86}$$

Steps 1 through 4 constitute one iteration of the Golub–Kahan algorithm. Typically, after a few iterations of this algorithm, the superdiagonal entry f_M becomes negligible. When f_M

becomes sufficiently small, we can *deflate* the matrix and apply the algorithm to the smaller matrix. The criterion for the smallness of f_M is usually of the following form:

$$|f_M| \leq \epsilon\,(|d_{M-1}| + |d_M|) \text{ where } \epsilon \text{ is the machine precision} \qquad (12.87)$$

The description just presented leaves much unsaid about the Golub–Kahan algorithm for the diagonaliztion of a square data matrix. For a more detailed treatment of the algorithm, the reader is referred to the original paper of Golub and Kahan (1965) or the book by Golub and Van Loan (1989).

Recently, there has been a significant improvement in the Golub–Kahan algorithm. The Golub–Kahan algorithm has the property that it computes every singular value of a bidiagonal matrix **B** with an absolute error bound of about $\epsilon\|\mathbf{B}\|$, where ϵ is the machine precision. Thus large singular values (those near $\|\mathbf{B}\|$) are computed with high relative accuracy, but small ones (those near $\epsilon\|\mathbf{B}\|$ or smaller) may have no relative accuracy at all. The new algorithm computes every singular value to high relative accuracy independent of its size. It also computes the singular vectors much more accurately. It is also approximately as fast as the old algorithm (and occasionally much faster). The new algorithm is a hybrid of the Golub–Kahan algorithm and a simplified version that corresponds to taking $\lambda = 0$ in Eq. (12.82). When $\lambda = 0$, the remainder of the algorithm can be stabilized so as to compute every matrix entry to high relative accuracy, whence the final accuracy of the singular values. The analysis of this algorithm can be found in Demmel and Kahan (1990) and Deift et al. (1989).

Summary of the QR Algorithm

The QR algorithm is not only mathematically elegant, but also a computationally powerful and highly versatile algorithm for SVD computation. Given a K-by-M data matrix **A**, the QR algorithm used to compute its SVD proceeds as follows:

1. Compute a sequence of Householder transformations that reduce the matrix **B** to upper bidiagonal form.
2. Apply the Golub–Kahan algorithm to the M-by-M nonzero submatrix resulting from step 1, and iterate this application until the superdiagonal elements become negligible in accordance with the criterion defined in Eq. (12.87).
3. The SVD of the data matrix **A** is determined as follows:
   - The diagonal elements of the matrix resulting from step 2 are the singular values of matrix **A**.
   - The product of the Householder transformations of step 1 and the Givens rotations of step 2 involved in premultiplication defines the left singular vectors of **A**. The product of the Householder transformations and Givens rotations involved in postmultiplication define the right-singular vectors of **A**.

TABLE 12.1 ILLUSTRATING THE FIRST TWO ITERATIONS OF THE GOLUB–KAHAN ALGORITHM

Iteration number	Matrix **B**		
0	1.0000	1.0000	0.0000
	0.0000	2.0000	1.0000
	0.0000	0.0000	3.0000
1	0.9155	0.6627	0.0000
	0.0000	2.0024	0.0021
	0.0000	0.0000	3.2731
2	0.8817	0.4323	0.0000
	0.0000	2.0791	0.0000
	0.0000	0.0000	3.2731

Example 3

Consider the real valued 3-by-3 bidiagonal matrix:

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix}$$

The iterative application of the Golub–Kahan algorithm to this matrix yields the sequence of results shown in Table 12.1 for $\epsilon = 10^{-4}$ in the stopping rule defined in Eq. (12.87). After two iterations of the algorithm, matrix **B** becomes small, at which point it is deflated. Specifically, we now work on the 2-by-2 leading principal submatrix:

$$\begin{matrix} 0.8817 & 0.4323 \\ 0.0000 & 2.0791 \end{matrix}$$

This submatrix is finally diagonalized in one step, yielding

$$\begin{matrix} 0.8596 & 0.0000 \\ 0.0000 & 2.1326 \end{matrix}$$

The singular values of the bidiagonal matrix are thus computed to be:

$$\sigma_1 = 0.8596$$
$$\sigma_2 = 2.1326$$
$$\sigma_3 = 3.2731$$

12.6 SUMMARY AND DISCUSSION

The *singular-value decomposition (SVD)* has become a fundamental tool in linear algebra, system theory, and signal processing (Kung et al., 1985; Deprettere, 1988; Van Loan, 1989; Haykin, 1989). Not only does the SVD permit an *elegant problem formulation, it*

also provides geometrical and algebraic insight together with a numerically robust implementation (Golub and Van Loan, 1989). It includes the eigenvalue decomposition of a nonnegative-definite matrix (e.g., correlation matrix) as a special case. In the context of our present discussion, the SVD provides a direct and numerically robust solution for the linear least-squares estimation problem, be it overdetermined or underdetermined; by "direct" we mean that the solution is obtained by applying the SVD *directly* to the data matrix.

The basic idea behind an algorithm used to compute the singular-value decomposition is to "nudge" a data matrix toward a diagonal form in a step-by-step fashion. The two most common iterative algorithms used to do this nudging are (1) the *cyclic Jacobi algorithm*, and (2) the *QR algorithm* (not to be confused with the QR-decomposition). The QR algorithm, in general, is computationally more *efficient* (i.e., requires less operations) than the cyclic Jacobi algorithm. On the other hand, the cyclic Jacobi algorithm is the preferred method when accuracy demands are extraordinary.

In Mathias (1995), building on and greatly simplifying the previous work by Demmel and Veselic (1989), it is shown that Jacobi's method (involving a sequence of elementary orthogonal matrices) is guaranteed to compute the eigenvalues and eigenvectors of a real-valued positive definite matrix more accurately than the QR algorithm. It is also shown that in the case of an M-by-N matrix with size M much bigger than N, Jacobi's method computes the singular values of the matrix essentially as quickly as the QR algorithm, but potentially much more accurately. With regard to the latter point, Jacobi's method and the QR algorithm start by reducing the matrix to an N-by-N matrix, which requires the same amount of work in both methods. Then the QR algorithm requires less work than Jacobi's method, but this exra work is just on N-by-N matrices and so it is negligible compared to the work required to reduce the M-by-N matrix to N-by-N.

There are two types of new algorithms for SVD computation which deserve special mention:

1. For singular values, the algorithm described in Fernando and Parlett, (1994) is several times faster, and more accurate, than its predecessor in LAPACK; for a short note on LAPACK; see Section 4.5. This new algorithm can be implemented in either parallel or pipelined form, with each iteration (performed on an M-by-M symmetric positive-definite matrix) nominally taking $O(\log_2 M)$ operations. The interesting point to note is that the development of the algorithm by Fernando and Parlett breaks away from the traditional *orthogonal paradigm* that has dominated the field of matrix computations since the 1960's. Specifically, the QR algorithm is abandoned in favor of the *Cholesky LR algorithm* that consists of successive applications of the Cholesky factorization. Given a symmetric positive-definite matrix $\mathbf{A}$, its Cholesky factorization may be written as

$$\mathbf{A} = \mathbf{L}^H\mathbf{L}$$

where $\mathbf{L}$ is a lower triangular matrix and $\mathbf{L}^H$ is its Hermitian transpose.

2. For singular vectors, the algorithm described in Gu and Eisenstat (1994) and Gu et al. (1994) is faster, and more accurate, than the QR algorithm. This second algorithm is based on a *divide-and-conquer* strategy that involves removing a whole column/row of the bidiagonal matrix (resulting from the Householder transformation of the data matrix) one at a time.

The two new algorithms mentioned here point to the fact that SVD computation is indeed an active area of research.

PROBLEMS

1. Repeat the calculation of the singular values and singular vectors of the matrix **A** given in Problem 9 of Chapter 11 by using the two-sided Jacobi algorithm.
2. Demonstrate that the sweep of orderings by rows is equivalent to the sweep of orderings by columns described in Example 1.
3. The transformation of a 2-by-2 complex matrix into a real one involves a plane rotation, followed by certain forms of premultiplication and postmultiplication.
   (a) Show that the combined effect of the plane rotation in Eq. (12.35) and the premultiplication in Eq. (12.44) is equivalent to a unitary matrix.
   (b) Show that the postmultiplying matrix on the left-hand side of Eq. (12.44) is also a unitary matrix.
4. Consider an M-by-M matrix **A** that is triangularized by the use of Householder transformations. After $M - 1$ steps, at most, the matrix **A** is triangularized as follows:

$$\mathbf{QA} = \mathbf{R}$$

   where **R** is an upper triangular matrix, and

$$\mathbf{Q} = \mathbf{Q}_{M-1}\mathbf{Q}_{M-2}\cdots\mathbf{Q}_1$$

   (a) Show that

$$\det(\mathbf{A}) = (-1)^{m-1}\det(\mathbf{R})$$

   (b) Using the fact that the Euclidean norm of a matrix is preserved under multiplication by a unitary matrix, and that each diagonal element of the triangular matrix **R** is the norm of the projection of that column on a certain subspace, show that

$$|\det(\mathbf{A})| \leq \prod_{i=1}^{M} \|\mathbf{a}_i\|$$

   where $\mathbf{a}_i$ is the ith column of matrix **A**. (This result is known as the *Hadamard theorem.*)
5. Consider the 4-by-3 data matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1.5 & 2 \\ 3 & 3 & 4 \\ 4 & 4.5 & 8 \end{bmatrix}$$

Using a sequence of Householder transformations, reduce the matrix to upper bidiagonal form.

6. Consider the data matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1.5 & 2 \\ 3 & 3 & 4 \end{bmatrix}$$

Reduce the matrix **A** to upper triangular form, using:

(a) Householder transformations.

(b) Givens rotations.

7. Use the Golub–Kahan algorithm to compute the singular-value decomposition of the bidiagonal matrix:

$$\mathbf{A} = \begin{bmatrix} 1.5 & 1.5 & 0 & 0 \\ 0 & 3 & 1.5 & 0 \\ 0 & 0 & 4.5 & 1.5 \\ 0 & 0 & 0 & 6 \end{bmatrix}$$

CHAPTER

13

Recursive Least-Squares Algorithm

In this chapter we extend the use of the method of least squares to develop a recursive algorithm for the design of adaptive transversal filters such that, given the least-squares estimate of the tap-weight vector of the filter at iteration $n - 1$, we may compute the updated estimate of this vector at iteration n upon the arrival of new data. We refer to the resulting algorithm as the *recursive least-squares (RLS) algorithm.*

The RLS algorithm may be viewed as a special case of the Kalman filter. Indeed, this special relationship between the RLS algorithm and the Kalman filter is considered later in the chapter. Our main mission in this chapter, however, is to develop the basic theory of the RLS algorithm as an important tool for linear adaptive filtering in its own right.

We begin the development of the RLS algorithm by reviewing some basic relations that pertain to the method of least squares. Then, by exploiting a relation in matrix algebra known as the *matrix inversion lemma*, we develop the RLS algorithm. An important feature of the RLS algorithm is that it utilizes information contained in the input data, extending back to the instant of time when the algorithm is initiated. The resulting rate of convergence is therefore typically an order of magnitude faster than the simple LMS algorithm. This improvement in performance, however, is achieved at the expense of a large increase in computational complexity.

13.1 SOME PRELIMINARIES

In *recursive* implementations of the method of least squares, we start the computation with *known initial conditions* and use the information contained in new data samples to *update* the old estimates. We therefore find that the length of observable data is variable. Accordingly, we express the *cost function* to be minimized as $\mathscr{E}(n)$, where n is the variable length of the observable data. Also, it is customary to introduce a *weighting factor* into the definition of the cost function $\mathscr{E}(n)$. We thus write

$$\mathscr{E}(n) = \sum_{i=1}^{n} \beta(n, i)|e(i)|^2 \tag{13.1}$$

where $e(i)$ is the difference between the *desired response* $d(i)$ and the *output* $y(i)$ produced by a transversal filter whose tap inputs (at time i) equal $u(i), u(i-1), \ldots, u(i-M+1)$, as in Fig. 13.1. That is, $e(i)$ is defined by

$$\begin{aligned} e(i) &= d(i) - y(i) \\ &= d(i) - \mathbf{w}^H(n)\mathbf{u}(i) \end{aligned} \tag{13.2}$$

where $\mathbf{u}(i)$ is the *tap-input vector at time i*, defined by

$$\mathbf{u}(i) = [u(i), u(i-1), \ldots, u(i-M+1)]^T \tag{13.3}$$

and $\mathbf{w}(n)$ is the *tap-weight vector at time n*, defined by

$$\mathbf{w}(n) = [w_0(n), w_1(n), \ldots, w_{M-1}(n)]^T \tag{13.4}$$

Note that the tap weights of the transversal filter remain *fixed* during the observation interval $1 \leq i \leq n$ for which the cost function $\mathscr{E}(n)$ is defined.

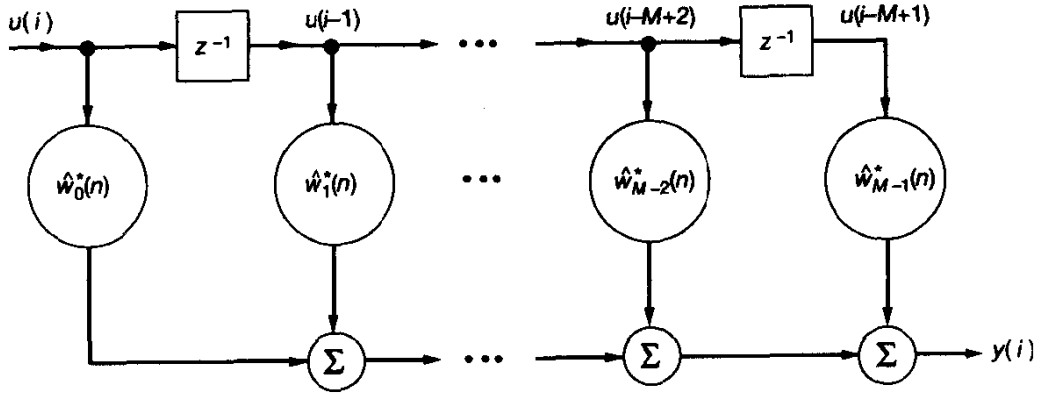


Figure 13.1 Transversal filter.

The weighting factor $\beta(n, i)$, in Eq. (13.1) has the property that

$$0 < \beta(n, i) \leq 1, \qquad i = 1, 2, \ldots, n \tag{13.5}$$

The use of the weighting factor $\beta(n, i)$, in general, is intended to ensure that data in the distant past are "forgotten" in order to afford the possibility of following the statistical variations of the observable data when the filter operates in a nonstationary environment. A special form of weighting that is commonly used is the *exponential weighting factor* or *forgetting factor* defined by

$$\beta(n, i) = \lambda^{n-i}, \qquad i = 1, 2, \ldots, n \tag{13.6}$$

where λ is a positive constant close to, but less than, 1. When λ equals 1, we have the ordinary method of least squares. The inverse of $1 - \lambda$ is, roughly speaking, a measure of the *memory* of the algorithm. The special case $\lambda = 1$ corresponds to *infinite memory*. Thus, in the *method of exponentially weighted least squares*, we minimize the cost function

$$\mathscr{E}(n) = \sum_{i=1}^{n} \lambda^{n-i} |e(i)|^2 \tag{13.7}$$

The optimum value of the tap-weight vector, $\hat{\mathbf{w}}(n)$, for which the cost function $\mathscr{E}(n)$ of Eq. (13.7) attains its minimum value is defined by the *normal equations* written in matrix form:

$$\boldsymbol{\Phi}(n)\hat{\mathbf{w}}(n) = \mathbf{z}(n) \tag{13.8}$$

The M-by-M correlation matrix $\boldsymbol{\Phi}(n)$ is now defined by

$$\boldsymbol{\Phi}(n) = \sum_{i=1}^{n} \lambda^{n-i} \mathbf{u}(i)\mathbf{u}^H(i) \tag{13.9}$$

The M-by-1 cross-correlation vector $\mathbf{z}(n)$ between the tap inputs of the transversal filter and the desired response is correspondingly defined by

$$\mathbf{z}(n) = \sum_{i=1}^{n} \lambda^{n-i} \mathbf{u}(i)d^*(i) \tag{13.10}$$

where the asterisk denotes complex conjugation.

The correlation matrix $\boldsymbol{\Phi}(n)$ of Eq. (13.9) differs from the time-averaged version of Eq. (11.45) in two respects.

1. The matrix product $\mathbf{u}(i)\mathbf{u}^H(i)$ inside the summation on the right-hand side of Eq. (11.45) is weighted by the exponential factor λ^{n-i}, which arises naturally from the adoption of Eq. (13.7) as the cost function.
2. The use of *prewindowing* is assumed, according to which the input data prior to time $i = 1$ are equal to zero, hence the use of $i = 1$ as the lower limit of the summation.

Similar remarks apply to the cross-correlation vector $\mathbf{z}(n)$ compared to its time-averaged counterpart of Chapter 11.

Isolating the term corresponding to $i = n$ from the rest of the summation on the right-hand side of Eq. (13.9), we may write

$$\boldsymbol{\Phi}(n) = \lambda\left[\sum_{i=1}^{n-1} \lambda^{n-1-i}\,\mathbf{u}(i)\mathbf{u}^H(i)\right] + \mathbf{u}(n)\mathbf{u}^H(n) \tag{13.11}$$

However, by definition, the expression inside the square brackets on the right-hand side of Eq. (13.11) equals the correlation matrix $\boldsymbol{\Phi}(n-1)$. Hence, we have the following recursion for updating the value of the correlation matrix of the tap inputs:

$$\boldsymbol{\Phi}(n) = \lambda\boldsymbol{\Phi}(n-1) + \mathbf{u}(n)\mathbf{u}^H(n) \tag{13.12}$$

where $\boldsymbol{\Phi}(n-1)$ is the "old" value of the correlation matrix, and the matrix product $\mathbf{u}(n)\mathbf{u}^H(n)$ plays the role of a "correction" term in the updating operation.

Similarly, we may use Eq. (13.10) to derive the following recursion for updating the cross-correlation vector between the tap inputs and the desired response:

$$\mathbf{z}(n) = \lambda\mathbf{z}(n-1) + \mathbf{u}(n)d^*(n) \tag{13.13}$$

To compute the least-square estimate $\hat{\mathbf{w}}(n)$ for the tap-weight vector in accordance with Eq. (13.8), we have to determine the inverse of the correlation matrix $\boldsymbol{\Phi}(n)$. In practice, however, we usually try to avoid performing such an operation as it can be very time consuming, particularly if the number of tap weights, M, is high. Also, we would like to be able to compute the least-squares estimate $\hat{\mathbf{w}}(n)$ for the tap-weight vector recursively for $n = 1, 2, \ldots, \infty$. We can realize both of these objectives by using a basic result in matrix algebra known as the *matrix inversion lemma*. We assume that the initial conditions have been chosen to ensure the nonsingularity of the correlation matrix $\boldsymbol{\Phi}(n)$; this issue is discussed later in Section 13.3.

13.2 THE MATRIX INVERSION LEMMA

Let $\mathbf{A}$ and $\mathbf{B}$ be two positive-definite M-by-M matrices related by

$$\mathbf{A} = \mathbf{B}^{-1} + \mathbf{C}\mathbf{D}^{-1}\mathbf{C}^H \tag{13.14}$$

where $\mathbf{D}$ is another positive-definite N-by-M matrix, and $\mathbf{C}$ is an M-by-N matrix. According to the *matrix inversion lemma*, we may express the inverse of the matrix $\mathbf{A}$ as follows:

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{B}\mathbf{C}(\mathbf{D} + \mathbf{C}^H\mathbf{B}\mathbf{C})^{-1}\mathbf{C}^H\mathbf{B} \tag{13.15}$$

The proof of this lemma is established by multiplying Eq. (13.14) by (13.15) and recognizing that the product of a square matrix and its inverse is equal to the identity matrix (see Problem 2). The matrix inversion lemma states that if we are given a matrix $\mathbf{A}$ as defined in Eq. (13.14), we can determine its inverse $\mathbf{A}^{-1}$ by using the relation of Eq. (13.15). In

effect, the lemma is described by this pair of equations. The matrix inversion lemma is also referred to in the literature as *Woodbury's identity*.[1]

In the next section we show how the matrix inversion lemma can be applied to obtain a recursive equation for computing the least-squares solution $\hat{\mathbf{w}}(n)$ for the tap-weight vector.

13.3 THE EXPONENTIALLY WEIGHTED RECURSIVE LEAST-SQUARES ALGORITHM

With the correlation matrix $\mathbf{\Phi}(n)$ assumed to be positive definite and therefore nonsingular, we may apply the matrix inversion lemma to the recursive equation (13.12). We first make the following identifications:

$$\mathbf{A} = \mathbf{\Phi}(n)$$

$$\mathbf{B}^{-1} = \lambda\mathbf{\Phi}(n-1)$$

$$\mathbf{C} = \mathbf{u}(n)$$

$$\mathbf{D} = 1$$

Then, substituting these definitions in the matrix inversion lemma of Eq. (13.15), we obtain the following recursive equation for the inverse of the correlation matrix:

$$\mathbf{\Phi}^{-1}(n) = \lambda^{-1}\mathbf{\Phi}^{-1}(n-1) - \frac{\lambda^{-2}\mathbf{\Phi}^{-1}(n-1)\mathbf{u}(n)\mathbf{u}^H(n)\mathbf{\Phi}^{-1}(n-1)}{1 + \lambda^{-1}\mathbf{u}^H(n)\mathbf{\Phi}^{-1}(n-1)\mathbf{u}(n)} \tag{13.16}$$

For convenience of computation, let

$$\mathbf{P}(n) = \mathbf{\Phi}^{-1}(n) \tag{13.17}$$

and

$$\mathbf{k}(n) = \frac{\lambda^{-1}\mathbf{P}(n-1)\mathbf{u}(n)}{1 + \lambda^{-1}\mathbf{u}^H(n)\mathbf{P}(n-1)\mathbf{u}(n)} \tag{13.18}$$

Using these definitions, we may rewrite Eq. (13.16) as follows:

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{u}^H(n)\mathbf{P}(n-1) \tag{13.19}$$

[1]The exact origin of the matrix inversion lemma is not known. Householder (1964) attributes it to Woodbury (1950). Nevertheless, application of the matrix inversion lemma in the filtering literature was first made by Kailath, who used a form of this lemma to prove the equivalence of the Wiener filter and the maximum-likelihood procedure for estimating the output of a random linear time-invariant channel that is corrupted by additive white Gaussian noise (Kailath, 1960). Early use of the matrix inversion lemma was also made by Ho (1963). Another interesting application of the matrix inversion lemma was made by Brooks and Reed, who used it to prove the equivalence of the Wiener filter, the maximum signal-to-noise ratio filter, and the likelihood ratio processor for detecting a signal in additive white Gaussian noise (Brooks and Reed, 1972).

The M-by-M matrix $\mathbf{P}(n)$ is referred to as the *inverse correlation matrix*. The M-by-1 vector $\mathbf{k}(n)$ is referred to as the *gain vector* for reasons that will become apparent later in the section. Equation (13.19) is the *Riccati equation* for the RLS algorithm.

By rearranging Eq. (13.18), we have

$$\begin{aligned}\mathbf{k}(n) &= \lambda^{-1}\mathbf{P}(n-1)\mathbf{u}(n) - \lambda^{-1}\mathbf{k}(n)\mathbf{u}^H(n)\mathbf{P}(n-1)\mathbf{u}(n) \\ &= [\lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{u}^H(n)\mathbf{P}(n-1)]\mathbf{u}(n)\end{aligned} \tag{13.20}$$

We see from Eq. (13.19) that the expression inside the brackets on the right-hand side of Eq. (13.20) equals $\mathbf{P}(n)$. Hence. we may simplify Eq. (13.20) to

$$\mathbf{k}(n) = \mathbf{P}(n)\mathbf{u}(n) \tag{13.21}$$

This result, together with $\mathbf{P}(n) = \boldsymbol{\Phi}^{-1}(n)$, may be used as the definition for the gain vector:

$$\mathbf{k}(n) = \boldsymbol{\Phi}^{-1}(n)\mathbf{u}(n) \tag{13.22}$$

In other words, the gain vector $\mathbf{k}(n)$ is defined as the tap-input vector $\mathbf{u}(n)$ transformed by the inverse of the correlation matrix $\boldsymbol{\Phi}(n)$.

Time Update for the Tap-Weight Vector

Next, we wish to develop a recursive equation for updating the least-squares estimate $\hat{\mathbf{w}}(n)$ for the tap-weight vector. To do this, we use Eqs. (13.8), (13.13), and (13.17) to express the least-squares estimate $\hat{\mathbf{w}}(n)$ for the tap-weight vector at iteration n as follows:

$$\begin{aligned}\hat{\mathbf{w}}(n) &= \boldsymbol{\Phi}^{-1}(n)\mathbf{z}(n) \\ &= \mathbf{P}(n)\mathbf{z}(n) \\ &= \lambda\mathbf{P}(n)\mathbf{z}(n-1) + \mathbf{P}(n)\mathbf{u}(n)d^*(n)\end{aligned} \tag{13.23}$$

Substituting Eq. (13.19) for $\mathbf{P}(n)$ in the first term only in the right-hand side of Eq. (13.23), we get

$$\begin{aligned}\hat{\mathbf{w}}(n) &= \mathbf{P}(n-1)\mathbf{z}(n-1) - \mathbf{k}(n)\mathbf{u}^H(n)\mathbf{P}(n-1)\mathbf{z}(n-1) \\ &\quad + \mathbf{P}(n)\mathbf{u}(n)d^*(n) \\ &= \boldsymbol{\Phi}^{-1}(n-1)\mathbf{z}(n-1) - \mathbf{k}(n)\mathbf{u}^H(n)\boldsymbol{\Phi}^{-1}(n-1)\mathbf{z}(n-1) \\ &\quad + \mathbf{P}(n)\mathbf{u}(n)d^*(n) \\ &= \hat{\mathbf{w}}(n-1) - \mathbf{k}(n)\mathbf{u}^H(n)\hat{\mathbf{w}}(n-1) + \mathbf{P}(n)\mathbf{u}(n)d^*(n)\end{aligned} \tag{13.24}$$

Finally, using the fact that $\mathbf{P}(n)\mathbf{u}(n)$ equals the gain vector $\mathbf{k}(n)$, as in Eq. (13.21), we get the desired recursive equation for updating the tap-weight vector:

$$\begin{aligned}\hat{\mathbf{w}}(n) &= \hat{\mathbf{w}}(n-1) + \mathbf{k}(n)[d^*(n) - \mathbf{u}^H(n)\hat{\mathbf{w}}(n-1)] \\ &= \hat{\mathbf{w}}(n-1) + \mathbf{k}(n)\xi^*(n)\end{aligned} \tag{13.25}$$

where $\xi(n)$ is the *a priori estimation error* defined by

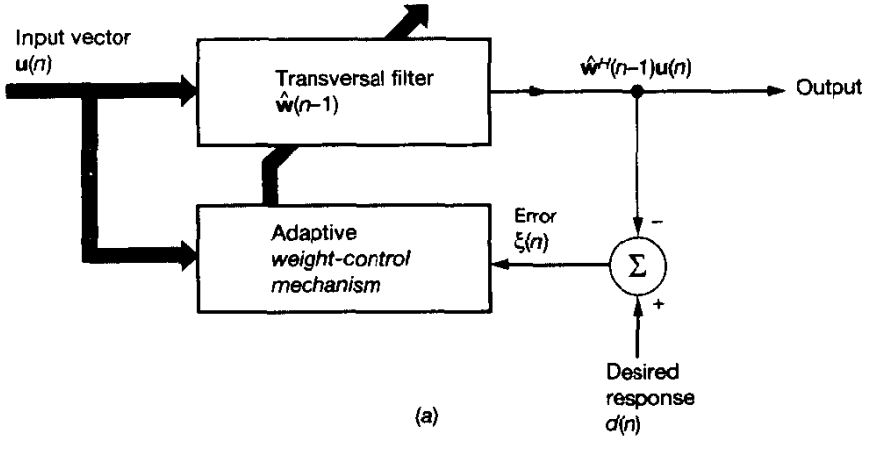
$$\begin{aligned}\xi(n) &= d(n) - \mathbf{u}^T(n)\hat{\mathbf{w}}^*(n-1) \\ &= d(n) - \hat{\mathbf{w}}^H(n-1)\mathbf{u}(n)\end{aligned} \tag{13.26}$$

The inner product $\hat{\mathbf{w}}^H(n-1)\mathbf{u}(n)$ represents an estimate of the desired response $d(n)$, based on the *old* least-squares estimate of the tap-weight vector that was made at time $n-1$.

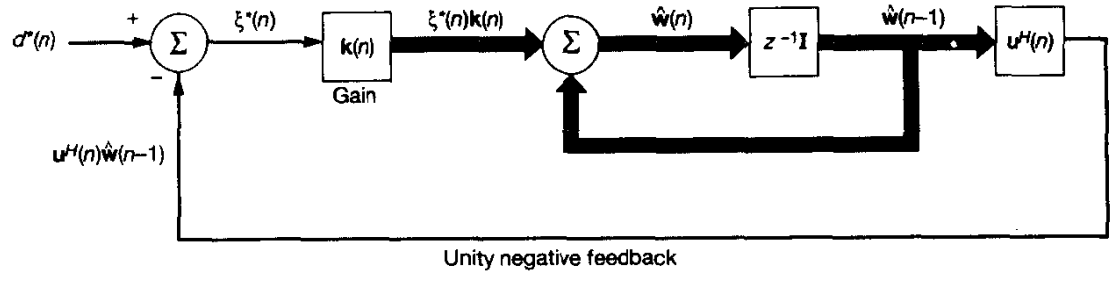
Equation (13.25) for the adjustment of the tap-weight vector and Eq. (13.26) for the *a priori* estimation error suggest the block-diagram representation depicted in Fig. 13.2(a) for the *recursive least-squares RLS algorithm.*

The *a priori* estimation error $\xi(n)$ is, in general, different from the *a posteriori estimation error*

$$e(n) = d(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n) \tag{13.27}$$



(a)



(b)

Figure 13.2 Representations of the RLS algorithm: (1) block diagram; (b) signal-flow graph.

TABLE 13.1 SUMMARY OF THE RLS ALGORITHM

Initialize the algorithm by setting

$\mathbf{P}(0) = \delta^{-1}\mathbf{I}$, δ = small positive constant

$\hat{\mathbf{w}}(0) = \mathbf{0}$

For each instant of time, $n = 1, 2, \ldots$, compute

$$\mathbf{k}(n) = \frac{\lambda^{-1}\mathbf{P}(n-1)\mathbf{u}(n)}{1 + \lambda^{-1}\mathbf{u}^H(n)\mathbf{P}(n-1)\mathbf{u}(n)}$$

$$\xi(n) = d(n) - \hat{\mathbf{w}}^H(n-1)\mathbf{u}(n)$$

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{k}(n)\xi^*(n)$$

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{u}^H(n)\mathbf{P}(n-1)$$

the computation of which involves the *current* least-squares estimate of the tap-weight vector available at time n. Indeed, we may view $\xi(n)$ as a "tentative" value of $e(n)$ before updating the tap-weight vector. Note, however, in the least-squares optimization that led to the recursive algorithm of Eq. (13.25) for the tap-weight vector, we actually minimized a cost function based on $e(n)$ and *not* $\xi(n)$.

Summary of the RLS Algorithm

Equations (13.18), (13.26), (13.25), and (13.19), collectively and in that order, constitute the *RLS algorithm*, as summarized in Table 13.1. We note that, in particular, Eq. (13.26) describes the filtering operation of the algorithm, whereby the transversal filter is excited to compute the *a priori* estimation error $\xi(n)$. Equation (13.25) describes the adaptive operation of the algorithm, whereby the tap-weight vector is updated by incrementing its old value by an amount equal to the complex conjugate of the *a priori* estimation error $\xi(n)$ times the time-varying gain vector $\mathbf{k}(n)$, hence the name "gain vector." Equations (13.18) and (13.19) enable us to update the value of the gain vector itself. An important feature of the RLS algorithm described by these equations is that the inversion of the correlation matrix $\mathbf{\Phi}(n)$ is replaced at each step by a simple scalar division. Figure 13.2(b) depicts a signal-flow-graph representation of the RLS algorithm that complements the block diagram of Fig. 13.2(a).

Initialization of the RLS Algorithm

The applicability of the RLS algorithm requires that we initialize the recursion of Eq. (13.19) by choosing a starting value $\mathbf{P}(0)$ that assures the nonsingularity of the correlation matrix $\mathbf{\Phi}(n)$. We may do this by evaluating the inverse

$$\left[\sum_{i=-n_0}^{0} \lambda^{-i}\mathbf{u}(i)\mathbf{u}^H(i)\right]^{-1}$$

where the tap-weight vector $\mathbf{u}(i)$ is obtained from an initial block of data for $-n_0 \leq i \leq 0$.

A simpler procedure, however, is to modify the expression slightly for the correlation matrix $\mathbf{\Phi}(n)$ by writing

$$\mathbf{\Phi}(n) = \sum_{i=1}^{n} \lambda^{n-i}\mathbf{u}(i)\mathbf{u}^H(i) + \delta\lambda^n\mathbf{I} \tag{13.28}$$

where $\mathbf{I}$ is the M-by-M identity matrix, and δ is a small positive constant. Thus putting $n = 0$ in Eq. (13.28), we have

$$\mathbf{\Phi}(0) = \delta\mathbf{I}$$

Correspondingly, for the initial value of $\mathbf{P}(n)$ equal to the inverse of the correlation matrix $\mathbf{\Phi}(n)$, we set

$$\mathbf{P}(0) = \delta^{-1}\mathbf{I} \tag{13.29}$$

The initialization described in Eq. (13.29) is equivalent to forcing the unknown data sample $u(-M + 1)$ equal to the value $\lambda^{(-M+1)/2}\delta^{1/2}$ instead of zero. In other words, during the initialization period we modify the prewindowing method by writing

$$u(n) = \begin{cases} \lambda^{(-M+1)/2}\delta^{1/2}, & n = -M + 1 \\ 0, & n < 0, n \neq -M + 1 \end{cases} \tag{13.30}$$

Note that for a transversal filter with M taps, the index $n = M + 1$ refers to the *last* tap in the filter. When the first nonzero data sample $u(i)$ enters the filter, the initializing tap input $u(-M + 1)$ leaves the filter and from then on the RLS algorithm takes over.

It only remains for us to choose an initial value for the tap-weight vector. It is customary to set

$$\hat{\mathbf{w}}(0) = \mathbf{0} \tag{13.31}$$

where $\mathbf{0}$ is the M-by-1 null vector.

The initialization procedure incorporating Eqs. (13.29) and (13.31) is referred to as a *soft-constrained initialization*. The positive constant δ is the only parameter required for this initialization. The recommended choice of δ is that it should be small compared to $0.01\sigma_u^2$, where σ_u^2 is the variance of a data sample $u(n)$. Such a choice is based on practical experience with the RLS algorithm, supported by a statistical analysis of the soft-constrained initialization of the algorithm (Hubing and Alexander, 1990). For large data lengths, the exact value of the initializing constant δ has an insignificant effect.

It is important to note that by using the initialization procedure defined by Eqs. (13.29) and (13.31), we are no longer computing a solution that minimizes the cost function $\xi(n)$ of Eq. (13.7) exactly. Instead, we are computing the solution that minimizes the modified cost function:

$$\mathscr{E}(n) = \delta\lambda^n\|\mathbf{w}(n)\|^2 + \sum_{i=1}^{n} \lambda^{n-i}\,|e(i)|^2$$

In other words, the RLS algorithm summarized in Table 13.1 yields the exact recursive solution to the following optimization problem (Sayed and Kailath, 1994):

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

where $e(i)$ is defined by Eq. (13.2).

13.4 UPDATE RECURSION FOR THE SUM OF WEIGHTED ERROR SQUARES

The minimum value of the sum of weighted error squares, $\mathscr{E}_{\min}(n)$, results when the tap-weight vector is set equal to the least-squares estimate $\hat{\mathbf{w}}(n)$. To compute $\mathscr{E}_{\min}(n)$, we may therefore use the relation [see first line of Eq. (10.40)]:

$$\mathscr{E}_{\min}(n) = \mathscr{E}_d(n) - \mathbf{z}^H(n)\hat{\mathbf{w}}(n) \tag{13.32}$$

where $\mathscr{E}_d(n)$ is defined by (using the notation of this chapter)

$$\begin{aligned}\mathscr{E}_d(n) &= \sum_{i=1}^{n} \lambda^{n-i}|d(i)|^2 \\ &= \lambda\,\mathscr{E}_d(n-1) + |d(n)|^2\end{aligned} \tag{13.33}$$

Therefore, substituting Eqs. (13.13), (13.25), and (13.33) in (13.32), we get

$$\begin{aligned}\mathscr{E}_{\min}(n) = \lambda[&\mathscr{E}_d(n-1) - \mathbf{z}^H(n-1)\hat{\mathbf{w}}(n-1)] \\ &+ d(n)[d^*(n) - \mathbf{u}^H(n)\hat{\mathbf{w}}(n-1)] \\ &- \mathbf{z}^H(n)\mathbf{k}(n)\xi^*(n)\end{aligned} \tag{13.34}$$

where, in the last term, we have restored $\mathbf{z}(n)$ to its original form. By definition, the expression inside the first set of brackets on the right-hand side of Eq. (13.34) equals $\mathscr{E}_{\min}(n-1)$. Also, by definition, the expression inside the second set of brackets equals the complex conjugate of the *a priori* estimation error $\xi(n)$. For the last term, we use the definition of the gain vector $\mathbf{k}(n)$ to express the inner product $\mathbf{z}^H(n)\mathbf{k}(n)$ as

$$\begin{aligned}\mathbf{z}^H(n)\mathbf{k}(n) &= \mathbf{z}^H(n)\boldsymbol{\Phi}^{-1}(n)\mathbf{u}(n) \\ &= [\boldsymbol{\Phi}^{-1}(n)\mathbf{z}(n)]^H\mathbf{u}(n) \\ &= \hat{\mathbf{w}}^H(n)\mathbf{u}(n)\end{aligned}$$

where (in the second line) we have used the Hermitian property of the correlation matrix $\boldsymbol{\Phi}(n)$, and (in the third line) we have used the fact that $\boldsymbol{\Phi}^{-1}(n)\mathbf{z}(n)$ equals the least-squares estimate $\hat{\mathbf{w}}(n)$. Accordingly, we may simplify Eq. (13.34) to

$$\begin{aligned}\mathscr{E}_{\min}(n) &= \lambda\mathscr{E}_{\min}(n-1) + d(n)\xi^*(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n)\xi^*(n) \\ &= \lambda\mathscr{E}_{\min}(n-1) + \xi^*(n)[d(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n)] \\ &= \lambda\mathscr{E}_{\min}(n-1) + \xi^*(n)e(n)\end{aligned} \tag{13.35}$$

where $e(n)$ is the *a posteriori* estimation error. Equation (13.35) is the recursion for updating the sum of weighted error squares. We thus see that the product of the complex conjugate of $\xi(n)$ and $e(n)$ represents the correction term in this update. Note that this product is real valued, which implies that we always have

$$\xi(n)e^*(n) = \xi^*(n)e(n) \tag{13.36}$$

Conversion Factor

The formula of Eq. (13.35) involves two different estimation errors: the *a priori* estimation error $\xi(n)$ and the *a posteriori* estimation error $e(n)$, which are naturally related. To establish the relationship between these two estimation errors, we start with the defining equation (13.27) and substitute the update equation (13.25), obtaining

$$\begin{aligned} e(n) &= d(n) - [\hat{\mathbf{w}}(n-1) + \mathbf{k}(n)\xi^*(n)]^H\mathbf{u}(n) \\ &= d(n) - \hat{\mathbf{w}}^H(n-1)\mathbf{u}(n) - \mathbf{k}^H(n)\mathbf{u}(n)\xi(n) \\ &= (1 - \mathbf{k}^H(n)\mathbf{u}(n))\xi(n) \end{aligned} \tag{13.37}$$

where, in the last line, we have made use of the definition given in Eq. (13.26). The ratio of the *a posteriori* estimation $e(n)$ to the *a priori* estimation $\xi(n)$ is called the *conversion factor*, denoted by $\gamma(n)$. We may thus write

$$\begin{aligned} \gamma(n) &= \frac{e(n)}{\xi(n)} \\ &= 1 - \mathbf{k}^H(n)\mathbf{u}(n) \end{aligned} \tag{13.38}$$

the value of which is uniquely determined by the gain vector $\mathbf{k}(n)$ and the tap-input vector $\mathbf{u}(n)$.

13.5 EXAMPLE: SINGLE-WEIGHT ADAPTIVE NOISE CANCELER

Consider the *single-weight, dual-input adaptive noise canceler* depicted in Fig. 13.3. The two inputs are represented by the *primary signal* $d(n)$ and the *reference signal* $u(n)$ that are characterized as follows. First, the primary signal consists of an *information-bearing signal* component and an additive *interference*. Second, the reference signal $u(n)$ is correlated with the interference and has no detectable contribution to the information-bearing signal. The requirement is to exploit the properties of the reference signal in relation to the primary signal to suppress the interference at the adaptive noise canceler output.

Application of the RLS algorithm yields the following set of equations for this canceler (after reorganization of terms):

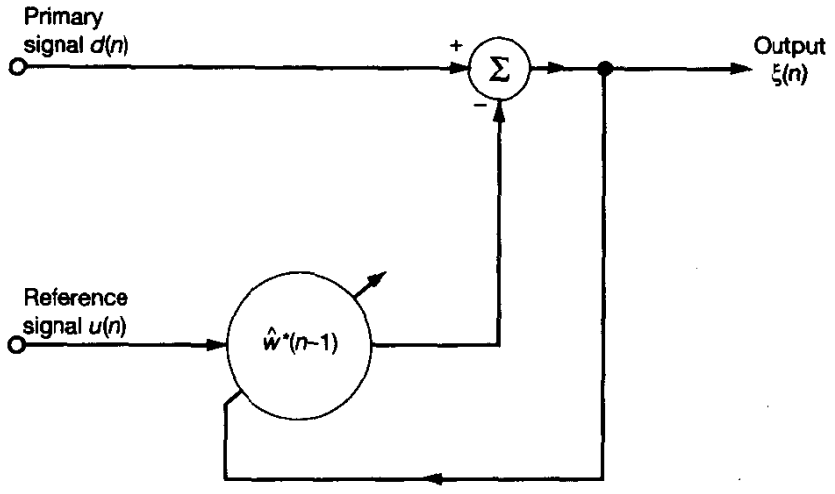


Figure 13.3 Single-weight adaptive noise canceler.

$$k(n) = \left[\frac{1}{\lambda\hat{\sigma}^2(n-1) + |u(n)|^2}\right]u(n) \tag{13.39}$$

$$\xi(n) = d(n) - \hat{w}^*(n-1)u(n) \tag{13.40}$$

$$\hat{w}(n) = \hat{w}(n-1) + k(n)\xi^*(n) \tag{13.41}$$

$$\hat{\sigma}^2(n) = \lambda\hat{\sigma}^2(n-1) + |u(n)|^2 \tag{13.42}$$

where $\hat{\sigma}^2(n)$ is an estimate of the error variance. It is the inverse of $P(n)$, the scalar version of the matrix $\mathbf{P}(n)$ in the RLS algorithm, as shown by

$$\hat{\sigma}^2(n) = P^{-1}(n) \tag{13.43}$$

It is informative to compare the algorithm described in Eqs. (13.39) to (13.42) with its counterpart obtained using the normalized LMS algorithm; the version of the normalized LMS algorithm of particular interest in the context of our present situation is that given in Eq. (9.144). The *major difference* between these two algorithms is that the constant a in the normalized LMS algorithm is replaced by the time-varying term $\lambda\hat{\sigma}^2(n-1)$ in the denominator of the gain factor $k(n)$ that controls the correction applied to the tap weight in Eq. (13.41).

13.6 CONVERGENCE ANALYSIS OF THE RLS ALGORITHM

In this section we demonstrate the convergence of the RLS algorithm operating in a stationary environment. The treatment presented here is rigorous, within the confines of the independence theory, the elements of which were described in Section 9.4. We say "rigor-

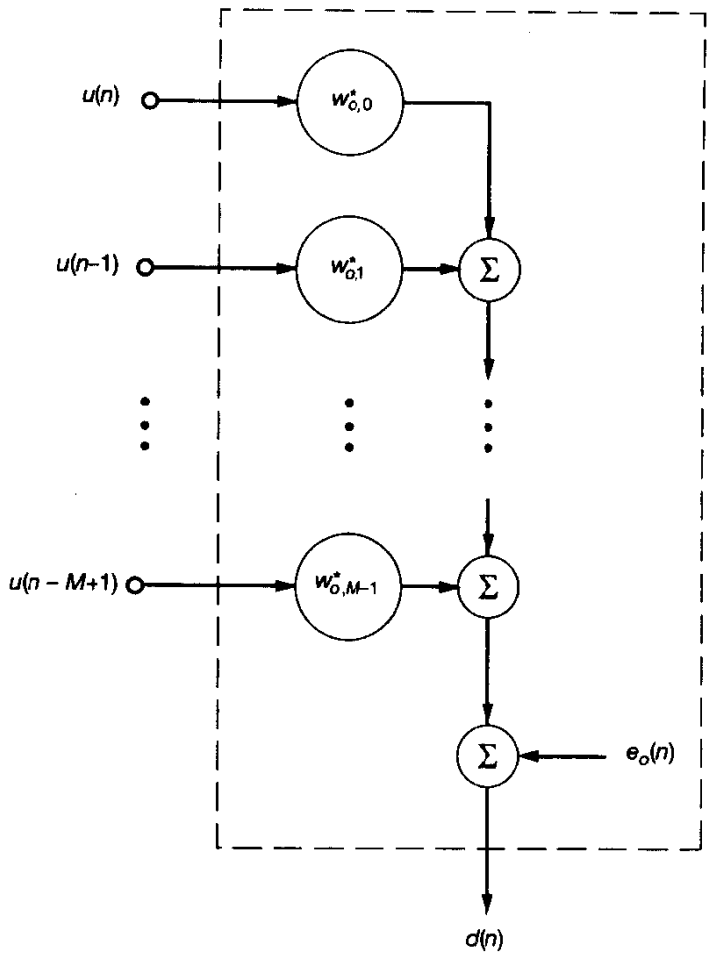


Figure 13.4 Multiple linear regression model.

ous" in the sense that we will *not* invoke the direct-averaging method, as was the case with the LMS algorithm.

To proceed with the analysis, we assume that the desired response $d(n)$ and the tap-input vector $\mathbf{u}(n)$ are related by the *multiple linear regression model* of Fig. 13.4. In particular, we may write

$$d(n) = e_o(n) + \mathbf{w}_o^H \mathbf{u}(n) \tag{13.44}$$

where the M-by-1 vector $\mathbf{w}_o$ denotes the *regression parameter vector* of the model, and $e_o(n)$ is the *measurement error*. The measurement error process $e_o(n)$ is white with zero mean and variance σ^2. The parameter vector $\mathbf{w}_o$ is constant. The latter assumption is equivalent to saying that the adaptive transversal filter operates in a stationary environment, with $\lambda = 1$.

Convergence of the RLS Algorithm in the Mean Value

Starting with $\mathbf{\Phi}(0) = \mathbf{0}$ that corresponds to $\mathbf{u}(0) = \mathbf{0}$ (i.e., prewindowing of the input data), we find that with the soft-constrained initialization procedure described in Section 13.3, the weight vector $\hat{\mathbf{w}}(n)$ computed by the RLS algorithm is almost exactly the same as that computed by the method of least-squares for $n \geq M$, where M is the number of taps in the adaptive transversal filter. Accordingly, we may use the normal equations to express $\hat{\mathbf{w}}(n)$ by the formula

$$\hat{\mathbf{w}}(n) = \mathbf{\Phi}^{-1}(n)\mathbf{z}(n), \qquad n \geq M \tag{13.45}$$

where, for $\lambda = 1$,

$$\mathbf{\Phi}(n) = \sum_{i=1}^{n} \mathbf{u}(i)\mathbf{u}^H(i) \tag{13.46}$$

and

$$\mathbf{z}(n) = \sum_{i=1}^{n} \mathbf{u}(i)d^*(i) \tag{13.47}$$

Substituting Eq. (13.44) in (13.47) yields

$$\begin{aligned} \mathbf{z}(n) &= \sum_{i=1}^{n} \mathbf{u}(i)\mathbf{u}^H(i)\mathbf{w}_o + \sum_{i=1}^{n} \mathbf{u}(i)e_o^*(i) \\ &= \mathbf{\Phi}(n)\mathbf{w}_0 + \sum_{i=1}^{n} \mathbf{u}(i)e_o^*(i) \end{aligned} \tag{13.48}$$

This, in turn, means that we may rewrite Eq. (13.45) as

$$\begin{aligned} \hat{\mathbf{w}}(n) &= \mathbf{\Phi}^{-1}(n)\mathbf{\Phi}(n)\mathbf{w}_o + \mathbf{\Phi}^{-1}(n) \sum_{i=1}^{n} \mathbf{u}(i)e_o^*(i) \\ &= \mathbf{w}_o + \mathbf{\Phi}^{-1}(n) \sum_{i=1}^{n} \mathbf{u}(i)e_o^*(i) \end{aligned} \tag{13.49}$$

Next, we invoke the expectation property of a random variable:

$$E[x] = E[E[x|y]] \tag{13.50}$$

where $E[x|y]$ is the conditional expectation of a random variable x, given another random variable y; the remaining expectation on the right-hand side of Eq. (13.50) is with respect to y. Hence, in light of this property, we may use Eq. (13.49) to express the expected value of $\hat{\mathbf{w}}(n)$ as follows:

$$\begin{aligned} E[\hat{\mathbf{w}}(n)] &= \mathbf{w}_o + E[\mathbf{\Phi}^{-1}(n) \sum_{i=1}^{n} \mathbf{u}(i)e_o^*(i)] \\ &= \mathbf{w}_o + E\left[E[\mathbf{\Phi}^{-1}(n) \sum_{i=1}^{n} \mathbf{u}(i)e_o^*(i) \,|\mathbf{u}(i), i = 1, 2, \ldots, n]\right] \end{aligned} \tag{13.51}$$

Recognizing that

- The time-averaged correlation matrix $\boldsymbol{\Phi}(n)$ is uniquely defined by the sequence of input vectors $\mathbf{u}(1),\mathbf{u}(2), \ldots ,\mathbf{u}(n)$
- The measurement error $e_o(i)$ is independent of the input vector $\mathbf{u}(i)$ for all i
- The measurement error $e(i)$ has zero mean

we may reduce Eq. (13.51) to

$$E[\hat{\mathbf{w}}(n)] = \mathbf{w}_o, \qquad n \geq M \tag{13.52}$$

Equation (13.52) states that the *RLS algorithm is convergent in the mean value* for $n \geq M$, where M is the number of taps in the adaptive transversal filter. Note that, unlike the LMS algorithm, the RLS algorithm does not have to wait for $n \to \infty$ for convergence in the mean value to be attained.

The Mean-squared Error in the Weight Vector $\hat{\mathbf{w}}(n)$

Consider next the convergence of the mean-squared error in the weight vector $\hat{\mathbf{w}}(n)$. To demonstrate this convergence, we first use Eq. (13.49) to express the *weight-error vector* as

$$\begin{aligned} \boldsymbol{\epsilon}(n) &= \hat{\mathbf{w}}(n) - \mathbf{w}_o \\ &= \boldsymbol{\Phi}^{-1}(n) \sum_{i=1}^{n} \mathbf{u}(i) e_o^*(i) \end{aligned} \tag{13.53}$$

Next, using the definition of the *weight-error correlation matrix*

$$\mathbf{K}(n) = E[\boldsymbol{\epsilon}(n)\boldsymbol{\epsilon}^H(n)] \tag{13.54}$$

we have

$$\mathbf{K}(n) = E\left[\boldsymbol{\Phi}^{-1}(n) \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbf{u}(i) e_o^*(i) e_o(j) \mathbf{u}^H(j) \boldsymbol{\Phi}^{-1}(n)\right] \tag{13.55}$$

Again, invoking the expectation property described in Eq. (13.50), we may rewrite Eq. (13.55) as

$$\mathbf{K}(n) = E\left[\boldsymbol{\Phi}^{-1}(n) \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbf{u}(i) E[e_o^*(i) e_0(j)] \mathbf{u}^H(n) \boldsymbol{\Phi}^{-1}(n)]\right] \tag{13.56}$$

Since the measurement error $e_o(i)$ is assumed to be drawn from a white-noise process of variance σ^2, we have

$$\cdot\ E[e_o(i) e_o^*(j)] = \begin{cases} \sigma^2, & j = i \\ 0, & j \neq i \end{cases} \tag{13.57}$$

It follows therefore that we may simplify Eq. (13.56) to

$$\begin{aligned}\mathbf{K}(n) &= \sigma^2 E\left[\boldsymbol{\Phi}^{-1}(n)\sum_{i=1}^{n}\mathbf{u}(i)\mathbf{u}^H(i)\boldsymbol{\Phi}^{-1}(n)\right] \\ &= \sigma^2 E[\boldsymbol{\Phi}^{-1}(n)\boldsymbol{\Phi}(n)\boldsymbol{\Phi}^{-1}(n)] \\ &= \sigma^2 E[\boldsymbol{\Phi}^{-1}(n)]\end{aligned} \tag{13.58}$$

Next, we invoke two elements of the *independence assumption* described in Section 9.4:

1. The input vectors $\mathbf{u}(1), \mathbf{u}(2), \ldots, \mathbf{u}(n)$ are *independently and identically distributed* (iid).
2. The input vectors $\mathbf{u}(1), \mathbf{u}(2), \ldots, \mathbf{u}(n)$ are drawn from a stochastic process with a *multivariate Gaussian distribution* of zero mean and ensemble-averaged correlation matrix $\mathbf{R}$.

Then, in light of the material presented in Appendix J, the correlation matrix $\boldsymbol{\Phi}(n)$ is described by a *complex Wishart distribution*, which is so named in honor of Wishart (1982). In particular, in that appendix it is shown that the expectation of the inverse correlation matrix $\boldsymbol{\Phi}^{-1}(n)$ is exactly

$$E[\boldsymbol{\Phi}^{-1}(n)] = \frac{1}{n-M-1}\mathbf{R}^{-1}, \qquad n > M+1 \tag{13.59}$$

Substituting Eq. (13.59) in (13.58), we may therefore express the weight-error correlation matrix $\mathbf{K}(n)$ as

$$\mathbf{K}(n) = \frac{\sigma^2}{n-M-1}\mathbf{R}^{-1}, \qquad n > M+1 \tag{13.60}$$

from which we readily deduce that

$$\begin{aligned}E[\boldsymbol{\epsilon}^H(n)\boldsymbol{\epsilon}(n)] &= \mathrm{tr}[\mathbf{K}(n)] \\ &= \frac{\sigma^2}{n-M-1}\mathrm{tr}[\mathbf{R}^{-1}] \\ &= \frac{\sigma^2}{n-M-1}\sum_{i=1}^{M}\frac{1}{\lambda_i}, \qquad n > M+1\end{aligned} \tag{13.61}$$

where the λ_i are the eigenvalues of the ensemble-averaged correlation matrix $\mathbf{R}$.

On the basis of Eq. (13.61), we may now make the following two important observations for $n > M + 1$:

1. The mean-squared error in the weight vector $\hat{\mathbf{w}}(n)$ is *magnified by the inverse of the smallest eigenvalue* $\lambda_{\min}$. Hence, to a first order of approximation, the sensi-

tivity of the RLS algorithm to eigenvalue spread is determined initially in proportion to the inverse of the smallest eigenvalue. Therefore, ill-conditioned least-squares problems may lead to *bad* convergence properties.

2. The mean-squared error in the weight-vector $\hat{\mathbf{w}}(n)$ decays almost linearly with the number of iterations, n. Hence, the estimate $\hat{\mathbf{w}}(n)$ produced by the RLS algorithm for the tap-weight vector converges in the norm (i.e., mean square) to the parameter vector $\mathbf{w}_o$ of the multiple linear regression model almost *linearly with time*.

Learning Curve of the RLS Algorithm

In the RLS algorithm there are two errors, the *a priori* estimation error $\xi(n)$ and the *a posteriori* estimation error $e(n)$, to be considered. Given the initial conditions of Section 13.3 we find that the mean-square values of these two errors vary differently with time n. At time $n = 1$, the mean-square value of $\xi(n)$ attains a *large* value, equal to the mean-square value of the desired response $d(n)$, and then *decays* with increasing n. The mean-square value of $e(n)$, on the other hand, attains a *small* value at $n = 1$, and then *rises* with increasing n. Accordingly, the choice of $\xi(n)$ as the error of interest yields a learning curve for the RLS algorithm that has the same general shape as that for the LMS algorithm. By so doing, we can then make a direct graphical comparison between the learning curves of the RLS and LMS algorithms. We will therefore base computation of the ensemble-averaged learning curve of the RLS algorithm on the *a priori* estimation error $\xi(n)$.

Eliminating the desired response $d(n)$ between Eqs. (13.26) and (13.44), we may express the *a priori* estimation error $\xi(n)$ as

$$\begin{aligned}\xi(n) &= e_o(n) - [\hat{\mathbf{w}}(n-1) - \mathbf{w}_o]^H\mathbf{u}(n) \\ &= e_o(n) - \boldsymbol{\epsilon}^H(n-1)\mathbf{u}(n)\end{aligned} \tag{13.62}$$

where $\boldsymbol{\epsilon}(n-1)$ is the weight-error vector at time $n-1$. As an *index of statistical performance* for the RLS algorithm, it is convenient to use the *a priori* estimation error $\xi(n)$ to define the *mean-squared error*:

$$J'(n) = E[|\xi(n)|^2] \tag{13.63}$$

The prime in the symbol $J'(n)$ is intended to distinguish the mean-square value of $\xi(n)$ from that of $e(n)$. Substituting Eq. (13.62) in (13.63), and then expanding terms, we get

$$\begin{aligned}J'(n) = E[|e_o(n)|^2] &+ E[\mathbf{u}^H(n)\boldsymbol{\epsilon}(n-1)\boldsymbol{\epsilon}^H(n-1)\mathbf{u}(n)] \\ &- E[\boldsymbol{\epsilon}^H(n-1)\mathbf{u}(n)e_o^*(n)] - E[e_o(n)\mathbf{u}^H(n)\boldsymbol{\epsilon}(n-1)]\end{aligned} \tag{13.64}$$

With the measurement $e_o(n)$ assumed to be of zero mean, the first expectation on the right-hand side of Eq. (13.64) is simply the variance of $e_o(n)$, which is denoted by σ^2. As for the remaining three expectations, we may make the following observations in light of the independence assumption described previously:

1. The estimate $\hat{\mathbf{w}}(n-1)$, and therefore the weight-error vector $\boldsymbol{\epsilon}(n-1)$, is independent of the tap-input vector $\mathbf{u}(n)$; the latter is assumed to be drawn from a wide-sense stationary process of zero mean. Hence, we may use this statistical independence together with well-known results from matrix algebra to express the second expectation on the right-hand side of Eq. (13.64) as follows:

$$\begin{aligned} E[\mathbf{u}^H(n)\boldsymbol{\epsilon}(n-1)\boldsymbol{\epsilon}^H(n-1)\mathbf{u}(n)] &= E[\mathrm{tr}\{\mathbf{u}^H(n)\boldsymbol{\epsilon}(n-1)\boldsymbol{\epsilon}^H(n-1)\mathbf{u}(n)\}] \\ &= E[\mathrm{tr}\{\mathbf{u}(n)\mathbf{u}^H(n)\boldsymbol{\epsilon}(n-1)\boldsymbol{\epsilon}^H(n-1)\}] \\ &= \mathrm{tr}\{E[\mathbf{u}(n)\mathbf{u}^H(n)\boldsymbol{\epsilon}(n-1)\boldsymbol{\epsilon}^H(n-1)]\} \\ &= \mathrm{tr}\{E[\mathbf{u}(n)\mathbf{u}^H(n)]E[\boldsymbol{\epsilon}(n-1)\boldsymbol{\epsilon}^H(n-1)]\} \\ &= \mathrm{tr}\{\mathbf{R}\mathbf{K}(n-1)\} \end{aligned} \tag{13.65}$$

   where, in the last line, we have made use of the definitions of the ensemble-averaged correlation matrix $\mathbf{R}$ and weight-error correlation matrix $\mathbf{K}(n-1)$.

2. The measurement error $e_o(n)$ depends on the tap-input vector $\mathbf{u}(n)$; this follows from a simple rearrangement of Eq. (13.44). The weight-error vector $\boldsymbol{\epsilon}(n-1)$ is therefore independent of both $\mathbf{u}(n)$ and $e_o(n)$. Accordingly, we may show that the third expectation on the right-hand side of Eq. (13.64) is zero by first reformulating it as follows:

$$E[\boldsymbol{\epsilon}^H(n-1)\mathbf{u}(n)e_o^*(n)] = E[\boldsymbol{\epsilon}^H(n-1)]E[\mathbf{u}(n)e_o^*(n)]$$

   We now recognize from the principle of orthogonality that all the elements of the tap-input vector $\mathbf{u}(n)$ are orthogonal to the measurement error $e_o(n)$. We therefore have

$$E[\boldsymbol{\epsilon}^H(n-1)\mathbf{u}(n)e_o^*(n)] = 0 \tag{13.66}$$

3. The fourth and final expectation on the right-hand side of Eq. (13.64) has the same mathematical form as that just considered in point 2, except for a trivial complex conjugation. We may therefore set this expectation equal to zero, too:

$$E[e_o(n)\mathbf{u}^H(n)\boldsymbol{\epsilon}(n-1)] = 0 \tag{13.67}$$

Thus, recognizing that $E[|e_o(n)|^2] = \sigma^2$, and using the results of Eqs. (13.65) to (13.67) in (13.64), we get the following simple formula for the mean-squared error in the RLS algorithm.

$$J'(n) = \sigma^2 + \mathrm{tr}[\mathbf{R}\mathbf{K}(n-1)] \tag{13.68}$$

Next, substituting Eq. (13.60) in (13.68), we get (for $\lambda = 1$)

$$J'(n) = \sigma^2 + \frac{M\sigma^2}{n-M-1}, \qquad n > M+1 \tag{13.69}$$

Based on this result, we may make the following deductions:

1. The ensemble-averaged learning curve of the RLS algorithm converges in about $2M$ iterations, where M is the number of taps in the transversal filter. This means that the rate of convergence of the RLS algorithm is typically an order of magnitude *faster* than that of the LMS algorithm.
2. As the number of iterations, n, approaches infinity, the mean-squared error $J'(n)$ approaches a final value equal to the variance σ^2 of the measurement error $e_o(n)$. In other words the RLS algorithm, in theory, produces zero excess mean-squared error (or, equivalently, zero misadjustment) when operating in a stationary environment.
3. Convergence of the RLS algorithm in the mean square is independent of the eigenvalues of the ensemble-averaged correlation matrix $\mathbf{R}$ of the input vector $\mathbf{u}(n)$.

It should be emphasized that the above-mentioned improvement in the rate of convergence of the RLS algorithm over the LMS algorithm holds only when the measurement error $e_o(n)$ is small compared to the desired response $d(n)$, that is, when the signal-to-noise ratio is high. Also, the zero misadjustment property of the RLS algorithm assumes that the exponential weighting factor λ equals unity; that is, the algorithm operates with infinite memory.

13.7 COMPUTER EXPERIMENT ON ADAPTIVE EQUALIZATION

For our computer experiment, we use the RLS algorithm with the exponential weighting factor $\lambda = 1$, for the adaptive equalization of a linear dispersive communication channel. The LMS version of this study was presented in Section 9.7. The block diagram of the system used in the study is depicted in Fig. 13.5. Two independent random-number generators are used, one, denoted by x_n, for probing the channel, and the other, denoted by $v(n)$, for simulating the effect of additive white noise at the receiver input. The sequence x_n is a Bernoulli sequence with $x_n = \pm 1$; the random variable x_n has zero mean and unit variance. The second sequence $v(n)$ has no zero mean; its variance σ_v^2 is determined by the desired signal-to-noise ratio. The equalizer has 11 taps. The impulse response of the channel is defined by

$$h_n = \begin{cases} \frac{1}{2}\left[1 + \cos\left(\frac{2\pi}{W}(n-2)\right)\right], & n = 1, 2, 3 \\ 0 & \text{otherwise} \end{cases}$$

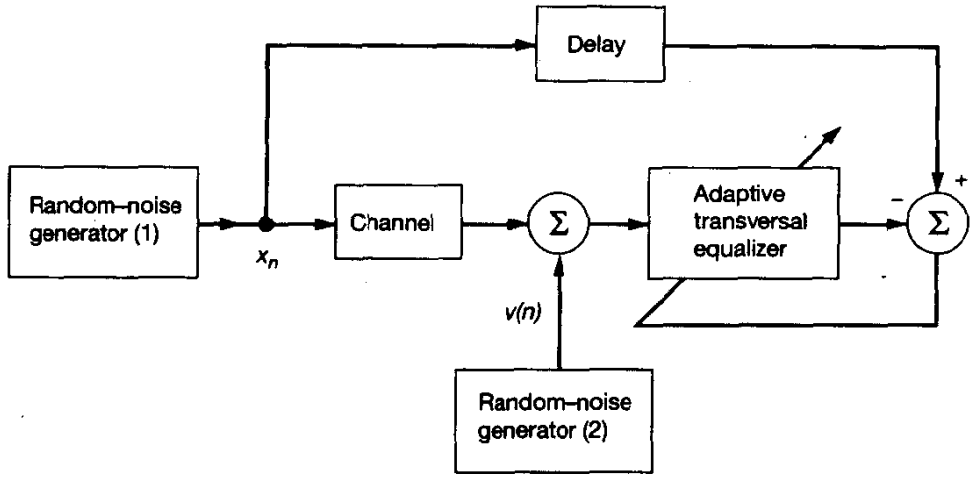


Figure 13.5 Block diagram of adaptive equalizer for computer experiment.

where W controls the amount of amplitude distortion and therefore the eigenvalue spread produced by the channel. The channel input x_n, after a delay of seven samples, provides the desired response for the equalizer (see Section 9.13 for details).

The experiment is in two parts: In part 1 the signal-to-noise ratio is high, and in part 2 it is low. In both parts of the experiment, the constant $\delta = 0.004$.

1. Signal-to-Noise Ratio = 30 dB. The results of the experiment for a fixed signal-to-noise ratio of 30 dB (equivalently, variance $\sigma_v^2 = 0.001$) and varying W or eigenvalue spread $\chi(\mathbf{R})$ were presented previously in Chapter 10; see Fig. 10.10. The four parts of that figure correspond to the parameter $W = 2.9$, 3.1, 3.3, and 3.5, or equivalently $\chi(\mathbf{R}) = 6.0782$, 11.1238, 21.7132, and 46.8216, respectively (see Table 9.2 for details). Each part of the figure includes learning curves for the LMS, DCT-LMS, and RLS algorithms. The present discussion pertains to the RLS and LMS algorithms. The learning curves of the RLS algorithm were obtained by ensemble-averaging the squared value of the *a priori* estimation error $\xi(n)$ for each iteration n, and those for the LMS algorithm were obtained by ensemble-averaging the squared value of the *a posteriori* estimation error $e(n)$. The ensemble-averaging was performed over 200 independent trials of the experiment. For the LMS algorithm, the step-size parameter $\mu = 0.075$ was used. Based on the results shown in Fig. 10.10, we may make the following observations:

- Convergence of the RLS algorithm is attained in about 20 iterations, approximately twice the number of taps in the transversal equalizer.
- Rate of convergence of the RLS algorithm is relatively insensitive to variations in the eigenvalue spread $\chi(\mathbf{R})$ compared to the LMS algorithm. This property is

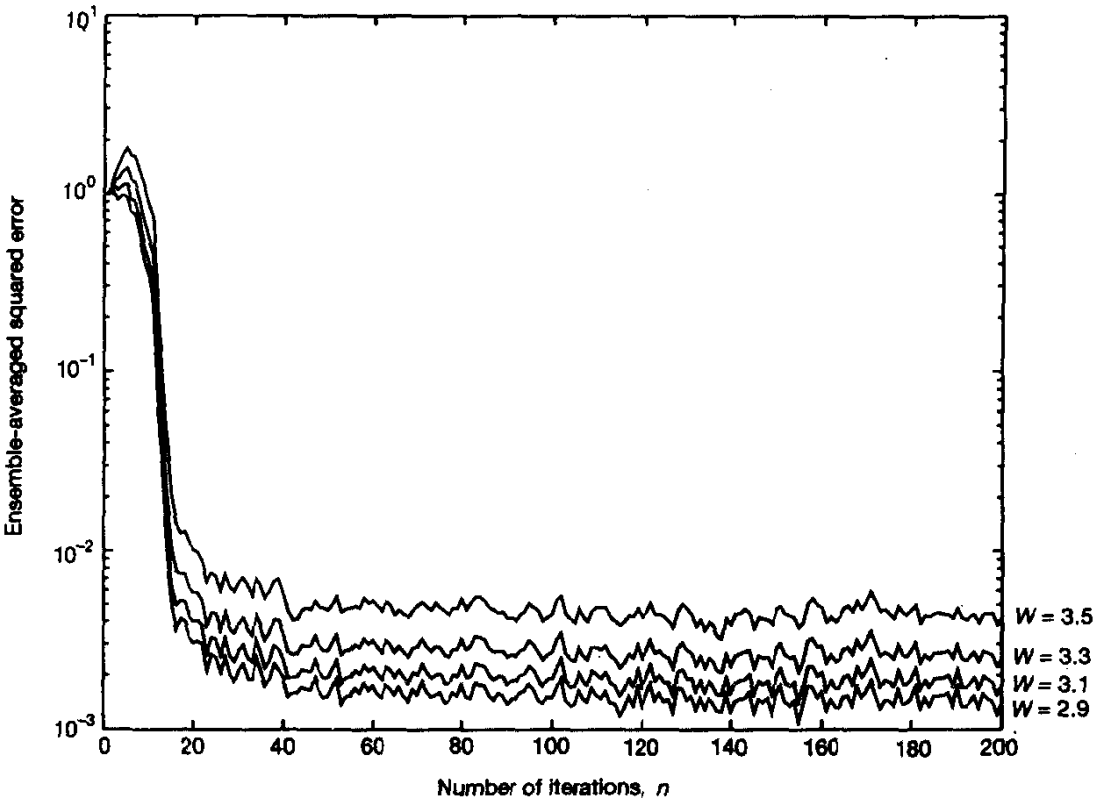


Figure 13.6 Learning curves for the RLS algorithm with four different eigenvalue spreads, and $\delta = 0.004$, $\lambda = 1.0$.

clearly illustrated in Fig. 13.6, where we have reproduced the learning curves of the RLS algorithm, corresponding to the four different values of the eigenvalue spread.

- The RLS algorithm converges much faster than the LMS algorithm.
- The steady-state value of the averaged squared error produced by the RLS algorithm is much smaller than in the case of the LMS algorithm, confirming what we said earlier: The RLS algorithm produces zero misadjustment, in theory.

The results presented in Figs. 10.10 and 13.6 clearly show the superior rate of convergence of the RLS over the LMS algorithm; for it to be realized, however, the signal-to-noise ratio has to be high. This advantage is lost when the signal-to-noise ratio is not high, as demonstrated next.

2. Signal-to-Noise Ratio = 10 dB. Figure 13.7 shows the learning curves for the RLS algorithm and the LMS algorithm (with the step-size parameter $\mu = 0.075$) for

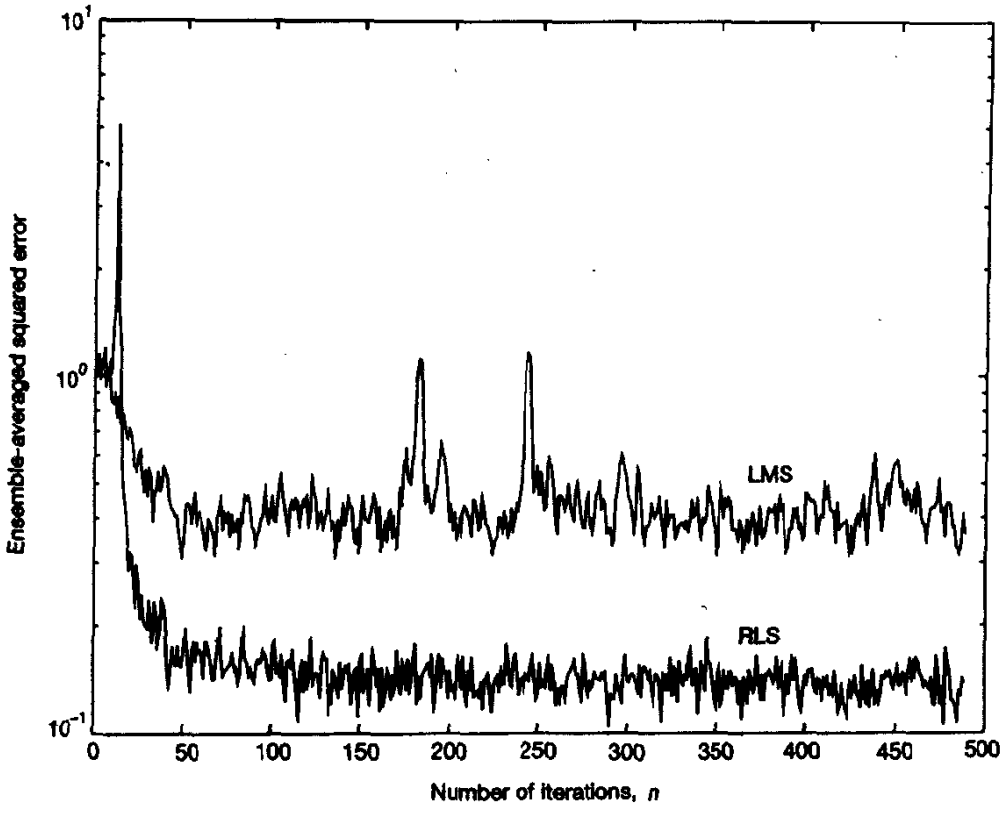


Figure 13.7 Learning curves for the RLS and LMS algorithms for $W = 3.1$ (i.e., eigenvalue spread $\chi(\mathbf{R}) = 11.1238$), and SNR = 10 dB. RLS: $\delta = 0.004$ and $\lambda = 1.0$. LMS: Step size parameter $\mu = 0.075$.

$W = 3.1$ and signal-to-noise ratio of 10 dB. Insofar as the rate of convergence is concerned, we now see that the RLS and LMS algorithms perform in roughly the same manner, both requiring about 40 iterations to converge.

13.8 STATE-SPACE FORMULATION OF THE RLS PROBLEM

The exponentially weighted RLS algorithm was derived from first principles in Section 13.3 using the matrix inversion lemma. The underlying mathematical model used in this derivation is a *deterministic* one, in that the only source of uncertainty in the model resides in the measurement error $e_o(n)$. The RLS algorithm may also be *deduced in its exact form* directly from the covariance Kalman filtering algorithm of Chapter 7 by using a state-space model that matches the RLS problem (Sayed and Kailath, 1994). The state-space model used here is naturally stochastic in its formulation. This alternative approach to the

solution of the RLS problem is important because it enables us to establish a highly valuable list of one-to-one correspondences between the RLS variables rooted in a linear regression model and the Kalman variables rooted in a state-space model. With such a list at our disposal, we may exploit the vast control literature on Kalman filters to solve the RLS problem in all of its different forms in a *unified manner*, which is precisely our ultimate objective.

A Comparison of Stochastic and Deterministic Models

To proceed, consider the unforced dynamical model described by Eqs. (7.74) to (7.76), reproduced here for convenience of presentation:

$$\mathbf{x}(n+1) = \lambda^{-1/2}\mathbf{x}(n) \tag{13.70}$$

$$y(n) = \mathbf{u}^H(n)\mathbf{x}(n) + v(n) \tag{13.71}$$

where $\mathbf{x}(n)$ is the state vector of the model, $y(n)$ is a scalar observation or reference signal, $\mathbf{u}^H(n)$ is the measurement matrix, and $v(n)$ is a scalar white noise process of zero mean and unit variance. The model parameter λ is a positive real constant. From Eq. (13.70), we readily see that

$$\mathbf{x}(n) = \lambda^{-n/2}\mathbf{x}(0) \tag{13.72}$$

where $\mathbf{x}(0)$ is the initial value of the state vector. Hence, evaluating Eq. (13.71) for time $n = 0, 1, \ldots$, and then utilizing Eq. (13.72) to express the state vectors at different times in terms of the common value $\mathbf{x}(0)$, we obtain the following stochastic system of linear simultaneous equations:

$$\begin{aligned} y(0) &= \mathbf{u}^H(0)\mathbf{x}(0) + v(0) \\ y(1) &= \lambda^{-1/2}\mathbf{u}^H(1)\mathbf{x}(0) + v(1) \\ &\vdots \\ y(n) &= \lambda^{-n/2}\mathbf{u}^H(n)\mathbf{x}(0) + v(n) \end{aligned} \tag{13.73}$$

Equivalently, we may write

$$\begin{aligned} y(0) &= \mathbf{u}^H(0)\mathbf{x}(0) + v(0) \\ \lambda^{1/2}y(1) &= \mathbf{u}^H(1)\mathbf{x}(0) + \lambda^{1/2}v(1) \\ &\vdots \\ \lambda^{n/2}y(n) &= \mathbf{u}^H(n)\mathbf{x}(0) + \lambda^{n/2}v(n) \end{aligned} \tag{13.74}$$

The system of Eqs. (13.74) represents a stochastic characterization of the unforced dynamical model pertaining to a Kalman filter point of view.

Consider next a deterministic formulation of the problem as seen from the RLS point of view. Adapting the linear regression model of Eq. (13.44) to the problem at hand, we may write the following deterministic system of linear simultaneous equations:

$$\begin{aligned} d^*(0) &= \mathbf{u}^H(0)\mathbf{w}_o + e_o^*(0) \\ d^*(1) &= \mathbf{u}^H(1)\mathbf{w}_o + e_o^*(1) \\ &\vdots \\ d^*(n) &= \mathbf{u}^H(n)\mathbf{w}_o + e_o^*(n) \end{aligned} \tag{13.75}$$

where $\mathbf{w}_o$ is the unknown parameter vector of the model, $\mathbf{u}(n)$ is the input vector, $d(n)$ is the reference signal or desired response, and $e_o(n)$ is the measurement error.

We thus have two systems of linear simultaneous equations for solving essentially the same problem. One system of equations is stochastic, rooted in Kalman filter theory; and the other system is deterministic, rooted in least-squares estimation theory. We would intuitively expect that both approaches lead to exactly the same solution for the problem at hand. Moreover, recognizing that these two systems of equations have the same mathematical form, it seems reasonable for us to set

$$\mathbf{x}(0) = \mathbf{w}_o \tag{13.76}$$

On this basis, a comparison between the stochastic equations (13.74) and the deterministic equations (13.75) immediately reveals the following one-to-one correspondences:

$$y(n) = \lambda^{-n/2} d^*(n) \tag{13.77}$$

$$v(n) = \lambda^{-n/2} e_o^*(n) \tag{13.78}$$

where the asterisk denotes complex conjugation, the variables on the left-hand side refer to the state-space model, and those on the right-hand side refer to the linear regression model.

A Comparison of Covariance Kalman Filtering and RLS Algorithm

We may expand on this list of one-to-one correspondences by comparing the covariance Kalman filtering algorithm summarized in Table 7.3 of Chapter 7 and the RLS algorithm summarized in Table 13.1 of this chapter. Indeed, a comparison of these two algorithms, line by line, immediately leads us to write [assuming $\mathbf{K}(1, 0) = \lambda^{-1}\delta^{-1}\mathbf{I}$ and $\hat{\mathbf{x}}(1|\mathcal{Y}_0) = \mathbf{0}$ in Table 7.3]:

$$\mathbf{K}(n-1) = \lambda^{-1}\mathbf{P}(n-1) \tag{13.79}$$

$$\mathbf{g}(n) = \lambda^{-1/2}\mathbf{k}(n) \tag{13.80}$$

$$\alpha(n) = \lambda^{-n/2}\xi^*(n) \tag{13.81}$$

$$\hat{\mathbf{x}}(n+1\,|\,\mathcal{Y}_n) = \lambda^{-(n+1)/2}\hat{\mathbf{w}}(n) \tag{13.82}$$

TABLE 13.2 SUMMARY OF CORRESPONDENCES BETWEEN KALMAN VARIABLES AND RLS VARIABLES

Kalman		RLS	
Description	Variable	Variable	Description
Initial value of state vector	$\mathbf{x}(0)$	$\mathbf{w}_o$	Unknown regression-coefficient vector
State vector	$\mathbf{x}(n)$	$\lambda^{-n/2}\mathbf{w}_o$	Exponentially weighted version of unknown coefficient-regression vector
Reference (observation) signal	$y(n)$	$\lambda^{-n/2}d^*(n)$	Desired response
Measurement noise	$v(n)$	$\lambda^{-n/2}e_o^*(n)$	Measurement error
One-step prediction of state vector	$\hat{\mathbf{x}}(n+1\mid\mathcal{Y}_n)$	$\lambda^{-(n+1)/2}\hat{\mathbf{w}}(n)$	Estimate of tap-weight vector
Correlation matrix of error in state prediction	$\mathbf{K}(n)$	$\lambda^{-1}\mathbf{P}(n)$	Inverse of correlation matrix of input vector
Kalman gain	$\mathbf{g}(n)$	$\lambda^{-1/2}\mathbf{k}(n)$	Gain vector
Innovation	$\alpha(n)$	$\lambda^{-n/2}\xi^*(n)$	*A priori* estimation error
Conversion factor	$r^{-1}(n)$	$\gamma(n)$	Conversion factor
Initial conditions	$\hat{\mathbf{x}}(1\mid\mathcal{Y}_0)=\mathbf{0}$ $\mathbf{K}(0)$	$\hat{\mathbf{w}}(0)=\mathbf{0}$ $\lambda^{-1}\mathbf{P}(0)$	Initial conditions

Moreover, comparing Eqs. (7.65) and (13.38), we see that the conversion factor $r^{-1}(n)$ in the specialized form of the covariance Kalman filtering algorithm[2] and the conversion factor $\gamma(n)$ in the RLS algorithm are exactly the same, as shown by

$$r^{-1}(n) = \gamma(n) \tag{13.83}$$

Thus, collecting the results described by Eqs. (13.77) to (13.83) and other related results under one umbrella, we may set up the one-to-one correspondences listed in Table 13.2 between the Kalman and RLS variables, assuming complex-valued data.[3] The left half of the table pertains to the Kalman variables and their descriptions, whereas the right half per-

[2]Adapting Eq. (7.65) to the specialized form of the covariance (Kalman) filtering algorithm summarized in Table 7.3, we have

$$r^{-1}(n) = \frac{e(n)}{\xi(n)} = \frac{1}{\mathbf{u}^H(n)\mathbf{K}(n-1)\mathbf{u}(n)+1}$$

[3]The list of correspondences presented in Table 13.2 is the same as that in the paper by Sayed and Kailath (1994); some minor differences are merely the result of differences in notation.

tains to the RLS variables. In making up the descriptions for the latter, we have focused on the variables of interest, ignoring (for the sake of simplicity) references to the operation of complex conjugation and multiplication by powers of the exponential weighting factor λ involved here.

13.9 SUMMARY AND DISCUSSION

In this chapter we first derived the recursive least-squares (RLS) algorithm as a natural extension of the method of least squares. The derivation was based on a lemma in matrix algebra known as the matrix inversion lemma.

The fundamental difference between the RLS algorithm and the LMS algorithm may be stated as follows: The step-size parameter μ in the LMS algorithm is replaced by $\mathbf{\Phi}^{-1}(n)$, that is, the inverse of the correlation matrix of the input vector $\mathbf{u}(n)$. This modification has a profound impact on the convergence behavior of the RLS algorithm for a stationary environment, as summarized here:

1. The rate of convergence of the RLS algorithm is typically an order of magnitude faster than that of the LMS algorithm.
2. The rate of convergence of the RLS algorithm is invariant to the eigenvalue spread (i.e., condition number) of the ensemble-averaged correlation matrix $\mathbf{R}$ of the input vector $\mathbf{u}(n)$.
3. The excess mean-squared error $J'_{ex}(n)$ of the RLS algorithm converges to zero as the number of iterations, n, approaches infinity.

The operation of the RLS algorithm described herein refers to a stationary environment with the exponential weighting factor $\lambda = 1$. The case of $\lambda \neq 1$ is considered in Chapter 16, where it is shown that Properties 1 and 2 still hold but the excess mean-squared error $J'_{ex}(n)$ is no longer zero. In any event, computation of the mean-squared error $J'(n)$, produced by the RLS algorithm, is based on the *a priori* estimation error $\xi(n)$.

Another important result that we established in this chapter is that, although the RLS algorithm is deterministic and the Kalman filter is stochastic, there exist one-to-one correspondences between their individual variables. In particular, we may use these correspondences to derive important variants of the RLS algorithm, in a unified manner, from their Kalman filter counterparts, as demonstrated in the next two chapters.

PROBLEMS

1. To permit a recursive implementation of the method of least squares, the window or weighting function $\beta(n, i)$ must have a suitable structure. Assume that $\beta(n, i)$ may be expressed as

$$\beta(n, i) = \lambda(i)\beta(n, i-1), \qquad i = 1, \ldots, n$$

where $\beta(n, n) = 1$. Hence, show that

$$\beta(n, i) = \prod_{k=i+1}^{n} \lambda^{-1}(k)$$

What is the form of $\lambda(k)$ for which $\beta(n, i) = \lambda^{n-i}$ is obtained?

2. Establish the validity of the matrix inversion lemma.
3. Consider a correlation matrix $\boldsymbol{\Phi}(n)$ defined by

$$\boldsymbol{\Phi}(n) = \mathbf{u}(n)\mathbf{u}^H(n) + \delta\mathbf{I}$$

where $\mathbf{u}(n)$ is a tap-input vector and δ is a small positive constant. Use the matrix inversion lemma to evaluate $\mathbf{P}(n) = \boldsymbol{\Phi}^{-1}(n)$.

4. Consider the modified definition of the correlation matrix $\boldsymbol{\Phi}(n)$ given in Eq. (13.28), which is reproduced here for convenience.

$$\boldsymbol{\Phi}(n) = \sum_{i=1}^{n} \lambda^{n-i}\mathbf{u}(i)\mathbf{u}^H(i) + \delta\lambda^n\mathbf{I}$$

where $\mathbf{u}(i)$ is the tap-input vector, λ is the exponential weighting factor, and δ is a small positive constant. Show that the use of this new definition for $\boldsymbol{\Phi}(n)$ leaves the equations that define the RLS algorithm completely unchanged.

5. Let $\xi(n)$ denote the *a priori* estimation error

$$\xi(n) = d(n) - \hat{\mathbf{w}}^H(n-1)\mathbf{u}(n)$$

where $d(n)$ is the desired response, $\mathbf{u}(n)$ is the tap-input vector, and $\hat{\mathbf{w}}(n-1)$ is the old estimate of the tap-weight vector. Let $e(n)$ denote the *a posteriori* estimation error

$$e(n) = d(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n)$$

where $\hat{\mathbf{w}}(n)$ is the current estimate of the tap-weight vector. For complex-valued data, both $\xi(n)$ and $e(n)$ are likewise complex valued. Show that the product $\xi(n)e^*(n)$ is always real valued.

6. Given the initial conditions of Section 13.3 for the RLS algorithm, explain the reasons for the fact that the mean-square value of the *a posteriori* estimation error $e(n)$ attains a small value at $n = 1$ and then rises with increasing n.
7. The last two entries in Table 13.2 pertain to the one-to-one correspondences between the initial conditions of the Kalman variables and those of the RLS variables. Justify the validity of these two entries.

CHAPTER

14

Square-Root Adaptive Filters

One of the problems encountered in applying the RLS algorithm of Chapter 13 is that of numerical instability, which can arise because of the way in which the Riccati difference equation is formulated. This same problem is also known to arise in the classical Kalman filtering algorithm for exactly the same reason. In Chapter 7 we pointed out that the instability (divergence) problem encountered in a Kalman filter can be ameliorated by using a square-root variant of the filter. At that point in the discussion, we deferred a detailed treatment of square-root Kalman filtering until we are ready for it. In this chapter we will take up a full discussion of this issue, which we do in the next section. The solution to the square-root Kalman filtering problem sets the stage for deriving the corresponding variants of the RLS algorithm in light of the one-to-one correspondences that exist between the Kalman variables and the RLS variables, established in the previous chapter.

14.1 SQUARE-ROOT KALMAN FILTERS

The recursions in a Kalman filter of the covariance type propagate the matrix $\mathbf{K}(n)$, which denotes the correlation matrix of the error in the filtered state estimate; this propagation takes place via the Riccati difference equation. The recursions in a root-square Kalman filter, on the other hand, propagate a lower triangular matrix $\mathbf{K}^{1/2}(n)$, defined as the *square root* of $\mathbf{K}(n)$. The relation between $\mathbf{K}(n)$ and $\mathbf{K}^{1/2}(n)$ is defined by

$$\mathbf{K}(n) = \mathbf{K}^{1/2}(n)\mathbf{K}^{H/2}(2) \tag{14.1}$$

where the upper triangular matrix $\mathbf{K}^{H/2}(n)$ is the Hermitian transpose of $\mathbf{K}^{1/2}(n)$. Unlike the situation that may exist in the covariance Kalman filter, the nonnegative definite character of $\mathbf{K}(n)$ as a correlation matrix is preserved by virtue of the fact that the product of any square matrix and its Hermitian transpose is always a nonnegative definite matrix.

In this section we derive the square-root forms of the covariance and information implementations of the Kalman filter. But, with variants of the RLS algorithm in mind, we will focus our attention on the special unforced dynamical model (Sayed and Kailath, 1994):

$$\mathbf{x}(n+1) = \lambda^{-1/2}\mathbf{x}(n+1) \tag{14.2}$$

$$y(n) = \mathbf{u}^H(n)\mathbf{x}(n) + v(n) \tag{14.3}$$

where $\mathbf{x}(n)$ is the state vector, the row vector $\mathbf{u}^H(n)$ is the measurement matrix, the scalar $y(n)$ is an observation or reference signal, and the scalar $v(n)$ is a white noise process of zero mean and unit variance; the positive real scalar λ is a constant of the model. However, before proceeding with the derivations, we digress briefly to consider a lemma in matrix algebra that is pivotal to our present discussion.

Matrix Factorization Lemma

Given any two N-by-M matrices, $\mathbf{A}$ and $\mathbf{B}$, with dimension $N \leq M$, the *matrix factorization lemma* states that (Stewart, 1973; Golub and Van Loan, 1989; Sayed and Kailath, 1994)

$$\mathbf{A}\mathbf{A}^H = \mathbf{B}\mathbf{B}^H \tag{14.4}$$

if, and only if, there exists a unitary matrix $\boldsymbol{\Theta}$ such that

$$\mathbf{B} = \mathbf{A}\boldsymbol{\Theta} \tag{14.5}$$

Assuming that the condition (14.5) holds, we readily find that

$$\mathbf{B}\mathbf{B}^H = \mathbf{A}\boldsymbol{\Theta}\boldsymbol{\Theta}^H\mathbf{A}^H \tag{14.6}$$

From the definition of a unitary matrix, we have

$$\boldsymbol{\Theta}\boldsymbol{\Theta}^H = \mathbf{I} \tag{14.7}$$

where $\mathbf{I}$ is the identity matrix. Hence, Eq. (14.6) reduces immediately to Eq. (14.4).

Conversely, the equality described in Eq. (14.4) implies that the matrices $\mathbf{A}$ and $\mathbf{B}$ must be related. We may prove the converse implication of the matrix factorization lemma by invoking the singular value decomposition theorem. According to this theorem, the matrix $\mathbf{A}$ may be factored as follows (see Section 11.10):

$$\mathbf{A} = \mathbf{U}_A\boldsymbol{\Sigma}_A\mathbf{V}_A^H \tag{14.8}$$

where $\mathbf{U}_A$ and $\mathbf{V}_A$ are N-by-N and M-by-M unitary matrices, respectively, and $\boldsymbol{\Sigma}_A$ is an N-by-M matrix defined by the singular values of matrix $\mathbf{A}$. Similarly, the second matrix $\mathbf{B}$ may be factored as follows:

$$\mathbf{B} = \mathbf{U}_B \boldsymbol{\Sigma}_B \mathbf{V}_B^H \tag{14.9}$$

The identity $\mathbf{A}\mathbf{A}^H = \mathbf{B}\mathbf{B}^H$ implies that we have

$$\mathbf{U}_A = \mathbf{U}_B \tag{14.10}$$

and

$$\boldsymbol{\Sigma}_A = \boldsymbol{\Sigma}_B \tag{14.11}$$

Now let

$$\boldsymbol{\Theta} = \mathbf{V}_A \mathbf{V}_B^H \tag{14.12}$$

Then, substituting Eqs. (14.8) and (14.10) in (14.12) in the matrix product $\mathbf{A}\boldsymbol{\Theta}$ yields a result equal to matrix $\mathbf{B}$ by virtue of Eq. (14.9), which is precisely the converse implication of the matrix factorization lemma.

Square-root Covariance Filter

Returning to the issue of square-root Kalman filtering, we note that the Riccati difference equation for the covariance Kalman filter may be expressed as follows (by combining the first and final lines of the algorithm summarized in Table 7.3):

$$\mathbf{K}(n) = \lambda^{-1}\mathbf{K}(n-1) - \lambda^{-1}\mathbf{K}(n-1)\mathbf{u}(n)r^{-1}(n)\mathbf{u}^H(n)\mathbf{K}(n-1) \tag{14.13}$$

The scalar $r(n)$ is the variance of the filtered estimation error; it is defined by

$$r(n) = \mathbf{u}^H(n)\mathbf{K}(n-1)\mathbf{u}(n) + 1 \tag{14.14}$$

There are four distinct matrix terms that constitute the right-hand side of the Riccati equation (14.13), in light of which we may introduce the following two-by-two block matrix:

$$\mathbf{M}(n) = \begin{bmatrix} \mathbf{u}^H(n)\mathbf{K}(n-1)\mathbf{u}(n) + 1 & \lambda^{-1/2}\mathbf{u}^H(n)\mathbf{K}(n-1) \\ \lambda^{-1/2}\mathbf{K}(n-1)\mathbf{u}(n) & \lambda^{-1}\mathbf{K}(n-1) \end{bmatrix} \tag{14.15}$$

Expressing the correlation matrix $\mathbf{K}(n-1)$ in its factored form:

$$\mathbf{K}(n-1) = \mathbf{K}^{1/2}(n-1)\mathbf{K}^{H/2}(n-1) \tag{14.16}$$

and recognizing that the matrix $\mathbf{M}(n)$ is a nonnegative-definite matrix, we may use the Cholesky factorization to express Eq. (14.15) as follows:

$$\mathbf{M}(n) = \begin{bmatrix} 1 & \mathbf{u}^H(n)\mathbf{K}^{1/2}(n-1) \\ \mathbf{0} & \lambda^{-1/2}\mathbf{K}^{1/2}(n-1) \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{K}^{H/2}(n-1)\mathbf{u}(n) & \lambda^{-1/2}\mathbf{K}^{H/2}(n-1) \end{bmatrix} \tag{14.17}$$

where $\mathbf{0}$ is the null vector.

The matrix product on the right-hand side of Eq. (14.17) may be interpreted as the product of matrix $\mathbf{A}$, say, and its Hermitian transpose $\mathbf{A}^H$. The stage is therefore set for invoking the matrix factorization lemma, according to which we may write

$$\begin{bmatrix} 1 & \mathbf{u}^H(n)\mathbf{K}^{1/2}(n-1) \\ \mathbf{0} & \lambda^{-1/2}\mathbf{K}^{1/2}(n-1) \end{bmatrix} \boldsymbol{\Theta}(n) = \begin{bmatrix} b_{11}(n) & \mathbf{0}^T \\ \mathbf{b}_{21}(n) & \mathbf{B}_{22}(n) \end{bmatrix} \tag{14.18}$$

where $\Theta(n)$ is a unitary rotation; and the scalar $b_{11}(n)$, the vector $\mathbf{b}_{21}(n)$, and the matrix $\mathbf{B}_{22}(n)$ denote the nonzero block elements of matrix $\mathbf{B}$. In Eq. (14.18), we may distinguish two arrays of numbers:

- A *prearray*, which is operated on by a unitary rotation.
- A *postarray*, which is characterized by a block zero entry resulting from the action of the unitary rotation. The postarray therefore has a "triangular" structure in a block sense.

To evaluate the unknown block elements b_{11}, $\mathbf{b}_{21}$, and $\mathbf{B}_{22}$ of the postarray, we proceed by squaring both sides of Eq. (14.18). Then, recognizing that $\Theta(n)$ is a unitary matrix, and therefore $\Theta(n)\Theta^H(n)$ equals the identity matrix for all n, we may write

$$\underbrace{\begin{bmatrix} 1 & \mathbf{u}^H(n)\mathbf{K}^{1/2}(n-1) \\ \mathbf{0} & \lambda^{-1/2}\mathbf{K}^{1/2}(n-1) \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{K}^{H/2}(n-1)\mathbf{u}(n) & \lambda^{-1/2}\mathbf{K}^{H/2}(n-1) \end{bmatrix}}_{\mathbf{A}^H}$$
$$= \underbrace{\begin{bmatrix} b_{11}(n) & \mathbf{0}^T \\ \mathbf{b}_{21}(n) & \mathbf{B}_{22}(n) \end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{bmatrix} b_{11}^*(n) & \mathbf{b}_{21}^H(n) \\ \mathbf{0} & \mathbf{B}_{22}^H(n) \end{bmatrix}}_{\mathbf{B}^H} \tag{14.19}$$

Thus, comparing the respective terms on both sides of the equality (14.19), we get the following identities:

$$|b_{11}(n)|^2 = \mathbf{u}^H(n)\mathbf{K}(n-1)\mathbf{u}(n) + 1 = r(n) \tag{14.20}$$

$$\mathbf{b}_{21}(n)b_{11}^*(n) = \lambda^{-1/2}\mathbf{K}(n-1)\mathbf{u}(n) \tag{14.21}$$

$$\mathbf{b}_{21}(n)\mathbf{b}_{21}^H(n) + \mathbf{B}_{22}(n)\mathbf{B}_{22}^H(n) = \lambda^{-1}\mathbf{K}(n-1) \tag{14.22}$$

Equations (14.20) to (14.22) may be satisfied by choosing

$$b_{11}(n) = r^{1/2}(n) \tag{14.23}$$

$$\mathbf{b}_{21}(n) = \lambda^{-1/2}\mathbf{K}(n-1)\mathbf{u}(n)r^{-1/2}(n) = \mathbf{g}(n)r^{1/2}(n) \tag{14.24}$$

$$\mathbf{B}_{22}(n) = \mathbf{K}^{1/2}(n) \tag{14.25}$$

where, in the second line, $\mathbf{g}(n)$ denotes the Kalman gain; see the first computation step of Table 7.3 for a definition of the Kalman gain $\mathbf{g}(n)$.

We may thus rewrite Eq. (14.18) as

$$\begin{bmatrix} 1 & \mathbf{u}^H(n)\mathbf{K}^{1/2}(n-1) \\ \mathbf{0} & \lambda^{-1/2}\mathbf{K}^{1/2}(n-1) \end{bmatrix} \Theta(n) = \begin{bmatrix} r^{1/2}(n) & \mathbf{0}^T \\ \mathbf{g}(n)r^{1/2}(n) & \mathbf{K}^{1/2}(n) \end{bmatrix} \tag{14.26}$$

The block elements of the prearray and postarray in Eq. (14.26) deserve close scrutiny, as they reveal some useful properties of their own:

- The block elements $\{\lambda^{-1/2}\mathbf{K}^{1/2}(n-1), \mathbf{u}^H(n)\,\mathbf{K}^{1/2}(n-1)\}$ of the prearray uniquely characterize the constitution of the quantities contained in the right-hand side of the Riccati difference equation (14.13), except for $r(n)$. Correspondingly, the block element $\mathbf{K}^{1/2}(n)$ of the postarray provides the quantity needed to update the prearray and therefore initiate the next iteration of the algorithm.
- Inclusion of the block elements $\{1, \mathbf{0}\}$ in the prearray induces the generation of two block elements in the postarray, namely, $\{r^{1/2}(n), \mathbf{g}(n)r^{1/2}(n)\}$. These elements make it possible to calculate two useful variables: the Kalman gain $\mathbf{g}(n)$ and the variance $r(n)$ of the filtered estimation error. The variance $r(n)$ is obtained simply by squaring the scalar entry $r^{1/2}(n)$. The Kalman gain $\mathbf{g}(n)$ is obtained equally simply by dividing the block entry $\mathbf{g}(n)r^{1/2}(n)$ by $r^{1/2}(n)$.

Building on the latter result, we may now readily update the state estimate as follows:

$$\hat{\mathbf{x}}(n+1\,|\,\mathcal{Y}_n) = \lambda^{-1/2}\hat{\mathbf{x}}(n\,|\,\mathcal{Y}_{n-1}) + \mathbf{g}(n)\alpha(n) \tag{14.27}$$

where $\alpha(n)$ is the innovation defined by

$$\alpha(n) = y(n) - \mathbf{u}^H(n\hat{\mathbf{x}}(n\,|\,\mathcal{Y}_{n-1}) \tag{14.28}$$

Table 14.1 presents a summary of the computations performed in the square-root covariance filtering algorithm (Sayed and Kailath, 1994). The initialization of the algorithm proceeds in exactly the same way as for the conventional covariance Kalman filtering algorithm (see Table 7.3).

Square-root Information Filters

Consider next the square-root implementation of Kalman's filtering algorithm, which propagates the inverse matrix $\mathbf{K}^{-1}(n)$ rather than $\mathbf{K}(n)$ itself. This form of propagation is useful, particularly when there exist large initial uncertainties (i.e., the initial value $\mathbf{\Pi}_0$ of the correlation matrix $\mathbf{K}(0)$ is large). A summary of the information filtering algorithm is presented in Table 7.4. The first two recursions of the algorithm are reproduced here for convenience of presentation:

$$\mathbf{K}^{-1}(n) = \lambda\mathbf{K}^{-1}(n-1) + \lambda\mathbf{u}(n)\mathbf{u}^H(n) \tag{14.29}$$

$$\mathbf{K}^{-1}(n)\hat{\mathbf{x}}(n+1\,|\,\mathcal{Y}_n) = \lambda^{1/2}\mathbf{K}^{-1}(n-1)\hat{\mathbf{x}}(n\,|\,\mathcal{Y}_{n-1}) + \lambda^{1/2}\mathbf{u}(n)y(n) \tag{14.30}$$

Let the inverse matrix $\mathbf{K}^{-1}(n)$ be expressed in its factored form:

$$\mathbf{K}^{-1}(n) = \mathbf{K}^{-H/2}(n)\mathbf{K}^{-1/2}(n) \tag{14.31}$$

TABLE 14.1 SUMMARY OF THE COMPUTATIONS PERFORMED IN SQUARE-ROOT VARIANTS OF THE KALMAN FILTER

1. *Square-root covariance filter:*

$$\begin{bmatrix} 1 & \mathbf{u}^H(n)\mathbf{K}^{1/2}(n-1) \\ \mathbf{0} & \lambda^{-1/2}\mathbf{K}^{1/2}(n-1) \end{bmatrix} \boldsymbol{\Theta}(n) = \begin{bmatrix} r^{1/2}(n) & \mathbf{0}^T \\ \mathbf{g}(n)r^{1/2}(n) & \mathbf{K}^{1/2}(n) \end{bmatrix}$$

$$\mathbf{g}(n) = (\mathbf{g}(n)r^{1/2}(n))\,(r^{1/2}(n))^{-1}$$

$$\alpha(n) = y(n) - \mathbf{u}^H(n)\hat{\mathbf{x}}(n \mid \mathcal{Y}_{n-1})$$

$$\hat{\mathbf{x}}(n+1 \mid \mathcal{Y}_n) = \lambda^{-1/2}\hat{\mathbf{x}}(n \mid \mathcal{Y}_{n-1}) + \mathbf{g}(n)\alpha(n)$$

2. *Square-root information filter:*

$$\begin{bmatrix} \lambda^{1/2}\mathbf{K}^{-H/2}(n-1) & \lambda^{1/2}\mathbf{u}(n) \\ \hat{\mathbf{x}}^H(n \mid \mathcal{Y}_{n-1})\mathbf{K}^{-H/2}(n-1) & y^*(n) \\ \mathbf{0}^T & 1 \end{bmatrix} \boldsymbol{\Theta}(n) = \begin{bmatrix} \mathbf{K}^{-H/2}(n) & \mathbf{0} \\ \hat{\mathbf{x}}^H(n+1 \mid \mathcal{Y}_n)\mathbf{K}^{-H/2}(n) & r^{-1/2}(n)\alpha^*(n) \\ \lambda^{1/2}\mathbf{u}^H(n)\mathbf{K}^{1/2}(n) & r^{-1/2}(n) \end{bmatrix}$$

$$\hat{\mathbf{x}}^H(n+1 \mid \mathcal{Y}_n) = (\hat{\mathbf{x}}^H(n+1 \mid \mathcal{Y}_n)\mathbf{K}^{-H/2}(n))(\mathbf{K}^{-H/2}(n))^{-1}$$

3. *Extended square-root information filter:*

$$\begin{bmatrix} \lambda^{1/2}\mathbf{K}^{-H/2}(n-1) & \lambda^{1/2}\mathbf{u}(n) \\ \hat{\mathbf{x}}^H(n \mid \mathcal{Y}_{n-1})\mathbf{K}^{-H/2}(n-1) & y^*(n) \\ \mathbf{0}^T & 1 \\ \lambda^{-1/2}\mathbf{K}^{1/2}(n-1) & \mathbf{0} \end{bmatrix} \boldsymbol{\Theta}(n) = \begin{bmatrix} \mathbf{K}^{-H/2}(n) & \mathbf{0} \\ \hat{\mathbf{x}}^H(n+1 \mid \mathcal{Y}_n)\mathbf{K}^{-H/2}(n) & r^{-1/2}(n)\alpha^*(N) \\ \lambda^{1/2}\mathbf{u}^H(n)\mathbf{K}^{1/2}(n) & r^{-1/2}(n) \\ \mathbf{K}^{1/2}(n) & -\mathbf{g}(n)r^{1/2}(n) \end{bmatrix}$$

$$\hat{\mathbf{x}}(n+1 \mid \mathcal{Y}_n) = \lambda^{-1/2}\hat{\mathbf{x}}(n \mid \mathcal{Y}_{n-1}) + (\mathbf{g}(n)r^{1/2}(n))(r^{-1/2}(n)\alpha^*(n))^*$$
$$= (\mathbf{K}^{1/2}(n))\,(\mathbf{K}^{-1/2}(n)\hat{\mathbf{x}}(n+1 \mid \mathcal{Y}_n))$$

Note: In all three cases, $\boldsymbol{\Theta}(n)$ is a unitary rotation that produces a block zero entry in the top row of the postarray.

For reasons that will become apparent presently, we find it more convenient to express Eqs. (14.29) and (14.30) in their Hermitian transposed forms, in which case we may express the four quantities on the right-hand sides of these equations in their individual factored forms as follows:

$$\lambda\mathbf{K}^{-H}(n-1) = (\lambda^{1/2}\mathbf{K}^{-H/2}(n-1))\,(\lambda^{1/2}\mathbf{K}^{-1/2}(n-1))$$

$$\lambda\,\mathbf{u}(n)\mathbf{u}^H(n) = (\lambda^{1/2}\mathbf{u}(n))\,(\lambda^{1/2}\mathbf{u}^H(n))$$

$$\lambda^{1/2}\hat{\mathbf{x}}^H(n \mid \mathcal{Y}_{n-1})\mathbf{K}^{-H}(n-1) = (\hat{\mathbf{x}}^H(n \mid \mathcal{Y}_{n-1})\mathbf{K}^{-H/2}(n-1))\,(\lambda^{1/2}\mathbf{K}^{-1/2}(n-1))$$

$$\lambda^{1/2}y^*(n)\mathbf{u}^H(n) = (y^*(n))\,(\lambda^{1/2}\mathbf{u}^H(n))$$

We may now identify four distinct factors as the block elements of the prearray, which are paired in the following manner:

- $\lambda^{1/2}\mathbf{K}^{-H/2}(n-1)$ and $\lambda^{1/2}\mathbf{u}(n)$, which are of dimensions M-by-M and M-by-1, respectively.

- $\hat{\mathbf{x}}^H(n \mid \mathcal{Y}_{n-1})\mathbf{K}^{-H/2}(n-1)$ and $y^*(n)$, which are of dimensions 1-by-M and 1-by-1, respectively.

The first pair of factors is naturally compatible by virtue of being made up of a matrix and a vector. The compatibility of the last pair of factors as row vectors is the reason for working with the Hermitian transposed forms of Eqs. (14.29) and (14.30). We may thus construct the following prearray:

$$\begin{bmatrix} \lambda^{1/2}\mathbf{K}^{-H/2}(n-1) & \lambda^{1/2}\mathbf{u}(n) \\ \hat{\mathbf{x}}^H(n \mid \mathcal{Y}_{n-1})\mathbf{K}^{-H/2}(n-1) & y^*(n) \\ \mathbf{0}^T & 1 \end{bmatrix}$$

The last row, made up of a block of zeros followed by a unity term, has been added in order to make room for the generation of other Kalman variables in the postarray (Morf and Kailath, 1975; Sayed and Kailath, 1994). Suppose next we choose a unitary rotation $\boldsymbol{\Theta}(n)$ that transforms this prearray so as to produce a block zero in the second entry of the postarray's top block row, as shown by

$$\begin{bmatrix} \lambda^{1/2}\mathbf{K}^{-H/2}(n-1) & \lambda^{1/2}\mathbf{u}(n) \\ \hat{\mathbf{x}}^H(n \mid \mathcal{Y}_{n-1})\mathbf{K}^{-H/2}(n-1) & y^*(n) \\ \mathbf{0}^T & 1 \end{bmatrix} \boldsymbol{\Theta}(n) = \begin{bmatrix} \mathbf{B}_{11}^H(n) & \mathbf{0} \\ \mathbf{b}_{21}^H(n) & b_{22}^*(n) \\ \mathbf{b}_{31}^H(n) & b_{32}^*(n) \end{bmatrix} \tag{14.32}$$

By proceeding in a manner similar to that described for the square-root covariance filter, that is, by squaring both sides of Eq. (14.32) and then comparing respective terms on both sides of the resulting equality, we may choose the block elements of the postarray as follows (see Problem 1):

$$\mathbf{B}_{11}^H(n) = \mathbf{K}^{-H/2}(n) \tag{14.33}$$

$$\mathbf{b}_{21}^H(n) = \hat{\mathbf{x}}^H(n+1 \mid \mathcal{Y}_n)\mathbf{K}^{-H/2}(n) \tag{14.34}$$

$$\mathbf{b}_{31}^H(n) = \lambda^{1/2}\mathbf{u}^H(n)\mathbf{K}^{1/2}(n) \tag{14.35}$$

$$b_{22}^*(n) = r^{-1/2}(n)\alpha^*(n) \tag{14.36}$$

$$b_{32}^*(n) = r^{-1/2}(n) \tag{14.37}$$

Accordingly, we may rewrite Eq. (14.32) in the form:

$$\begin{bmatrix} \lambda^{1/2}\mathbf{K}^{-H/2}(n-1) & \lambda^{1/2}\mathbf{u}(n) \\ \hat{\mathbf{x}}^H(n \mid \mathcal{Y}_{n-1})\mathbf{K}^{-H/2}(n-1) & y^*(n) \\ \mathbf{0}^T & 1 \end{bmatrix} \boldsymbol{\Theta}(n) = \begin{bmatrix} \mathbf{K}^{-H/2}(n) & \mathbf{0} \\ \hat{\mathbf{x}}^H(n+1 \mid \mathcal{Y}_n)\mathbf{K}^{-H/2}(n) & r^{-1/2}(n)\alpha^*(n) \\ \lambda^{1/2}\mathbf{u}^H(n)\mathbf{K}^{1/2}(n) & r^{-1/2}(n) \end{bmatrix} \tag{14.38}$$

The block elements of the postarray provide two sets of useful results:

1. *Updated block elements of the prearray*:
   - The updated square root $\mathbf{K}^{-H/2}(n)$ is given by $\mathbf{B}_{11}^{H}(n)$.
   - The updated matrix product $\hat{\mathbf{x}}^{H}(n+1 \mid \mathcal{Y}_n)\mathbf{K}^{-H/2}(n)$ is given by $\mathbf{b}_{21}^{H}(n)$.
2. *Other Kalman variables*:
   - The conversion factor, $r^{-1}(n)$, is obtained by squaring $b_{32}(n)$, which is real.
   - The innovation, $\alpha(n)$, is obtained by dividing $b_{22}(n)$ by $b_{32}(n)$.

The updated value of the state estimate $\hat{\mathbf{x}}(n+1 \mid \mathcal{Y}_n)$ is computed from the upper triangular system of equation (14.34), where $\mathbf{K}^{-H/2}(n)$ is known by virtue of Eq. (14.33). Specifically, the individual elements of $\hat{\mathbf{x}}(n+1 \mid \mathcal{Y}_n)$ are computed by using the method of *back substitution* that exploits the upper triangular structure of the square root $\mathbf{K}^{-H/2}(n)$.

Table 14.1 presents a summary of the square-root information filtering algorithm; initialization of the algorithm proceeds in the usual way.

Extended Square-root Information Filter

The need for back-substitution, required in the square-root information filtering algorithm for computing the elements of the updated state estimate $\hat{\mathbf{x}}(n+1 \mid \mathcal{Y}_n)$, may be avoided by "expanding" the prearray of Eq. (14.38) as follows (Sayed and Kailath, 1994):

$$\begin{bmatrix} \lambda^{1/2}\mathbf{K}^{-H/2}(n-1) & \lambda^{1/2}\mathbf{u}(n) \\ \hat{\mathbf{x}}^{H}(n \mid \mathcal{Y}_{n-1})\mathbf{K}^{-H/2}(n-1) & y^{*}(n) \\ \mathbf{0}^{T} & 1 \\ \lambda^{-1/2}\mathbf{K}^{1/2}(n-1) & \mathbf{0} \end{bmatrix}$$

The last block line of this prearray is borrowed from the square-root Kalman filtering algorithm; see the last line of the prearray in Eq. (14.18). Then, following a procedure similar to that described for the conventional square-root information filtering algorithm, we may show that (see Problem 3):

$$\begin{bmatrix} \lambda^{1/2}\mathbf{K}^{-H/2}(n-1) & \lambda^{1/2}\mathbf{u}(n) \\ \hat{\mathbf{x}}^{H}(n \mid \mathcal{Y}_{n-1})\mathbf{K}^{-H/2}(n-1) & y^{*}(n) \\ \mathbf{0}^{T} & 1 \\ \lambda^{-1/2}\mathbf{K}^{1/2}(n-1) & \mathbf{0} \end{bmatrix} \boldsymbol{\Theta}(n)$$

$$= \begin{bmatrix} \mathbf{K}^{-H/2}(n) & \mathbf{0} \\ \hat{\mathbf{x}}^{H}(n+1 \mid \mathcal{Y}_n)\mathbf{K}^{-H/2}(n) & r^{-1/2}(n)\alpha^{*}(n) \\ \lambda^{1/2}\mathbf{u}^{H}(n)\mathbf{K}^{1/2}(n) & r^{-1/2}(n) \\ \mathbf{K}^{1/2}(n) & -\mathbf{g}(n)r^{1/2}(n) \end{bmatrix} \tag{14.39}$$

where, as before, $\Theta(n)$ is a unitary rotation that produces a block zero entry in the top block row of the postarray.

Now we can see the benefit of using the extended prearray, as described here. Specifically, the updated value of the state estimate may be computed as follows:

$$\begin{aligned}\hat{\mathbf{x}}(n+1 \mid \mathcal{Y}_n) &= \lambda^{-1/2}\hat{\mathbf{x}}(n \mid \mathcal{Y}_{n-1}) + \mathbf{g}(n)\alpha(n) \\ &= \lambda^{-1/2}\hat{\mathbf{x}}(n \mid \mathcal{Y}_{n-1}) + (\mathbf{g}(n)r^{1/2}(n))(r^{-1/2}(n)\alpha^*(n))^* \\ &= (\mathbf{K}^{1/2}(n))\,(\mathbf{K}^{-1/2}(n)\hat{\mathbf{x}}(n+1 \mid \mathcal{Y}_n))\end{aligned} \tag{14.40}$$

where the quantities $\{\mathbf{g}(n)r^{1/2}(n),\ r^{-1/2}(n)\alpha^*(n)\}$ and $\{\mathbf{K}^{1/2}(n),\ \mathbf{K}^{-1/2}(n)\hat{\mathbf{x}}(n+1|\mathcal{Y}_n)\}$ are read directly from the postarray in Eq. (14.39). The last two lines of Eq. (14.40) provide two different but equivalent methods for computing the updated state estimate. Thus the cumbersome operation of back substitution is replaced by a simple multiplication.

A summary of the *extended square-root information filtering algorithm* is presented in Table 14.1; initialization of the algorithm proceeds in the usual manner.

The square-root covariance filter, the conventional square-root information filter, and the extended square-root information filter share a common feature. The number of operations (multiplications and additions) needed to proceed from one iteration of the algorithm to the next, in all three cases, is $O(M^2)$, where M is the state dimension. This computational complexity is of the same order as that of the conventional Riccati-based Kalman filtering algorithm.

14.2 BUILDING SQUARE-ROOT ADAPTIVE FILTERING ALGORITHMS ON THEIR KALMAN FILTER COUNTERPARTS

The square-root variants of the Kalman filter described in the previous section provide the general framework for the derivation of known square-root adaptive filtering algorithms for exponentially weighted recursive least-squares (RLS) estimation. We say so in light of the one-to-one correspondences that exist between the Kalman variables and RLS variables, as demonstrated in Chapter 13.

The square-root adaptive filtering algorithms for RLS estimation are known as the *QR–RLS algorithm, extended QR–RLS algorithm*, and *inverse QR–RLS algorithm*. The reason for this terminology is that the derivation of these algorithms has traditionally relied, in one form or another, on the use of an orthogonal triangularization process known in matrix algebra as the *QR decomposition*. The motivation for using the QR decomposition in adaptive filtering is to exploit its good numerical properties.

For a matrix $\mathbf{A}(n)$, say, the QR decomposition may be written as follows (Stewart, 1973; Golub and Van Loan, 1989):

$$\mathbf{Q}(n)\mathbf{A}(n) = \begin{bmatrix}\mathbf{R}(n) \\ \mathbf{O}\end{bmatrix} \tag{14.41}$$

where $\mathbf{Q}(n)$ is a unitary matrix, $\mathbf{R}(n)$ is an upper triangular matrix, and $\mathbf{O}$ is the null matrix. The pervasive use of the symbols $\mathbf{Q}$ and $\mathbf{R}$ in such a transformation has led, in the course of time, to the common use of "QR decomposition." By the same token, adaptive RLS filtering algorithms based on the QR decomposition, in a broad sense, became known as "QR–RLS algorithms." Prior to the 1994 paper by Sayed and Kailath, the QR–RLS algorithms for exponentially weighted RLS estimation were derived starting from the prewindowed version of a data matrix, which was then triangularized by applying the QR decomposition. The paper by Sayed and Kailath revealed for the first time how these different adaptive filtering algorithms can indeed be deduced directly from their square-root Kalman filter counterparts, thereby achieving two highly desirable objectives:

- The unified treatment of QR–RLS adaptive filtering algorithms for exponentially weighted RLS estimation
- Consolidating the bridge between the deterministic RLS estimation theory and the stochastic Kalman filter theory

In the remainder of this chapter, we follow the paper by Sayed and Kailath in deriving the different QR–RLS adaptive filtering algorithms. However, the order in which these algorithms are considered follows the traditional development of RLS estimation theory rather than the order of square-root Kalman filters summarized in Table 14.1.

14.3 QR–RLS ALGORITHM

The *QR–RLS algorithm*, or more precisely, the *QR decomposition-based RLS algorithm*, derives its name from the fact that the computation of the least-squares weight vector in a finite-duration impulse response (FIR) filter implementation of the adaptive filtering algorithm is accomplished by working directly with the incoming data matrix via the QR decomposition rather than working with the (time-averaged) correlation matrix of the input data as in the standard RLS algorithm (Gentleman and Kung, 1981; McWhirter, 1983; Haykin, 1991). Accordingly, the QR–RLS algorithm is numerically more stable than the standard RLS algorithm.

Assuming the use of prewindowing on the input data, the *data matrix* is defined by

$$\mathbf{A}^H(n) = [\mathbf{u}(1), \mathbf{u}(2), \ldots, \mathbf{u}(M), \ldots, \mathbf{u}(n)]$$

$$= \begin{bmatrix} u(1) & u(2) & \cdots & u(M) & \cdots & u(n) \\ 0 & u(1) & \cdots & u(M-1) & \cdots & u(n-1) \\ \cdot & \cdot & & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & & \cdot \\ 0 & 0 & \cdots & u(1) & \cdots & u(n-M+1) \end{bmatrix} \tag{14.42}$$

where M is the number of FIR filter coefficients. Correspondingly, the correlation matrix of the input data is defined by

$$\begin{aligned}\boldsymbol{\Phi}(n) &= \sum_{i=1}^{n} \lambda^{n-i}\mathbf{u}(i)\mathbf{u}^H(i) \\ &= \mathbf{A}^H(n)\boldsymbol{\Lambda}(n)\mathbf{A}(n)\end{aligned} \tag{14.43}$$

The matrix $\boldsymbol{\Lambda}(n)$ is called the *exponential weighting matrix*, defined by

$$\boldsymbol{\Lambda}(n) = \text{diag}[\lambda^{n-1}, \lambda^{n-2}, \ldots, 1] \tag{14.44}$$

where λ is the *exponential weighting factor*. Equation (14.43) represents a generalization of Eq. (10.45) used in the method of least squares.

From Chapter 13 we recall that the matrix $\mathbf{P}(n)$, used in deriving the RLS algorithm, is defined as the inverse of the correlation matrix $\boldsymbol{\Phi}(n)$, as shown by [see Eq. (13.17)]

$$\mathbf{P}(n) = \boldsymbol{\Phi}^{-1}(n) \tag{14.45}$$

From Chapter 13, we also note the following correspondences between the Kalman variables and RLS variables:

KALMAN VARIABLE	RLS VARIABLE	DESCRIPTION
$\mathbf{K}^{-1}(n)$	$\lambda\mathbf{P}^{-1}(n) = \lambda\boldsymbol{\Phi}(n)$	Correlation matrix
$r^{-1}(n)$	$\gamma(n)$	Conversion factor
$\mathbf{g}(n)$	$\lambda^{-1/2}\mathbf{k}(n)$	Gain vector
$\alpha(n)$	$\lambda^{-n/2}\,\xi^*(n)$	*A priori* estimation error
$y(n)$	$\lambda^{-n/2}\,d^*(n)$	Desired response
$\hat{\mathbf{x}}(n \mid \mathcal{Y}_{n-1})$	$\lambda^{-n/2}\hat{\mathbf{w}}(n-1)$	Estimate of tap-weight vector

From the first line of this table, we immediately see that the QR–RLS algorithm corresponds to the square-root information filtering algorithm (14.38) of Kalman filter theory.

Before proceeding to formulate the QR–RLS algorithm in light of this correspondence, we find it convenient to make a change of notation. According to the normal equations, the least squares estimate of the tap-weight vector, $\hat{\mathbf{w}}(n)$, is defined by [see Eq. (10.35)]

$$\boldsymbol{\Phi}(n)\hat{\mathbf{w}}(n) = \mathbf{z}(n) \tag{14.46}$$

where $\mathbf{z}(n)$ is the cross-correlation vector between the desired response $d(n)$ and input data vector $\mathbf{u}(n)$. Let $\boldsymbol{\Phi}(n)$ be expressed in its factored form:

$$\boldsymbol{\Phi}(n) = \boldsymbol{\Phi}^{1/2}(n)\,\boldsymbol{\Phi}^{H/2}(n) \tag{14.47}$$

Then, premultiplying both sides of Eq. (14.46) by the square root $\boldsymbol{\Phi}^{-1/2}(n)$, we may introduce a new vector variable defined by

$$\mathbf{p}(n) = \boldsymbol{\Phi}^{H/2}(n)\,\hat{\mathbf{w}}(n) = \boldsymbol{\Phi}^{-1/2}(n)\,\mathbf{z}(n) \tag{14.48}$$

We are now ready to formulate the QR–RLS algorithm for linear adaptive filtering. Specifically, we may translate Eq. (14.38) pertaining to the square-root information filtering algorithm into the corresponding prearray-to-postarray transformation for the QR–RLS algorithm as follows (after cancelation of common terms):

$$\begin{bmatrix} \lambda^{1/2}\boldsymbol{\Phi}^{1/2}(n-1) & \mathbf{u}(n) \\ \lambda^{1/2}\mathbf{p}^H(n-1) & d(n) \\ \mathbf{0}^T & 1 \end{bmatrix} \boldsymbol{\Theta}(n) = \begin{bmatrix} \boldsymbol{\Phi}^{1/2}(n) & \mathbf{0} \\ \mathbf{p}^H(n) & \xi(n)\,\gamma^{1/2}(n) \\ \mathbf{u}^H(n)\boldsymbol{\Phi}^{-H/2}(n) & \gamma^{1/2}(n) \end{bmatrix} \tag{14.49}$$

Basically $\boldsymbol{\Theta}(n)$ is any unitary rotation that operates on the elements of the input data vector $\mathbf{u}(n)$ in the prearray, *annihilating* them one by one so as to produce a block zero entry in the top block row of the postarray. Naturally, the lower triangular structure of the square root of the correlation matrix, namely, $\boldsymbol{\Phi}^{1/2}$, is preserved in its exact form before and after the transformation. This is indeed the very essence of the QR decomposition for RLS estimation, hence the name "QR–RLS algorithm."

Having computed the updated block values $\boldsymbol{\Phi}^{1/2}(n)$ and $\mathbf{p}^H(n)$, we may then solve for the least-squares weight vector $\hat{\mathbf{w}}(n)$ by using the formula [see Eq. (14.48)]

$$\hat{\mathbf{w}}^H(n) = \mathbf{p}^H(n)\,\boldsymbol{\Phi}^{-1/2}(n) \tag{14.50}$$

The computation of this solution is accomplished using the *method of back substitution* that exploits the lower triangular structure of $\boldsymbol{\Phi}^{1/2}(n)$. Note, however, that this computation is feasible only for time $n > M$, for which the data matrix $\mathbf{A}(n)$, and therefore $\boldsymbol{\Phi}^{1/2}(n)$, is of full column rank.

To initialize the QR–RLS algorithm, we may set $\boldsymbol{\Phi}^{1/2}(0) = \mathbf{O}$ and $\mathbf{p}(0) = \mathbf{0}$. The *exact initialization* of the QR–RLS algorithm occupies the period $\mathbf{O} \le n \le M$ for which the *a posteriori* estimation error $e(n)$ is zero. At iteration $n = M$, the initialization is completed, whereafter $e(n)$ may assume a nonzero value.

A summary of the QR–RLS algorithm is presented in Table 14.2, including details of the initialization and other matters of interest.

Implementation Considerations

Thus far we have not focused on the particulars of the unitary rotation $\boldsymbol{\Theta}(n)$, other than to require that it be chosen to produce a block zero entry in the top block row of the postarray. A unitary matrix that befits this requirement is the transformation based on the *Givens rotation* discussed in Chapter 12. Through successive applications of the Givens rotation, we may develop a systematic procedure for the efficient annihilation of the block entry $\mathbf{u}(n)$ in the prearray, as prescribed by Eq. (14.49).

Moreover, the use of Givens rotations lends itself to a *parallel implementation* in the form of a *systolic array*, the idea of which was developed originally by Kung and Leiserson (1978). A systolic array consists of an array of individual *processing cells* arranged as a regular structure. Each cell in the array is provided with local memory of its own, and each cell is connected only to its nearest neighbors. The array is designed such that regular streams of data are clocked through it in a highly rhythmic fashion, much like the

TABLE 14.2 SUMMARY OF THE QR–RLS ALGORITHM AND ITS EXTENDED FORM FOR EXPONENTIALLY WEIGHTED RLS ESTIMATION

Inputs:

input signal vector = $\{\mathbf{u}(1), \mathbf{u}(2), \ldots, \mathbf{u}(n)\}$

desired response = $\{d(1)\, d(2), \ldots, d(n)\}$

Known parameter:

exponential weighting factor = λ

Initial conditions:

$\boldsymbol{\Phi}^{1/2}(0) = \mathbf{O}$

$\mathbf{p}(0) = \mathbf{0}$

1. *QR–RLS algorithm*:

For $n = 1, 2, \ldots$, compute

$$\begin{bmatrix} \lambda^{1/2}\boldsymbol{\Phi}^{1/2}(n-1) & \mathbf{u}(n) \\ \lambda^{1/2}\mathbf{p}^H(n-1) & d(n) \\ \mathbf{0}^T & 1 \end{bmatrix} \boldsymbol{\Theta}(n) = \begin{bmatrix} \boldsymbol{\Phi}^{1/2}(n) & \mathbf{0} \\ \mathbf{p}^H(n) & \xi(n)\gamma^{1/2}(n) \\ \mathbf{u}^H(n)\boldsymbol{\Phi}^{-H/2}(n) & \gamma^{1/2}(n) \end{bmatrix}$$

$$\hat{\mathbf{w}}^H(n) = \mathbf{p}^H(n)\, \boldsymbol{\Phi}^{-1/2}(n)$$

2. *Extended QR–RLS algorithm*:

For $n = 1, 2, \ldots$, compute

$$\begin{bmatrix} \lambda^{1/2}\boldsymbol{\Phi}^{1/2}(n-1) & \mathbf{u}(n) \\ \lambda^{1/2}\mathbf{p}^H(n-1) & d(n) \\ \mathbf{0}^T & 1 \\ \lambda^{-1/2}\boldsymbol{\Phi}^{-H/2}(n-1) & \mathbf{0} \end{bmatrix} \boldsymbol{\Theta}(n) = \begin{bmatrix} \boldsymbol{\Phi}^{1/2}(n) & \mathbf{0} \\ \mathbf{p}^H(n) & \xi(n)\gamma^{1/2}(n) \\ \mathbf{u}^H(n)\boldsymbol{\Phi}^{-H/2}(n) & \gamma^{1/2}(n) \\ \boldsymbol{\Phi}^{-H/2}(n) & -\mathbf{k}(n)\gamma^{-1/2}(n) \end{bmatrix}$$

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + (\mathbf{k}(n)\, \gamma^{-1/2}(n))(\xi(n)\, \gamma^{-1/2}(n))^*$$

Note: In both cases, $\boldsymbol{\Theta}(n)$ is a unitary rotation that operates on the prearray to produce a block zero entry in the top block row of the postarray.

pumping action of the human heart, hence the name "systolic" (Kung, 1982). The important point to note here is that systolic arrays are well suited for implementing complex signal processing algorithms such as the QR–RLS algorithm, particularly when the requirement is to operate in *real time* and at *high data bandwidths*.

Turning to the QR–RLS algorithm as described herein, we may identify two systolic implementations of it, referred to as implementation I and implementation II. Basically, these two implementations differ from each other in their specific computation products.

Systolic Array Implementation I

Figure 14.1 shows a systolic array structure for implementing a simplified form of the QR–RLS algorithm for the case when the weight vector $\hat{\mathbf{w}}(n)$ has three elements (i.e., $M = 3$). The simplification merely involves deleting the last rows of the prearray and the

postarray in Eq. (14.49), with a corresponding reduction in the dimensions of the unitary matrix $\mathbf{\Theta}(n)$. Specifically, we now have[1]

$$\begin{bmatrix} \lambda^{1/2}\mathbf{\Phi}^{1/2}(n-1) & \mathbf{u}(n) \\ \lambda^{1/2}\mathbf{p}^H(n-1) & d(n) \end{bmatrix} \mathbf{\Theta}(n) = \begin{bmatrix} \mathbf{\Phi}^{1/2}(n) & \mathbf{0} \\ \mathbf{p}^H(n) & \xi(n)\gamma^{1/2}(n) \end{bmatrix} \tag{14.51}$$

The systolic structure of Fig. 14.1 is arranged with two specific points in mind. First, data flow through it from left to right, consistent with all other adaptive filters considered in previous chapters. Second, the systolic array operates directly on the input data that are represented by successive values of the input data vector $\mathbf{u}(n)$ and the desired response $d(n)$.

The structure of Fig. 14.1 consists of two distinct sections: a *triangular systolic array* and a *linear systolic array* (Gentleman and Kung, 1981). The entire systolic array is controlled by a single *clock*. Each section of the array consists of two types of processing cells: *internal cells* (squares) and *boundary cells* (circles). The specific arithmetic functions of these cells are defined later. Each cell receives its input data from the directions indicated for one clock cycle, performs the specific arithmetic functions, and then, on the next clock cycle, delivers the resulting output values to neighboring cells as indicated. A distinctive feature of systolic arrays is that each processing cell is always kept active as data flow across the array. The triangular systolic array section implements the Givens rotations part of the QR–RLS algorithm, whereas the linear systolic array section computes the weight vector *at the end of the entire recursion*. If we were to compute the weight vector at each iteration of the QR–RLS algorithm, which we indeed can, the operation of the systolic array processor in Fig. 14.1 would be prohibitively slow, hence the idea of deferring the computation of $\hat{\mathbf{w}}(n)$ to the end of the recursion.

The dashed squares shown in Fig. 14.1 are merely included to represent delays in the transfer of data from the triangular array to the linear section. These delays are needed to ensure that the data transfer takes place at the correct instants of time.

Consider first the operation of the triangular systolic array section labeled *ABC* in Fig. 14.1. The boundary and internal cells of this section are given in Fig. 14.2. Basically,

[1] The $(M+1)$-by-$(M+1)$ unitary matrix $\mathbf{\Theta}$ in Eq. (14.51) for the QR–RLS algorithm is implemented as a sequence of M Givens rotations, each of which is configured to annihilate a particular element of the M-by-1 vector $\mathbf{u}(n)$ in the prearray. We may thus write

$$\mathbf{\Theta} = \prod_{k=1}^{M} \mathbf{\Theta}_k$$

where $\mathbf{\Theta}_k$ consists of a unitary matrix except for four strategic elements located at the points where the pair of rows k and $M+1$ intersects the pair of columns k and $M+1$. These four elements, denoted by θ_{kk}, $\theta_{M+1,k}$, $\theta_{k,M+1}$, and $\theta_{M+1,M+1}$ are defined as follows:

$$\theta_{kk} = \theta_{M+1,M+1} = c_k$$

$$\theta_{M+1,k} = s_k^*$$

$$\theta_{k,M+1} = -s_k$$

where $k = 1, 2, \ldots, M$. The cosine parameter c_k is real, whereas the sine parameter s_k is complex. The remarks made in this footnote also apply to the unitary matrix $\mathbf{\Theta}$ in Eq. (14.51) for the extended QR–RLS algorithm.

The design equations for the cells in the triangular array in Fig. 14.2 for the QR–RLS algorithm and the corresponding ones in Fig. 14.9 for the extended QR–RLS algorithm are based on this particular description of the unitary matrix $\mathbf{\Theta}$.

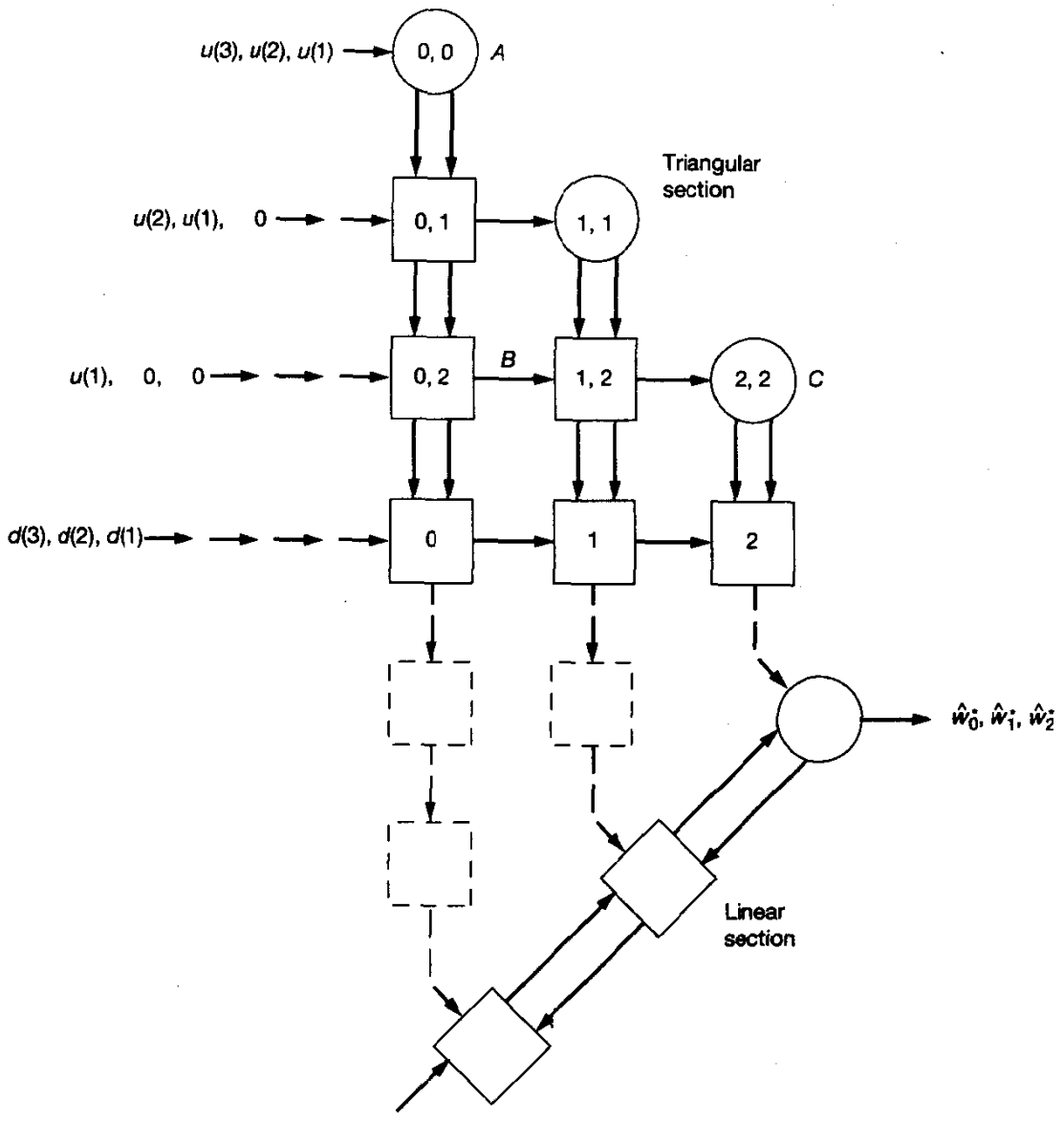


Figure 14.1 Systolic array implementation I of the QR-RLS algorithm for $M = 3$. Note: The dashed squares represent delays in the transfer of computations from the triangular section to the linear section.

the internal cells perform only additions and multiplications, as described in Fig. 14.2(b). The boundary cells, on the other hand, are considerably more complex, in that they compute square roots and reciprocals, as described in Fig. 14.2(a). Each cell of the triangular systolic array section (depending on its location) stores a particular element of the lower triangular matrix $\mathbf{\Phi}^{1/2}(n)$, which, at the outset of the least-squares recursion, is initialized to zero and thereafter updated every clock cycle. The function of each column of processing cells in the triangular systolic array section is to *rotate* one column of the stored triangular matrix with a vector of data received from the left in such a way that the leading *ele-*

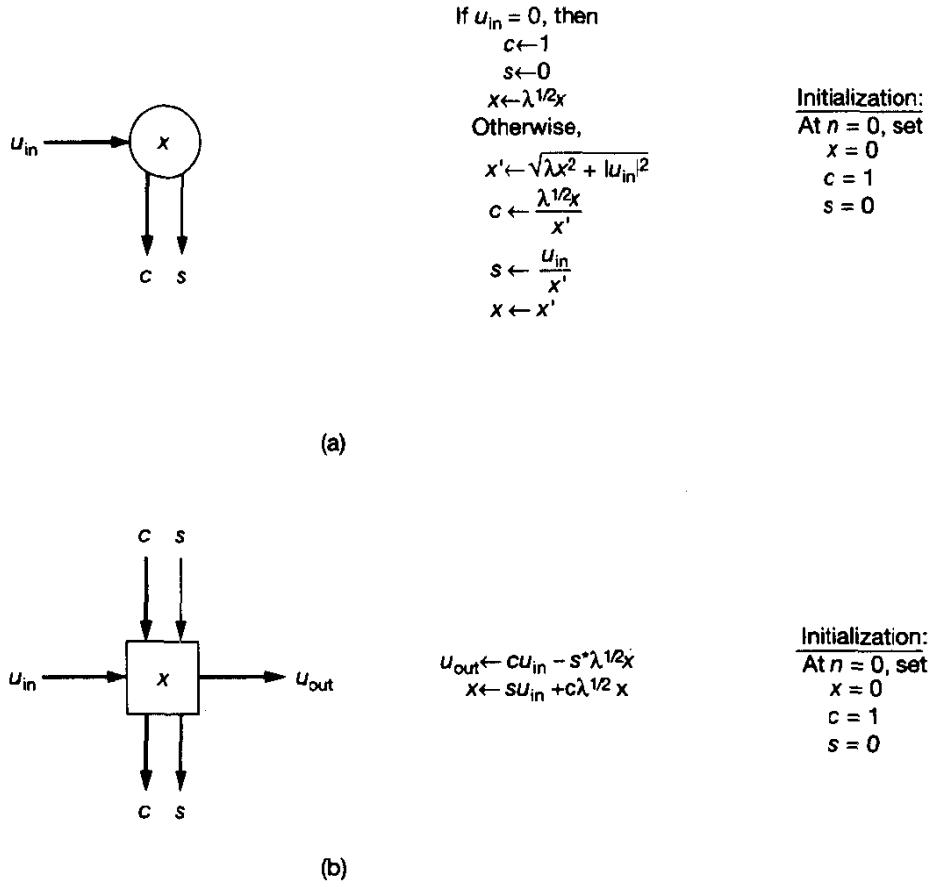


Figure 14.2 Cells for systolic array I: (a) boundary cell; (b) internal cell. *Note*: The stored value x is initialized to be zero (i.e., real). For the boundary cell, it always remains *real*; this is consistent with the property that the diagonal elements of the upper triangular matrix $\mathbf{R}$ are all real. Hence, the formulas for the rotation parameters c and s computed by the boundary cell can be simplified considerably, as shown in part (a). Also, note that the values x stored in the array are elements of the lower triangular matrix $\mathbf{R}^H$; hence, we may identify $r^* = x$ for all elements of the triangular array.

ment of the received data vector is annihilated. The reduced data vector is then passed to the right on to the next column of cells. The boundary cell in each column of the section computes the pertinent rotation parameters and then passes them downward on the next clock cycle. The internal cells subsequently apply the same rotation to all other elements in the received data vector. Since a delay of one clock cycle per cell is incurred in passing the rotation parameters downward along a column, it is necessary that the input data vectors enter the triangular systolic array in a *skewed order*, as illustrated in Fig. 14.1 for the case of $M = 3$. This arrangement of the input data ensures that as each column vector $\mathbf{u}(n)$ of the data matrix $\mathbf{A}^H(n)$ propagates through the array, it interacts with the previously stored triangular matrix $\mathbf{\Phi}^{1/2}(n-1)$ and thereby undergoes the sequence of Givens rotations denoted by $\mathbf{\Theta}(n)$, as required. Accordingly, all the elements of the column vector $\mathbf{u}(n)$

are annihilated, one by one, and an updated lower triangular matrix $\mathbf{\Phi}^{1/2}(n)$ is produced and stored in the process.

The systolic array operates in a highly pipelined manner, whereby, as (time-skewed) input data vectors enter the array from the left, we find that in effect each such vector defines a processing *wavefront* that moves across the array. It should therefore be appreciated that, on any particular clock cycle, elements of the pertinent lower triangular matrix $\mathbf{\Phi}^{1/2}(n)$ only exist along the corresponding wave-front. This phenomenon is illustrated later in Example 1.

At the same time that the orthogonal triangularization process is being performed by the triangular systolic array section labeled *ABC* in Fig. 14.1, the row vector $\mathbf{p}^H(n)$ is computed by the appended bottom row of internal cells.

When the entire orthogonal triangularization process is completed, the data flow *stops*, and then the stored data can be *clocked out* for subsequent processing by the linear systolic array section. The dashed lines in Fig. 14.3 depict the clock-out paths for the final values of the elements of both $\mathbf{\Phi}^{1/2}(n)$ and $\mathbf{p}^H(n)$ into the linear section of the systolic processor.

The linear section of the processor computes the Hermitian transposed least-squares weight vector, namely $\hat{\mathbf{w}}^H(n)$. For convenience of presentation, let

$$\mathbf{R}^H(n) = \mathbf{\Phi}^{1/2}(n) \tag{14.52}$$

Then in accordance with Eq. (14.49), the elements of the vector $\hat{\mathbf{w}}^H(n)$ are computed by using the method of back substitution (Kung and Leiserson, 1978). Taking the Hermitian transpose of both sides of Eq. (14.52), we have

$$\mathbf{\Phi}^{H/2}(n) = \mathbf{R}(n) \tag{14.53}$$

We may then write

$$\begin{aligned} z_i^{(M-1)} &= 0 \\ z_i^{(k-1)} &= z_i^{(k)} + r_{ik}^*(n)\hat{w}_k^*(n), \qquad k = M-1, \ldots, i; \quad i = M-1, \ldots, 0 \\ \hat{w}_i^*(n) &= \frac{p_i^*(n) - z_i^{(i)}}{r_{ii}(n)} \end{aligned} \tag{14.54}$$

where the $z_i^{(k)}$ are intermediate variables, the $r_{ik}(n)$ are elements of upper triangular matrix $\mathbf{R}(n)$, the $p_i(n)$ are elements of the vector $\mathbf{p}(n)$, and the $\hat{w}_k(n)$ are elements of the weight vector $\hat{\mathbf{w}}(n)$. The linear systolic array section consists of one boundary cell and $(M-1)$ internal cells that perform the arithmetic functions defined in Fig. 14.4, in accordance with Eq. (14.54). The boundary cell performs subtraction and division, whereas the internal cells perform additions and multiplications. The elements of the complex-conjugated weight vector leave the linear array every second clock cycle with $\hat{w}_{M-1}^*(n)$ leaving first, followed by $\hat{w}_{M-2}{}^*(n)$, and so on right up to $\hat{w}_0{}^*(n)$. In effect, the elements of the weight vector $\hat{\mathbf{w}}^H(n)$ are read out *backward*. Thus, by chaining the linear and triangular systolic array sections together in the manner shown in Fig. 14.1, we produce a device capable of solving the exact least-squares problem recursively.

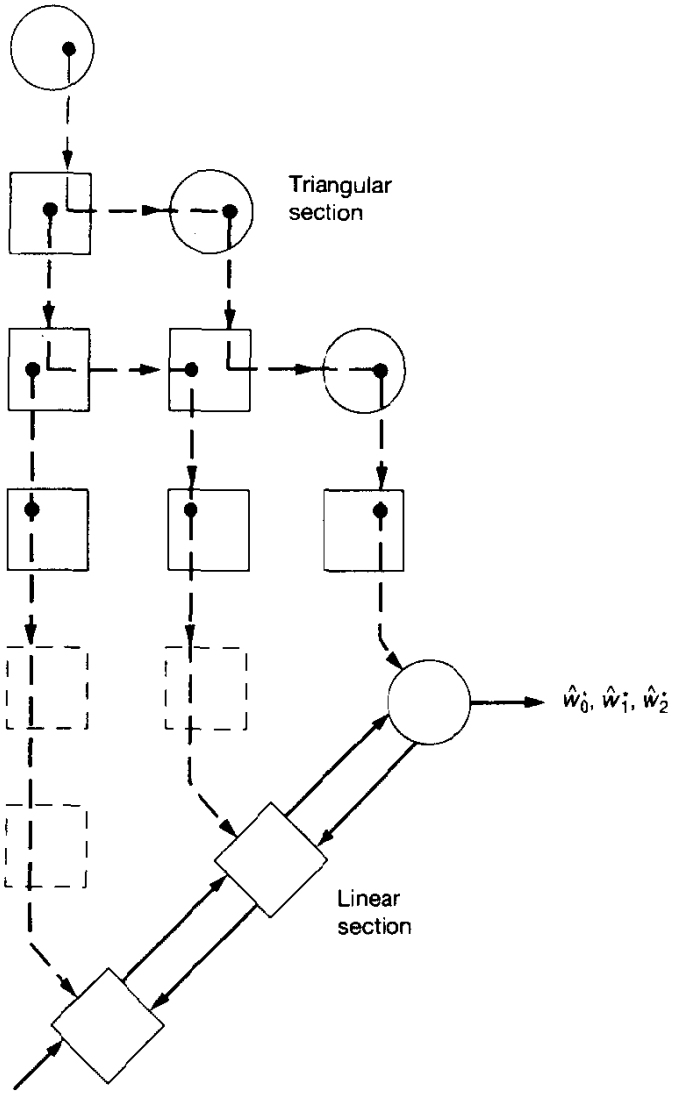


Figure 14.3 Clock-out paths of the triangular section into the linear section of the systolic array. The dots represent cell contents.

The following points are noteworthy in the context of the two-section systolic processor of Fig. 14.1 for the general case of *complex-valued data*:

1. Initially, zeros are stored in all boundary and internal cells. Also, the parameters of the Givens rotation at the output of each boundary cell (and therefore every other cell in the triangular systolic array section) are initially set at the values $c_{\text{out}} = 1$ and $s_{\text{out}} = 0$. The initialization of the complete systolic array shown in Fig. 14.1 occupies a total of $3M$ *clock cycles*, where M is the dimension of the weight vector.

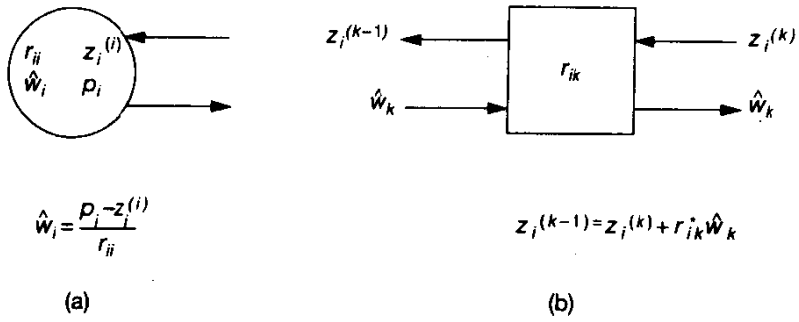


Figure 14.4 Cells for linear systolic array: (a) boundary cell; (b) internal cell.

2. The element $r_{ik}^*(n)$ of the lower triangular matrix $\mathbf{R}^H(n) = \mathbf{\Phi}^{1/2}(n)$ is computed in the kth cell on the ith row of the triangular section at time $n + (i - 1) + (k - 1)$, where $i, k = 0, 1, \ldots, M - 1$; note that the diagonal elements of $\mathbf{R}^H(n)$ are all *real valued*. The element $p_k^*(n)$ of the row vector $\mathbf{p}^H(n)$ is computed in the kth cell of the row of cells appended to the triangular section at time $n + M + k$, where $k = 0, 1, \ldots, M - 1$.
3. The systolic processor of Fig. 14.1 experiences *delays* in the various stages of computation. In particular, for the general case of an M-by-1 weight vector, $2M$ clock cycles are required to compute the Givens rotations, and another $2M$ clock cycles are required to clock out and compute the weight vector. Accordingly, a *total delay* or *latency of* $4M$ *clock cycles* is experienced in computing the complete M-by-1 weight vector $\hat{\mathbf{w}}(n)$. For real-time operations requiring the use of a large M, this latency may be too large and therefore unacceptable.
4. The element values of the matrix $\mathbf{R}^H(n)$ do not all propagate from the triangular section straight down to the linear section. Rather, except for $r_{i,M-1}^*(n)$, $i = 0, 1, \ldots, M - 1$, the propagation of all the other elements of $\mathbf{R}^H(n)$ follows zig-zag paths, as illustrated in Fig. 14.3 for the case of $M = 3$.
5. The linear systolic array section has a lower data throughput than the triangular systolic array section. Consequently, the elements of the least-squares weight vector $\hat{\mathbf{w}}(n)$ are computed serially only when the entire sequence of computations in the triangular section comes to an end, largely for the sake of ensuring computational efficiency. It is, of course, a straightforward matter to modify these two array structures to make "on-the-fly" computation of $\hat{\mathbf{w}}(n)$ possible by the addition of extra data paths at the cost of additional computations; Varvisiotis et al. (1989) describe a scheme for parallel implementation of the linear section.

Example 1

In this example, we illustrate the operation of the systolic array structure of Fig. 14.1 for the case when the input data are *real valued* and $M = 3$.

TABLE 14.3 INPUTS AND STATES OF THE TRIANGULAR SYSTOLIC ARRAY FOR $M = 3$ AND REAL DATA

States at time n_-			Inputs at time n	States at time n_+		
$r_{00}(n-1)$			$u(n)$	$r_{00}(n)$		
$r_{01}(n-2)$	$r_{11}(n-3)$		$u(n-2)$	$r_{01}(n-1)$	$r_{11}(n-2)$	
$r_{02}(n-3)$	$r_{12}(n-4)$	$r_{22}(n-5)$	$u(n-4)$	$r_{02}(n-2)$	$r_{12}(n-3)$	$r_{22}(n-4)$
$p_0(n-4)$	$p_1(n-5)$	$p_2(n-6)$	$d(n-3)$	$p_0(n-3)$	$p_1(n-4)$	$p_2(n-5)$

Note: The initialization procedure can also be represented by this stage graph, provided that we set $u(k) = 0$, $r_{ij}(k) = 0$, and $p_i(k) = 0$, when $k < 1$.

The inputs and states of the triangular systolic array for $M = 3$ are summarized in Table 14.3. In particular, we show the states of the individual cells in this section at time n_-, the external inputs applied to them at time n, and the states of the individual cells at time n_+.

The elements of column 1 of the lower triangular 3-by-3 matrix $\mathbf{\Phi}^{1/2}(n) = \mathbf{R}^T(n)$ for $M = 3$, that is, the elements $r_{00}(n)$, $r_{01}(n)$, and $r_{02}(n)$, are computed in the cells of column 1 of the triangular section at times n, $n + 1$, and $n + 2$, respectively. The elements of column 2 of $\mathbf{R}^T(n)$, that is, the elements $r_{11}(n)$ and $r_{12}(n)$, are computed in the cells of column 2 of the triangular section at times $n + 2$ and $n + 3$, respectively. The remaining element of $\mathbf{R}^T(n)$, that is, $r_{33}(n)$, is computed in the only cell of column 3 of the triangular section at time $n + 4$. The elements of the 1-by-3 vector $\mathbf{p}^T(n)$, that is, $p_0(n)$, $p_1(n)$, and $p_2(n)$, are computed in the elements of the row of cells appended to the triangular section at times $n + 4$, $n + 5$, and $n + 6$, respectively. When the orthogonal triangularization of the weighted data matrix is completed, the data flow is terminated, and the stored contents of the internal and boundary cells are clocked out in the manner described in Fig. 14.3. In particular, the clocked-out data are processed by a linear systolic array. For the example at hand, Fig. 14.5 presents the details of the operation of this section. The clock cycle numbers included in Fig. 14.5 are measured from the instant when the linear section begins its operation.

Systolic Array Implementation II

The triangular systolic array section shown in Fig. 14.1 may be viewed as a *partial* implementation of the transformation described in Eq. (14.49) that constitutes what the QR–RLS algorithm is all about. Figure 14.6 shows a systolic array implementation of this transformation in *full* (McWhirter, 1983). The internal cells of this structure are identical

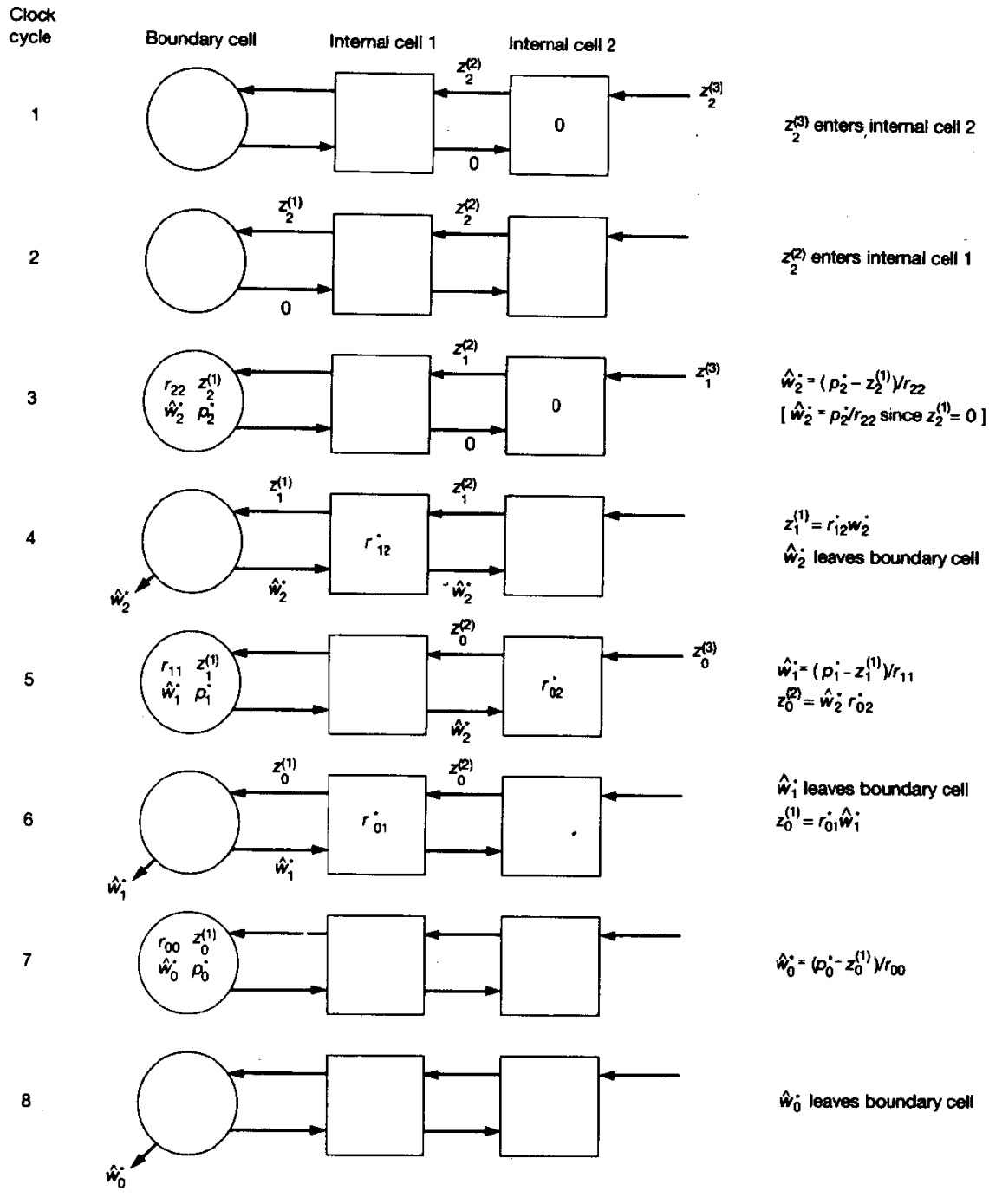


Figure 14.5 Operation of the linear systolic array for $M = 3$.

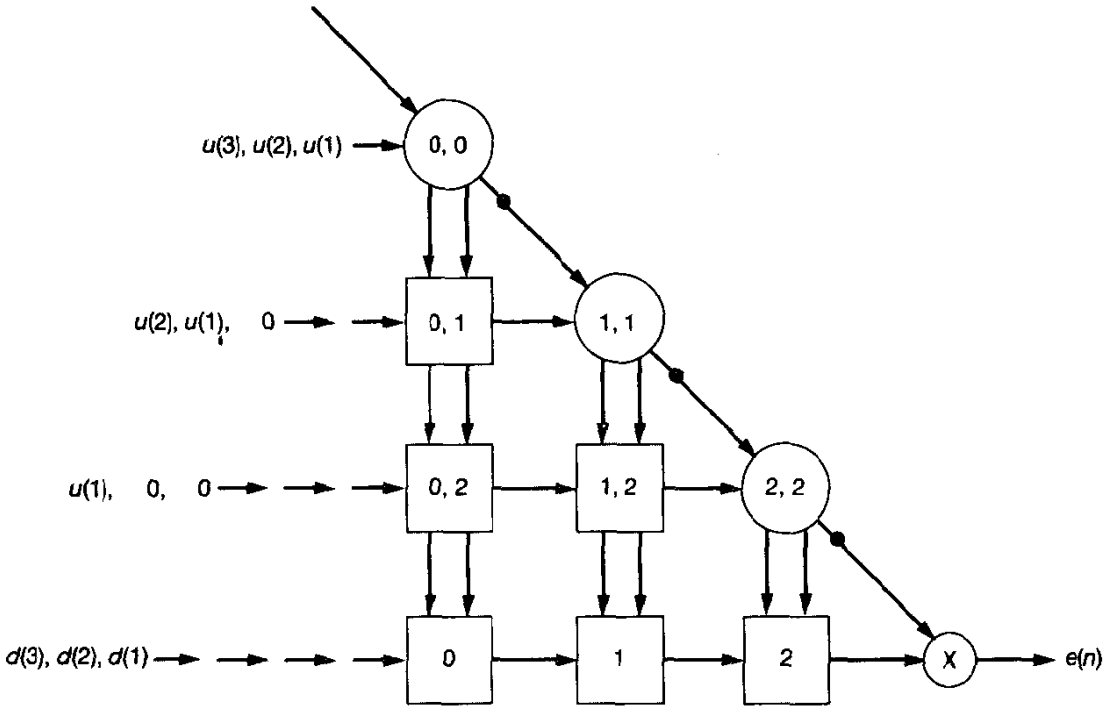


Figure 14.6 Systolic array implementation II of the QR–RLS algorithm. The dots along the diagonal of the array represent storage elements. This processing delay, which is a consequence of the temporal skew imposed on the input data, may be incorporated within the associated boundary cell.

to those in the triangular section of Fig. 14.1, but the boundary cells are now more complex, as shown in Fig. 14.7 to account for the structure of the full postarray in Eq. (14.49).

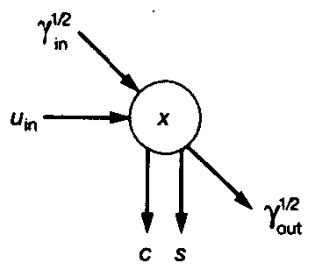
The computations of $\Phi^{1/2}(n)$ and $\mathbf{p}^H(n)$ proceed along exactly the same lines as those described for the systolic structure of Fig. 14.1.

From Fig. 14.7(a), we note that the *k*th boundary cell in the systolic array structure of Fig. 14.6 performs the computation:

$$\gamma_{\text{out},k}^{1/2} = c_k(n)\gamma_{\text{in},k}^{1/2}(n), \qquad k = 1, 2, \ldots, M \tag{14.55}$$

where $c_k(n)$ is the cosine rotation parameter of that cell. Accordingly, with a set of M boundary cells connected together as in Fig. 14.6, the output of the last boundary cell produced in response to a unit input applied to the first boundary cell, may be expressed as follows:

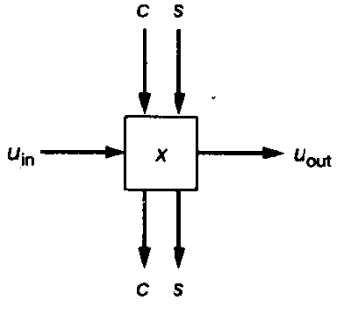
$$\begin{aligned}\gamma^{1/2}(n) &= \gamma_{\text{out},M}^{1/2}(n) \,|\gamma_{\text{in},1}^{1/2}(n) = 1 \\ &= \prod_{k=1}^{M} c_k(n)\end{aligned} \tag{14.56}$$



If $u_{in} = 0$, then
$c \leftarrow 1$,
$s \leftarrow 0$, and
$\gamma_{out}^{1/2} \leftarrow \gamma_{in}^{1/2}$
$x \leftarrow \lambda^{1/2} x$
Otherwise,
$x' \leftarrow \sqrt{\lambda x^2 + |u_{in}|^2}$
$c \leftarrow \dfrac{\lambda^{1/2} x}{x'}$
$s \leftarrow \dfrac{u_{in}}{x'}$
$x \leftarrow x'$
$\gamma_{out}^{1/2} \leftarrow c\gamma_{in}^{1/2}$

Initialization:
At $n = 0$, set
$x = 0$
$c = 1$
$s = 0$
$\gamma_{in} = 1$

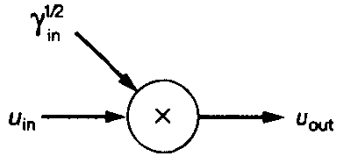
(a)



$u_{out} \leftarrow cu_{in} - s^* \lambda^{1/2} x$
$x \leftarrow su_{in} + c\lambda^{1/2} x$

Initialization:
At $n = 0$, set
$x = 0$
$c = 1$
$s = 0$

(b)



$u_{out} \leftarrow \gamma_{in}^{1/2} u_{in}$

(c)

Figure 14.7 Cells for systolic array II: (a) boundary cell; (b) internal cell; (c) final cell. *Note*: The stored valve x is initialized to be zero (i.e., real). For the boundary cell, it always remains real. Hence, the formulas for the rotation parameters c and s computed by the boundary cell can be simplified considerably, as shown in part (a). Note also that in parts (a) and (b), the values x stored in the array are elements of $\mathbf{R}^H$; hence, $r^* = x$ for all elements of the array.

In a corresponding way, the last cell in the row of internal cells appended to the triangular section produces an output equal to $\xi(n)\gamma^{1/2}(n)$.

The structure of Fig. 14.6 includes a new element referred to as the *final processing cell*, which is indicated by a small circle. This cell produces an output simply by multiplying its two inputs. Thus, with inputs equal to $\gamma^{1/2}(n)$ and $\xi(n)\gamma^{1/2}(n)$, the final processing cell in Fig. 14.6 produces an output equal to the *a posteriori* estimation error $e(n)$, in accordance with the relation [see Eq. (13.38)]

$$\begin{aligned} e(n) &= \xi(n)\gamma(n) \\ &= (\xi(n)\gamma^{1/2}(n))(\gamma^{1/2}(n)) \end{aligned} \tag{14.57}$$

As the time-skewed input data vectors enter the systolic array of Fig. 14.6, we find that updated estimation errors are produced at the output of the array at the rate of one every clock cycle. The estimation error produced on a given clock cycle corresponds, of course, to the particular element of the desired response vector $d(n)$ that entered the array M clock cycles previously.

It is noteworthy that the *a priori* estimation error $\xi(n)$ may be obtained by *dividing* the output that emerges from the last cell in the appended (bottom) row of internal cells by the output from the last boundary cell. Also, the conversion factor $\gamma(n)$ may be obtained simply by squaring the output that emerges from the last boundary cell.

Figure 14.8 summarizes, in a diagrammatic fashion, the flow of signals in the systolic array of Fig. 14.6. The figure includes the external inputs $\mathbf{u}(n)$ and $d(n)$, the resulting

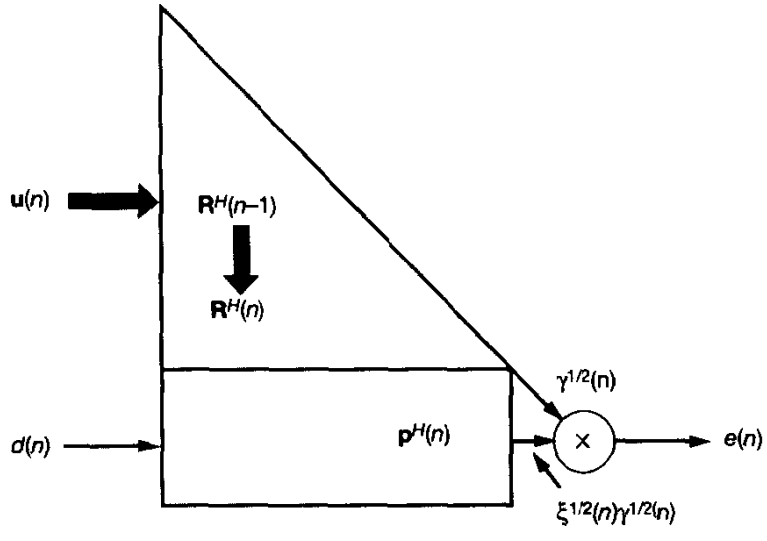


Figure 14.8 Diagrammatic representation of the flow of signals in systolic array II of Fig. 14.6.

transformations in the internal states of the triangular section and appended row of internal cells, the respective outputs of these two sections, and the overall output of the complete processor. Note that the systolic processor I of Fig. 14.1 and the systolic processor II of Fig. 14.6 are mathematically equivalent; however, they have different numerical properties.

A distinctive feature of the systolic structure shown in Fig. 14.6 is that, unlike the systolic structure of Fig. 14.1, computation of the *a posteriori* estimation error does not require knowledge of the weight vector $\hat{\mathbf{w}}(n)$. Clearly, the structure of Fig. 14.6 may be extended to include a linear systolic section (as in Fig. 14.1) so as to compute $\hat{\mathbf{w}}(n)$, if so required. However, there is a simpler method of computing the weight vector $\hat{\mathbf{w}}(n)$ as a useful *by-product* of the direct error (residual) extraction capability inherent in the systolic process of Fig. 14.6. The method for extracting the weight vector $\hat{\mathbf{w}}(n)$ is referred to as *serial weight flushing* (Ward et al., 1986; Shepherd and McWhirter, 1991). To explain the method, let $\mathbf{u}(n)$ denote the input vector and $d(n)$ denote the desired response, both at time n. Given that the weight vector at this time is $\hat{\mathbf{w}}(n)$, the corresponding *a posteriori* estimation error is

$$e(n) = d(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n) \tag{14.58}$$

Suppose that the state of the array is *frozen* at time n_+, immediately *after* the systolic computation at time n is completed. Specifically, any update of stored values in the array is suppressed; otherwise, it is permitted to function normally in all other respects. At time n_+, we also set the desired response $d(n)$ equal to zero. We now define an input vector that consists of a string of zeros, except for the ith element that is set equal to unity, as shown by

$$\mathbf{u}^H(n_+) = [0 \ldots 010 \ldots 0] \qquad \begin{matrix} \uparrow \\ i\text{th element} \end{matrix} \tag{14.59}$$

Then, substituting these values in Eq. (14.58), we get

$$e(n_+) = -\hat{w}_i^*(n_+) \tag{14.60}$$

In other words, except for a trivial sign change, we may compute the ith element of the M-by-1 weight vector $\mathbf{w}^H(n)$ by freezing the state of the processor at time n, and subsequently setting the desired response equal to zero and feeding the processor with an input vector whose ith element is unity and the remaining $M - 1$ elements are all zero. The essence of all of this is that the Hermitian-transposed weight vector $\hat{\mathbf{w}}^H(n)$ may be viewed as the *impulse response* of the *nonadaptive* (i.e., frozen) form of the systolic array processor in the sense that it can be generated as the system output produced by inputting an $(M - 1)$-by-$(M - 1)$ identity matrix to the main triangular array and a zero vector to the bottom row of the array in Fig. 14.6 (Shepherd and McWhirter, 1991). To "flush" the entire M-by-1 weight vector $\hat{\mathbf{w}}^H(n)$ out of the systolic processor in Fig. 14.6, the procedure is therefore simply to halt the update of all stored values and input a data matrix that consists of a unit diagonal matrix (i.e., identity matrix) of dimension M.

14.4 EXTENDED QR–RLS ALGORITHM

The systolic array structure of Fig. 14.6 is suitable for adaptive filtering applications such as adaptive beamforming and acoustic echo cancelation, where the primary function is to compute the *a posteriori* estimation error without explicit knowledge of the least-squares weight vector. However, in other adaptive filtering applications such as system identification and spectrum analysis, knowledge of the weight vector on a *continuing* basis is a necessary requirement. Although, indeed, we may cater to this requirement by appending a linear section to the systolic array structure of Fig. 14.6 in the manner described in Fig. 14.1, the use of such a procedure is computationally inefficient because the data throughput of the linear section is lower than that of the triangular section. A preferable approach is to modify the QR–RLS algorithm so as to avoid the need for the cumbersome method of back substitution. This should be possible in light of what we know about the extended square-root information filter, which computes the state estimate directly from the postarray without invoking back substitution. The modified form of QR–RLS algorithm derived in this way is called the *extended QR–RLS algorithm* (Hudson et al., 1989; Yang and Böhme, 1992).

To be specific, consider the transformation of Eq. (14.39) that pertains to the extended square-root information filter. We may formulate the prearray-to-postarray transformation for the extended QR–RLS algorithm by using the one-to-one correspondences that exist between the Kalman variables and the RLS variables, and so write the following (Sayed and Kailath, 1994):

$$\begin{bmatrix} \lambda^{1/2}\mathbf{\Phi}^{1/2}(n-1) & \mathbf{u}(n) \\ \lambda^{1/2}\mathbf{p}^{H}(n-1) & d(n) \\ \mathbf{0}^{T} & 1 \\ \lambda^{-1/2}\mathbf{\Phi}^{-H/2}(n-1) & \mathbf{0} \end{bmatrix} \mathbf{\Theta}(n) = \begin{bmatrix} \mathbf{\Phi}^{1/2}(n) & \mathbf{0} \\ \mathbf{p}^{H}(n) & \xi(n)\gamma^{1/2}(n) \\ \mathbf{u}^{H}(n)\mathbf{\Phi}^{-H/2}(n) & \gamma^{1/2}(n) \\ \mathbf{\Phi}^{-H/2}(n) & -\mathbf{k}(n)\gamma^{-1/2}(n) \end{bmatrix} \tag{14.61}$$

Moreover, we readily see from the second line of Eq. (14.40) that the updated least-squares value of the weight vector is computed using the recursion:

$$\begin{aligned} \hat{\mathbf{w}}(n) &= \hat{\mathbf{w}}(n-1) + \mathbf{k}(n)\xi^{*}(n) \\ &= \hat{\mathbf{w}}(n-1) + (\mathbf{k}(n)\gamma^{-1/2}(n))\,(\xi(n)\gamma^{1/2}(n))^{*} \end{aligned} \tag{14.62}$$

where the quantities $\mathbf{k}(n)\gamma^{-1/2}(n)$ and $\xi(n)\lambda^{1/2}(n)$ are read directly from the postarray in Eq. (14.61). Note, however, unlike the QR–RLS algorithm of Section 14.3, both $\mathbf{\Phi}^{1/2}$ and $\mathbf{\Phi}^{-H/2}$ are propagated in the extended QR–RLS algorithm. Accordingly, these two algorithms may behave differently in finite-precision arithmetic; we will have more to say on this issue in Chapter 17.

A summary of the extended QR–RLS algorithm in presented in Table 14.2.

Systolic Array Implementation

Figure 14.9 (drawn for the case of filter order $M = 3$) presents a systolic array implementation of the extended QR–RLS algorithm (Yang and Böhme, 1992; Sayed and Kailath, 1994). This structure consists of two triangular sections appended to each other:

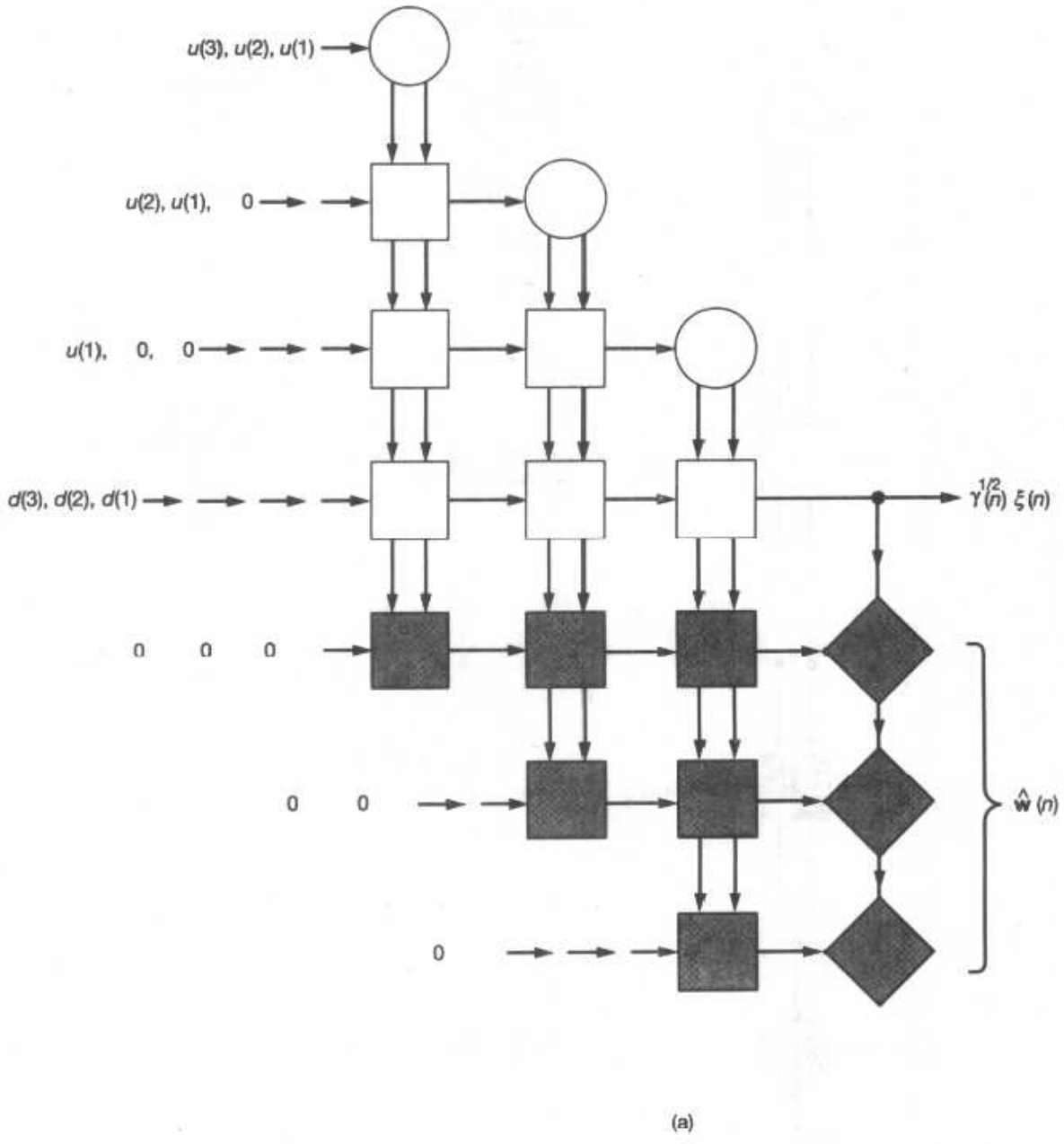
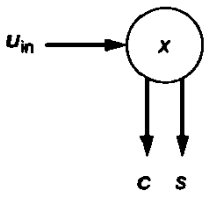
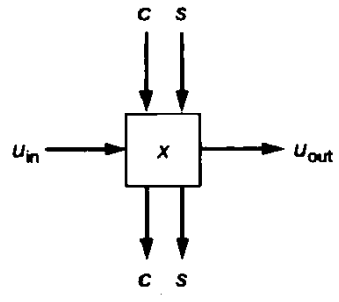


Figure 14.9 (a) Systolic array implementation of the extended QR–RLS algorithm. (b) Cells for the lower triangular section (unshaded). (c) Cells for the upper triangular section (shaded). Parts (b) and (c) of the figure are shown on the next page.



If $u_{in} = 0$, then
$c \leftarrow 1$
$s \leftarrow 0$
$x \leftarrow \lambda^{1/2} x$
Otherwise,
$x' \leftarrow \sqrt{\lambda x^2 + |u_{in}|^2}$
$c \leftarrow \frac{\lambda^{1/2} x}{x'}$
$s \leftarrow \frac{u_{in}}{x'}$
$x \leftarrow x'$

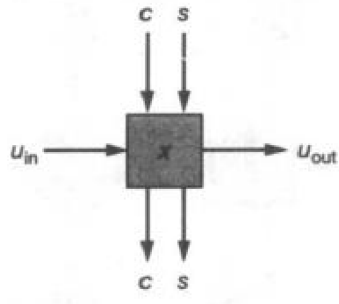
Initialization:
At $n = 0$, set
$x = 0$
$c = 1$
$s = 0$



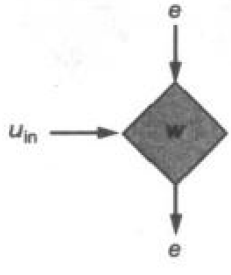
$u_{out} \leftarrow c u_{in} - s^* \lambda^{1/2} x$
$x \leftarrow s u_{in} + c \lambda^{1/2} x$

Initialization:
At $n = 0$, set
$x = 0$
$c = 1$
$s = 0$

(b)



$u_{out} \leftarrow c u_{in} - s^* \lambda^{-1/2} x$
$x \leftarrow s u_{in} + c \lambda^{-1/2} x$



$w \leftarrow w - e^* u_{in}$

(c)

Figure 14.9 (*Cont.*)

- The top triangular section, shown unshaded in Fig. 14.9(a), operates in exactly the same way as the triangular section of Fig. 14.1. The computations performed by the boundary and internal cells of this section are described in Fig. 14.2; they are reproduced in Fig. 14.9(b) for convenience of presentation. The function of the top triangular section is to compute the quantity $\xi(n)\gamma^{1/2}(n)$.
- The bottom triangular section, shown lightly shaded in Fig. 14.9(a), consists of its own boundary and internal cells. The computations performed by the latter cells are described in Fig. 14.9(c). This second triangular section rotates stored values of $\lambda^{-1/2}\Phi^{-H/2}(n-1)$ and an externally applied vector of zeros, yielding the updated $\Phi^{-H/2}(n)$ and the desired quantity $\mathbf{k}(n)\gamma^{-1/2}(n)$.

The correction needed to update the weight vector is obtained simply by complex conjugating the top triangular section's output $\xi(n)\gamma^{1/2}(n)$ and then multiplying it by the bottom triangular section's output $\mathbf{k}(n)\gamma^{-1/2}(n)$, in accordance with Eq. (14.62). These computations are performed in the diamond-shaped boundary cells of the bottom triangular section.

14.5 ADAPTIVE BEAMFORMING

From previous discussions of adaptive beamforming, we recall that the objective of this spatial form of adaptive filtering is to modify the individual outputs of an array of sensors so as to produce an overall far-field pattern that optimizes, in some statistical sense, the reception of a target signal along a direction of interest. As with any adaptive filter, this optimization is achieved by suitable modifications of a set of weights built into the construction of the array. However, unlike other adaptive filtering applications, adaptive beamforming does not require explicit knowledge of the weights. This suggests a possible area of application for the QR–RLS algorithm implemented in the form of a systolic array, particularly the structure described in Fig. 14.6.

In this section we focus on an important type of adaptive beamforming known as *minimum variance distortionless response (MVDR) beamforming*. The key question, of course, is how to formulate the QR–RLS algorithm, and therefore the triangular systolic array of Fig. 14.6, so as to perform the MVDR beamforming task.

The MVDR Problem

Consider a linear array of M uniformly spaced sensors whose outputs are individually weighted and then summed to produce the beamformer output

$$e(i) = \sum_{l=1}^{M} w_l^*(n)u_l(i) \tag{14.63}$$

where $u_l(i)$ is the output of sensor l at time i, and $w_l(n)$ is the associated (complex) weight. To simplify the mathematical presentation, we consider the simple case of a single look direction. Let $s_1(\phi), s_2(\phi), \ldots, s_M(\phi)$ be the elements of a prescribed *steering vector* $\mathbf{s}(\phi)$; the electrical angle ϕ is determined by the look direction of interest. In particular, the element $s_l(\phi)$ is the output of sensor l of the array under the condition that there is no signal other than that due to a source of interest. We may thus state the MVDR problem as follows:

Minimize the cost function

$$\mathscr{E}(n) = \sum_{i=1}^{n} \lambda^{n-i} |e(i)|^2 \tag{14.64}$$

subject to the constraint

$$\sum_{l=1}^{M} w_l^*(n) s_l(\phi) = 1 \qquad \text{for all } n \tag{14.65}$$

Using matrix notation, we may redefine the cost function $\mathscr{E}(n)$ of Eq. (14.64) as

$$\mathscr{E}(n) = \boldsymbol{\epsilon}^H(n)\boldsymbol{\Lambda}(n)\boldsymbol{\epsilon}(n) \tag{14.66}$$

where $\boldsymbol{\Lambda}(n)$ is the exponential weighting matrix, and $\boldsymbol{\epsilon}(n)$ is the vector of constrained beamformer outputs. According to Eq. (14.63), the beamformer output vector $\boldsymbol{\epsilon}(n)$ is related to the data matrix $\mathbf{A}(n)$ by

$$\begin{aligned} \boldsymbol{\epsilon}(n) &= [e(1), e(2), \ldots, e(n)]^H \\ &= \mathbf{A}(n)\mathbf{w}(n) \end{aligned} \tag{14.67}$$

where $\mathbf{w}(n)$ is the weight vector, and the data matrix $\mathbf{A}(n)$ is defined in terms of the *snapshots* $\mathbf{u}(1), \mathbf{u}(2), \ldots, \mathbf{u}(n)$ by

$$\begin{aligned} \mathbf{A}^H(n) &= [\mathbf{u}(1), \mathbf{u}(2), \ldots, \mathbf{u}(n)] \\ &= \begin{bmatrix} u_1(1) & u_1(2) & \cdots & u_1(n) \\ u_2(1) & u_2(2) & \cdots & u_2(n) \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ u_M(1) & u_M(2) & \cdots & u_M(n) \end{bmatrix} \end{aligned} \tag{14.68}$$

We may now restate the MVDR problem in matrix terms as follows:

Given the data matrix $\mathbf{A}(n)$ and the exponential weighting matrix $\boldsymbol{\Lambda}(n)$, minimize the cost function

$$\mathscr{E}(n) = \| \boldsymbol{\Lambda}^{1/2}(n)\mathbf{A}(n)\mathbf{w}(n) \|^2 \tag{14.69}$$

with respect to the weight vector $\mathbf{w}(n)$, subject to the constraint

$$\mathbf{w}^H(n)\mathbf{s}(\phi) = 1 \qquad \text{for all } n$$

where $\mathbf{s}(\phi)$ is the steering vector for a prescribed electrical angle ϕ.

The solution to this constrained optimization problem is described by the MVDR formula (see Section 10.9)

$$\hat{\mathbf{w}}(n) = \frac{\boldsymbol{\Phi}^{-1}(n)\mathbf{s}(\phi)}{\mathbf{s}^H(\phi)\boldsymbol{\Phi}^{-1}(n)\mathbf{s}(\phi)} \tag{14.70}$$

where $\boldsymbol{\Phi}(n)$ is the M-by-M correlation matrix of the exponentially weighted sensor outputs averaged over n snapshots, and which is related to the data matrix $\mathbf{A}(n)$ as follows:

$$\boldsymbol{\Phi}(n) = \mathbf{A}^H(n)\boldsymbol{\Lambda}(n)\mathbf{A}(n) \tag{14.71}$$

Systolic MVDR Beamformer

Let the correlation matrix $\boldsymbol{\Phi}(n)$ be expressed in its factored form:

$$\boldsymbol{\Phi}(n) = \boldsymbol{\Phi}^{1/2}(n)\boldsymbol{\Phi}^{H/2}(n) \tag{14.72}$$

Correspondingly, we may rewrite Eq. (14.70) as follows:

$$\hat{\mathbf{w}}(n) = \frac{\boldsymbol{\Phi}^{-H/2}(n)\boldsymbol{\Phi}^{-1/2}(n)\mathbf{s}(\phi)}{\mathbf{s}^H(\phi)\boldsymbol{\Phi}^{-H/2}(n)\boldsymbol{\Phi}^{-1/2}(n)\mathbf{s}(\phi)} \tag{14.73}$$

To simplify matters, we define the auxiliary vector:

$$\mathbf{a}(n) = \boldsymbol{\Phi}^{-1/2}(n)\mathbf{s}(\phi) \tag{14.74}$$

We now note that the denominator of Eq. (14.73) is a real-valued scalar equal to the squared Euclidean norm of the auxiliary vector $\mathbf{a}(n)$. As for the numerator, it is equal to the Hermitian-transposed square root $\boldsymbol{\Phi}^{-H/2}(n)$ postmultiplied by the auxiliary vector $\mathbf{a}(n)$. We may thus simplify Eq. (14.73) to

$$\hat{\mathbf{w}}(n) = \frac{\boldsymbol{\Phi}^{-H/2}(n)\mathbf{a}(n)}{\|\mathbf{a}(n)\|^2} \tag{14.75}$$

The MVDR beamformer output, or in adaptive filtering terminology, the *a posteriori* estimation error $e(n)$ produced at time n in response to the snapshot $\mathbf{u}(n)$ is given by

$$\begin{aligned} e(n) &= \hat{\mathbf{w}}^H(n)\mathbf{u}(n) \\ &= \frac{\mathbf{a}^H(n)\boldsymbol{\Phi}^{-1/2}(n)\mathbf{u}(n)}{\|\mathbf{a}(n)\|^2} \end{aligned} \tag{14.76}$$

Let $e'(n)$ denote a new estimation error, defined by

$$e'(n) = \mathbf{a}^H(n)\boldsymbol{\Phi}^{-1/2}(n)\mathbf{u}(n) \tag{14.77}$$

We may then reduce Eq. (14.76) to

$$e(n) = \frac{e'(n)}{\|\mathbf{a}(n)\|^2} \tag{14.78}$$

This equation shows that the MVDR beamformer output $e(n)$ is uniquely defined by two quantities: $e'(n)$ and $\mathbf{a}(n)$.

At this point in the discussion, we find it informative to recall the formula for the *a posteriori* estimation error $e(n)$ actually computed by the QR–RLS algorithm. By definition, we have

$$e(n) = d(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n) \tag{14.79}$$

where $d(n)$ is the desired response, $\hat{\mathbf{w}}(n)$ is the least-squares weight vector, and $\mathbf{u}(n)$ is the input data vector. Substituting Eq. (14.50) in (14.79) yields

$$e(n) = d(n) - \mathbf{p}^H(n)\boldsymbol{\Phi}^{-1/2}(n)\mathbf{u}(n) \tag{14.80}$$

Thus, comparing Eqs. (14.77) and (14.80), we readily deduce the correspondences between the QR–RLS adaptive filtering and MVDR beamforming variables listed in Table 14.4.

The stage is now set for a recasting of the QR–RLS algorithm to suit the MVDR beamforming problem. First of all, we reformulate the prearray in Eq. (14.49) in light of the correspondences in Table 14.4 and so express it as

$$\begin{bmatrix} \lambda^{1/2}\boldsymbol{\Phi}^{1/2}(n-1) & \mathbf{u}(n) \\ \lambda^{1/2}\mathbf{a}^H(n-1) & 0 \\ \mathbf{0}^T & 1 \end{bmatrix}$$

Next, we determine the postarray that goes with this prearray by proceeding in the same manner as that described in Section 14.3. We may thus write

$$\begin{bmatrix} \lambda^{1/2}\boldsymbol{\Phi}^{1/2}(n-1) & \mathbf{u}(n) \\ \lambda^{1/2}\mathbf{a}^H(n-1) & 0 \\ \mathbf{0}^T & 1 \end{bmatrix} \boldsymbol{\Theta}(n) = \begin{bmatrix} \boldsymbol{\Phi}^{1/2}(n) & \mathbf{0} \\ \mathbf{a}^H(n) & -e'(n)\gamma^{-1/2}(n) \\ \mathbf{u}^H(n)\boldsymbol{\Phi}^{-H/2}(n) & \gamma^{1/2}(n) \end{bmatrix} \tag{14.81}$$

We now see that the two quantities of interest to the MVDR problem may be obtained from the postarray of Eq. (14.81) as follows:

- The updated auxiliary vector $\mathbf{a}(n)$ is read directly from the second row of the postarray.
- The estimation error $e'(n)$ is given by

$$e'(n) = (e'(n)\gamma^{-1/2}(n))(\gamma^{1/2}(n)) \tag{14.82}$$

where $e'(n)\gamma^{-1/2}(n)$ and $\gamma^{1/2}(n)$ are read directly from the nonzero entries of the second column of the postarray.

TABLE 14.4 CORRESPONDENCES BETWEEN THE QR-RLS ADAPTIVE FILTERING AND MVDR BEAMFORMING VARIABLES

QR-RLS adaptive filtering	MVDR beamforming	Description
$e(n)$	$-e'(n)$	Estimation error
$d(n)$	0	Desired response
$\mathbf{p}(n)$	$\mathbf{a}(n)$	Auxiliary vector
$\mathbf{u}(n)$	$\mathbf{u}(n)$	Snapshot

Finally, we may implement the MVDR beamformer using the systolic array structure shown in Fig. 14.10, which is basically the same as that of Fig. 14.6 except for some minor changes (McWhirter and Shepherd, 1989). Specifically, $d(n)$ is set equal to zero for all n. With this change in place, we note the following from Fig. 14.10:

- The auxiliary vector $\mathbf{a}(n)$ is generated and stored in the bottom row of cells.
- The output of the final cell is identically equal to $-e'(n)$.

With a continuing sequence of snapshots $\mathbf{u}(n)$, $\mathbf{u}(n+1), \ldots$, applied to the systolic array processor in Fig. 14.10, a corresponding sequence of estimation errors $e(n)$, $e(n+1), \ldots$ is generated by the MVDR beamformer in accordance with Eq. (14.78).

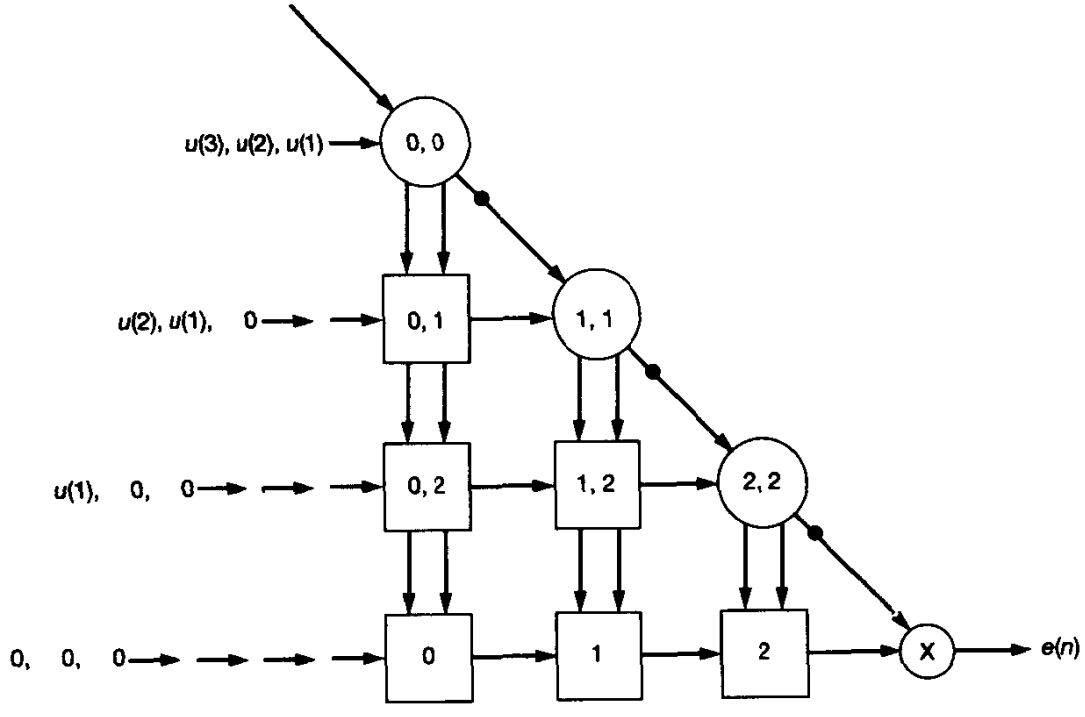


Figure 14.10 Systolic array for solving the MVDR beamforming problem.

Computer Experiment

We now illustrate the performance of the systolic array implementation of an adaptive MVDR beamformer by considering a linear array of five uniformly spaced sensors. The spacing d between adjacent elements equals one half of the received wavelength. The array operates in an environment that consists of a target signal and a single source of interference, which originate from different sources. The exponential weighting factor $\lambda = 1$.

The aims of the experiment are twofold:

1. To examine the evolution of the adapted spatial response (pattern) of the beamformer with time.
2. To evaluate the effect of varying the interference-to-target ratio on the interference-nulling capability of the beamformer.

The directions of the target and source of interference are as follows:

Excitation	Angle of incidence, θ, measured with respect to normal to the array (radians)
Target	$\sin^{-1}(0.2)$
Interference	0

The steering vector is defined by

$$\mathbf{s}^T(\phi) = [1, e^{-j\phi}, e^{-j2\phi}, e^{-j3\phi}, e^{-j4\phi}] \tag{14.83}$$

where the electrical angle ϕ is defined in terms of the angle of incidence θ as follows:

$$\phi = \pi \sin\theta \tag{14.84}$$

The data set used for the experiment consists of three components: a target signal, elemental receiver noise, and an interfering signal. The target signal and the interfering signal originate in the far field of the array antenna and are therefore represented by plane waves impinging on the array along their respective directions. Let these directions be denoted by angles θ_1 and θ_2, measured (in radians) with respect to the normal to the array antenna. The elemental signals of the array antenna are thus expressed in baseband form as follows:

$$u(n) = A_1 \exp(jn\phi_1) + A_2 \exp(jn\phi_2 + \psi) + v(n), \qquad n = 1, 2, 3, 4, 5 \tag{14.85}$$

where A_1 is the amplitude of the target signal and A_2 is the amplitude of the interfering signal. The electrical angles ϕ_1 and ϕ_2 are related to the individual angles of arrival θ_1 and θ_2, respectively, by Eq. (14.84). Since the target and interfering signals are uncorrelated,

the phase difference ψ associated with the second component in Eq. (14.85) is a random variable uniformly distributed over the interval $(0, 2\pi]$. Lastly, the additive receiver noise $v(n)$ is a complex-valued Gaussian random variable with zero mean and unit variance. The target-to-noise ratio is held constant at 10 dB; the interference-to-noise ratio is variable, assuming the values 40, 30, and 20 dB.

Figure 14.11 shows the effects of varying the target-to-interference ratio and the number of snapshots (excluding those needed for initialization) on the adapted response of the beamformer. The response is obtained by plotting $20\log_{10}|e(n)e^{j\Phi}|$ versus the electrical angle ϕ; multiplication by the exponential factor $e^{j\Phi}$ provides a means of spatially sampling the beamformer output. The results are presented in three parts, corresponding to 20, 100, and 200 snapshots; and each part corresponds to the three different values of interference-to-noise ratio, namely, 40, 30, and 20 dB.

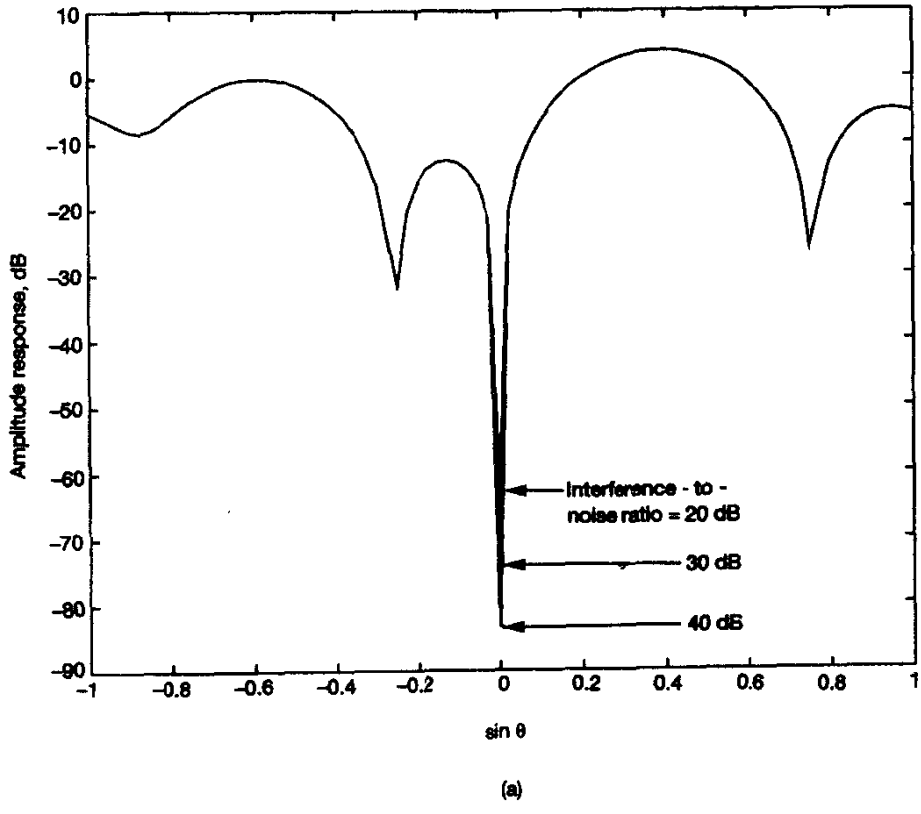
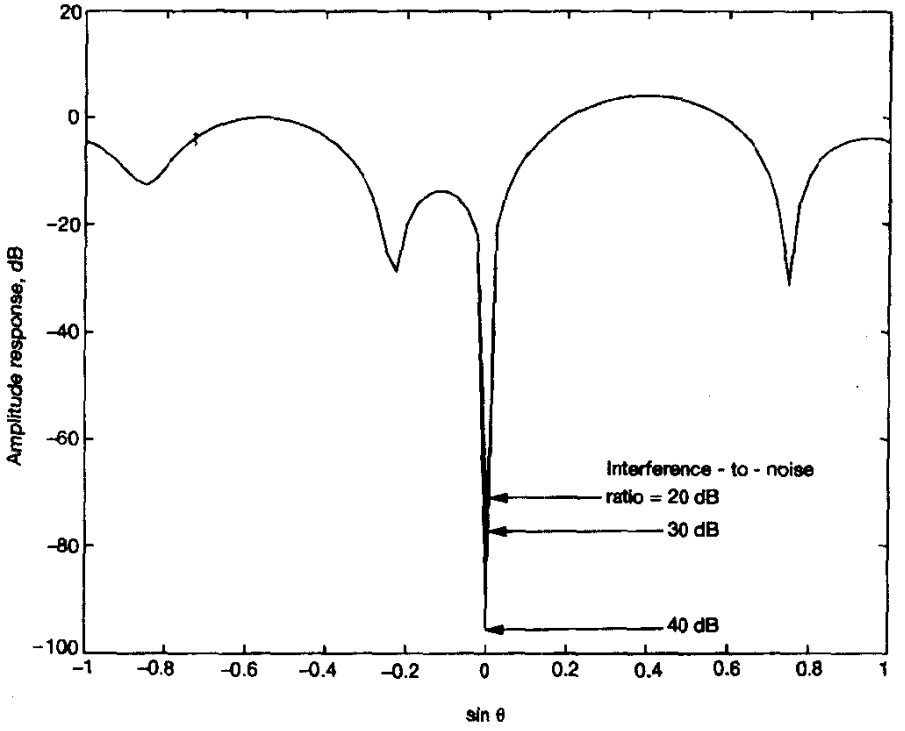


Figure 14.11 Results of the computer experiment on the spatial response of the systolic MVDR beamformer for varying interference-to-noise ratio and different number of snapshots: (a) $n = 20$; (b) $n = 100$; and (c) $n = 200$. Parts (b) and (c) of the figure are shown on the next two pages.



(b)

Figure 14.11 (*Cont.*)

Based on these results, we may make the following observations:

- The response of the beamformer along the target is held fixed at a value of one under all conditions, as required.
- With as few as 30 snapshots, including initialization, the beamformer exhibits a reasonably effective nulling capability, which continually improves as the beamformer processes more snapshots.
- The response of the beamformer is relatively insensitive to variations in the interference-to-target ratio.

14.6 INVERSE QR–RLS ALGORITHM

We now come to our last square-root adaptive filtering algorithm, known as the *inverse QR–RLS algorithm*. This algorithm derives its name from the fact that, instead of operating on the correlation matrix $\Phi(n)$ as in the conventional QR–RLS algorithm or extended

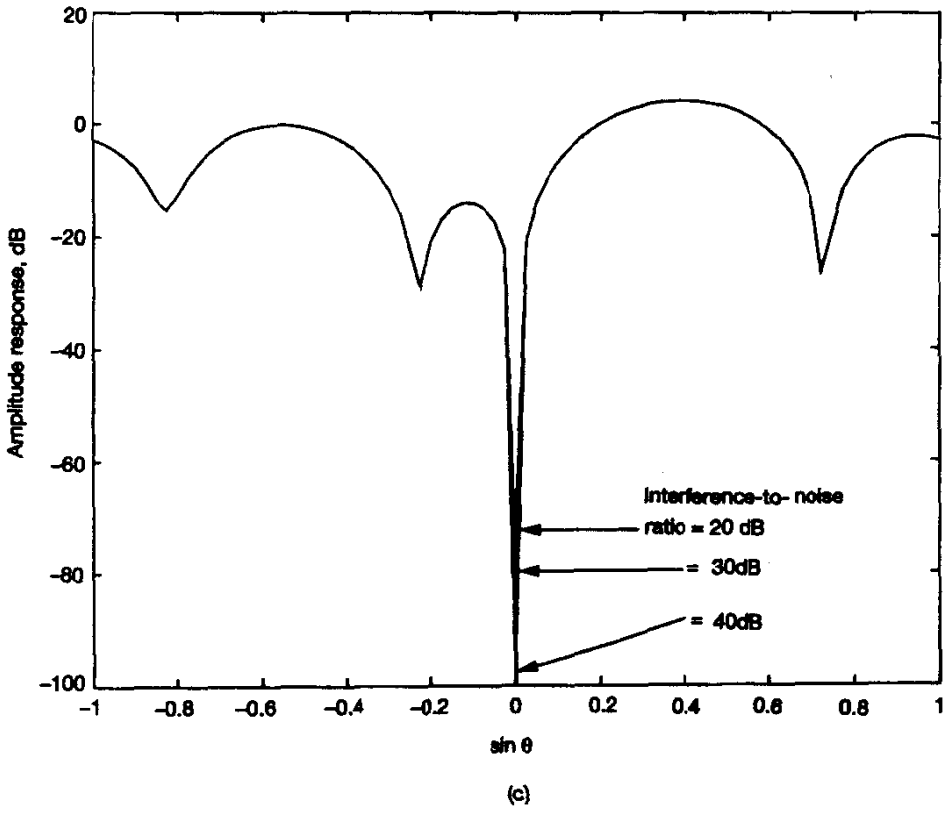


Figure 14.11 (*Cont.*)

QR–RLS algorithm, the operation is performed on the "inverse" of $\Phi(n)$. In other words, the inverse QR–RLS algorithm operates on $\mathbf{P}(n) = \Phi^{-1}(n)$ (Alexander and Ghirnikar, 1993; Pan and Plemmons, 1989). In light of the one-to-one correspondences between the Kalman variables and RLS variables, this means that the inverse QR–RLS algorithm is basically a reformulation of the square-root covariance (Kalman) filtering algorithm (Sayed and Kailath, 1994).

Referring to Eq. (14.26), which pertains to the square-root covariance filtering algorithm, we readily see that the corresponding prearray-to-postarray transformation for the inverse QR–RLS algorithm may be written as follows (after canceling common terms):

$$\begin{bmatrix} 1 & \lambda^{-1/2}\mathbf{u}^H(n)\mathbf{P}^{1/2}(n-1) \\ \mathbf{0} & \lambda^{-1/2}\mathbf{P}^{1/2}(n-1) \end{bmatrix} \Theta(n) = \begin{bmatrix} \gamma^{-1/2}(n) & \mathbf{0}^T \\ \mathbf{k}(n)\gamma^{-1/2}(n) & \mathbf{P}^{1/2}(n) \end{bmatrix} \tag{14.86}$$

where $\Theta(n)$ is a unitary rotation that operates on the block entry $\lambda^{-1/2}\mathbf{u}^H(n)\mathbf{P}^{1/2}(n-1)$ in the prearray by annihilating its elements, one by one, so as to produce a block zero entry in the first row of the postarray. The gain vector $\mathbf{k}(n)$ of the RLS algorithm is readily obtained from the entries in the first column of the postarray by writing

$$\mathbf{k}(n) = (\mathbf{k}(n)\gamma^{-1/2}(n))(\gamma^{-1/2}(n))^{-1} \tag{14.87}$$

TABLE 14.5 SUMMARY OF THE INVERSE QR-RLS ALGORITHM

Inputs:

input signal vector = $\{\mathbf{u}(1), \mathbf{u}(2), \ldots, \mathbf{u}(n)\}$
desired response = $\{d(1), d(2), \ldots, d(n)\}$

Known parameters:

exponential weighting factor = λ

Initial conditions:

$\mathbf{P}^{1/2}(0) = \delta^{-1/2}\mathbf{I}$, δ = small positive constant
$\hat{\mathbf{w}}(0) = \mathbf{0}$

Computations:

For $n = 1, 2, \ldots$, compute

$$\begin{bmatrix} 1 & \lambda^{-1/2}\mathbf{u}^H(n)\mathbf{P}^{1/2}(n-1) \\ \mathbf{0} & \lambda^{-1/2}\mathbf{P}^{1/2}(n-1) \end{bmatrix}\boldsymbol{\Theta}(n) = \begin{bmatrix} \gamma^{-1/2}(n) & \mathbf{0}^T \\ \mathbf{k}(n)\gamma^{-1/2}(n) & \mathbf{P}^{1/2}(n) \end{bmatrix}$$

where $\boldsymbol{\Theta}(n)$ is a unitary rotation that produces a zero entry in the first row of the post-array.

$$\mathbf{k}(n) = (\mathbf{k}(n)\gamma^{-1/2}(n))\,(\gamma^{-1/2}(n))^{-1}$$

$$\xi(n) = d(n) - \hat{\mathbf{w}}^H(n-1)\mathbf{u}(n)$$

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{k}(n)\xi^*(n)$$

Hence, the least-squares weight vector may be updated in accordance with the recursion;

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{k}(n)\xi^*(n) \tag{14.88}$$

where the *a priori* estimation error $\xi(n)$ is defined in the usual way:

$$\xi(n) = d(n) - \hat{\mathbf{w}}^H(n-1)\mathbf{u}(n) \tag{14.89}$$

A summary of the inverse QR–RLS algorithm, including initial conditions, is presented in Table 14.5.

Here we note that since the square root $\boldsymbol{\Phi}^{1/2}(n)$ is lower triangular in accordance with Eq. (14.47), its inverse matrix $\boldsymbol{\Phi}^{-1/2}(n) = \mathbf{P}^{1/2}(n)$ is upper triangular.

The inverse QR–RLS algorithm differs from both the conventional QR–RLS algorithm and the extended QR–RLS algorithm in a fundamental way. Specifically, the input data vector $\mathbf{u}(n)$ does not appear by itself as a block entry in the prearray of the algorithm; rather, it is multiplied by $\lambda^{-1/2}\mathbf{P}^{1/2}(n-1)$. Hence, the input data vector $\mathbf{u}(n)$ has to be preprocessed prior to performing the rotations described in Eq. (14.86). The *preprocessor* to do this consists of simply computing the inner product of $\mathbf{u}(n)$ with each of the columns of the square-root matrix $\mathbf{P}^{1/2}(n-1)$ scaled by $\lambda^{-1/2}$. The preprocessor can be structured to take advantage of the upper triangular form of $\mathbf{P}^{-1/2}(n-1)$.

The inverse QR–RLS algorithm lends itself to parallel implementation in the form of two sections connected together (Alexander and Ghirnikar, 1993):

- A *triangular systolic array*, which operates on the preprocessed input vector $\lambda^{-1/2}\mathbf{P}^{H/2}(n-1)\mathbf{u}(n)$ in accordance with Eq. (14.86). Nonzero elements of the updated matrix $\mathbf{P}^{H/2}(n)$ are stored in the internal cells of the systolic array. The two other products of the systolic computation are $\gamma^{-1/2}(n)$ and $\mathbf{k}(n)\gamma^{-1/2}(n)$.
- A *linear section*, which is appended to the triangular section for the purpose of operating on the latter two products of the systolic computation to produce the elements of the updated weight vector $\hat{\mathbf{w}}(n)$ in accordance with Eqs. (14.87), (14.89), and (14.88), in that order.

The combination of these two sections is designed to operate in a completely parallel fashion.

14.7 SUMMARY AND DISCUSSION

In this chapter we discussed the derivations of three square-root adaptive filtering algorithms for exponentially weighted recursive least-squares (RLS) estimation in a unified manner. The algorithms are known as the QR–RLS algorithm, the extended QR–RLS algorithm, and the inverse QR–RLS algorithm. These algorithms bear one-to-one correspondences with the square-root information filter, the extended square-root information filter, and the square-root covariance filter, respectively, that represent square-root variants of the celebrated Kalman filter. These correspondences were exploited in the derivations of different variants of the RLS algorithm presented here.

The inverse QR–RLS algorithm is a natural extension of the standard RLS algorithm. It may therefore be legitimately referred to as the *square-root RLS algorithm.*

The QR–RLS algorithm and inverse QR–RLS algorithm propagate a *single* square root, namely, $\mathbf{\Phi}^{1/2}(n)$ and $\mathbf{P}^{1/2}(n) = \mathbf{\Phi}^{-1/2}(n)$, respectively. On the other hand, the extended QR–RLS algorithm propagates two square roots: $\mathbf{\Phi}^{1/2}(n)$ and the Hermitian transpose of $\mathbf{P}^{1/2}(n)$. This raises numerical difficulties for the extended QR–RLS algorithm, as discussed in Chapter 17.

A common feature of the QR–RLS algorithm, extended QR–RLS algorithm, and inverse QR–RLS algorithm is that, in varying degrees, they lend themselves to parallel implementation in the form of systolic arrays. Naturally, the actual details of the systolic array implementations depend on which algorithm is being considered. In particular, there are some basic differences that should be carefully noted. The conventional QR–RLS and extended QR–RLS algorithms operate directly on the input data. On the other hand, in the inverse QR–RLS algorithm the input data vector $\mathbf{u}(n)$ is transformed by the square-root matrix $\mathbf{P}^{1/2}(n) = \mathbf{\Phi}^{-1/2}(n)$ before it can be processed by the systolic array. This adds computational complexity to the parallel implementation of the inverse QR–RLS algorithm.

The parallel implementations of both the extended QR–RLS algorithm and inverse QR–RLS algorithms permit the computation of the least-squares weight vector in an efficient manner in their own individual ways. Accordingly, these two square-root adaptive filtering algorithms are well suited for applications such as system identification, spectrum estimation, and adaptive equalization, where knowledge of the weight vector is a necessary requirement. In contrast, computation of the weight vector in the conventional

QR–RLS algorithm involves the method of back substitution, which can be performed "on-the-fly" at the cost of additional computation. For this reason, the scope of practical applications for the QR–RLS algorithm is restricted to those areas such as adaptive beamforming and acoustic echo cancelation, where it is *not* necessary to have explicit knowledge of the weight vector.

Finally, the point that needs to be stressed is that all three QR–RLS algorithms preserve the desirable convergence properties of the standard RLS algorithm, namely, a fast rate of convergence and insensitivity to variations in the eigenvalue spread of the correlation matrix of incoming data.

One final comment is in order. The systolic array implementations of the QR decomposition involved in the design of the variants of the RLS algorithm described in this chapter were all based on the Givens rotation. This form of rotation provides one method for constructing the unitary rotation $\Theta(n)$. From Chapter 12 we recall that the Householder transformation (reflection) provides another method for constructing the unitary rotation $\Theta(n)$. According to an error analysis under finite-precision computations reported by Wilkinson (1965), the Householder transformation is superior to the Givens rotation. It is therefore of interest to know if a systolic implementation can be extended to the Householder transformation for QR decomposition–based RLS algorithms. Indeed, Liu et al. (1992) describe a two-level pipelined implementation of the Householder transformation on a systolic array with only local connections. The systolic array is, however, of a block-oriented kind, with the block size providing a new variable. In particular, improved numerical stability is attained by increasing the block size, but at the expense of increased latency.

PROBLEMS

1. Starting with the prearray-to-postarray transformation described in Eq. (14.32) for the square-root information filter, derive the equalities defined in Eqs. (14.33) to (14.37).
2. In this problem we revisit the square-root information filter. Specifically, the term $v(n)$ in the state-space model of Eqs. (14.2) and (14.3) is assumed to be a random variable of zero mean and variance $Q(n)$. Show that the square-root information filter may now be formulated as follows:

$$\mathbf{K}^{-1}(n) = \lambda(\mathbf{K}^{-1}(n-1) + Q^{-1}(n)\mathbf{u}(n)\mathbf{u}^H(n))$$

$$\mathbf{K}^{-1}(n)\hat{\mathbf{x}}(n+1\,|\,\mathcal{Y}_n) = \lambda^{1/2}(\mathbf{K}^{-1}(n-1)\hat{\mathbf{x}}(n\,|\,\mathcal{Y}_{n-1}) + Q^{-1}(n)\mathbf{u}(n)y(n))$$

   which includes Eqs. (14.29) and (14.30) as a special case.
3. Justify the validity of the prearray-to-postarray transformation described in Eq. (14.39) for the extended square-root information filter.
4. Let the n-by-n unitary matrix $\mathbf{Q}(n)$ involved in the QR-decomposition of the data matrix $\mathbf{A}(n)$ be partitioned as follows

$$\mathbf{Q}(n) = \begin{bmatrix} \mathbf{Q}_1(n) \\ \mathbf{Q}_2(n) \end{bmatrix}$$

   where $\mathbf{Q}_1(n)$ has the same number of rows as the upper triangular matrix $\mathbf{R}(n)$ in the QR-decomposition of $\mathbf{A}(n)$. Assume that the exponential weighting factor $\lambda = 1$.

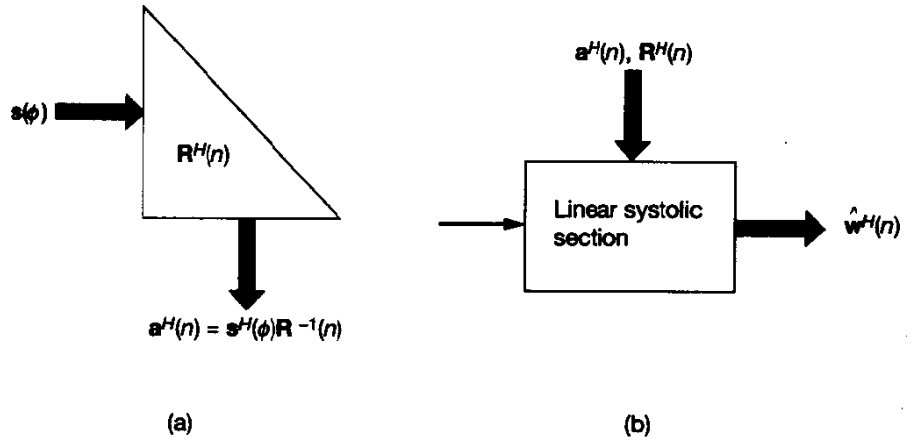


Figure P14.1

According to the method of least squares presented in Chapter 11, the projection operator is

$$\mathbf{P}(n) = \mathbf{A}(n)(\mathbf{A}^H(n)\mathbf{A}(n))^{-1}\mathbf{A}^H(n)$$

Show that for the problem at hand:

$$\mathbf{P}(n) = \mathbf{Q}_1^H(n)\mathbf{Q}_1(n)$$

How is the result modified for the case where $0 < \lambda \leq 1$?

5. In the context of the systolic array in Fig. 14.1, explain the reasons for the following:
   (a) A total of $2M$ clock cycles are required to compute the Givens rotations in the triangular section of the systolic array in Fig. 14.1.
   (b) The latency for the entire array is $4M$ clock cycles.
6. Explain the way in which the systolic array structure of Fig. 14.6 may be used to operate as a prediction-error filter.
7. Discuss the use of a linear section, based on forward substitution, for solving the equation $\mathbf{R}^H(n)\mathbf{a}(n) = \mathbf{s}$ for the vector $\mathbf{a}(n)$; the matrix $\mathbf{R}$ is an M-by-M upper triangular matrix and $\mathbf{s}$ is an M-by-1 vector.
8. Figure P14.1 depicts a block diagram representation of an MVDR beamforming algorithm. Specifically, the triangular array in part (a) of this figure is frozen at time n and the steering vector $\mathbf{s}(\phi)$ is input into the array. The stored $\mathbf{R}^H(n)$ of the array and its output $\mathbf{a}^H(n)$ are applied to a linear systolic section as in part (b) of the figure. Do the following:
   (a) Show that the output of the triangular array is
   $$\mathbf{a}^H(n) = \mathbf{s}^H(\phi)\mathbf{R}^{-1}(n)$$
   (b) Using the method of back substitution, show that the linear systolic array produces the Hermitian transposed weight vector $\mathbf{w}^H(n)$ as its output.
9. Referring to the systolic MVDR beamformer described in Section 14.5, show that
   $$\lambda\|\mathbf{a}(n-1)\|^2 = \lambda^2\|\mathbf{a}(n)\|^2 + |\epsilon(n)|^2$$
   where $\|\mathbf{a}(n)\|$ is the Euclidean norm of the auxiliary vector $\mathbf{a}(n)$, λ is the exponential weighting factor, and $\epsilon(n)$ is some estimation error.

CHAPTER

15

Order-Recursive Adaptive Filters

In this chapter we develop another important class of adaptive filters, the design of which is based on algorithms that involve both *order-update* and *time-update recursions*. The algorithms are rooted in recursive least-squares estimation theory[1] and therefore retain two unique attributes of the RLS algorithm: A fast rate of convergence, and insensitivity to variations in the eigenvalue spread of the underlying correlation matrix of the input data. However, unlike the RLS algorithm, the computational complexity of the algorithms considered in this chapter increases *linearly* with the number of adjustable filter parameters. This highly desirable property is a direct result of order recursiveness, which gives the adaptive filter a *computationally efficient, modular, latticelike structure*. In particular, as the filter order is increased from m to $m + 1$, say, the lattice filter permits us to carry over certain information gathered from the previous computations pertaining to the filter order m.

In deriving the order-recursive adaptive filters considered herein, we follow the same approach that we pursued in the previous chapter dealing with square-root adaptive filters. Specifically, we start from a state-space model of lattice filtering, which makes it possible to exploit the relevant aspects of Kalman filter theory. In so doing, we further consolidate the unification of adaptive filters along the lines described by Sayed and Kailath

[1]There is another type of order-recursive adaptive filtering algorithm, called the *gradient adaptive lattice (GAL) algorithm*, which is rooted in stochastic approximation. The derivation of GAL algorithms follows an approach similar to that of the least-mean-square (LMS) algorithm; for details, see Appendix G.

(1994). Before embarking on this development, however, we first present some background material relating to the forward and backward predictions of input data, which is fundamental to the underlying theory of order-recursive adaptive filters.

15.1 ADAPTIVE FORWARD LINEAR PREDICTION

Consider a forward linear predictor of order m, depicted in Fig. 15.1(a), whose tap-weight vector $\hat{\mathbf{w}}_{f,m}(n)$ is optimized in the least-squares sense over the entire observation interval $1 \leq i \leq n$. Let $f_m(n)$ denote the forward prediction error produced by the predictor at time n in response to the tap-input vector $\mathbf{u}_m(n-1)$ of size m, as shown by

$$f_m(n) = u(n) - \hat{\mathbf{w}}_{f,m}^H(n)\mathbf{u}_m(n-1) \tag{15.1}$$

According to this definition, $u(n)$ plays the role of "desired response" for forward linear prediction. The compositions of input vector $\mathbf{u}_m(n-1)$ and weight vector $\hat{\mathbf{w}}_m(n)$ are as follows, respectively:

$$\mathbf{u}_m(n-1) = [u(n-1), u(n-2), \ldots, u(n-m)]^T$$

$$\hat{\mathbf{w}}_{f,m}(n) = [w_{f,m,1}(n), w_{f,m,2}(n), \ldots, w_{f,m,m}(n)]^T$$

We refer to $f_m(n)$ as the *forward a posteriori prediction error* since its computation is based on the current value of the forward predictor's tap-weight vector, $\hat{\mathbf{w}}_{f,m}(n)$. Correspondingly, we may define the *forward a priori prediction error* as

$$\eta_m(n) = u(n) - \hat{\mathbf{w}}_{f,m}^H(n-1)\mathbf{u}_m(n-1) \tag{15.2}$$

the computation of which is based on the past value of the forward predictor's tap-weight vector, $\hat{\mathbf{w}}_{f,m}(n-1)$. In effect, $\eta_m(n)$ represents a form of innovation.

In Table 15.1 are listed the correspondences between the various quantities characterizing linear estimation in general and those characterizing forward linear prediction in particular, with the RLS algorithm in mind. With the aid of this table, it is a straightforward matter to modify the RLS algorithm developed in Sections 13.3 and 13.4 to write the recursions for adaptive forward linear prediction. Specifically, we deduce the following recursion for updating the tap-weight vector of the forward predictor:

$$\hat{\mathbf{w}}_{f,m}(n) = \hat{\mathbf{w}}_{f,m}(n-1) + \mathbf{k}_m(n-1)\eta_m^*(n) \tag{15.3}$$

where $\eta_m(n)$ is the forward *a priori* prediction error defined in Eq. (15.2), and $\mathbf{k}_m(n-1)$ is the past value of the *gain vector* defined by

$$\mathbf{k}_m(n-1) = \boldsymbol{\Phi}_m^{-1}(n-1)\mathbf{u}_m(n-1) \tag{15.4}$$

The matrix $\boldsymbol{\Phi}_m^{-1}(n-1)$ is the inverse of the correlation matrix $\boldsymbol{\Phi}_m(n-1)$ of the input data, with the latter matrix being defined by

$$\boldsymbol{\Phi}_m(n-1) = \sum_{i=1}^{n-1} \lambda^{n-1-i}\mathbf{u}_m(i)\mathbf{u}_m^H(i) \tag{15.5}$$

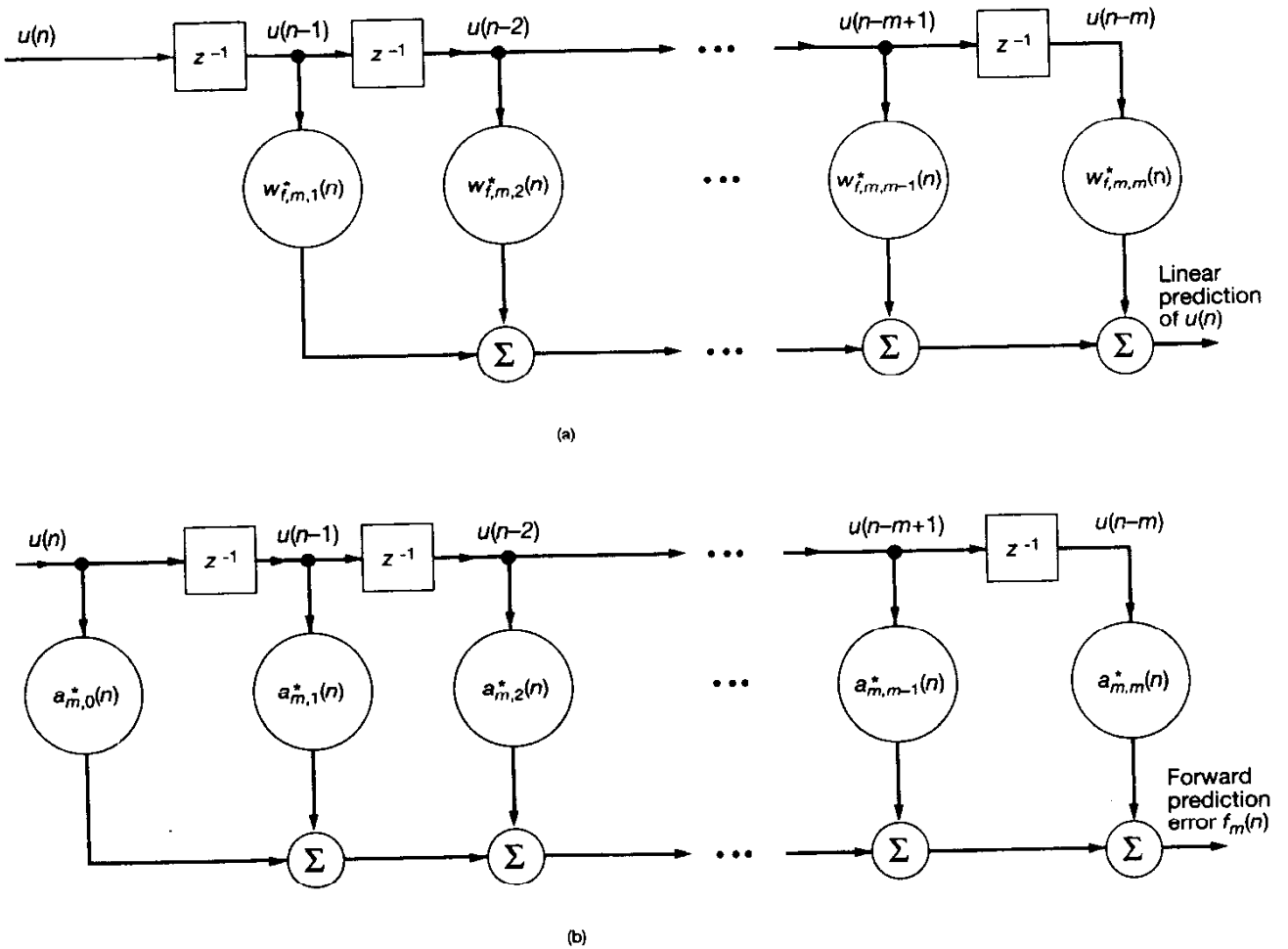


Figure 15.1 (a) Forward predictor of order m; (b) corresponding prediction-error fitter.

TABLE 15.1 SUMMARY OF CORRESPONDENCES BETWEEN LINEAR ESTIMATION, FORWARD PREDICTION, AND BACKWARD PREDICTION

Quantity	Linear estimation (general)	Forward linear prediction of order m	Backward linear prediction of order m
Tap-input vector	$\mathbf{u}(n)$	$\mathbf{u}_m(n-1)$	$\mathbf{u}_m(n)$
Desired response	$d(n)$	$u(n)$	$u(n-m)$
Tap-weight vector	$\hat{\mathbf{w}}(n)$	$\hat{\mathbf{w}}_{f,m}(n)$	$\hat{\mathbf{w}}_{b,m}(n)$
A posteriori estimation error	$e(n)$	$f_m(n)$	$b_m(n)$
A priori estimation error	$\xi(n)$	$\eta_m(n)$	$\beta_m(n)$
Gain vector	$\mathbf{k}(n)$	$\mathbf{k}_m(n-1)$	$\mathbf{k}_m(n)$
Minimum value of sum of weighted error squares	$\mathscr{E}_{\min}(n)$	$\mathscr{F}_m(n)$	$\mathscr{B}_m(n)$

The use of subscript m is intended to signify the order of the prediction process. We follow this practice here and in the rest of the chapter, since some of the recursions to be developed involve an order update.

The adaptive forward linear prediction problem just described is in terms of a predictor characterized by the tap-weight vector $\hat{\mathbf{w}}_{f,m}(n)$. Equivalently, we may describe the problem by specifying a *forward prediction-error filter*, as depicted in Fig. 15.1(b). Let $\mathbf{a}_m(n)$ denote the $(m+1)$-by-1 tap-weight vector of the prediction-error filter of order m. This tap-weight vector is related to that of the forward predictor in Fig. 15.1(a) by

$$\mathbf{a}_m(n) = \begin{bmatrix} 1 \\ -\hat{\mathbf{w}}_{f,m}(n) \end{bmatrix} \tag{15.6}$$

Then we may redefine the forward *a posteriori* prediction and forward *a priori* prediction errors as follows, respectively:

$$f_m(n) = \mathbf{a}_m^H(n)\mathbf{u}_{m+1}(n) \tag{15.7}$$

and

$$\eta_m(n) = \mathbf{a}_m^H(n-1)\mathbf{u}_{m+1}(n) \tag{15.8}$$

where the input vector $\mathbf{u}_{m+1}(n)$ of size $m+1$ is partitioned in the following way:

$$\mathbf{u}_{m+1}(n) = \begin{bmatrix} u(n) \\ \mathbf{u}_m(n-1) \end{bmatrix} \tag{15.9}$$

The tap-weight vector $\hat{\mathbf{w}}_{f,m}(n)$ of the forward predictor is the solution obtained by minimizing the sum of weighted forward *a posteriori* prediction-error squares for $1 \le i \le n$,

$$\mathscr{F}_m(n) = \sum_{i=1}^{n} \lambda^{n-i} |f_m(i)|^2 \tag{15.10}$$

Equivalently, the tap-weight vector $\mathbf{a}_m(n)$ of the prediction-error filter is the solution to the same minimization problem, subject to the constraint that the first element of $\mathbf{a}_m(n)$ equals unity, in accordance with Eq. (15.6).

Finally, using (13.35), we get the following recursion for updating the minimum value of the sum of weighted forward prediction-error squares (i.e., forward prediction-error energy):

$$\mathcal{F}_m(n) = \lambda \mathcal{F}_m(n-1) + \eta_m(n) f_m^*(n) \tag{15.11}$$

where the product term $\eta_m(n) f_m^*(n)$ is real valued.

15.2 ADAPTIVE BACKWARD LINEAR PREDICTION

Consider next the *backward linear predictor of order m*, depicted in Fig. 15.2(a), whose tap-weight vector $\hat{\mathbf{w}}_{b,m}(n)$ is optimized in the least-squares sense over the entire observation interval $1 \leq i \leq n$. Let $b_m(n)$ denote the backward prediction error produced by this predictor at time n in response to the tap-input vector $\mathbf{u}_m(n)$ of size m, as shown by

$$b_m(n) = u(n-m) - \hat{\mathbf{w}}_{b,m}^H(n)\mathbf{u}_m(n) \tag{15.12}$$

According to this definition, $u(n-m)$ plays the role of desired response for backward linear prediction, and

$$\mathbf{u}_m(n) = [u(n), u(n-1), \ldots, u(n-m+1)]^T$$

$$\hat{\mathbf{w}}_{b,m}(n) = [\hat{w}_{b,m,1}(n), \hat{w}_{b,m,2}(n), \ldots, \hat{w}_{b,m,m}(n)]^T$$

We refer to $b_m(n)$ as the *a posteriori backward prediction error* since its computation is based on the current value of the backward predictor's tap-weight vector, $\hat{\mathbf{w}}_{b,m}(n)$. Correspondingly, we may define the *backward a priori prediction error* as

$$\beta_m(n) = u(n-m) - \hat{\mathbf{w}}_{b,m}^H(n-1)\mathbf{u}_m(n) \tag{15.13}$$

the computation of which is based on the past value of the backward predictor's tap-weight vector, $\hat{\mathbf{w}}_{b,m}(n-1)$.

In Table 15.1 are also listed the correspondences between the quantities characterizing linear estimation in general and those characterizing backward linear prediction in particular. To write the recursions for adaptive backward linear prediction, we may again modify the RLS algorithm developed in Sections 13.3 and 13.4 in light of these correspondences. Thus, we deduce the following recursion for updating the tap-weight vector of the backward predictor:

$$\hat{\mathbf{w}}_{b,m}(n) = \hat{\mathbf{w}}_{b,m}(n-1) + \mathbf{k}_m(n)\beta_m^*(n) \tag{15.14}$$

where $\beta_m(n)$ is the backward *a priori* prediction error defined in Eq. (15.12), and $\mathbf{k}_m(n)$ is the current value of the *gain vector* defined by

$$\mathbf{k}_m(n) = \boldsymbol{\Phi}_m^{-1}(n)\mathbf{u}_m(n) \tag{15.15}$$

The matrix $\mathbf{\Phi}_m^{-1}(n)$ is the inverse of the correlation matrix $\mathbf{\Phi}_m(n)$ of the input data, with the latter matrix being defined by

$$\mathbf{\Phi}_m(n) = \sum_{i=1}^{n} \lambda^{n-i} \mathbf{u}_m(i) \mathbf{u}_m^H(i) \tag{15.16}$$

The description of the backward linear prediction problem just presented is in terms of a backward predictor characterized by the tap-weight vector $\hat{\mathbf{w}}_{b,m}(n)$. Equivalently, we may describe the problem in terms of a *backward prediction-error filter*, as depicted in Fig. 15.2(b). Let the prediction-error filter of order m be characterized by a tap-weight vector $\mathbf{c}_m(n)$, which is related to that of the backward predictor in Fig. 15.2(a) as follows:

$$\mathbf{c}_m(n) = \begin{bmatrix} -\hat{\mathbf{w}}_{b,m}(n) \\ 1 \end{bmatrix} \tag{15.17}$$

Thus, with an input vector $\mathbf{u}_{m+1}(n)$ of size $m + 1$, the backward *a posteriori* prediction and backward *a priori* prediction errors may be rewritten as follows, respectively:

$$b_m(n) = \mathbf{c}_m^H(n) \mathbf{u}_{m+1}(n) \tag{15.18}$$

and

$$\beta_m(n) = \mathbf{c}_m^H(n-1) \mathbf{u}_{m+1}(n) \tag{15.19}$$

In this case, the input vector $\mathbf{u}_{m+1}(n)$ is partitioned in the following way:

$$\mathbf{u}_{m+1}(n) = \begin{bmatrix} \mathbf{u}_m(n) \\ u(n-m) \end{bmatrix} \tag{15.20}$$

The tap-weight vector $\hat{\mathbf{w}}_{b,m}(n)$ of the backward predictor is obtained by minimizing the sum of weighted backward *a posteriori* prediction-error squares for $1 \leq i \leq n$,

$$\mathcal{R}_m(n) = \sum_{i=1}^{n} \lambda^{n-i} |b_m(i)|^2 \tag{15.21}$$

Equivalently, the tap-weight vector $\mathbf{c}_m(n)$ of the backward prediction-error filter is the solution to the same minimization problem, subject to the constraint that the last element of $\mathbf{c}_m(n)$ equals unity, in accordance with Eq. (15.17).

Also, using Eq. (13.35), we get the following recursion for updating the minimum value of the sum of weighted backward prediction-error squares (i.e., backward prediction-error energy):

$$\mathcal{R}_m(n) = \lambda \mathcal{R}_m(n-1) + \beta_m(n) b_m^*(n) \tag{15.22}$$

where the product term $\beta_m(n) b_m^*(n)$ is real valued.

In closing this discussion of the recursive least-squares prediction problem, it is of interest to note that in the case of backward prediction, the input vector $\mathbf{u}_{m+1}(n)$ is partitioned with the desired response $u(n-m)$ as the last entry, as shown in Eq. (15.20). On the other hand, in the case of forward linear prediction the input vector $\mathbf{u}_{m+1}(n)$ is parti-

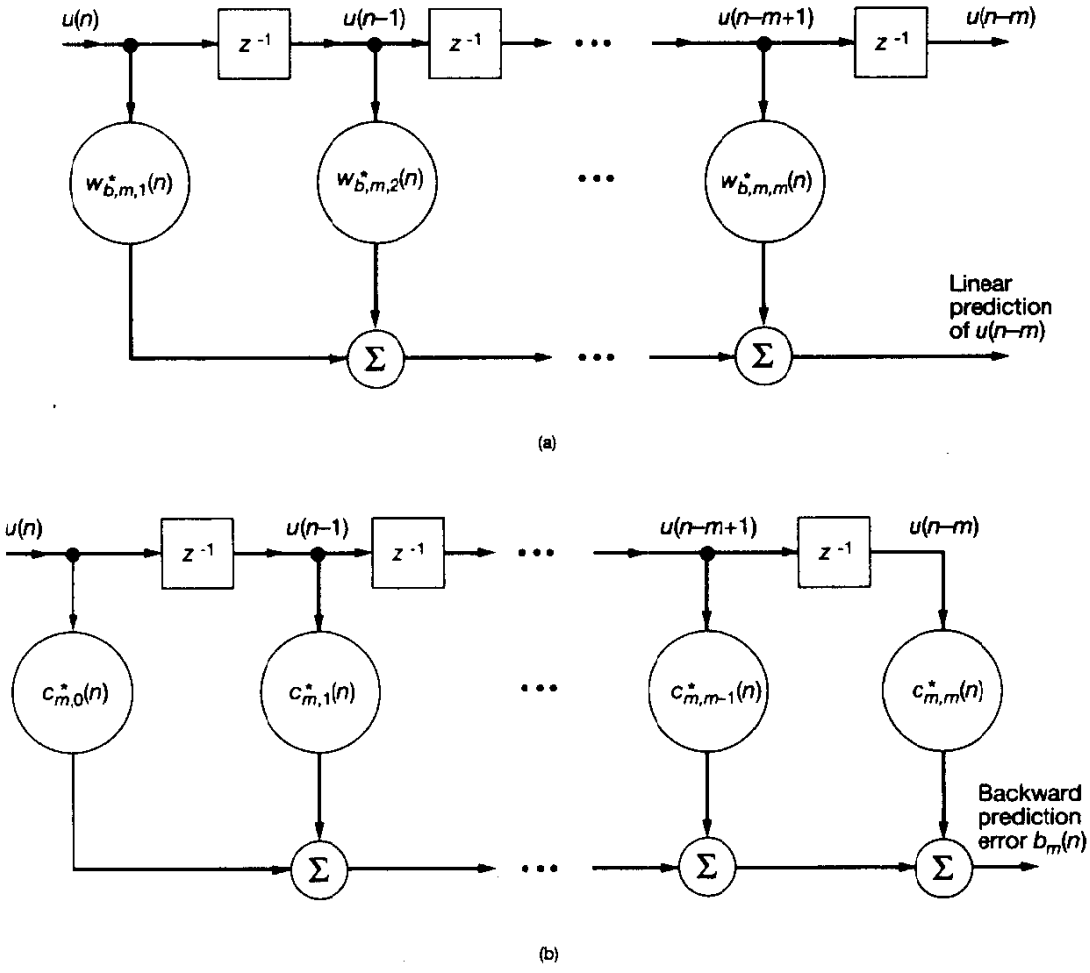


Figure 15.2 (a) Backward predictor of order m; (b) corresponding backward prediction-error filter.

tioned with the desired response $u(n)$ as the leading entry, as shown in Eq. (15.9). Note also that the update recursion for the tap-weight vector $\hat{\mathbf{w}}_{b,m}(n)$ of the backward linear predictor in Eq. (15.14) requires knowledge of the current value $\mathbf{k}_m(n)$ of the gain vector. On the other hand, the update recursion for the tap-weight vector $\hat{\mathbf{w}}_{f,m}(n)$ of the forward linear predictor in Eq. (15.3) requires knowledge of the old value $\mathbf{k}_m(n-1)$ of the gain vector.

15.3 CONVERSION FACTOR

The definition of the m-by-1 vector,

$$\mathbf{k}_m(n) = \boldsymbol{\Phi}_m^{-1}(n)\mathbf{u}_m(n)$$

may also be viewed as the solution of a special case of the normal equations for least-squares estimation. To be specific, the gain vector $\mathbf{k}_m(n)$ defines the tap-weight vector

of a transversal filter that contains m taps and that operates on the input data $u(1)$, $u(2)$, . . . , $u(n)$ to produce the least-squares estimate of a special desired response that equals

$$d(i) = \begin{cases} 1, & i = n \\ 0, & i = 1, 2, \ldots, n-1 \end{cases} \tag{15.23}$$

The n-by-1 vector whose elements equal the $d(i)$ of Eq. (15.23) is called the *first coordinate vector*. This vector has the property that its inner product with any time-dependent vector reproduces the upper or "most recent" element of that vector.

Substituting Eq. (15.23) in (13.10), we find that the m-by-1 cross-correlation vector $\mathbf{z}_m(n)$ between the m tap inputs of the transversal filter and the desired response equals $\mathbf{u}_m(n)$. This therefore confirms the gain vector $\mathbf{k}_m(n)$ as the special solution of the normal equations that arises when the desired response is defined by Eq. (15.23).

For the problem described here, define the *estimation error*

$$\begin{aligned} \gamma_m(n) &= 1 - \mathbf{k}_m^H(n)\mathbf{u}_m(n) \\ &= 1 - \mathbf{u}_m^H(n)\boldsymbol{\Phi}_m^{-1}(n)\mathbf{u}_m(n) \end{aligned} \tag{15.24}$$

The estimation error $\gamma_m(n)$ represents the output of a transversal filter whose tap-weight vector equals the gain vector $\mathbf{k}_m(n)$ and which is excited by the tap-input vector $\mathbf{u}_m(n)$, as depicted in Fig. 15.3. Since the filter output has the structure of a Hermitian form, it follows that the estimation error $\gamma_m(n)$ is a real-valued scalar. Moreover, $\gamma_m(n)$ has the important property that it is bounded by zero and one; that is

$$0 < \gamma_m(n) \leq 1 \tag{15.25}$$

This property is readily proved by substituting the recursion of Eq. (13.16) for the inverse matrix $\boldsymbol{\Phi}_m^{-1}(n-1)$ in Eq. (15.24), and then simplifying to obtain the result

$$\gamma_m(n) = \frac{1}{1 + \lambda^{-1}\mathbf{u}_m^H(n)\boldsymbol{\Phi}_m^{-1}(n-1)\mathbf{u}_m(n)} \tag{15.26}$$

The Hermitian form $\mathbf{u}_m^H(n)\boldsymbol{\Phi}_m^{-1}(n-1)\mathbf{u}_m(n) \geq 0$. Consequently, the estimation error $\gamma_m(n)$ is bounded as in (15.25).

It is noteworthy that $\gamma_m(n)$ also equals the sum of weighted error squares resulting from use of the transversal filter in Fig. 15.3, whose tap-weight vector equals the gain vector $\mathbf{k}_m(n)$, to obtain the least-squares estimate of the first coordinate vector (see Problem 1).

Other Useful Interpretations of $\gamma_m(n)$

Depending on the approach taken, the parameter $\gamma_m(n)$ may be given three other entirely different interpretations:

1. The parameter $\gamma_m(n)$ may be viewed as a *likelihood variable* (Lee et al., 1981). This interpretation follows from a statistical formulation of the tap-input vector in terms of its log-likelihood function, under the assumption that the tap-inputs have a joint Gaussian distribution (see Problem 11).

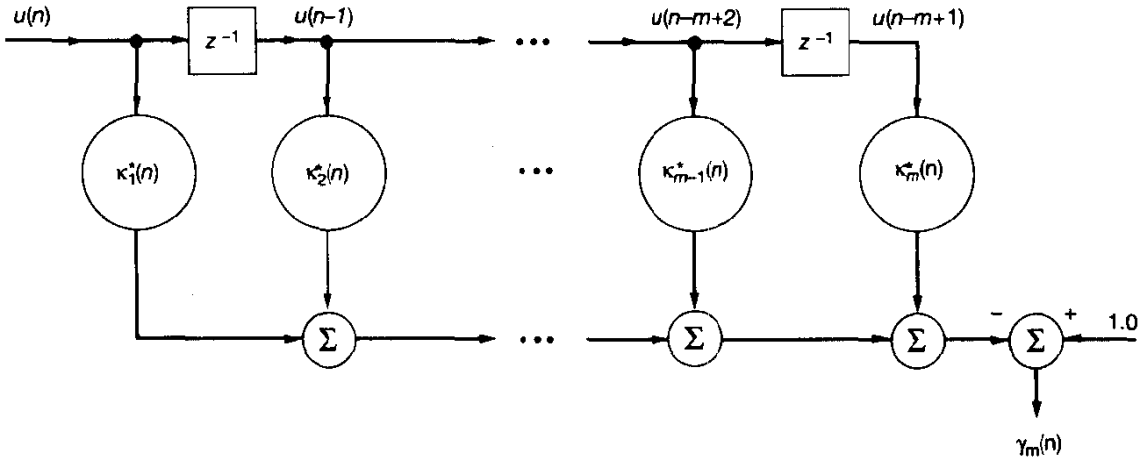


Figure 15.3 Transversal filter for defining the estimation error $\gamma_m(n)$.

2. The parameter $\gamma_m(n)$ may be interpreted as an *angle variable* (Lee et al., 1981; Carayannis et al., 1983). This interpretation follows from Eq. (15.24). In particular, following the discussion presented in Chapter 14, we may express the (positive) square root of $\gamma_m(n)$ as

$$\gamma_m^{1/2}(n) = \prod_{i=1}^{m} \cos\phi_i(n)$$

where $\phi_i(n)$ represents the angle of a plane (Givens) rotation [see Eq. (14.56)].

3. The parameter $\gamma_m(n)$ may be interpreted as a *conversion factor* (Carayannis et al., 1983). According to this interpretation, the availability of $\gamma_m(n)$ helps us determine the value of an *a posteriori* estimation error, given the value of the corresponding *a priori* estimation error.

It is this third interpretation that we pursue here. Indeed, it is because of this interpretation that we have adopted the terminology "conversion factor" as a description for $\gamma_m(n)$.

Three Kinds of Estimation Error

In linear least-squares estimation theory, there are three kinds of estimation error to be considered: the ordinary estimation error (involved in the estimation of some desired response), the forward prediction error, and the backward prediction error. Correspondingly, $\gamma_m(n)$ has three useful interpretations as a conversion factor, as described next.

1. For recursive least-squares estimation, we have

$$\gamma_m(n) = \frac{e_m(n)}{\xi_m(n)} \tag{15.27}$$

where $e_m(n)$ is the *a posteriori* estimation error and $\xi_m(n)$ is the *a priori* estimation error. This relation is readily proved by postmultiplying the Hermitian transposed sides of Eq. (13.25) by $\mathbf{u}_m(n)$, using Eq. (13.26) for the *a priori* estimation error $\xi_m(n)$, Eq. (13.27) for the *a posteriori* estimation error $e_m(n)$, and the first line of Eq. (15.24) for the variable $\gamma_m(n)$. Equation (15.27) states that given the *a priori* estimation error $\xi_m(n)$ as computed in the RLS algorithm, we may determine the corresponding value of the *a posteriori* estimation error $e_m(n)$ by multiplying $\xi_m(n)$ by $\gamma_m(n)$. We may therefore view $\xi_m(n)$ as a tentative value of the estimation error $e_m(n)$ and $\gamma_m(n)$ as the multiplicative correction.

2. For adaptive forward linear prediction, we have

$$\gamma_m(n-1) = \frac{f_m(n)}{\eta_m(n)} \tag{15.28}$$

This relation is readily proved by postmultiplying the Hermitian transposed sides of Eq. (15.3) by $\mathbf{u}_m(n-1)$, and then using the definitions of Eqs. (15.1), (15.2), (15.4), and (15.24). Equation (15.28) states that given the forward *a priori* prediction error $\eta_m(n)$, we may compute the forward *a posteriori* prediction error $f_m(n)$ by multiplying $\eta_m(n)$ by the delayed estimation error $\gamma_m(n-1)$. We may therefore view $\eta_m(n)$ as a tentative value for the forward *a posteriori* prediction error $f_m(n)$ and $\gamma_m(n-1)$ as the multiplicative correction.

3. For adaptive backward linear prediction, we have

$$\gamma_m(n) = \frac{b_m(n)}{\beta_m(n)} \tag{15.29}$$

This third relation is readily proved by postmultiplying the Hermitian transposed sides of Eq. (15.14) by $\mathbf{u}_m(n)$, and then using the definitions of Eqs. (15.12), (15.13), (15.15), and (15.24). Equation (15.29) states that given the backward *a priori* prediction error $\beta_m(n)$, we may compute the backward *a posteriori* prediction error $b_m(n)$ by multiplying $\beta_m(n)$ by the estimation error $\gamma_m(n)$. We may therefore view $\beta_m(n)$ as a tentative value for the backward prediction error $b_m(n)$ and $\gamma_m(n)$ as the multiplicative correction.

The discussion above points out the unique role of the variable $\gamma_m(n)$ in that it is the *common* factor (either in its regular or delayed form) in the conversion of an *a priori* estimation error into the corresponding *a posteriori* estimation error, be it in the context of ordinary estimation, forward prediction, or backward prediction. Accordingly, we may refer to $\gamma_m(n)$ as a *conversion factor*. Indeed, it is remarkable that through the use of this conversion factor we are able to compute the *a posteriori* errors $e_m(n)$, $f_m(n)$, and $b_m(n)$ at time n before the tap-weight vectors of the pertinent filters that produce them have been actually computed (Carayannis et al., 1983).

15.4 LEAST-SQUARES LATTICE PREDICTOR

Returning to the time-shifting property of the input data, we note from Eq. (15.20) that the input vector $\mathbf{u}_m(n)$ for a backward linear predictor of order m and the input vector $\mathbf{u}_{m+1}(n)$ for a backward linear predictor of order $m + 1$ have exactly the same first m entries. Likewise, we note from Eq. (15.9) that the input vector $\mathbf{u}_m(n - 1)$ for a forward linear predictor of order m and the input vector $\mathbf{u}_{m+1}(n)$ for a forward linear predictor of order $m + 1$ have exactly the same m last entries. These observations prompt us to raise the following fundamental question: In the course of increasing the prediction order from $m - 1$ to m, say, is it possible to carryover information gathered from previous computations pertaining to the prediction order $m - 1$? The answer to this question is an emphatic yes, and it is embodied in a modular filtering structure known as the *least-squares lattice predictor*.

To derive this important filtering structure and its algorithmic design, we propose to proceed as follows. In this section, we use the *principle of orthogonality* to derive the basic equations that characterize the least-squares lattice predictor. Then, under the unifying umbrella of *Kalman filter theory*, we derive various algorithms for its design in subsequent sections of the chapter.

To begin with, consider the situation depicted in Fig. 15.4, involving a pair of forward and backward prediction-error filters of order $m - 1$. They are both fed by the same input vector $\mathbf{u}_m(i)$. The forward prediction-error filter, characterized by the tap-weight vector $\mathbf{a}_{m-1}(n)$, produces $f_{m-1}(i)$ at its output. The backward prediction-error filter, characterized by the tap-weight vector $\mathbf{c}_{m-1}(n)$, produces $b_{m-1}(i)$ at its output. The input data $u(i)$ occupy the observation interval $1 \leq i \leq n$. The problem we wish to address may be stated as follows:

- Given the forward prediction error $f_{m-1}(i)$ and backward prediction error $b_{m-1}(i)$, determine their order-updated values $f_m(i)$ and $b_m(i)$, respectively, in a computationally efficient manner.

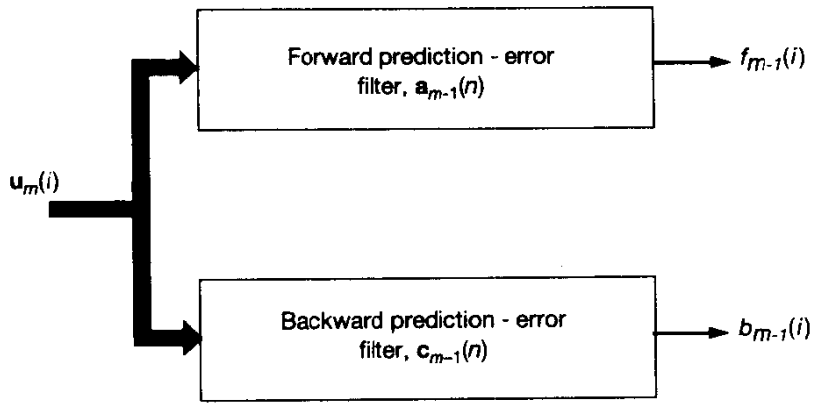


Figure 15.4 Setting the stage for formulating the least-squares lattice predictor.

By "computationally efficient," we mean the following: The input vector in Fig. 15.4 is enlarged by adding the past sample $u(i - m)$ and the prediction order is thereby increased by one; yet the computations involved in evaluating $f_{m-1}(i)$ and $b_{m-1}(i)$ remain completely intact.

The forward prediction error $f_{m-1}(i)$ is determined by the tap inputs $u(i)$, $u(i - 1)$, $\ldots$, $u(i - m + 1)$. The order-updated forward prediction error $f_m(i)$ requires knowledge of the additional tap input (i.e., past sample) $u(i - m)$. The backward prediction error $b_{m-1}(i)$ is determined by the same tap inputs as those involved in $f_{m-1}(i)$. If therefore we were to delay $b_{m-1}(i)$ by one time unit, the additional past sample $u(i - m)$ needed for computing $f_m(i)$ would be found in the composition of the *delayed* backward prediction error $b_{m-1}(i - 1)$. Thus, treating $b_{m-1}(i - 1)$ as the input to a *one-tap least-squares filter*, $f_{m-1}(i)$ as the desired response and $f_m(i)$ as the residual resulting from the least-squares estimation, we may write (see Fig. 15.5(a))

$$f_m(i) = f_{m-1}(i) + \kappa_{f,m}^*(n)\, b_{m-1}(i-1), \qquad i = 1, 2, \ldots, n \tag{15.30}$$

where $\kappa_{f,m}(n)$ is the filter's scalar coefficient to be determined. The format of Eq. (15.30) is similar to that of the corresponding order-update derived in Chapter 6 for a lattice predictor operating on stationary inputs. However, the formula for $\kappa_{f,m}(n)$ is different. For the determination of this coefficient, we turn to the principle of orthogonality discussed in Chapter 11 in the context of linear least-squares estimation. According to this principle, the estimation error (i.e., residual) produced by a linear least-squares filter in response to a set of inputs is orthogonal to each of those inputs in a time-averaged sense over the entire observation interval of interest. Thus, applying the principle of orthogonality to the input $b_{m-1}(i - 1)$ and residual $f_m(i)$ of the linear forward prediction problem postulated in Eq. (15.30), we get

$$\sum_{i=1}^{n} \lambda^{n-i} f_m^*(i)\, b_{m-1}(i-1) = 0 \tag{15.31}$$

Hence, substituting Eq. (15.30) in (15.31) and then solving for $\kappa_{f,m}(n)$, we get

$$\kappa_{f,m}(n) = -\frac{\sum_{i=1}^{n} \lambda^{n-i} f_{m-1}^*(i)\, b_{m-1}(i-1)}{\sum_{i=1}^{n} \lambda^{n-i} |b_{m-1}(i-1)|^2} \tag{15.32}$$

The denominator of this formula is the sum of weighted backward prediction-error squares for order $m - 1$:

$$\begin{aligned} \mathcal{B}_{m-1}(n-1) &= \sum_{i=1}^{n-1} \lambda^{n-1-i} |b_{m-1}(i)|^2 \\ &= \sum_{i=1}^{n} \lambda^{n-i} |b_{m-1}(i-1)|^2 \end{aligned} \tag{15.33}$$

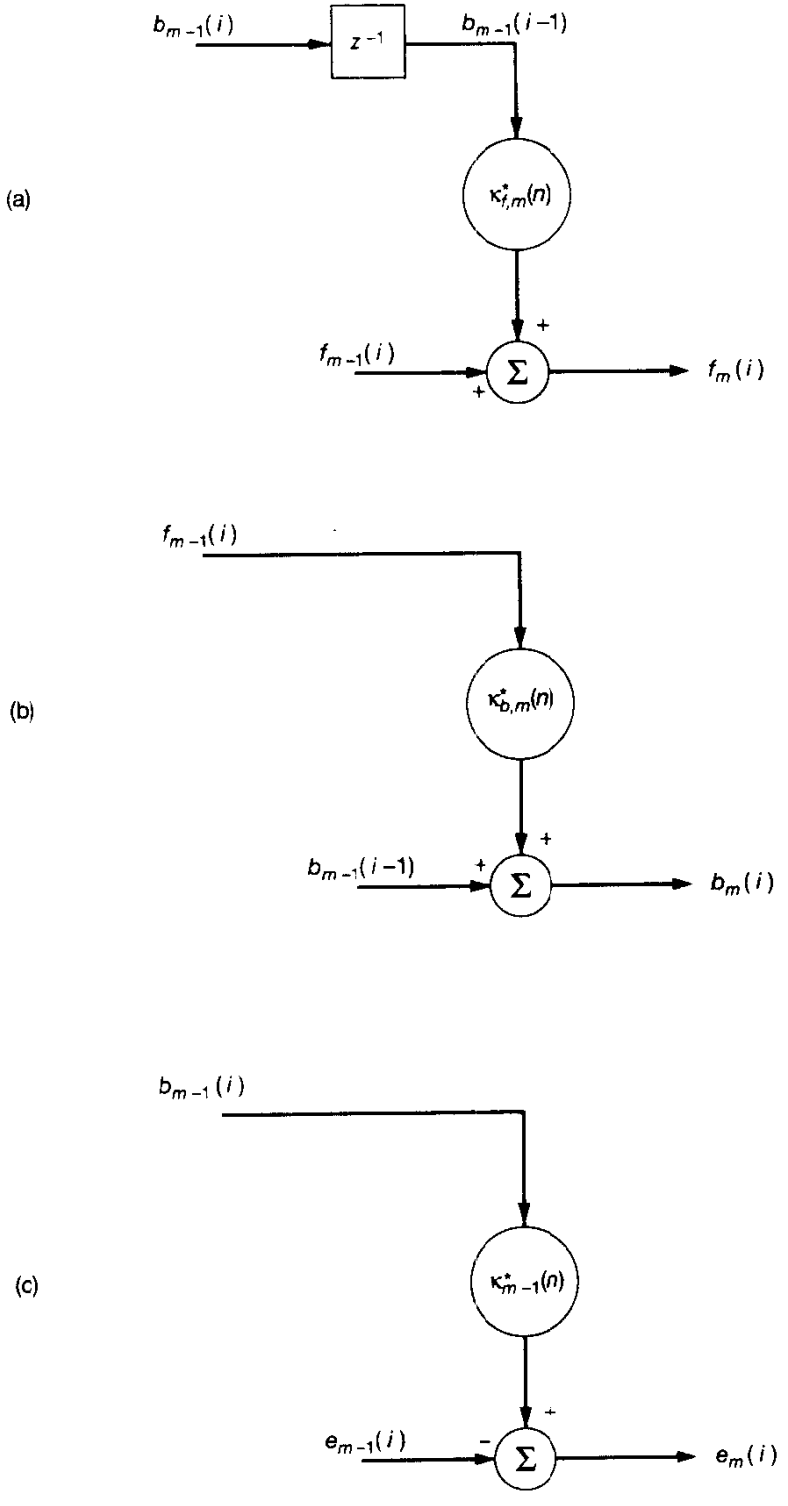


Figure 15.5 Single-coefficient linear combiners for (a) forward prediction, (b) backward prediction, and (c) joint-process estimation.

where, in the last line, we have used the fact that

$$b_{m-1}(0) = 0 \qquad \text{for all } m \geq 1$$

by virtue of prewindowing the input data. For the numerator of Eq. (15.32), we introduce a new definition:

$$\Delta_{m-1}(n) = \sum_{i=1}^{n} \lambda^{n-i} f_{m-1}^{*}(i)\, b_{m-1}(i-1) \tag{15.34}$$

Using the definitions of Eqs. (15.33) and (15.34) in Eq. (15.32), the formula for the scalar coefficient $\kappa_{f,m}(n)$ takes on the compact form:

$$\kappa_{f,m}(n) = -\frac{\Delta_{m-1}(n)}{\mathcal{B}_{m-1}(n-1)} \tag{15.35}$$

Consider next the issue of computing the order-updated backward prediction error $b_m(i)$. As before, we may work with the forward prediction error $f_{m-1}(i)$ and delayed backward prediction error $b_{m-1}(i-1)$, except for the fact that their filtering roles are now interchanged. Specifically, we have a one-tap least-squares filter with $f_{m-1}(i)$ acting as the input, $b_{m-1}(i-1)$ as the desired response, and $b_m(i)$ as the residual of the filtering process. That is, we write (see Fig. 15.5(b))

$$b_m(i) = b_{m-1}(i-1) + \kappa_{b,m}^{*}(n) f_{m-1}(i), \qquad i = 1, 2, \ldots, n \tag{15.36}$$

where $\kappa_{b,m}(n)$ is the filter's scalar coefficient to be determined. The format of Eq. (15.36) is also similar to the corresponding order update derived in Chapter 6 for a lattice predictor operating on stationary inputs. However, the formula for $\kappa_{b,m}(n)$ is different. In particular, we determine $\kappa_{b,m}(n)$ by applying the principle of orthogonality to the input $f_{m-1}(i)$ and residual $b_m(i)$ in the backward prediction problem postulated in Eq. (15.36), and thus write

$$\sum_{i=1}^{n} \lambda^{n-i} b_m^{*}(i) f_{m-1}(i) = 0 \tag{15.37}$$

Hence, substituting Eq. (15.36) in (15.37) and then solving for $\kappa_{b,m}(n)$, we get

$$\kappa_{b,m}(n) = -\frac{\displaystyle\sum_{i=1}^{n} \lambda^{n-i} b_{m-1}^{*}(i-1) f_{m-1}(i)}{\displaystyle\sum_{i=1}^{n} \lambda^{n-i} |f_{m-1}(i)|^2} \tag{15.38}$$

The numerator of this formula is the complex conjugate of the quantity $\Delta_{m-1}(n)$ defined in Eq. (15.34). The denominator is recognized as the sum of weighted forward prediction-error squares for order $m-1$:

$$\mathcal{F}_{m-1}(n) = \sum_{i=1}^{n} \lambda^{n-i} |f_{m-1}(i)|^2 \tag{15.39}$$

Accordingly, we may recast the formula of Eq. (15.38) in the compact form

$$\kappa_{b,m}(n) = -\frac{\Delta^*_{m-1}(n)}{\mathcal{F}_{m-1}(n)} \tag{15.40}$$

The results described in Eqs. (15.30) and (15.36) are basic to the least-squares lattice predictor. For their physical interpretation, define the n-by-1 prediction-error vectors:

$$\mathbf{f}_m(n) = [f_m(1), f_m(2), \ldots, f_m(n)]^T$$

$$\mathbf{b}_m(n) = [b_m(1), b_m(2), \ldots, b_m(n)]^T$$

$$\mathbf{b}_m(n-1) = [0, b_m(1), \ldots, b_m(n-1)]^T$$

where prediction order $m = 0, 1, 2, \ldots$. Then, on the basis of Eqs. (15.30) and (15.36) we may make the following statements in the terminology of *projection* theory:

- The result of projecting the vector $\mathbf{f}_{m-1}(n)$ onto $\mathbf{b}_{m-1}(n-1)$ is represented by the residual vector $\mathbf{f}_m(n)$; the *forward reflection coefficient* $\kappa_{f,m}(n)$ is the parameter needed to do this projection.
- The result of projecting the vector $\mathbf{b}_{m-1}(n-1)$ onto $\mathbf{f}_{m-1}(n)$ is represented by the residual vector $\mathbf{b}_m(n)$; the *backward reflection coefficient* $\kappa_{b,m}(n)$ is the parameter needed to do this second projection.

To put the finishing touch to this part of the discussion, we evaluate Eqs. (15.30) and (15.36) for the end of the observation interval $i = n$. We thus have the following pair of interrelated order-update recursions:

$$f_m(n) = f_{m-1}(n) + \kappa^*_{f,m}(n) b_{m-1}(n-1) \tag{15.41}$$

$$b_m(n) = b_{m-1}(n-1) + \kappa^*_{b,m}(n) f_{m-1}(n) \tag{15.42}$$

where $m = 1, 2, \ldots, M$, and M is the *final prediction order*. When $m = 0$ there is no prediction being preformed on the input data; this corresponds to the *initial conditions* described by

$$f_0(n) = b_0(n) = u(n) \tag{15.43}$$

where $u(n)$ is the input datum at time n. Thus, as we vary the prediction order m from zero all the way up to the final value M, we get the multistage least-squares lattice predictor of Fig. 15.6, whose number of stages equals M. An important feature of the least-squares lattice predictor described herein is its *modular* structure, the implication which is that the computational complexity scales *linearly* with the prediction order.

Least-Squares Lattice Version of the Levinson–Durbin Recursion

The forward prediction error $f_m(n)$ and backward prediction error $b_m(n)$ are defined by Eqs. (15.7) and (15.18), reproduced for convenience of presentation on the next page:

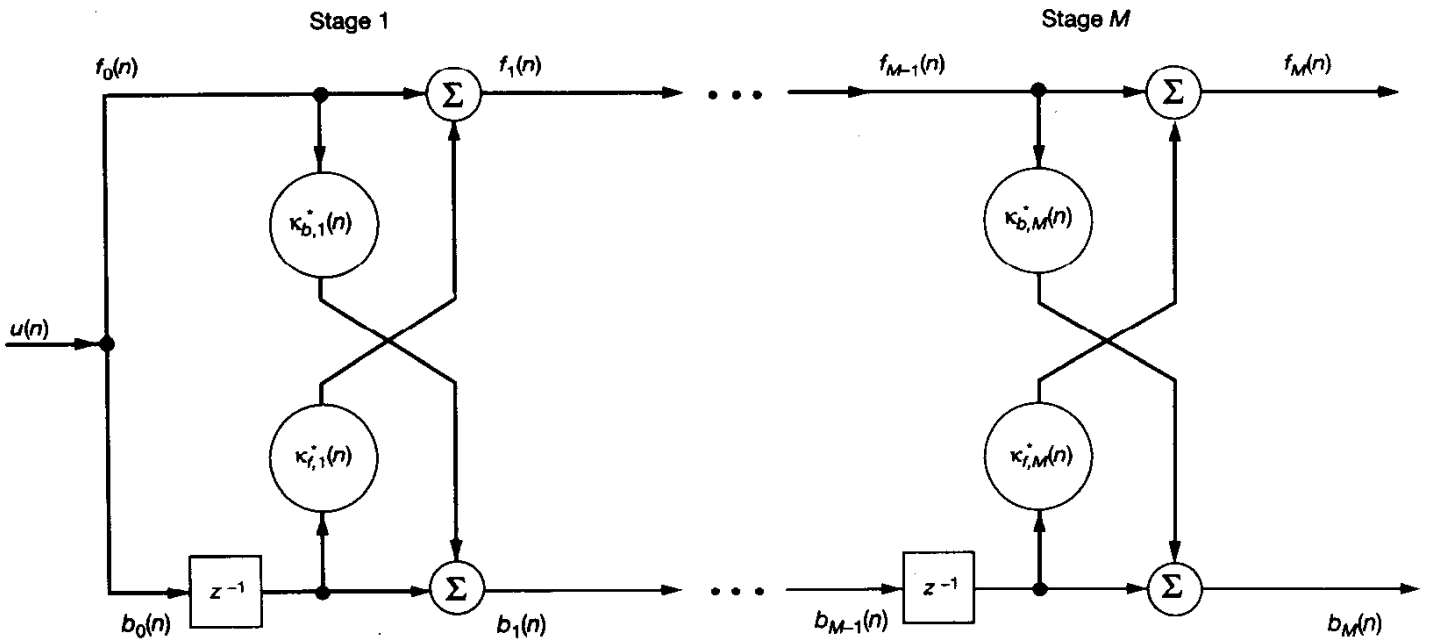


Figure 15.6 Multistage lattice predictor.

$$f_m(n) = \mathbf{a}_m^H(n)\,\mathbf{u}_{m+1}(n)$$

$$b_m(n) = \mathbf{c}_m^H(n)\,\mathbf{u}_{m+1}(n)$$

where $\mathbf{a}_m(n)$ and $\mathbf{c}_m(n)$ are the tap-weight vectors of the corresponding forward and backward prediction-error filters, respectively. The forward prediction error $f_{m-1}(n)$ and delayed backward prediction error $b_{m-1}(n-1)$, pertaining to a lower prediction order, are defined as follows, respectively:

$$\begin{aligned} f_{m-1}(n) &= \mathbf{a}_{m-1}^H(n)\,\mathbf{u}_m(n) \\ &= \begin{bmatrix} \mathbf{a}_{m-1}(n) \\ 0 \end{bmatrix}^H \begin{bmatrix} \mathbf{u}_m(n) \\ u(n-m) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{a}_{m-1}(n) \\ 0 \end{bmatrix}^H \mathbf{u}_{m+1}(n) \end{aligned}$$

$$\begin{aligned} b_{m-1}(n-1) &= \mathbf{c}_{m-1}^H(n-1)\,\mathbf{u}_m(n-1) \\ &= \begin{bmatrix} 0 \\ \mathbf{c}_{m-1}(n-1) \end{bmatrix}^H \begin{bmatrix} u(n) \\ \mathbf{u}_m(n-1) \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ \mathbf{c}_{m-1}(n-1) \end{bmatrix}^H \mathbf{u}_{m+1}(n) \end{aligned}$$

The four prediction errors defined above share a common input vector, namely $\mathbf{u}_{m+1}(n)$. Therefore, substituting their defining equations into Eqs. (15.41) and (15.42) and then comparing terms on the two sides of the resultants, we deduce the following pair of order updates:

$$\mathbf{a}_m(n) = \begin{bmatrix} \mathbf{a}_{m-1}(n) \\ 0 \end{bmatrix} + \kappa_{f,m}(n) \begin{bmatrix} 0 \\ \mathbf{c}_{m-1}(n-1) \end{bmatrix} \tag{15.44}$$

$$\mathbf{c}_m(n) = \begin{bmatrix} 0 \\ \mathbf{c}_{m-1}(n) \end{bmatrix} + \kappa_{b,m}(n) \begin{bmatrix} \mathbf{a}_{m-1}(n) \\ 0 \end{bmatrix} \tag{15.45}$$

where $m = 1, 2, \ldots, M$. Equations (15.44) and (15.45) may be viewed as the least-squares version of the *Levinson–Durbin recursion* discussed in Chapter 6. Recognizing that the last element of $\mathbf{c}_{m-1}(n-1)$ and the first element of $\mathbf{a}_{m-1}(n)$ equal unity, by definition, we readily see from Eqs. (15.44) and (15.45) that

$$\kappa_{f,m}(n) = a_{m,m}(n) \tag{15.46}$$

$$\kappa_{b,m}(n) = c_{m,0}(n) \tag{15.47}$$

where $a_{m,m}(n)$ is the last element of the vector $\mathbf{a}_m(n)$, and $c_{m,0}(n)$ is the first element of the vector $\mathbf{c}_m(n)$. Unlike the situation described in Chapter 6 for a stationary environment, we generally find that in a least-squares lattice predictor:

$$\kappa_{f,m}(n) \neq \kappa_{b,m}^*(n)$$

In any event, the order updates of Eqs. (15.44) and (15.49) reveal a remarkable property of a least-squares lattice predictor of final order M: In an implicit sense, it embodies a chain of forward prediction-error filters of order 1,2, . . . , M, and a chain of backward prediction-error filters 1, 2, . . . , M, all in one modular structure.

A Time-Update Recursion for $\Delta_{m-1}(n)$

From Eqs (15.35) and (15.40) we see that the reflection coefficients $\kappa_{f,m}(n)$ and $\kappa_{b,m}(n)$ of the least-squares lattice predictor are uniquely determined by three quantities: $\Delta_{m-1}(n)$, $\mathcal{F}_{m-1}(n)$, and $\mathcal{B}_{m-1}(n-1)$. Equations (15.11) and (15.22) provide us with the means to time-update the latter two quantities. We need the corresponding time-update recursion for $\Delta_{m-1}(n)$. A derivation of this recursion is presented in what follows.

Let $\mathbf{\Phi}_{m+1}(n)$ denote the $(m+1)$-by-$(m+1)$ correlation matrix of the tap-input vector $\mathbf{u}_{m+1}(i)$ applied to the forward prediction-error filter of order m, where $1 \le i \le n$. Let $\mathbf{a}_m(n)$ denote the tap-weight vector of this filter, and $\mathcal{F}_m(n)$ denote the corresponding sum of weighted prediction-error squares. We may characterize this filter by the *augmented normal equations* (see Chapter 11)

$$\mathbf{\Phi}_{m+1}(n)\mathbf{a}_m(n) = \begin{bmatrix} \mathcal{F}_m(n) \\ \mathbf{0}_m \end{bmatrix} \tag{15.48}$$

where $\mathbf{0}_m$ is the m-by-1 null vector. The correlation matrix $\mathbf{\Phi}_{m+1}(n)$ may be partitioned in two different ways, depending on how we interpret the first or last element of the tap-input vector $\mathbf{u}_{m+1}(i)$. The form of partitioning that we like to use first is the one that enables us to relate the tap-weight vector $\mathbf{a}_m(n)$, pertaining to prediction order m, to the tap-weight vector $\mathbf{a}_{m-1}(n)$, pertaining to prediction order $m-1$. This aim is realized by using

$$\mathbf{\Phi}_{m+1}(n) = \left[\begin{array}{c|c} \mathbf{\Phi}_m(n) & \boldsymbol{\phi}_2(n) \\ \hline \boldsymbol{\phi}_2^H(n) & \mathcal{U}_2(n) \end{array}\right] \tag{15.49}$$

where $\mathbf{\Phi}_m(n)$ is the m-by-m correlation matrix of the tap-input vector $\mathbf{u}_m(i)$, $\boldsymbol{\phi}_2(n)$ is the m-by-1 cross-correlation vector between $\mathbf{u}_m(i)$ and $u(i-m)$, and $\mathcal{U}_2(n)$ is the sum of weighted squared values of the input $u(i-m)$ for $1 \le i \le n$. Note that $\mathcal{U}_2(n)$ is zero for $n - m \le 0$. We postmultiply both sides of Eq. (15.49) by an $(m+1)$-by-1 vector whose first m elements are defined by the vector $\mathbf{a}_{m-1}(n)$ and whose last element equals zero. We may thus write

$$\begin{aligned} \mathbf{\Phi}_{m+1}(n)\left[\begin{array}{c} \mathbf{a}_{m-1}(n) \\ \hline 0 \end{array}\right] &= \left[\begin{array}{c|c} \mathbf{\Phi}_m(n) & \boldsymbol{\phi}_2(n) \\ \hline \boldsymbol{\phi}_2^H(n) & \mathcal{U}_2(n) \end{array}\right]\left[\begin{array}{c} \mathbf{a}_{m-1}(n) \\ \hline 0 \end{array}\right] \\ &= \begin{bmatrix} \mathbf{\Phi}_m(n)\mathbf{a}_{m-1}(n) \\ \boldsymbol{\phi}_2^H(n)\mathbf{a}_{m-1}(n) \end{bmatrix} \end{aligned} \tag{15.50}$$

Both $\mathbf{\Phi}_m(n)$ and $\mathbf{a}_{m-1}(n)$ have the same time argument n. Furthermore, in the first line of Eq. (15.50), they are both positioned in such a way that when the matrix multiplication is performed, $\mathbf{\Phi}_m(n)$ becomes postmultiplied by $\mathbf{a}_{m-1}(n)$. For a forward prediction-error fil-

ter of order $m-1$, evaluated at time n, the set of augmented normal equations defined in Eq. (15.48) takes the form

$$\boldsymbol{\Phi}_m(n)\mathbf{a}_{m-1}(n) = \begin{bmatrix} \mathcal{F}_{m-1}(n) \\ \mathbf{0}_{m-1} \end{bmatrix}$$

Define the scalar

$$\Delta_{m-1}(n) = \boldsymbol{\phi}_2^H(n)\mathbf{a}_{m-1}(n) \tag{15.51}$$

which is the same parameter defined previously in Eq. (15.34); see Problem 2. Accordingly, we may rewrite Eq. (15.50) as

$$\boldsymbol{\Phi}_{m+1}(n)\begin{bmatrix} \mathbf{a}_{m-1}(n) \\ 0 \end{bmatrix} = \begin{bmatrix} \mathcal{F}_{m-1}(n) \\ \mathbf{0}_{m-1} \\ \Delta_{m-1}(n) \end{bmatrix} \tag{15.52}$$

Consider next the backward prediction-error filter of order m. Let $\mathbf{c}_m(n)$ denote its tap-weight vector, and $\mathcal{B}_m(n)$ denote the corresponding sum of weighted prediction-error squares. This filter is characterized by the augmented normal equations written in the matrix form:

$$\boldsymbol{\Phi}_{m+1}(n)\mathbf{c}_m(n) = \begin{bmatrix} \mathbf{0}_m \\ \mathcal{B}_m(n) \end{bmatrix} \tag{15.53}$$

where $\boldsymbol{\Phi}_{m+1}(n)$ is as defined previously, and $\mathbf{0}_m$ is the m-by-1 null vector. This time we use the other partitioned form of the correlation matrix $\boldsymbol{\Phi}_m(n)$, as shown by

$$\boldsymbol{\Phi}_{m+1}(n) = \left[\begin{array}{c|c} \mathcal{U}_1(n) & \boldsymbol{\phi}_1^H(n) \\ \hline \boldsymbol{\phi}_1(n) & \boldsymbol{\Phi}_m(n-1) \end{array}\right] \tag{15.54}$$

where $\mathcal{U}_1(n)$ is the sum of weighted squared values of the input $u(i)$ for the time interval $1 \le i \le n$, $\boldsymbol{\phi}_1(n)$ is the m-by-1 cross-correlation vector between $u(i)$ and the tap-input vector $\mathbf{u}_m(i-1)$, and $\boldsymbol{\Phi}_m(n-1)$ is the m-by-m correlation matrix of $\mathbf{u}_m(i-1)$. Correspondingly, we postmultiply $\boldsymbol{\Phi}_{m+1}(n)$ by an $(m+1)$-by-1 vector whose first element is zero and whose m remaining elements are defined by the tap-weight vector $\mathbf{c}_{m-1}(n-1)$ that pertains to a backward prediction-error filter of order $m-1$. We may thus write

$$\begin{aligned} \boldsymbol{\Phi}_{m+1}(n)\begin{bmatrix} 0 \\ \mathbf{c}_{m-1}(n-1) \end{bmatrix} &= \begin{bmatrix} \mathcal{U}_1(n) & \boldsymbol{\phi}_1^H(n) \\ \boldsymbol{\phi}_1(n) & \boldsymbol{\Phi}_m(n-1) \end{bmatrix}\begin{bmatrix} 0 \\ \mathbf{c}_{m-1}(n-1) \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{\phi}_1^H(n)\mathbf{c}_{m-1}(n-1) \\ \boldsymbol{\Phi}_m(n-1)\mathbf{c}_{m-1}(n-1) \end{bmatrix} \end{aligned} \tag{15.55}$$

Both $\boldsymbol{\Phi}_m(n-1)$ and $\mathbf{c}_{m-1}(n-1)$ have the same time argument, $n-1$. Also, they are both positioned in the first line of Eq. (15.55) in such a way that, when the matrix multiplication is performed, $\boldsymbol{\Phi}_{m-1}(n-1)$ becomes postmultiplied by $\mathbf{c}_{m-1}(n-1)$. For a backward prediction-error filter of order $m-1$, evaluated at time $n-1$, the set of augmented normal equations in Eq. (15.53) takes the form

$$\mathbf{\Phi}_m(n-1)\mathbf{c}_{m-1}(n-1) = \begin{bmatrix} \mathbf{0}_{m-1} \\ \mathcal{B}_{m-1}(n-1) \end{bmatrix}$$

Define the second scalar

$$\Delta'_{m-1}(n) = \mathbf{\Phi}_1^H(n)\mathbf{c}_{m-1}(n-1) \tag{15.56}$$

where the prime is intended to distinguish this new parameter from $\Delta_{m-1}(n)$. Accordingly, we may rewrite Eq. (15.55) as

$$\mathbf{\Phi}_{m+1}(n)\begin{bmatrix} 0 \\ \mathbf{c}_{m-1}(n-1) \end{bmatrix} = \begin{bmatrix} \Delta'_{m-1}(n) \\ \mathbf{0}_{m-1} \\ \mathcal{B}_{m-1}(n-1) \end{bmatrix} \tag{15.57}$$

The parameters $\Delta_{m-1}(n)$ and $\Delta'_{m-1}(n)$, defined by Eqs. (15.51) and (15.56), respectively, are in actual fact the complex conjugate of one another; that is,

$$\Delta'_{m-1}(n) = \Delta^*_{m-1}(n) \tag{15.58}$$

where $\Delta^*_{m-1}(n)$ is the complex conjugate of $\Delta_{m-1}(n)$. We prove this relation in three stages:

1. We premultiply both sides of Eq. (15.52) by the row vector

$$[0, \mathbf{c}^H_{m-1}(n-1)]$$

where the superscript H denotes Hermitian transposition. The result of this matrix multiplication is the scalar

$$\begin{aligned}[0, \mathbf{c}^H_{m-1}(n-1)]\mathbf{\Phi}_{m+1}(n)\begin{bmatrix} \mathbf{a}_{m-1}(n) \\ 0 \end{bmatrix} &= [0, \mathbf{c}^H_{m-1}(n-1)]\begin{bmatrix} \mathcal{F}_{m-1}(n) \\ \mathbf{0}_{m-1} \\ \Delta_{m-1}(n) \end{bmatrix} \\ &= \Delta_{m-1}(n)\end{aligned} \tag{15.59}$$

where we have used the fact that the last element of $\mathbf{c}_{m-1}(n-1)$ equals unity.

2. We apply Hermitian transposition to both sides of Eq. (15.57), and use the Hermitian property of the correlation matrix $\mathbf{\Phi}_{m+1}(n)$, thereby obtaining

$$[0, \mathbf{c}^H_{m-1}(n-1)]\mathbf{\Phi}_{m+1}(n) = [\Delta'^*_{m-1}(n), \mathbf{0}^T_{m-1}, \mathcal{B}_{m-1}(n-1)]$$

where $\Delta'^*_{m-1}(n)$ is the complex conjugate of $\Delta'_{m-1}(n)$, and $\mathcal{B}_{m-1}(n-1)$ is real valued. Next we use this relation to evaluate the scalar

$$\begin{aligned}&[0, \mathbf{c}^H_{m-1}(n-1)]\mathbf{\Phi}_{m+1}(n)\begin{bmatrix} \mathbf{a}_{m-1}(n) \\ 0 \end{bmatrix} \\ &\quad = [\Delta'^*_{m-1}(n), \mathbf{0}^T_{m-1}, \mathcal{B}_{m-1}(n-1)]\begin{bmatrix} \mathbf{a}_{m-1}(n) \\ 0 \end{bmatrix} \\ &\quad = \Delta'^*_{m-1}(n)\end{aligned} \tag{15.60}$$

where we have used the fact that the first element of $\mathbf{a}_{m-1}(n)$ equals unity.

3. Comparison of Eqs. (15.59) and (15.60) immediately yields the relation of Eq. (15.58) between the parameters $\Delta_{m-1}(n)$ and $\Delta'_{m-1}(n)$.

We are now equipped with the relations needed to derive the desired time-update for recursive computation of the parameter $\Delta_{m-1}(n)$.

Consider the m-by-1 tap-weight vector $\mathbf{a}_{m-1}(n-1)$ that pertains to a forward prediction-error filter of order $m-1$, evaluated at time $n-1$. The reason for considering time $n-1$ will become apparent presently. Since the leading element of the vector $\mathbf{a}_{m-1}(n-1)$ equals unity, we may express $\Delta_{m-1}(n)$ as follows [see Eqs. (15.58) and (15.60)]:

$$\Delta_{m-1}(n) = [\Delta_{m-1}(n), \mathbf{0}_{m-1}^T, \mathcal{B}_{m-1}(n-1)]\begin{bmatrix}\mathbf{a}_{m-1}(n-1)\\ 0\end{bmatrix} \tag{15.61}$$

Taking the Hermitian transpose of both sides of Eq. (15.57), recognizing the Hermitian property of $\mathbf{\Phi}_{m+1}(n)$, and using the relation of Eq. (15.58), we get

$$[0, \mathbf{c}_{m-1}^H(n-1)]\,\mathbf{\Phi}_{m+1}(n) = [\Delta_{m-1}(n), \mathbf{0}_{m-1}^T, \mathcal{B}_{m-1}(n-1)] \tag{15.62}$$

Hence, substitution of Eq. (15.62) in (15.61) yields

$$\Delta_{m-1}(n) = [0, \mathbf{c}_{m-1}^H(n-1)]\mathbf{\Phi}_{m+1}(n)\begin{bmatrix}\mathbf{a}_{m-1}(n-1)\\ 0\end{bmatrix} \tag{15.63}$$

But the correlation matrix $\mathbf{\Phi}_{m+1}(n)$ may be time-updated as follows [see Eq. (13.12)]:

$$\mathbf{\Phi}_{m+1}(n) = \lambda\mathbf{\Phi}_{m+1}(n-1) + \mathbf{u}_{m+1}(n)\mathbf{u}_{m+1}^H(n) \tag{15.64}$$

Accordingly, we may use this relation for $\mathbf{\Phi}_{m+1}(n)$ to rewrite Eq. (15.63) as

$$\begin{aligned}\Delta_{m-1}(n) = {} & \lambda[0, \mathbf{c}_{m-1}^H(n-1)]\mathbf{\Phi}_{m+1}(n-1)\begin{bmatrix}\mathbf{a}_{m-1}(n-1)\\ 0\end{bmatrix}\\ & + [0, \mathbf{c}_{m-1}^H(n-1)]\mathbf{u}_{m+1}(n)\mathbf{u}_{m+1}^H(n)\begin{bmatrix}\mathbf{a}_{m-1}(n-1)\\ 0\end{bmatrix}\end{aligned} \tag{15.65}$$

Next we recognize from the definition of forward *a priori* prediction error that

$$\begin{aligned}\mathbf{u}_{m+1}^H(n)\begin{bmatrix}\mathbf{a}_{m-1}(n-1)\\ 0\end{bmatrix} &= [\mathbf{u}_m^H(n), u^*(n-m)]\begin{bmatrix}\mathbf{a}_{m-1}(n-1)\\ 0\end{bmatrix}\\ &= \mathbf{u}_m^H(n)\mathbf{a}_{m-1}(n-1)\\ &= \eta_{m-1}^*(n)\end{aligned} \tag{15.66}$$

and from the definition of the backward *a posteriori* prediction error that

$$\begin{aligned}[0, \mathbf{c}_{m-1}^H(n-1)]\mathbf{u}_{m+1}(n) &= [0, \mathbf{c}_{m-1}^H(n-1)]\begin{bmatrix}u(n)\\ \mathbf{u}_m(n-1)\end{bmatrix}\\ &= \mathbf{c}_{m-1}^H(n-1)\mathbf{u}_m(n-1)\\ &= b_{m-1}(n-1)\end{aligned} \tag{15.67}$$

Also, by substituting $n - 1$ for n in Eq. (15.52), we have

$$\mathbf{\Phi}_{m+1}(n-1)\begin{bmatrix}\mathbf{a}_{m-1}(n-1)\\ 0\end{bmatrix} = \begin{bmatrix}\mathcal{F}_{m-1}(n-1)\\ \mathbf{0}_{m-1}\\ \Delta_{m-1}(n-1)\end{bmatrix}$$

Hence, using this relation and the fact that the last element of the tap-weight vector $\mathbf{c}_{m-1}(n-1)$, pertaining to the backward prediction-error filter, equals unity, we may write the first term on the right-hand side of Eq. (15.65), except for λ, as

$$\begin{aligned}[0, \mathbf{c}_{m-1}^H(n-1)]\mathbf{\Phi}_{m+1}(n-1)&\begin{bmatrix}\mathbf{a}_{m-1}(n-1)\\ 0\end{bmatrix}\\ &= [0, \mathbf{c}_{m-1}^H(n-1)]\begin{bmatrix}\mathcal{F}_{m-1}(n-1)\\ \mathbf{0}_{m-1}\\ \Delta_{m-1}(n-1)\end{bmatrix}\\ &= \Delta_{m-1}(n-1)\end{aligned} \tag{15.68}$$

Finally, substituting Eqs. (15.66), (15.67), and (15.68) in (15.65), we may express the time-update recursion for $\Delta_{m-1}(n)$ simply as

$$\Delta_{m-1}(n) = \lambda\Delta_{m-1}(n-1) + b_{m-1}(n-1)\eta_{m-1}^*(n) \tag{15.69}$$

which is the desired result. Note that Eq. (15.69) for $\Delta_{m-1}(n)$ is similar to Eq. (15.11) for $\mathcal{F}_m(n)$ and Eq. (15.22) for $\mathcal{B}_m(n)$ in that in each of these three updates, the correction term involves the product of an *a posteriori* and an *a priori* prediction error.

Exact Decoupling Property of the Least-Squares Lattice Predictor

Another important property of a least-squares lattice predictor consisting of m stages is that the backward prediction errors $b_0(n), b_1(n), \ldots, b_m(n)$ produced at the various stages of the predictor are *uncorrelated* (orthogonal) with each other in a time-averaged sense at all instants of time. In other words, the least-squares lattice predictor transforms a correlated input data sequence $\{u(n), u(n-1), \ldots, u(n-m)\}$ into a new sequence of uncorrelated backward prediction errors, as shown by

$$\{u(n), u(n-1), \ldots, u(n-m)\} \rightleftharpoons \{b_0(n), b_1(n), \ldots, b_m(n)\} \tag{15.70}$$

The transformation shown here is *reciprocal*, which means that the least-squares lattice predictor preserves the full information content of the input data.

Consider a backward prediction-error filter of order m. Let the $(m+1)$-by-1 tap-weight vector of the filter, optimized in the least-squares sense over the time interval $1 \le i \le n$, be denoted by $\mathbf{c}_m(n)$. In expanded form, we have

$$\mathbf{c}_m(n) = [c_{m,m}(n), c_{m,m-1}(n), \ldots, 1]^T$$

Let $b_m(i)$ denote the backward *a posteriori* prediction error produced at the output of the filter in response to the $(m + 1)$-by-1 input vector $\mathbf{u}_{m+1}(i)$. The expanded form of the input vector is shown by

$$\mathbf{u}_{m+1}(i) = [u(i), u(i-1), \ldots, u(i-m)]^T, \qquad i > m$$

We may thus express the error $b_m(i)$ as

$$\begin{aligned} b_m(i) &= \mathbf{c}_m^H(n)\mathbf{u}_{m+1}(i) \\ &= \sum_{k=0}^{m} c_{m,k}^*(n)u(i-m+k) \qquad \begin{matrix} m < i \le n \\ m = 0, 1, 2, \ldots \end{matrix} \end{aligned} \tag{15.71}$$

Let $\mathbf{b}_{m+1}(i)$ denote the $(m + 1)$-by-1 backward *a posteriori* prediction-error vector, defined by

$$\mathbf{b}_{m+1}(i) = [b_0(i), b_1(i), \ldots, b_m(i)]^T, \qquad \begin{matrix} m < i \le n \\ m = 0, 1, 2, \ldots \end{matrix}$$

Substituting Eq. (15.71) into this vector, we may express the transformation of the input data into the corresponding set of backward *a posteriori* prediction errors as follows:

$$\mathbf{b}_{m+1}(i) = \mathbf{L}_m(n)\mathbf{u}_{m+1}(i) \tag{15.72}$$

where the $(m + 1)$-by-$(m + 1)$ *transformation matrix* $\mathbf{L}_m(n)$ is defined by the lower triangular matrix:

$$\mathbf{L}_m(n) = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ c_{1,1}^*(n) & 1 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ c_{m,m}^*(n) & c_{m,m-1}^* & \cdots & 1 \end{bmatrix} \tag{15.73}$$

The subscript m in the symbol $\mathbf{L}_m(n)$ refers to the highest order of backward prediction-error filter involved in its constitution. Note also the following points:

- The nonzero elements of row l of matrix $\mathbf{L}_m(n)$ are defined by the tap weights of a backward prediction-error filter of order $(l - 1)$.
- The diagonal elements of matrix $\mathbf{L}_m(n)$ equal unity; this follows from the fact that the last tap weight of a backward prediction-error filter equals unity.
- The determinant of matrix $\mathbf{L}_m(n)$ equals one for all m; hence, the inverse matrix $\mathbf{L}_m^{-1}(n)$ exists, and the reciprocal nature of the transformation in Eq. (15.70) is confirmed.

By definition, the correlation between the backward prediction errors pertaining to different orders, k and m, say, is given by the exponentially weighted time average

$$\begin{aligned}\phi_{km}(n) &= \sum_{i=1}^{n} \lambda^{n-i} b_k(i) b_m^*(i) \\ &= \sum_{i=1}^{n} \lambda^{n-i} \mathbf{c}_k^H(n) \mathbf{u}_k(i) b_m^*(i) \\ &= \mathbf{c}_k^H(n) \sum_{i=1}^{n} \lambda^{n-i} \mathbf{u}_k(i) b_m^*(i)\end{aligned} \tag{15.74}$$

where $\mathbf{c}_k(n)$ is the tap-weight vector of the backward prediction-error filter of order k that is responsible for generating the error $b_k(n)$. Without loss of generality, we may assume that $m > k$. Then, recognizing that the elements of the input vector $\mathbf{u}_k(n)$ are involved in generating the backward prediction represented by the error $b_m(n)$, optimized in the least-squares sense, we readily deduce from the principle of orthogonality that the correlation $\phi_{km}(n)$ is zero for $m > k$. In other words, for $m \neq k$, the backward prediction errors $b_k(n)$ and $b_m(n)$ are uncorrelated with each other in a time-averaged sense.

This remarkable property makes the least-squares lattice predictor an ideal device for *exact least-squares joint-process estimation*. Specifically, we may exploit the sequence of backward prediction errors produced by the lattice structure of Fig. 15.6 to perform the least-squares estimation of a desired response in the order-recursive manner described in Fig. 15.7. In particular, for order (stage) m we may write

$$e_m(n) = e_{m-1}(n) - \kappa_{m-1}^*(n) b_{m-1}(n), \qquad m = 1, 2, \ldots, M+1 \tag{15.75}$$

For the *initial condition* of joint-process estimation, we have

$$e_0(n) = d(n) \tag{15.76}$$

The parameters $\kappa_{m-1}(n)$, $m = 1, 2, \ldots, M+1$ are called *joint-process regression coefficients*. Thus, the least-squares estimation of desired response $d(n)$ may proceed on a stage-by-stage basis, alongside the linear prediction process.

Equation (15.75) represents a *single-order linear combiner*, as depicted in Fig. 15.5(c). Here we have purposely used the time index i in place of n for the estimation variables in order to be consistent with the notations used in parts (a) and (b) of the figure. The point to note here is that $b_{m-1}(i)$ may be viewed as the input and $e_{m-1}(i)$ as the desired response for $1 \leq i \leq n$. (The symbol used to denote the joint-process regression coefficient in Fig. 15.5(c) should not be confused with that used in Chapter 6 to denote the reflection coefficient of a lattice predictor.)

15.5 ANGLE-NORMALIZED ESTIMATION ERRORS

The formulation of the least-squares lattice predictor presented in the previous section was based on the forward *a posteriori* prediction error $f_m(n)$ and the backward *a posteriori* prediction error $b_m(n)$. The resulting order-recursive relations are defined in terms of the cur-

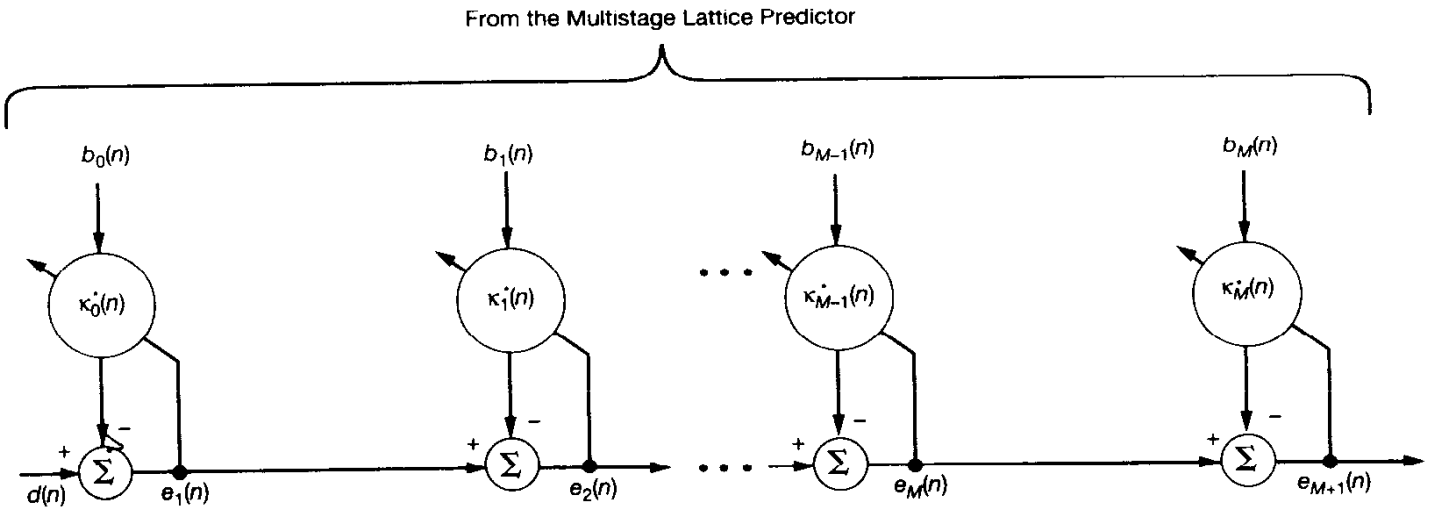


Figure 15.7 Joint-process estimation using a sequence of backward prediction errors.

rent value of forward reflection coefficient $\kappa_{f,m}(n)$ and the current value of backward reflection coefficient $\kappa_{b,m}(n)$. We may equally well formulate the least-squares lattice predictor in terms of the forward *a priori* prediction error $\eta_m(n)$ and the backward *a priori* prediction error $\beta_m(n)$. In the latter case, the order-recursive relations are defined in terms of the past value of the forward reflection coefficient $\kappa_{f,m}(n-1)$ and the past value of the backward reflection coefficient $\kappa_{b,m}(n-1)$.

From a developmental point of view, it would be highly desirable to formulate the least-squares lattice prediction problem in a way that is invariant to the choice of *a posteriori* or *a priori* prediction errors. This objective is attained by introducing the notion of *angle-normalized estimation errors*. Specifically, with three different forms of estimation in mind, we define the following set of angle-normalized estimation errors for a least-squares lattice predictor of order m:

- *Angle-normalized forward prediction error*:

$$\epsilon_{f,m}(n) = \gamma_m^{1/2}(n-1)\eta_m(n) = \frac{f_m(n)}{\gamma_m^{1/2}(n-1)} \tag{15.77}$$

  where $\gamma_m(n-1)$ is the past value of the conversion factor.

- *Angle-normalized backward prediction error*:

$$\epsilon_{b,m}(n) = \gamma_m^{1/2}(n)\beta_m(n) = \frac{b_m(n)}{\gamma_m^{1/2}(n)} \tag{15.78}$$

  where $\gamma_m(n)$ is the current value of the conversion factor.

- *Angle-normalized joint-process estimation error*:

$$\epsilon_m(n) = \gamma_m^{1/2}(n)\xi_m(n) = \frac{e_m(n)}{\gamma_m^{1/2}(n)} \tag{15.79}$$

  where $e_m(n)$ and $\xi_m(n)$ are the *a posteriori* and *a priori* values of the joint-process estimation error, respectively.

The term "angle" used in these definitions refers to an interpretation of the conversion factor as the cosine of an angle; see Section 15.3 for more details. In any event, the important point to note here is that in basing the algorithmic development of least-squares lattice filtering on angle-normalized estimation errors, we no longer have to distinguish between the *a posteriori* and *a priori* versions of the different estimation errors.

15.6 FIRST-ORDER STATE-SPACE MODELS FOR LATTICE FILTERING

With the background material presented in the previous sections at our disposal, we are ready to embark on the derivation of algorithms for the design of order-recursive adaptive filters based on least-squares estimation. The approach used here builds on the one-to-one correspondences between RLS variables and Kalman variables. To do so, we clearly need

to formulate state-space representations for least-squares prediction using a lattice structure and its extension for joint-process estimation. For reasons explained in the previous section, we wish to formulate the state-space models in terms of angle-normalized estimation errors.

Consider the following two n-by-1 vectors of angle-normalized prediction errors of order $m - 1$:

$$\boldsymbol{\epsilon}_{f,m-1}(n) = \begin{bmatrix} \epsilon_{f,m-1}(1) \\ \epsilon_{f,m-1}(2) \\ \vdots \\ \epsilon_{f,m-1}(n) \end{bmatrix}, \quad \boldsymbol{\epsilon}_{b,m-1}(n-1) = \begin{bmatrix} 0 \\ \epsilon_{b,m-1}(1) \\ \vdots \\ \epsilon_{b,m-1}(n-1) \end{bmatrix} \tag{15.80}$$

For initialization of the least-squares lattice predictor, we typically set

$$\mathcal{F}_{m-1}(0) = \mathcal{B}_{m-1}(-1) = \delta$$

$$\Delta_{m-1}(0) = 0$$

The constant δ is usually small enough to have a negligible effect on $\mathcal{F}_{m-1}(n)$ and $\mathcal{B}_{m-1}(n-1)$, and so it may be ignored. Accordingly, we may draw the following conclusions from Eqs. (15.11), (15.22), and (15.69), respectively:

1. The sum of weighted forward prediction-error squares $\mathcal{F}_{m-1}(n)$ is equal to the exponentially weighted squared norm of the corresponding angle-normalized vector $\boldsymbol{\epsilon}_{f,m-1}(n)$:

$$\mathcal{F}_{m-1}(n) = \boldsymbol{\epsilon}_{f,m-1}^H(n)\, \boldsymbol{\Lambda}(n)\, \boldsymbol{\epsilon}_{f,m-1}(n) \tag{15.81}$$

where $\boldsymbol{\Lambda}(n)$ is the n-by-n *exponential weighting matrix*:

$$\boldsymbol{\Lambda}(n) = \text{diag}[\lambda^{n-1}, \lambda^{n-2}, \ldots, 1]$$

2. The sum of weighted backward prediction-error squares $\mathcal{B}_{m-1}(n-1)$ is equal to the exponentially weighted squared norm of the corresponding angle-normalized vector $\boldsymbol{\epsilon}_{b,m-1}(n-1)$:

$$\mathcal{B}_{m-1}(n-1) = \boldsymbol{\epsilon}_{b,m-1}^H(n-1)\, \boldsymbol{\Lambda}(n)\, \boldsymbol{\epsilon}_{b,m-1}(n-1) \tag{15.82}$$

3. The parameter $\Delta_{m-1}^*(n)$, representing a form of cross-correlation, is equal to the exponentially weighted inner product of the angle-normalized vectors $\boldsymbol{\epsilon}_{b,m-1}(n-1)$ and $\boldsymbol{\epsilon}_{f,m-1}(n)$:

$$\Delta_{m-1}^*(n) = \boldsymbol{\epsilon}_{b,m-1}^H(n-1)\, \boldsymbol{\Lambda}(n)\, \boldsymbol{\epsilon}_{f,m-1}(n) \tag{15.83}$$

These facts mean that the forward reflection coefficient

$$\kappa_{f,m}^*(n) = -\frac{\Delta_{m-1}^*(n)}{\mathcal{B}_{m-1}(n-1)}$$

is also equal to

$$\kappa_{f,m}^{*}(n) = -\frac{\epsilon_{b,m-1}^{H}(n-1)\Lambda(n)\epsilon_{f,m-1}(n)}{\epsilon_{b,m}^{H}(n-1)\Lambda(n)\epsilon_{b,m-1}(n-1)} \tag{15.84}$$

In other words, the scalar $\kappa_{f,m}^{*}(n)$ can be interpreted as the coefficient that we need in order to project $\epsilon_{f,m-1}(n)$ onto $\epsilon_{b,m-1}(n-1)$.This suggests that we may replace the problem of projecting the *a posteriori* prediction vector $\mathbf{f}_{m-1}(n)$ onto the *a posteriori* prediction vector $\mathbf{b}_{m-1}(n-1)$ by the equivalent problem of projecting the angle-normalized prediction vector $\epsilon_{f,m-1}(n)$ onto the angle-normalized prediction vector $\epsilon_{b,m-1}(n-1)$. A similar conclusion follows for the problem of projecting $\mathbf{b}_{m-1}(n-1)$ onto $\mathbf{f}_{m-1}(n)$ and also for the joint-process estimation problem.

Accordingly, referring to the signal-flow-graphs of Figs. 15.5(a), 15.5(b), and 15.5(c), we may formulate a combination of *three first-order state-space models* for stage m of the least-squares lattice filtering process, based on three projections:

1. For forward linear prediction, $\epsilon_{f,m-1}(n)$ is projected onto $\epsilon_{b,m-1}(n-1)$.
2. For backward linear prediction, $\epsilon_{b,m-1}(n-1)$ is projected onto $\epsilon_{f,m-1}(n)$.
3. For joint-process linear estimation, $\epsilon_{m-1}(n)$ is projected onto $\epsilon_{b,m-1}(n)$.

Thus, bearing in mind the one-to-one correspondences between the Kalman variables and RLS variables that were established in Chapter 13, the state-space characterization of stage m of the least-squares lattice filtering process may be described in three parts as follows (Sayed and Kailath, 1994):

1. *Forward prediction*:

$$x_1(n+1) = \lambda^{-1/2}x_1(n) \tag{15.85}$$

$$y_1(n) = \epsilon_{b,m-1}^{*}(n-1)x_1(n) + v_1(n) \tag{15.86}$$

where $x_1(n)$ is the state-variable, and the reference signal $y_1(n)$ is defined by

$$y_1(n) = \lambda^{-n/2}\epsilon_{f,m-1}^{*}(n) \tag{15.87}$$

The scalar measurement noise $v_1(n)$ is a random variable with zero mean and unit variance.

2. *Backward prediction*:

$$x_2(n+1) = \lambda^{-1/2}x_2(n) \tag{15.88}$$

$$y_2(n) = \epsilon_{f,m-1}^{*}(n)x_2(n) + v_2(n) \tag{15.89}$$

where $x_2(n)$ is the second state-variable, and the second reference signal (observation) $y_2(n)$ is defined by

$$y_2(n) = \lambda^{-n/2}\epsilon_{b,m-1}^{*}(n) \tag{15.90}$$

As with $v_1(n)$, the scalar measurement noise $v_2(n)$ is a random variable with zero mean and unit variance.

3. *Joint-process estimation*:

$$x_3(n+1) = \lambda^{-1/2} x_3(n) \tag{15.91}$$

$$y_3(n) = \epsilon^*_{b,m-1}(n) x_3(n) + v_3(n) \tag{15.92}$$

where $x_3(n)$ is the third and final state-variable, and the corresponding reference signal (observation) $y_3(n)$ is defined by

$$y_3(n) = \lambda^{-n/2} \epsilon^*_{m-1}(n) \tag{15.93}$$

As before, the scalar measurement noise $v_3(n)$ is a random variable with zero mean and unit variance. The noise variables $v_1(n)$, $v_2(n)$, and $v_3(n)$ are all independent of each other.

On the basis of the state-space models described above, we may formulate the list of one-to-one correspondences, shown in Table 15.2, between Kalman variables and three sets of least-squares lattice (LSL) variables, assuming a prediction order of $m-1$. The three sets of LSL variables refer to forward prediction, backward prediction, and joint-process estimation.

The first four lines of Table 15.2 follow readily from the state-space models of Eqs. (15.85) to (15.93) and Table 13.2 listing the one-to-one correspondences between Kalman

TABLE 15.2 SUMMARY OF ONE-TO-ONE CORRESPONDENCES BETWEEN KALMAN VARIABLES AND LSL VARIABLES IN STAGE m OF THE LATTICE PREDICTOR

Kalman Variable	LSL Variable		
	Forward Prediction	Backward Prediction	Joint-process Estimation
$y(n)$	$\lambda^{-n/2}\epsilon^*_{f,m-1}(n)$	$\lambda^{-n/2}\epsilon^*_{b,m-1}(n-1)$	$\lambda^{-n/2}\epsilon^*_{m-1}(n)$
$\mathbf{u}^H(n)$	$\epsilon^*_{b,m-1}(n-1)$	$\epsilon^*_{f,m-1}(n)$	$\epsilon^*_{b,m-1}(n)$
$\hat{\mathbf{x}}(n\|\mathcal{Y}_{n-1})$	$-\lambda^{-n/2}\kappa_{f,m}(n-1)$	$-\lambda^{-n/2}\kappa_{b,m}(n-1)$	$\lambda^{-n/2}\kappa_{m-1}(n-1)$
$\mathbf{K}(n-1)$	$\lambda^{-1}\mathcal{B}^{-1}_{m-1}(n-2)$	$\lambda^{-1}\mathcal{F}^{-1}_{m-1}(n-1)$	$\lambda^{-1}\mathcal{B}^{-1}_{m-1}(n-1)$
$r(n)$	$\dfrac{\gamma_{m-1}(n-1)}{\gamma_m(n-1)}$	$\dfrac{\gamma_{m-1}(n-1)}{\gamma_m(n)}$	$\dfrac{\gamma_{m-1}(n)}{\gamma_m(n)}$
$\alpha(n)$	$\lambda^{-n/2}\gamma^{1/2}_{m-1}(n-1)\,\eta^*_m(n)$	$\lambda^{-n/2}\gamma^{1/2}_{m-1}(n-1)\,\beta^*_m(n)$	$\lambda^{-n/2}\gamma^{1/2}_{m-1}(n)\,\xi^*_m(n)$

variables and RLS variables. To verify the remaining two lines of correspondences in Table 15.2, we may proceed as follows for the case of forward linear prediction:

1. According to Kalman filter theory, the innovation $\alpha(n)$ is defined by

$$\alpha(n) = y(n) - \mathbf{u}^H(n)\hat{\mathbf{x}}(n|\mathcal{Y}_{n-1}) \tag{15.94}$$

From the first three lines of Table 15.2, we have the following one-to-one correspondences for forward prediction:

$$y(n) \leftrightarrow \lambda^{-n/2}\, \epsilon^*_{f,m-1}(n)$$

$$\mathbf{u}^H(n) \leftrightarrow \epsilon^*_{b,m-1}(n-1)$$

$$\hat{\mathbf{x}}(n|\mathcal{Y}_{n-1}) \leftrightarrow -\lambda^{-n/2}\, \kappa_{f,m}(n-1)$$

where the double-headed arrows signify one-to-one correspondences. Therefore, substituting these correspondences into the right-hand side of Eq. (15.94), we get

$$\alpha(n) \leftrightarrow \lambda^{-n/2}(\epsilon^*_{f,m-1}(n) + \kappa_{f,m}(n)\, \epsilon^*_{b,m-1}(n-1))$$

Next, using the relations of Eqs. (15.77) and (15.78), we may equivalently write

$$\alpha(n) \leftrightarrow \lambda^{-n/2}\gamma^{1/2}_{m-1}(n-1)(\eta^*_{m-1}(n) + \kappa_{f,m}(n-1)\beta^*_{m-1}(n-1)) \tag{15.95}$$

We now recognize the following input–output relations for stage m of a lattice predictor described in terms of *a priori* prediction errors:

$$\eta_m(n) = \eta_{m-1}(n) + \kappa^*_{f,m}(n-1)\beta_{m-1}(n-1) \tag{15.96}$$

$$\beta_m(n) = \beta_{m-1}(n-1) + \kappa^*_{b,m}(n-1)\eta_{m-1}(n) \tag{15.97}$$

The fundamental difference between Eqs. (15.96) and (15.97) and their counterparts, Eqs. (15.41) and (15.42), is that the former pair is based on past values $\kappa_{f,m}(n-1)$ and $\kappa_{b,m}(n-1)$ of the forward and backward reflection coefficients, whereas the latter pair is based on their current values $\kappa_{f,m}(n)$ and $\kappa_{b,m}(n)$. Thus, in light of Eq. (15.96) we conclude from (15.95) that for forward prediction of order $m-1$:

$$\alpha(n) \leftrightarrow \lambda^{-n/2}\gamma^{1/2}_{m-1}(n-1)\, \eta^*_m(n) \tag{15.98}$$

2. According to Kalman filter theory, the filtered estimation error is given by

$$e(n) = y(n) - \mathbf{u}^H(n)\hat{\mathbf{x}}(n|\mathcal{Y}n) \tag{15.99}$$

where the filtered state vector is defined by

$$\hat{\mathbf{x}}(n|\mathcal{Y}_n) = \mathbf{F}(n, n+1)\hat{\mathbf{x}}(n+1)|\mathcal{Y}_n)$$

For the problem at hand, the transition matrix is [see Eq. (15.85)]

$$\mathbf{F}(n+1, n) = \lambda^{-1/2}$$

Since

$$\mathbf{F}(n+1, n)\mathbf{F}(n, n+1) = \mathbf{I}$$

and

$$\hat{\mathbf{x}}(n+1|\mathcal{Y}_n) \leftrightarrow -\lambda^{-(n+1)/2}\kappa_{f,m}(n)$$

it follows that

$$\hat{\mathbf{x}}(n|\mathcal{Y}_n) \leftrightarrow -\lambda^{-n/2}\kappa_{f,m}(n)$$

We may therefore use Eq. (15.99) to write for forward linear prediction:

$$e(n) \leftrightarrow \lambda^{-n/2}\,(\epsilon^*_{f,m-1}(n) + \kappa_{f,m}(n)\,\epsilon^*_{b,m-1}(n-1))$$

Again, using the relations of Eqs. (15.77) and (15.78), we may equivalently write

$$e(n) \leftrightarrow \lambda^{-n/2}\,\gamma_{m-1}^{-1/2}\,(n-1)\,(f^*_{m-1}(n) + \kappa_{f,m}(n)\,b^*_{m-1}(n-1)) \tag{15.100}$$

But, from Eq. (15.41) we have

$$f_m(n) = f_{m-1}(n) + \kappa^*_{f,m}(n)\,b_{m-1}(n-1)$$

in light of which, we conclude from Eq. (15.100) that for forward prediction of order $m-1$:

$$e(n) \leftrightarrow \lambda^{-n/2}\,\gamma_{m-1}^{-1/2}\,(n-1)\,f^*_m(n) \tag{15.101}$$

3. In Kalman filter theory, the conversion factor $r^{-1}(n)$ is defined by the ratio $e(n)/\alpha(n)$. We may therefore use (15.98) and (15.101) to write the following one-to-one correspondence for forward prediction of order $m-1$:

$$r(n) \leftrightarrow \frac{\gamma_{m-1}(n-1)}{\gamma_m(n-1)} \tag{15.102}$$

Thus, Eqs. (15.102) and (15.98) provide the basis for the last two lines of correspondences between the Kalman and LSL variables for forward prediction listed in Table 15.2. By proceeding in a manner similar to that described above, we may fill in the remaining correspondences pertaining to backward prediction and joint-process estimation; this is left as an exercise for the reader.

15.7 QR-DECOMPOSITION–BASED LEAST-SQUARES LATTICE FILTERS

Equipped with the state-space models for least-squares lattice filtering described in Section 15.6 and the square-root information (Kalman) filter developed in Chapter 14, we are at long last positioned to get on with the derivation of our first and most important order-recursive adaptive filter. The writings of arrays for the filter and their expansions are presented in three parts, dealing with adaptive forward prediction, adaptive backward prediction, and adaptive joint-process estimation, in that order.

Array for Adaptive Forward Prediction

Adapting Eq. (14.39) to suit the forward prediction model described by the state-space equations (15.85) to (15.87), with the aid of Table 15.2 listing the one-to-one correspondences between the Kalman variables and LSL variables (for forward prediction), we may write the following array for stage m of the least-squares lattice predictor (Sayed and Kailath, 1994):

$$\begin{bmatrix} \lambda^{1/2}\mathcal{B}_{m-1}^{1/2}(n-2) & \epsilon_{b,m-1}(n-1) \\ \lambda^{1/2}p_{f,m-1}^{*}(n-1) & \epsilon_{f,m-1}(n) \\ 0 & \gamma_{m-1}^{1/2}(n-1) \end{bmatrix} \Theta_{b,m-1}(n-1) \tag{15.103}$$

$$= \begin{bmatrix} \mathcal{B}_{m-1}^{1/2}(n-1) & 0 \\ p_{f,m-1}^{*}(n) & \epsilon_{f,m}(n) \\ b_{m-1}^{*}(n-1)\mathcal{B}_{m-1}^{-1/2}(n-1) & \gamma_{m}^{1/2}(n-1) \end{bmatrix}$$

where we have done the following things. First, the common factors $\lambda^{1/2}$ and $\lambda^{-n/2}$ have been cancelled from the pre- and postarrays in the first and second rows, respectively. Second, we have multiplied the pre- and postarray in the third row by $\gamma_{m-1}^{1/2}(n-1)$. The scalar quantities $\mathcal{B}_{m-1}(n-1)$ and $p_{f,m-1}(n)$ appearing in the postarray are defined as follows:

1. The real-valued quantity $\mathcal{B}_{m-1}(n-1)$ is the autocorrelation of the delayed, angle-normalized backward prediction error $\epsilon_{b,m-1}(n-1)$ for a lag of zero:

$$\begin{aligned} \mathcal{B}_{m-1}(n-1) &= \sum_{i=1}^{n-1} \lambda^{n-1-i}\epsilon_{b,m-1}(i-1)\epsilon_{b,m-1}^{*}(i-1) \\ &= \lambda\mathcal{B}_{m-1}(n-2) + \epsilon_{b,m-1}(n-1)\epsilon_{b,m-1}^{*}(n-1) \end{aligned} \tag{15.104}$$

   The quantity $\mathcal{B}_{m-1}(n-1)$ may also be interpreted as the minimum value of the sum of weighted backward *a posteriori* prediction-error squares, which is defined in accordance with RLS theory as follows [see Eq. (15.22)]:

$$\mathcal{B}_{m-1}(n-1) = \lambda\mathcal{B}_{m-1}(n-2) + \beta_{m-1}(n-1)b_{m-1}^{*}(n-1) \tag{15.105}$$

   Note that the product term $\beta_{m-1}(n-1)b_{m-1}^{*}(n-1)$ is always real, in that we can write

$$\beta_{m-1}(n-1)b_{m-1}^{*}(n-1) = \beta_{m-1}^{*}(n-1)b_{m-1}(n-1)$$

2. The complex-valued quantity $p_{f,m-1}(n)$ is, except for the factor $\mathcal{B}_{m-1}^{-1/2}(n-1)$, the cross-correlation between the angle-normalized forward and backward prediction errors, as shown by

$$p_{f,m-1}(n) = \frac{\Delta_{m-1}(n)}{\mathcal{B}_{m-1}^{1/2}(n-1)} \tag{15.106}$$

where

$$\Delta_{m-1}(n) = \sum_{i=1}^{n} \lambda^{n-i} \epsilon_{b,m-1}(i-1)\epsilon^*_{f,m-1}(i) \tag{15.107}$$
$$= \lambda \Delta_{m-1}(n-1) + \epsilon_{b,m-1}(n-1)\epsilon^*_{f,m-1}(n)$$

Indeed $p_{f,m-1}(n)$ is related to the forward reflection coefficient $\kappa_{f,m}(n)$ for prediction order m as follows (Haykin, 1991):

$$\kappa_{f,m}(n) = -\frac{\Delta_{m-1}(n)}{\mathcal{B}_{m-1}(n-1)} \tag{15.108}$$
$$= -\frac{p_{f,m-1}(n)}{\mathcal{B}^{1/2}_{m-1}(n-1)}$$

The 2-by-2 matrix $\Theta_{b,m-1}(n-1)$ in Eq. (15.103) is any unitary rotation that reduces the (1,2) entry in the postarray to zero; that is, it annihilates the entry $\epsilon_{b,m-1}(n-1)$ in the prearray. This requirement is readily satisfied by using a Givens rotation, as described here:

$$\Theta_{b,m-1}(n-1) = \begin{bmatrix} c_{b,m-1}(n-1) & -s_{b,m-1}(n-1) \\ s^*_{b,m-1}(n-1) & c_{b,m-1}(n-1) \end{bmatrix} \tag{15.109}$$

where the cosine and sine parameters are themselves defined by

$$c_{b,m-1}(n-1) = \frac{\lambda^{1/2}\mathcal{B}^{1/2}_{m-1}(n-2)}{\mathcal{B}_{m-1}(n-1)} \tag{15.110}$$

$$s_{b,m-1}(n-1) = \frac{\epsilon_{b,m-1}(n-1)}{\mathcal{B}^{1/2}_{m-1}(n-1)} \tag{15.111}$$

Thus, using Eq. (15.109) in (15.103), we get the following update relations in addition to that of Eq. (15.104):

$$p^*_{f,m-1}(n) = c_{b,m-1}(n-1)\lambda^{1/2}p^*_{f,m-1}(n-1) + s^*_{b,m-1}(n-1)\epsilon_{f,m-1}(n) \tag{15.112}$$

$$\epsilon_{f,m}(n) = c_{b,m-1}(n-1)\epsilon_{f,m-1}(n) - s_{b,m-1}(n-1)\lambda^{1/2}p^*_{f,m-1}(n-1) \tag{15.113}$$

$$\gamma^{1/2}_m(n-1) = c_{b,m-1}(n-1)\gamma^{1/2}_{m-1}(n-1) \tag{15.114}$$

Equations (15.104) and (15.110) to (15.114) constitute the set of relations for a square-root information filtering solution to the adaptive forward linear problem in a least-squares lattice sense.

Array for Adaptive Backward Prediction

Consider next the adaptive backward prediction model described by the state-space equations (15.88) to (15.90). Then, with the aid of the one-to-one correspondences between the Kalman variables and LSL variables (for backward prediction) listed in Table 15.2, we

may use Eq. (14.39) to write the following array for stage m of the least-squares lattice predictor (Sayed and Kailath, 1994):

$$\begin{bmatrix} \lambda^{1/2}\mathcal{F}_{m-1}^{1/2}(n-1) & \epsilon_{f,m-1}(n) \\ \lambda^{1/2}p_{b,m-1}^{*}(n-1) & \epsilon_{b,m-1}(n-1) \\ 0 & \gamma_{m-1}^{1/2}(n-1) \end{bmatrix} \Theta_{f,m-1}(n) = \begin{bmatrix} \mathcal{F}_{m-1}^{1/2}(n) & 0 \\ p_{b,m-1}^{*}(n) & \epsilon_{b,m}(n) \\ f_{m-1}^{*}(n)\mathcal{F}_{m-1}^{-1/2}(n) & \gamma_{m}^{1/2}(n) \end{bmatrix} \tag{15.115}$$

The two new scalar quantities $\mathcal{F}_{m-1}(n)$, and $p_{b,m-1}(n)$ appearing in the postarray of Eq. (15.115) are defined as follows:

1. The real-valued quantity $\mathcal{F}_{m-1}(n)$ is the autocorrelation of the angle-normalized forward prediction error $\epsilon_{f,m-1}(n)$ for a lag of zero:

$$\begin{aligned} \mathcal{F}_{m-1}(n) &= \sum_{i=1}^{n} \lambda^{n-i}\epsilon_{f,m-1}(i)\epsilon_{f,m-1}^{*}(i) \\ &= \lambda\mathcal{F}_{m-1}(n-1) + \epsilon_{f,m-1}(n)\epsilon_{f,m-1}^{*}(n) \end{aligned} \tag{15.116}$$

   It may also be interpreted as the minimum value of the sum of forward prediction-error squares, which is defined in accordance with the RLS theory as follows [see Eq. (15.11)]:

$$\mathcal{F}_{m-1}(n) = \lambda\mathcal{F}_{m-1}(n-1) + \eta_{m-1}(n)f_{m-1}^{*}(n) \tag{15.117}$$

   As in the case of Eq. (15.105), the product $\eta_{m-1}(n)f_{m-1}^{*}(n)$ is always real-valued.

2. The complex-valued quantity $p_{b,m-1}(n)$ is, except for the factor $\mathcal{F}_{m-1}^{-1/2}(n)$, the complex conjugate of the cross-correlation between the angle-normalized forward and backward prediction errors, defined in Eq. (15.107); that is,

$$p_{b,m-1}(n) = \frac{\Delta_{m-1}^{*}(n)}{\mathcal{F}_{m-1}^{1/2}(n)} \tag{15.118}$$

   The quantity $p_{b,m-1}(n)$ is also related to the backward reflection coefficient for prediction order m by the formula (Haykin, 1991):

$$\begin{aligned} \kappa_{b,m}(n) &= -\frac{\Delta_{m-1}^{*}(n)}{\mathcal{F}_{m-1}(n)} \\ &= -\frac{p_{b,m-1}(n)}{\mathcal{F}_{m-1}^{1/2}(n)} \end{aligned} \tag{15.119}$$

The 2-by-2 matrix $\Theta_{f,m-1}(n)$ is a unitary rotation that reduces the (1,2) entry of the postarray in Eq. (15.115) to zero; that is, it annihilates the entry $\epsilon_{f,m-1}(n)$ in the prearray of this same equation. This requirement may be satisfied by using a Givens rotation described as follows:

$$\Theta_{f,m-1}(n) = \begin{bmatrix} c_{f,m-1}(n) & -s_{f,m-1}(n) \\ s_{f,m-1}^{*}(n) & c_{f,m-1}(n) \end{bmatrix} \tag{15.120}$$

where the cosine and sine parameters are themselves defined by

$$c_{f,m-1}(n) = \frac{\lambda^{1/2}\mathcal{F}_{m-1}^{1/2}(n-1)}{\mathcal{F}_{m-1}^{1/2}(n)} \tag{15.121}$$

$$s_{f,m-1}(n) = \frac{\epsilon_{f,m-1}(n)}{\mathcal{F}_{m-1}^{1/2}(n)} \tag{15.122}$$

Thus, substituting Eq. (15.120) in (15.115), we readily deduce the following recursions:

$$p_{b,m-1}^{*}(n) = c_{f,m-1}(n)\lambda^{1/2}p_{b,m-1}^{*}(n-1) + s_{f,m-1}^{*}(n)\epsilon_{b,m-1}(n-1) \tag{15.123}$$

$$\epsilon_{b,m}(n) = c_{f,m-1}(n)\epsilon_{b,m-1}(n-1) - s_{f,m-1}(n)\lambda^{1/2}p_{b,m-1}^{*}(n-1) \tag{15.124}$$

$$\gamma_{m}^{1/2}(n) = c_{f,m-1}(n)\gamma_{m-1}^{1/2}(n-1) \tag{15.125}$$

Equations (15.115) and (15.121) to (15.125) constitute the set of recursions for a square-root information filtering solution to the adaptive backward problems in a least-squares lattice sense.

Array for Joint-Process Estimation

Finally, consider the joint-process estimation problem described by the state-space equations (15.91) to (15.93), pertaining to stage m of the least-squares lattice filtering process. Thus, with the aid of the one-to-one correspondences between the Kalman variables and LSL variables (for joint-process estimation) listed in Table 15.2, we may use the array of Eq. (14.39) to write (Sayed and Kailath, 1994)

$$\begin{bmatrix} \lambda^{1/2}\mathcal{B}_{m-1}^{1/2}(n-1) & \epsilon_{b,m-1}(n) \\ \lambda^{1/2}p_{m-1}^{*}(n-1) & \epsilon_{m-1}(n) \\ 0 & \gamma_{m-1}^{1/2}(n) \end{bmatrix} \Theta_{b,m-1}(n) = \begin{bmatrix} \mathcal{B}_{m-1}^{1/2}(n) & 0 \\ p_{m-1}^{*}(n) & \epsilon_{m}(n) \\ b_{m-1}^{*}(n)\mathcal{B}_{m-1}^{-1/2}(n) & \gamma_{m}^{1/2}(n) \end{bmatrix} \tag{15.126}$$

In Eq. (15.126), there is only one new quantity that we have to describe, namely, $p_{m-1}(n)$. This new quantity is, except for the factor $\mathcal{B}_{m-1}^{-1/2}(n)$, the cross-correlation between the angle-normalized backward prediction error and angle-normalized joint-estimation error, as shown by

$$p_{m-1}(n) = \frac{1}{\mathcal{B}_{m-1}^{1/2}(n)} \sum_{i=1}^{n} \lambda^{n-i}\epsilon_{b,m-1}(i)\epsilon_{b,m-1}^{*}(i) \tag{15.127}$$

The regression coefficient $\kappa_{m-1}(n)$ for prediction order $m-1$ is correspondingly defined by (Haykin, 1991)

$$\kappa_{m-1}(n) = \frac{p_{m-1}(n)}{\mathcal{B}_{m-1}^{1/2}(n)} \tag{15.128}$$

The 2-by-2 matrix $\Theta_{b,m-1}(n)$ is a unitary rotation designed to annihilate the entry $\epsilon_{b,m-1}(n)$ in the prearray of Eq. (15.126). To do this, we may use the same Givens rotation as that in Eq. (15.109), except for a shift in time by one unit. Specifically, we may write

$$\Theta_{b,m-1}(n) = \begin{bmatrix} c_{b,m-1}(n) & -s_{b,m-1}(n) \\ s^*_{b,m-1}(n) & c_{b,m-1}(n) \end{bmatrix} \tag{15.129}$$

where

$$c_{b,m-1}(n) = \frac{\lambda^{1/2}\mathcal{B}^{1/2}_{m-1}(n-1)}{\mathcal{B}^{1/2}_{m-1}(n)} \tag{15.130}$$

$$s_{b,m-1}(n) = \frac{\epsilon_{b,m-1}(n)}{\mathcal{B}^{1/2}_{m-1}(n)} \tag{15.131}$$

Hence, substituting Eq. (15.129) in (15.126), we get the following new recursions

$$p^*_{m-1}(n) = c_{b,m-1}(n)\lambda^{1/2}p^*_{m-1}(n-1) + s^*_{b,m-1}(n)\epsilon_{m-1}(n) \tag{15.132}$$

$$\epsilon_m(n) = c_{b,m-1}(n)\epsilon_{m-1}(n) - s_{b,m-1}(n)\lambda^{1/2}p^*_{m-1}(n-1) \tag{15.133}$$

Equations (15.104), (15.110), and (15.111) with time $n-1$ replaced with n, together with Eqs. (15.132) and (15.133), constitute the set of recursions for a square-root information filtering solution to the joint-process estimation problem in a least-squares lattice sense.

Summary of the QRD–LSL Algorithm

Table 15.3 presents a summary of the angle-normalized QRD–LSL algorithm,[2] based on the arrays of Eqs. (15.103), (15.115), and (15.126). Note that the forward and backward predictions are performed for $m = 1, 2, \ldots, M$, whereas those for the joint-process estimation are performed for $m = 1, 2, \ldots, M+1$, where M is the final prediction order.

[2]The idea of a fast QR-decomposition–based algorithm for recursive least-squares estimation was first presented by Cioffi (1988). A detailed derivation of the algorithm is described in Cioffi (1990). In the latter paper, Cioffi presents a geometric approach for the derivation that is reminiscent of his earlier work on fast transversal filters. The algorithm derived by Cioffi is of a Kalman or matrix-oriented type. Several other authors have presented seemingly simple algebraic derivations and other versions of the QRD-fast RLS algorithm (Bellanger, 1988; Proudler et al., 1988, 1989; Regalia and Bellanger, 1991). The paper by Proudler et al. (1989) is of particular interest in that it develops a novel implementation of the QRD-RLS algorithm using a lattice structure. A similar fast algorithm has been derived independently by Ling (1989) using the modified Gram–Schmidt orthogonalization procedure. The connection between the modified Gram–Schmidt orthogonalization and QR-decomposition is discussed in Shepherd and McWhirter (1991).

Haykin (1991) presented a development of the QRD–LSL algorithm that is based on a hybridization of ideas due to Proudler et al. (1989) and Regalia and Bellanger (1991). Specifically, the development followed Proudler et al. in deriving QR-decomposition–based solutions to forward and backward linear prediction problems, and Regalia and Bellanger in solving the joint-process estimation problem. By so doing, the complications in the procedure by Proudler et al. that uses forward linear prediction errors for joint-process estimation are avoided. The structure of the QRD–LSL algorithm derived by Haykin follows a philosophy directly analogous to that described for conventional LSL algorithms later in the chapter.

TABLE 15.3 SUMMARY OF THE QRD–LSL ALGORITHM

1. *Computations*

(a) *Predictions*: For each time instant $n = 1, 2, \ldots$, perform the following computations and repeat for each prediction order $m = 1, 2, \ldots, M$, where M is the final prediction order:

$$\begin{bmatrix} \lambda^{1/2}\mathcal{B}_{m-1}^{1/2}(n-2) & \epsilon_{b,m-1}(n-1) \\ \lambda^{1/2}p_{f,m-1}^{*}(n-1) & \epsilon_{f,m-1}(n) \\ 0 & \gamma_{m-1}^{1/2}(n-1) \end{bmatrix} \Theta_{b,m-1}(n-1) = \begin{bmatrix} \mathcal{B}_{m-1}^{1/2}(n-1) & 0 \\ p_{f,m-1}^{*}(n) & \epsilon_{f,m}(n) \\ b_{m-1}^{*}(n-1)\mathcal{B}_{m-1}^{-1/2}(n-1) & \gamma_{m}^{1/2}(n-1) \end{bmatrix}$$

$$\begin{bmatrix} \lambda^{1/2}\mathcal{F}_{m-1}^{1/2}(n-1) & \epsilon_{f,m-1}(n) \\ \lambda^{1/2}p_{b,m-1}^{*}(n-1) & \epsilon_{b,m-1}(n-1) \end{bmatrix} \Theta_{f,m-1}(n) = \begin{bmatrix} \mathcal{F}_{m-1}^{1/2}(n) & 0 \\ p_{b,m-1}^{*}(n) & \epsilon_{b,m}(n) \end{bmatrix}$$

(b) *Filtering*: For each time instant $n = 1, 2, \ldots$, perform the following computations and repeat for each prediction order $m = 1, 2, \ldots, M + 1$, where M is the final prediction order:

$$\begin{bmatrix} \lambda^{1/2}\mathcal{B}_{m-1}^{1/2}(n-1) & \epsilon_{b,m-1}(n) \\ \lambda^{1/2}p_{m-1}^{*}(n-1) & \epsilon_{m-1}(n) \end{bmatrix} \Theta_{b,m-1}(n) = \begin{bmatrix} \mathcal{B}_{m-1}^{1/2}(n) & 0 \\ p_{m-1}^{*}(n) & \epsilon_{m}(n) \end{bmatrix}$$

2. *Initialization*

(a) *Auxiliary parameter initialization*: For order $m = 1, 2, \ldots, M$, set

$$p_{f,m-1}(0) = p_{b,m-1}(0) = 0$$

and for order $m = 1, 2, \ldots, M + 1$, set

$$p_{m-1}(0) = 0$$

(b) *Soft-constraint initialization*: For order $m = 0, 1, \ldots, M$, set

$$\mathcal{B}_m(-1) = \mathcal{B}_m(0) = \delta$$

$$\mathcal{F}_m(0) = \delta$$

where δ is a small positive constant.

(c) *Data initialization*: For $n = 1, 2, \ldots$, compute

$$\epsilon_{f,0}(n) = \epsilon_{b,0}(n) = u(n)$$

$$\epsilon_0(n) = d(n)$$

$$\gamma_0(n) = 1$$

where $u(n)$ is the input and $d(n)$ is the desired response at time n.

That is, the joint-process estimation involves one final set of computations pertaining to $m = M + 1$. Note also that in the second and third arrays of Table 15.3, we have omitted the particular rows that involve updating the conversion factor. This requirement is taken care of by the first array.

Table 15.3 includes the *initialization* procedure, which is of a *soft-constraint* form. This form of initialization is consistent with that adopted for the conventional RLS algorithm, as described in Chapter 13. The algorithm proceeds with a set of *initial values* determined by the input datum $u(n)$ and desired response $d(n)$, as shown by

$$\epsilon_{f,0}(n) = \epsilon_{b,0}(n) = u(n)$$

and

$$\epsilon_0(n) = d(n)$$

The initial value for the conversion factor is chosen as

$$\gamma_0(n) = 1$$

15.8 FUNDAMENTAL PROPERTIES OF THE QRD–LSL FILTER

The QRD–LSL algorithm summarized in Table 15.3 is so called in recognition of three facts. First, the unitary transformations in Eqs. (15.103), (15.115), and (15.130), in which the (1,2) entry of each postarray is reduced to zero, are all examples of the *QR-decomposition*. Second, the algorithm is rooted in *recursive least-squares estimation*. Third, the computations performed by the algorithm proceed on a stage-by-stage fashion, with each stage having the form of a *lattice*. By virtue of these facts, the QRD–LSL algorithm is endowed with a highly desirable set of operational and implementational characteristics:

- *Good numerical properties*, which are inherited from the QR-decomposition part of the algorithm
- *Good convergence properties* (i.e., fast rate of convergence, and insensitivity to variations in the eigenvalue spread of the underlying correlation matrix of the input data), which are due to the recursive least-squares nature of the algorithm.
- A *high level of computational efficiency*, which results from the modular, lattice-like structure of the prediction process

The unique combination of these characteristics makes the QRD–LSL algorithm a powerful adaptive filtering algorithm.

The latticelike structure of the QRD–LSL algorithm, using a sequence of Givens rotations, is clearly illustrated by the multistage signal-flow graph of Fig. 15.8. In particular, we see that stage m of the predictor section of the algorithm involves the computations of the angle-normalized prediction errors: $\epsilon_{f,m}(n)$ and $\epsilon_{b,m}(n)$, where the prediction order $m = 1, 2, \ldots, M$. On the other hand, the filtering section of the algorithm involves the computation of the angle-normalized joint-process estimation error $\epsilon_m(n)$, where $m = 1, 2, \ldots, M + 1$. The details of these computations are depicted in the signal-flow graphs shown in Fig. 15.9, which further emphasize the inherent lattice nature of the QRD–LSL algorithm.

The boxes labeled $z^{-1}\mathbf{I}$ in Fig. 15.8 signify *storage*, which is needed to accommodate the fact that the Givens rotations involved in the adaptive forward prediction process are delayed with respect to those involved in the joint-process estimation process by one time unit. Note, however, that the joint-process estimation process involves one last Givens rotation, all by itself.

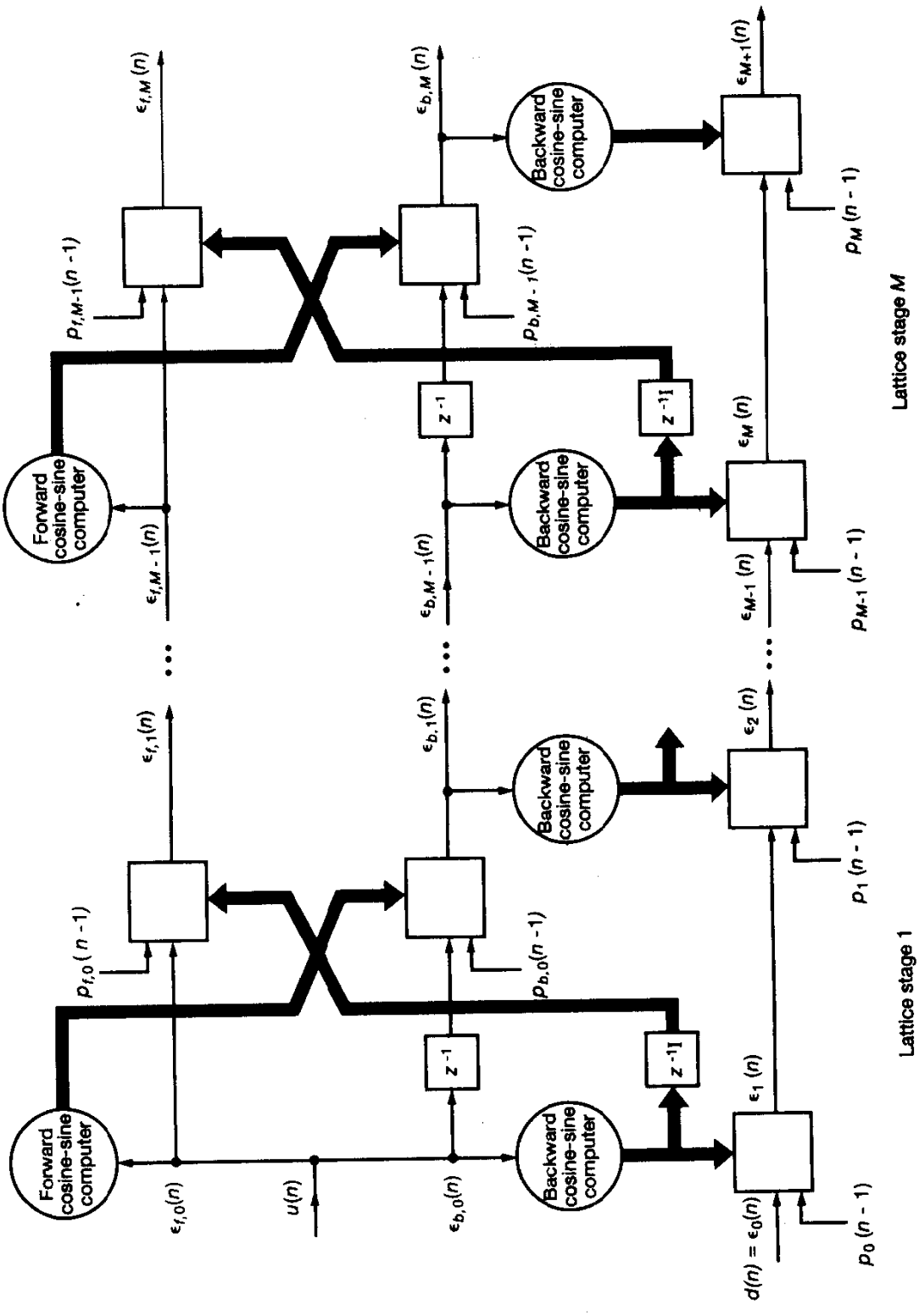


Figure 15.8 Signal-flow graph of the QRD–LSL algorithm.

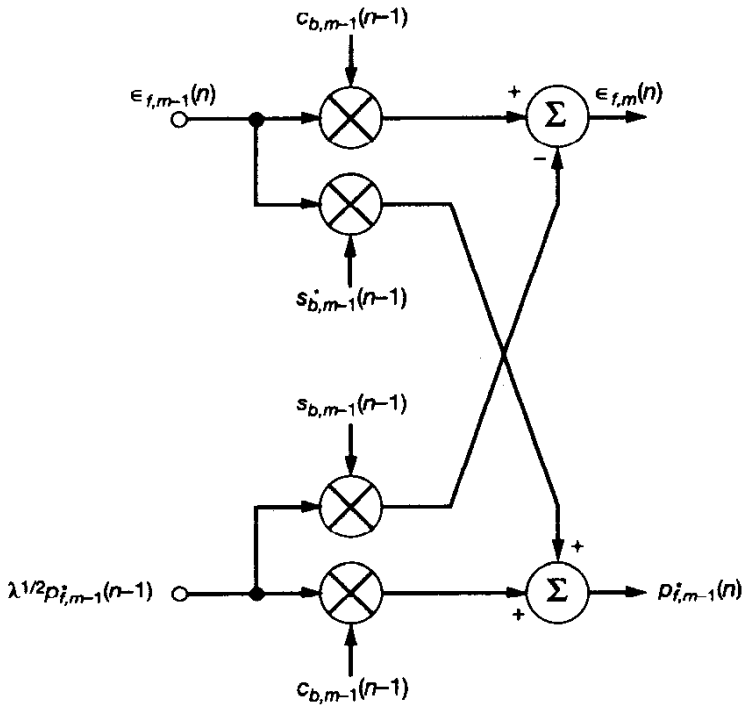
From the signal-flow graph of Fig. 15.8 we clearly see that the total number of Givens rotations needed for the computation of $\epsilon_{M+1}(n)$ is $2M + 1$, which increases linearly with the final prediction order M. However, the price paid for the high level of computational efficiency of the fast algorithm described herein, compared to the conventional RLS algorithm described in Chapter 13, is that of having to write a more elaborate set of instructions.

Examining the signal-flow graphs of Figs. 15.8 and 15.9, we may identify two different sets of recursions in the formulation of the QRD–LSL algorithm:

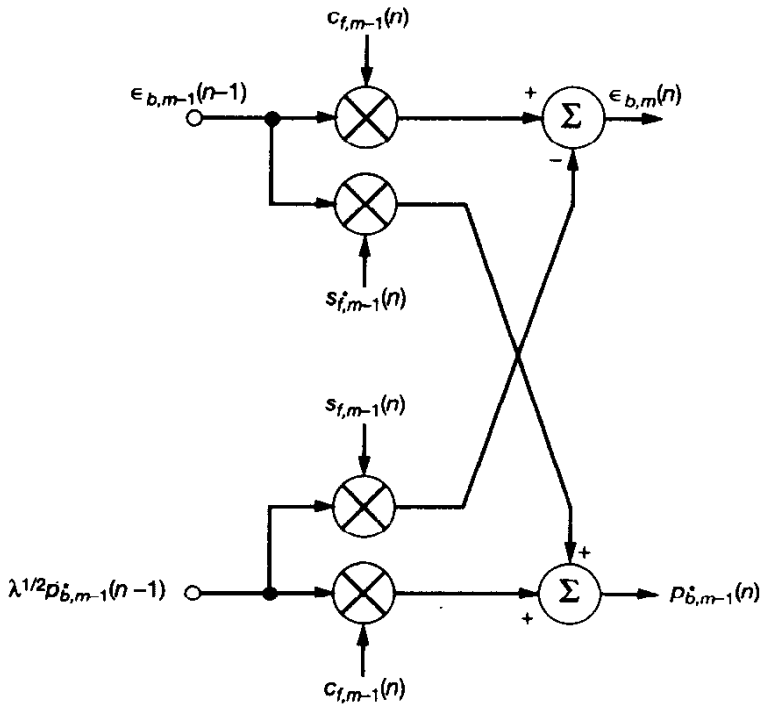
1. *Order updates (recursions).* At each stage of the algorithm, *order updates* are performed on the angle-normalized estimation errors. Specifically, a series of m order updates applied to the initial values $\epsilon_{f,0}(n)$ and $\epsilon_{b,0}(n)$ yields the final values $\epsilon_{f,M}(n)$ and $\epsilon_{b,M}(n)$, respectively, where M is the final prediction order. To compute the final value of the angle-normalized joint-process estimation error $\epsilon_{M+1}(n)$, another series of $M + 1$ order updates are applied to the initial value $\epsilon_0(n)$. This latter set of updates includes the use of the sequence of angle-normalized backward prediction-errors $\epsilon_{b,0}(n), \epsilon_{b,1}(n), \ldots, \epsilon_{b,M}(n)$. The final order update pertains to the computation of the square root of the conversion factor, $\gamma_{M+1}^{1/2}(n)$, which involves the application of $M + 1$ order updates to the initial value $\gamma_0^{1/2}(n)$. The availability of the final values $\epsilon_{M+1}(n)$ makes it possible to compute the final value $e_{M+1}(n)$ of the joint-process estimation error.
2. *Time updates (recursions).* The computations of $\epsilon_{f,m}(n)$ and $\epsilon_{b,m}(n)$ as outputs of predictor stage m of the algorithm involve the use of the auxiliary parameters $p_{f,m-1}(n)$ and $p_{b,m-1}(n)$, respectively, for $m = 1, 2, \ldots, M$. Similarly, the computation of $\epsilon_m(n)$ involves the auxiliary parameter $p_{m-1}(n)$ for $m = 1, 2, \ldots, M + 1$. The computations of these three auxiliary parameters themselves have the following common features:
   - They are all governed by *first-order difference equations.*
   - The coefficients of the equation are *time varying.* For exponential weighting (i.e., $\lambda \leq 1$), the coefficients are bounded in absolute value by one. Hence, the solution of the equation is convergent.
   - The term playing the role of "excitation" is represented by some form of an estimation error.
   - In the prewindowing method, all three auxiliary parameters are equal to zero for $n \leq 0$.

   Consequently, the auxiliary parameters $p_{f,m-1}(n)$ and $p_{b,m-1}(n)$ for $m = 1, 2, \ldots, M$, and $p_m(n)$ for $m = 0, 1, \ldots, M$, may be computed recursively in time.

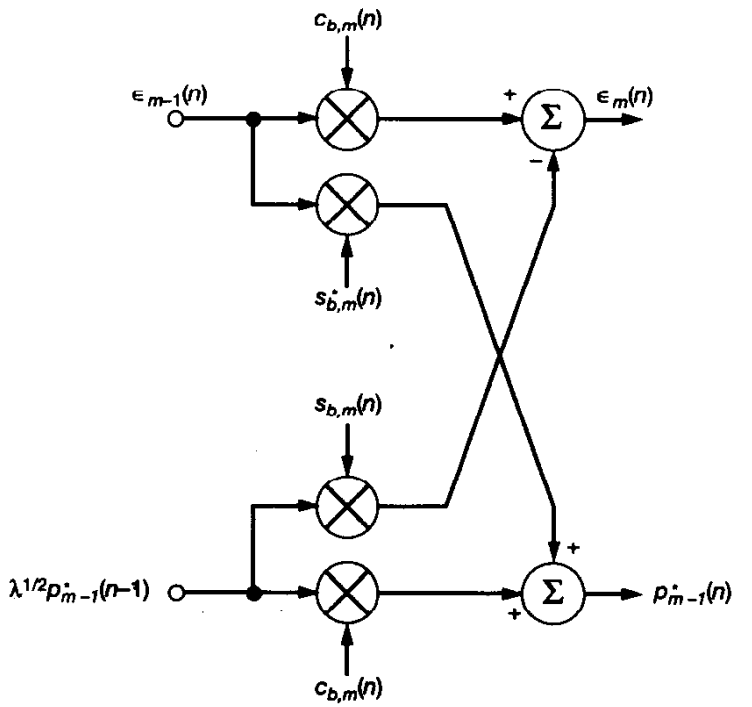
The auxiliary parameters $p_{f,m-1}(n)$, $p_{b,m-1}(n)$, and $p_{m-1}(n)$ in the QRD–LSL algorithm perform functions that are analogous to those of the forward reflection coefficient $\kappa_{f,m}(n)$, the backward reflection coefficient $\kappa_{b,m}(n)$, and the joint-process regression coefficient $\kappa_{m-1}(n)$, for prediction order m, respectively. Indeed, these three sets of parameters


$c_{b,m-1}(n-1)$
$\epsilon_{f,m-1}(n)$
$\epsilon_{f,m}(n)$
$s^*_{b,m-1}(n-1)$
$s_{b,m-1}(n-1)$
$\lambda^{1/2}p^*_{f,m-1}(n-1)$
$p^*_{f,m-1}(n)$
$c_{b,m-1}(n-1)$

(a)


$c_{f,m-1}(n)$
$\epsilon_{b,m-1}(n-1)$
$\epsilon_{b,m}(n)$
$s^*_{f,m-1}(n)$
$s_{f,m-1}(n)$
$\lambda^{1/2}p^*_{b,m-1}(n-1)$
$p^*_{b,m-1}(n)$
$c_{f,m-1}(n)$

(b)



(c)

Figure 15.9 Signal-flow graphs for computing normalized variables in the QRD–LSL algorithm: (a) Angle-normalized forward prediction error $\epsilon_{f,m}(n)$. (b) Angle-normalized backward prediction error $\epsilon_{b,m}(n)$. (c) Angle-normalized joint-process estimation error $\epsilon_m(n)$.

are related to each other by Eqs. (15.108), (15.118), and (15.128), respectively, which enables us to compute $\kappa_{f,m}(n)$, $\kappa_{b,m}(n)$, and $\kappa_{m-1}(n)$ *indirectly*, if so desired.

Finally, and perhaps most importantly, the angle-normalized QRD–LSL algorithm summarized in Table 15.3 plays a central role in the derivation of the whole family of recursive least-squares lattice (LSL) algorithms. We say so because all other existing recursive LSL algorithms using *a posteriori* estimation errors or *a priori* estimation errors (or combinations thereof) may be viewed as rewritings of the QRD–LSL algorithm (Sayed and Kailath, 1994). The validity of this statement is demonstrated in Sections 15.10 and 15.11.

15.9 COMPUTER EXPERIMENT ON ADAPTIVE EQUALIZATION

In this computer experiment, we study the use of the QRD–LSL algorithm for *adaptive equalization* of a linear channel that produces unknown distortion. The parameters of the channel are the same as those used to study the RLS algorithm in Section 13.8 for a similar application. The results of the experiment should therefore help us to make an assessment of the performance of this order-recursive algorithm compared to the standard RLS algorithm.

The parameters of the QRD–LSL algorithm studied here are identical to those used for the RLS algorithm in Section 13.8:

Exponential weighting factor:	$\lambda = 1$ (for stationary data)
Prediction order:	$M = 10$
Number of equalizer taps:	$M + 1 = 11$
Initializing constant:	$\delta = 0.004$

The computer simulations were run for four different values of the channel parameter W defined in Eq. (9.105), namely, $W = 2.9$, 3.1, 3.3, and 3.5. These values of W correspond to the following eigenvalue spreads of the underlying correlation matrix $\mathbf{R}$ of the channel output (equalizer input): $\chi(\mathbf{R}) = 6.0782$, 11.1238, 21.7132, and 46.8216, respectively. The signal-to-noise ratio measured at the channel output was 30 dB. For more details of the experimental setup, the reader is referred to Sections 9.16 and 13.8.

Learning Curves

Figure 15.10 presents the superposition of learning curves of the QRD–LSL algorithm for the exponential weighting factor $\lambda = 1$, and four different values of the channel parameter $W = 2.9$, 3.1, 3.3, and 3.5. Each learning curve was obtained by ensemble-averaging the squared value of the final *a priori* estimation error (i.e., the innovation) $\xi_{M+1}(n)$ over 200 independent trials of the experiment for a final prediction order $M = 10$. To compute the *a priori* estimation error $\xi_{M+1}(n)$, we first recognize that

$$\xi_{M+1}(n) = \frac{e_{M+1}(n)}{\gamma_{M+1}(n)}$$

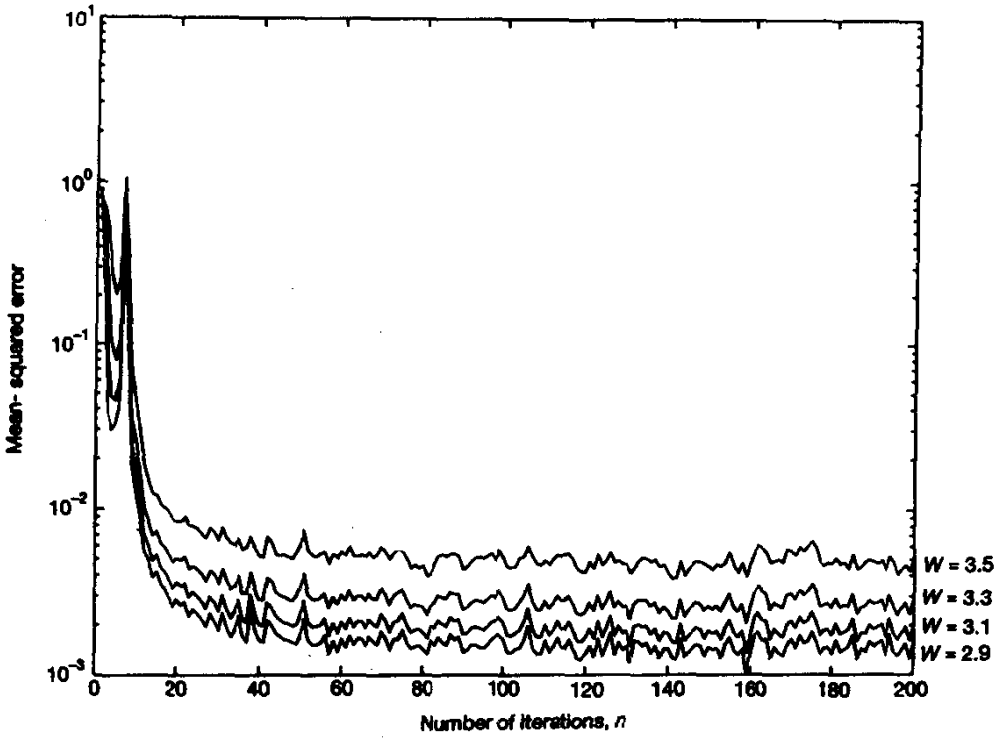


Figure 15.10 Learning curves of the QRD–LSL algorithm for the adaptive equalization experiment.

Equivalently, we may write

$$\xi_{M+1}(n) = \frac{\epsilon_{M+1}(n)}{\gamma_{M+1}^{1/2}(n)}$$

where $\epsilon_{M+1}(n)$ is the final value of the angle-normalized joint-process estimation error, and γ_{M+1} is the associated conversion factor. Note that for a final prediction order M, the number of taps involved in the joint-process estimation is $M + 1$.

For each eigenvalue spread, the learning curve of the QRD–LSL algorithm follows a path practically identical to that of the RLS algorithm, once the initialization is completed. This is readily confirmed by comparing the plots of Fig. 15.10 with those of Fig. 13.7. In both cases, double-precision arithmetic was used so that finite-precision effects are negligible.

Conversion Factor

In Fig. 15.11, we show the superposition of four ensemble-averaged plots of the conversion factor $\gamma_{M+1}(n)$ (for the final stage) versus the number of iterations n, corresponding to the four different values of the eigenvalue spread $\chi(\mathbf{R})$ as defined above. The curves plotted here are obtained by ensemble-averaging $\gamma_{M+1}(n)$ over 200 independent trials of

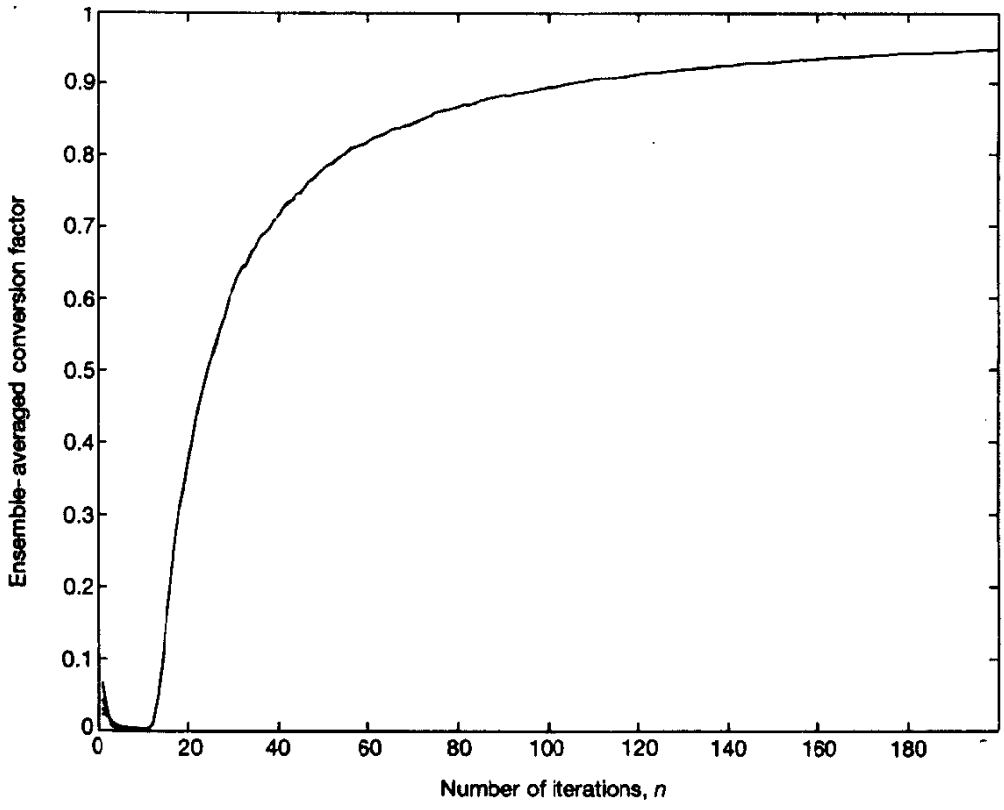


Figure 15.11 Ensemble-averaged conversion factor $\gamma_{M+1}(n)$ for varying eigenvalve spread.

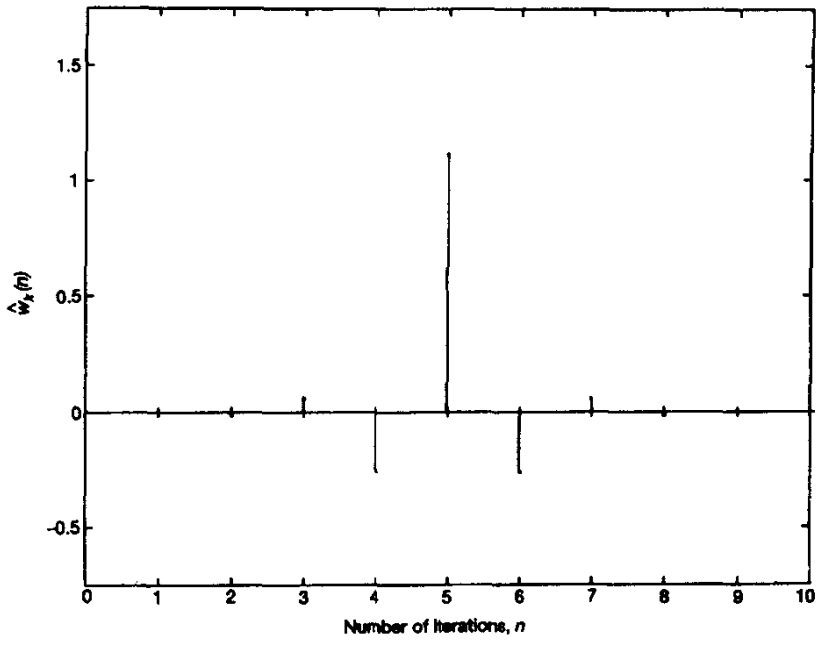
the experiment. It is noteworthy that the time variation of this ensemble-averaged conversion factor $E[\gamma_m(n)]$, follows an *inverse* law, as shown by

$$E[\gamma_m(n)] \simeq 1 - \frac{m}{n} \qquad \text{for } m = 1, 2, \ldots, M+1, \text{ and } n \geq m$$

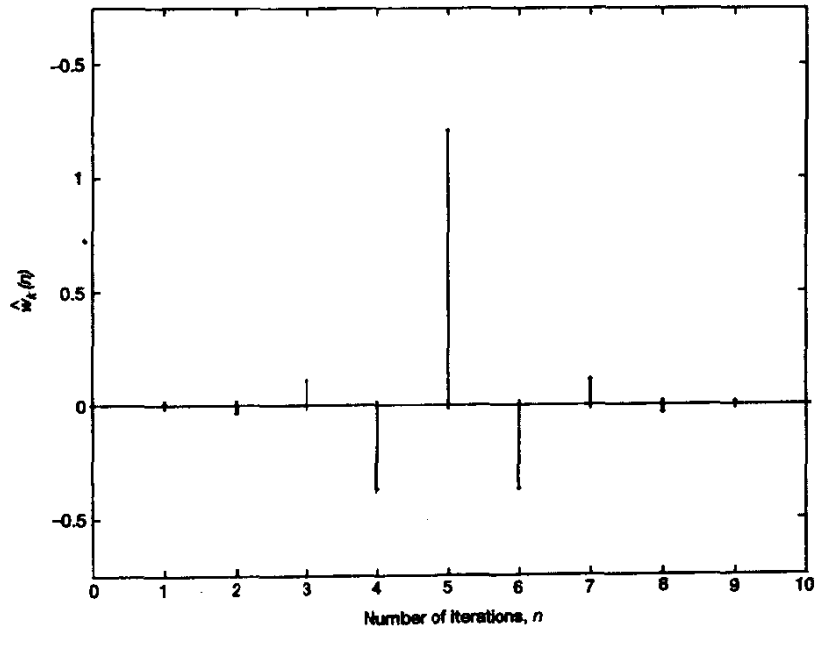
This equation provides a good fit to the experimentally computed curve shown in Fig. 15.11 particularly for n large compared to the predictor order $m = M + 1$. The reader is invited to check the validity of this fit. Note that the experimental plots of the conversion factor $\gamma_{M+1}(n)$ are insensitive to variations in the eigenvalue spread of the correlation matrix of the equalizer input for $n \geq 10$.

Impulse Response

In Fig. 15.12 we have plotted the ensemble-averaged impulse response of the adaptive equalizer after $n = 500$ iterations for each of the four eigenvalue spreads. As before, this ensemble-averaging was performed over 200 independent trials of the experiment. The

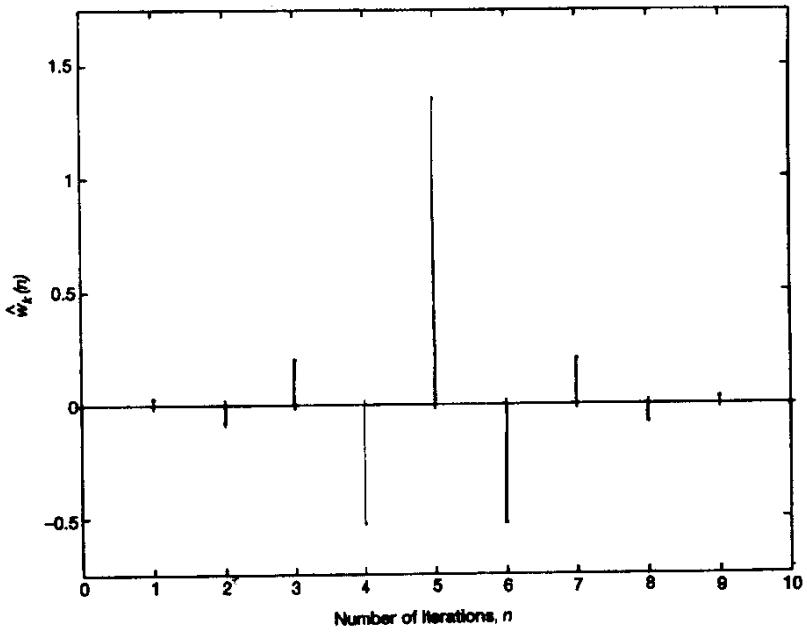


(a)

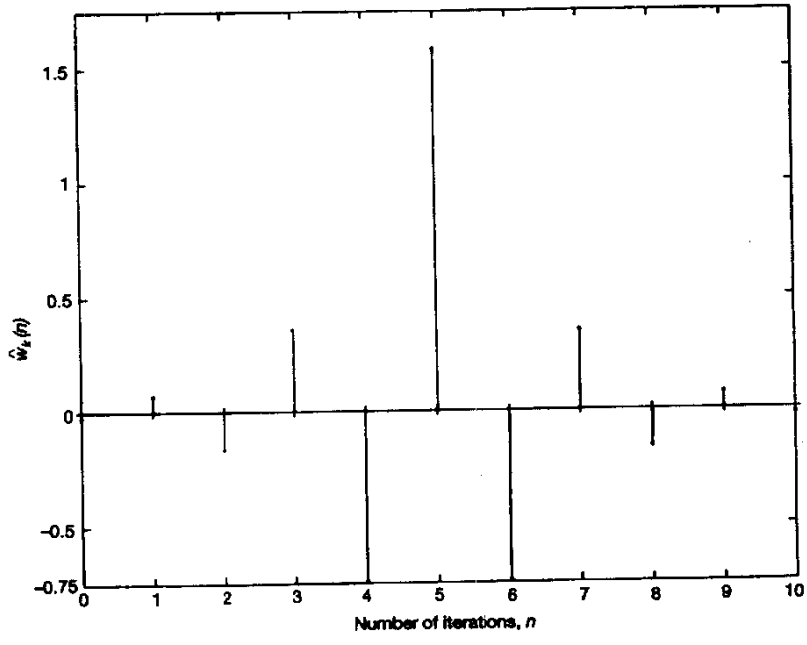


(b)

Figure 15.12 Ensemble-averaged impulse response of the adaptive equalizer for varying eigenvalve spread: (a) $W = 2.9$; $\chi(\mathbf{R}) = 6.0782$. (b) $W = 3.1$; $\chi(\mathbf{R}) = 11.1238$. (c) $W = 3.3$; $\chi(\mathbf{R}) = 21.7132$. (d) $W = 3.5$; $\chi(\mathbf{R}) = 46.8216$. Parts (c) and (d) of the figure are presented on the next page.



(c)



(d)

Figure 15.12 (*contd.*)

results of Fig. 15.12 for the QRD–LSL algorithm are, for all practical purposes, indistinguishable from the corresponding results of Fig. 9.20 for the LMS algorithm for a similar application.

15.10 EXTENDED QRD–LSL ALGORITHM

The QRD–LSL algorithm summarized in Table 15.3 permits computations of the forward and backward reflection coefficients and joint-process regression coefficients in an indirect manner; see Eqs. (15.108), (15.118), and (15.128). If *direct* computations of these coefficients are desired, we may use the *extended QRD–LSL algorithm* that follows from the use of the extended square-root information filter. Specifically, we may expand the three arrays of Table 15.3 as described here.

1. *Adaptive forward prediction*

$$\begin{bmatrix} \lambda^{1/2}\mathcal{B}_{m-1}^{1/2}(n-2) & \epsilon_{b,m-1}(n-1) \\ \lambda^{1/2}p_{f,m-1}^{*}(n-1) & \epsilon_{f,m-1}(n) \\ 0 & \gamma_{m-1}^{1/2}(n-1) \\ \lambda^{-1/2}\mathcal{B}_{m-1}^{-1/2}(n-2) & 0 \end{bmatrix} \Theta_{b,m-1}(n-1)$$

$$= \begin{bmatrix} \mathcal{B}_{m-1}^{1/2}(n-1) & 0 \\ p_{f,m-1}^{*}(n) & \epsilon_{f,m}(n) \\ b_{m-1}^{*}(n-1)\mathcal{B}_{m-1}^{-1/2}(n-1) & \gamma_{m}^{1/2}(n-1) \\ \mathcal{B}_{m-1}^{-1/2}(n-1) & -\bar{k}_{b,m-1}(n-1) \end{bmatrix} \tag{15.134}$$

The complex-valued quantity $\bar{k}_{b,m-1}(n-1)$ in the last row of the postarray is the *normalized gain factor*, which is determined by backward prediction variables; it is defined by

$$\bar{k}_{b,m-1}(n-1) = \gamma_{m}^{-1/2}(n-1)k_{b,m-1}(n-1) \tag{15.135}$$

The normalized gain factor $\bar{k}_{b,m-1}(n-1)$ may be used to update the forward reflection coefficient in accordance with the (Kalman filtering) formula:

$$\begin{aligned} \kappa_{f,m}(n) &= \kappa_{f,m}(n-1) + k_{b,m-1}(n-1)\eta_{m}^{*}(n) \\ &= \kappa_{f,m}(n-1) + (\gamma_{m}^{-1/2}(n-1)k_{b,m-1}(n-1))\,(\gamma_{m}^{1/2}(n-1)\eta_{m}(n))^{*} \\ &= \kappa_{f,m}(n-1) + \bar{k}_{b,m-1}(n-1)\epsilon_{f,m}^{*}(n) \end{aligned} \tag{15.136}$$

where the entries $\bar{k}_{b,m-1}(n-1)$ and $\epsilon_{f,m}(n)$ are read directly from the last and second rows of the postarray in Eq. (15.134).

2. *Adaptive backward prediction*

$$\begin{bmatrix} \lambda^{1/2}\mathcal{F}_{m-1}^{1/2}(n-1) & \epsilon_{f,m-1}(n) \\ \lambda^{1/2}p_{b,m-1}^{*}(n-1) & \epsilon_{b,m-1}(n-1) \\ 0 & \gamma_{m-1}^{1/2}(n-1) \\ \lambda^{-1/2}\mathcal{F}_{m-1}^{-1/2}(n-1) & 0 \end{bmatrix} \Theta_{f,m-1}(n)$$

$$= \begin{bmatrix} \mathcal{F}_{m-1}^{1/2}(n) & 0 \\ p_{b,m-1}^{*}(n) & \epsilon_{b,m}(n) \\ f_{m-1}^{*}(n)\mathcal{F}_{m-1}^{-1/2}(n) & \gamma_{m}^{1/2}(n) \\ \mathcal{F}_{m-1}^{-1/2}(n) & -\bar{k}_{f,m-1}(n) \end{bmatrix} \tag{15.137}$$

The complex-valued quantity $\bar{k}_{f,m-1}(n)$, appearing in the last row of the postarray, is the normalized gain factor determined by forward prediction variables; it is defined by

$$\bar{k}_{f,m-1}(n) = \gamma_{m}^{-1/2}(n)k_{f,m-1}(n) \tag{15.138}$$

As such, it may be used to update the backward reflection coefficient in accordance with the (Kalman filtering) formula:

$$\begin{aligned} \kappa_{b,m}(n) &= \kappa_{b,m}(n-1) + k_{f,m-1}(n)\beta_{m}^{*}(n) \\ &= \kappa_{b,m}(n-1) + (\gamma_{m}^{-1/2}(n)k_{f,m-1}(n))\,(\gamma_{m}^{1/2}(n)\beta_{m}(n))^{*} \\ &= \kappa_{b,m}(n-1) + \bar{k}_{f,m-1}(n)\epsilon_{b,m}^{*}(n) \end{aligned} \tag{15.139}$$

where the entries $\bar{k}_{f,m-1}(n)$ and $\epsilon_{b,m}(n)$ are read directly from the last and second rows of the postarray in Eq. (15.137).

3. *Joint-process estimation*

$$\begin{bmatrix} \lambda^{1/2}\mathcal{B}_{m-1}^{1/2}(n-1) & \epsilon_{b,m-1}(n) \\ \lambda^{1/2}p_{m-1}^{*}(n-1) & \epsilon_{m-1}(n) \\ 0 & \gamma_{m-1}^{1/2}(n) \\ \lambda^{-1/2}\mathcal{B}_{m-1}^{-1/2}(n-1) & 0 \end{bmatrix} \Theta_{b,m-1}(n)$$

$$= \begin{bmatrix} \mathcal{B}_{m-1}^{1/2}(n) & 0 \\ p_{m-1}^{*}(n) & \epsilon_{m}(n) \\ b_{m-1}^{*}(n)\mathcal{B}_{m-1}^{-1/2}(n) & \gamma_{m}^{1/2}(n) \\ \mathcal{B}_{m-1}^{-1/2}(n) & -\bar{k}_{b,m-1}(n) \end{bmatrix} \tag{15.140}$$

The regression coefficient $\kappa_{m-1}(n)$ is computed directly in terms of the normalized gain factor $\bar{k}_{b,m-1}(n)$, appearing in the last row of the postarray, as follows:

$$\begin{aligned}\kappa_{m-1}(n) &= \kappa_{m-1}(n-1) + k_{b,m-1}(n)\xi_m^*(n) \\ &= \kappa_{m-1}(n-1) + (k_{b,m-1}(n)\gamma_m^{-1/2}(n))\,(\xi_m(n)\gamma_m{}^{1/2}(n))^* \\ &= \kappa_{m-1}(n-1) + \bar{k}_{b,m-1}(n)\epsilon_m^*(n)\end{aligned} \tag{15.141}$$

where the entries $\bar{k}_{b,m-1}(n)$ and $\epsilon_m(n)$ are read directly from the last and second rows of the postarray of Eq. (15.140).

Despite the fact that the extended QRD–LSL algorithm is able to compute the forward and backward reflection coefficients and joint-process regression coefficients directly, the QRD–LSL algorithm of Table 15.3 is to be preferred over it for two practical reasons:

1. An inherently simpler structure
2. A potentially better numerical behavior

The first point is obvious in light of what we have already said. The second point needs to be explained. Each array of the QRD–LSL algorithm propagates a *single* square root, namely, $\mathcal{B}_{m-1}^{1/2}(n-2)$ or $\mathcal{F}_{m-1}^{1/2}(n-1)$. On the other hand, each array of the extended QRD–LSL algorithm propagates one or the other of these two square roots and, in addition, the inverse of that particular square root. Therefore, finite-precision effects may cause a numerical discrepancy to arise between each of these two square roots and its inverse, unless proper precautions are taken.

15.11 RECURSIVE LEAST-SQUARES LATTICE FILTERS USING *A POSTERIORI* ESTIMATION ERRORS

In a generic sense, the family of recursive LSL algorithms may be divided into two subgroups: those that involve the use of unitary rotations, and those that do not. A well-known algorithm that belongs to the latter subgroup is the standard *recursive LSL algorithm using a posteriori estimation errors* (Morf, 1974; Morf and Lee, 1978; Lee et al., 1981). The estimation errors used in this algorithm are represented by the *a posteriori* forward prediction error $f_m(n)$, the *a posteriori* backward prediction error $b_m(n)$, and the *a posteriori* joint-process estimation error $e_m(n)$, where $m = 0, 1, 2, \ldots, M$, as shown in the signal-flow graphs of Figs. 15.6 and 15.7.

We may derive this algorithm (and for that matter, other recursive LSL algorithms) from the QRD–LSL algorithm of Table 15.3 by using a simple two-step procedure:

- The three arrays of the QRD–LSL algorithm, dealing with adaptive forward prediction, adaptive backward prediction, and adaptive joint-process estimation are squared; the effects of unitary rotations are thereby completely removed from the algorithm.

- Certain terms (depending on the algorithm of interest) on both sides of the resultant arrays are retained, and then compared.

Examples of this procedure were presented in Chapter 14 dealing with square-root adaptive filters.

Applying this procedure to the three arrays of Table 15.3 that describe the angle-normalized QRD–LSL algorithm, and expressing the results in terms of the *a posteriori* estimation errors, we get the following three sets of recursions:

1. *Adaptive forward prediction*:

$$\mathcal{B}_{m-1}(n-1) = \lambda \mathcal{B}_{m-1}(n-2) + \frac{|b_{m-1}(n-1)|^2}{\gamma_{m-1}(n-1)} \tag{15.142}$$

$$\Delta_{m-1}(n) = \lambda \Delta_{m-1}(n-1) + \frac{b_{m-1}(n-1) f^*_{m-1}(n)}{\gamma_{m-1}(n-1)} \tag{15.143}$$

$$\kappa_{f,m}(n) = -\frac{\Delta_{m-1}(n)}{\mathcal{B}_{m-1}(n-1)} \tag{15.144}$$

$$f_m(n) = f_{m-1}(n) + \kappa^*_{f,m}(n) b_{m-1}(n-1) \tag{15.145}$$

$$\gamma_m(n-1) = \gamma_{m-1}(n-1) - \frac{|b_{m-1}(n-1)|^2}{\mathcal{B}_{m-1}(n-1)} \tag{15.146}$$

where in the second line we have made use of the relation between $\Delta_{m-1}(n)$ and $p_{f,m-1}(n)$ given in Eq. (15.106), and in the third line we have made use of the definition of the forward reflection coefficient $\kappa_{f,m}(n)$ given in Eq. (15.108).

2. *Adaptive backward prediction*:

$$\mathcal{F}_{m-1}(n) = \lambda \mathcal{F}_{m-1}(n-1) + \frac{|f_{m-1}(n)|^2}{\gamma_{m-1}(n-1)} \tag{15.147}$$

$$\kappa_{b,m}(n) = -\frac{\Delta^*_{m-1}(n)}{\mathcal{F}_{m-1}(n)} \tag{15.148}$$

$$b_m(n) = b_{m-1}(n-1) + \kappa^*_{b,m}(n) f_{m-1}(n) \tag{15.149}$$

where in the second line we have made use of the relation between $\Delta_{m-1}(n)$ and $p_{b,m-1}(n)$ given in Eq. (15.118), and also the definition of the backward reflection coefficient $\kappa_{b,m}(n)$ given in Eq. (15.119).

3. *Adaptive joint-process estimation*:

$$\pi_{m-1}(n) = \lambda \pi_{m-1}(n-1) + \frac{b_{m-1}(n) e^*_{m-1}(n)}{\gamma_{m-1}(n)} \tag{15.150}$$

$$\kappa_{m-1}(n) = \frac{\pi_{m-1}(n)}{\mathcal{B}_{m-1}(n)} \tag{15.151}$$

$$e_m(n) = e_{m-1}(n) - \kappa^*_{m-1}(n) b_{m-1}(n) \tag{15.152}$$

where $\pi_{m-1}(n)$ is defined in terms of $p_{m-1}(n)$ by

$$\pi_{m-1}(n) = \mathcal{B}_{m-1}^{1/2}(n)p_{m-1}(n) \tag{15.153}$$

and the joint-process regression coefficient $\kappa_{m-1}(n)$ is defined in terms of $p_{m-1}(n)$ by

$$\kappa_{m-1}(n) = \frac{p_{m-1}(n)}{\mathcal{B}_{m-1}^{1/2}(n)} \tag{15.154}$$

Summary of the Recursive LSL Algorithm Using *A Posteriori* Estimation Errors

The complete list of order- and time-update recursions constituting the recursive LSL algorithm (based on *a posteriori* estimation errors) is summarized in Table 15.4. This summary includes the recursions and relations of Eqs. (15.143), (15.142), (15.147), (15.144), (15.148), (15.145), (15.149), (15.146), (15.150), (15.151), and (15.152), in that order.

Since the LSL algorithm summarized in Table 15.4 involves division by updated parameters at some of the steps, care must be taken to ensure that these values are not allowed to become too small. Unless a high-precision computer is used, selection of the constant δ [determining the initial values $\mathcal{F}_0(0)$ and $\mathcal{B}_0(0)$] may have a severe effect on the initial transient performance of the LSL algorithm. Friedlander (1982) suggests using some form of *thresholding*, in that if the divisor (in any computation of LSL algorithm) is less than this preassigned threshold, the corresponding term involving that divisor is set to be zero. This remark also applies to other versions of the recursive LSL algorithm, e.g., those summarized in Table 15.5.

Initialization of the Recursive LSL Algorithm

To initialize the recursive LSL algorithm using *a posteriori* estimation errors, we start with the elementary case of zero prediction order, for which we have [see Eq. (15.43)]

$$f_0(n) = b_0(n) = u(n)$$

where $u(n)$ is the lattice predictor input at time n.

The remaining set of initial values pertain to the sums of weighted *a posteriori* prediction-error squares for zero prediction order. Specifically, setting $m - 1 = 0$ in Eq. (15.117) yields

$$\mathcal{F}_0(n) = \lambda\mathcal{F}_0(n - 1) + |u(n)|^2 \tag{15.155}$$

Similarly, setting $m - 1 = 0$ and replacing $n - 1$ with n in Eqs. (15.105) yields

$$\mathcal{B}_0(n) = \lambda\mathcal{B}_0(n - 1) + |u(n)|^2 \tag{15.156}$$

With the conversion factor $\gamma_m(n - 1)$ bounded by zero and 1, a logical choice for the zeroth-order value of this parameter is

$$\gamma_0(n - 1) = 1 \tag{15.157}$$

TABLE 15.4 SUMMARY OF THE RECURSIVE LSL ALGORITHM USING *A POSTERIORI* ESTIMATION ERRORS

Predictions:
For $n = 1, 2, 3, \ldots$, compute the various order updates in the sequence $m = 1, 2, \ldots, M$, where M is the final order of the least-squares lattice predictor:

$$\Delta_{m-1}(n) = \lambda\Delta_{m-1}(n-1) + \frac{b_{m-1}(n-1)f^*_{m-1}(n)}{\gamma_{m-1}(n-1)}$$

$$\mathcal{B}_{m-1}(n-1) = \lambda\mathcal{B}_{m-1}(n-2) + \frac{|b_{m-1}(n-1)|^2}{\gamma_{m-1}(n-1)}$$

$$\mathcal{F}_{m-1}(n) = \lambda\mathcal{F}_{m-1}(n-1) + \frac{|f_{m-1}(n)|^2}{\gamma_{m-1}(n-1)}$$

$$\kappa_{f,m}(n) = -\frac{\Delta_{m-1}(n)}{\mathcal{B}_{m-1}(n-1)}$$

$$\kappa_{b,m}(n) = -\frac{\Delta^*_{m-1}(n)}{\mathcal{F}_{m-1}(n)}$$

$$f_m(n) = f_{m-1}(n) + \kappa^*_{f,m}(n)b_{m-1}(n-1)$$

$$b_m(n) = b_{m-1}(n-1) + \kappa^*_{b,m}(n)f_{m-1}(n)$$

$$\gamma_m(n-1) = \gamma_{m-1}(n-1) - \frac{|b_{m-1}(n-1)|^2}{\mathcal{B}_{m-1}(n-1)}$$

Filtering:
For $n = 1, 2, 3, \ldots$ compute the various order updates in the sequence $m = 1, 2, \ldots, M+1$:

$$\pi_{m-1}(n) = \lambda\pi_{m-1}(n-1) + \frac{b_{m-1}(n)e^*_{m-1}(n)}{\gamma_{m-1}(n)}$$

$$\kappa_{m-1}(n) = \frac{\pi_{m-1}(n)}{\mathcal{B}_{m-1}(n)}$$

$$e_m(n) = e_{m-1}(n) - \kappa^*_{m-1}(n)b_{m-1}(n)$$

Initialization:
1. To initialize the algorithm, at time $n = 0$ set

$$\Delta_{m-1}(0) = 0$$

$$\mathcal{F}_{m-1}(0) = \delta \qquad \delta = \text{small positive constant}$$

$$\mathcal{B}_{m-1}(-1) = \delta$$

$$\gamma_0(0) = 1$$

2. At each instant $n \geq 1$, generate the various zeroth-order variables as follows:

$$f_0(n) = b_0(n) = u(n)$$

$$\mathcal{F}_0(n) = \mathcal{B}_0(n) = \lambda\mathcal{F}_0(n-1) + |u(n)|^2$$

$$\gamma_0(n-1) = 1$$

3. For joint-process estimation, initialize the algorithm by setting at time $n = 0$

$$\pi_{m-1}(0) = 0$$

At each instant $n \geq 1$, generate the zeroth-order variable

$$e_0(n) = d(n)$$

Note: For prewindowed data, the input $u(n)$ and desired response $d(n)$ are both zero for $n \leq 0$.

We complete the initialization of the algorithm for forward and backward predictions by using the following conditions at time $n = 0$:

$$\Delta_{m-1}(0) = 0 \tag{15.158}$$

and

$$\mathcal{F}_{m-1}(0) = \mathcal{B}_{m-1}(-1) = \delta \tag{15.159}$$

where δ is a small positive constant. The constant δ is used to ensure nonsingularity of the correlation matrix $\mathbf{\Phi}_m(n)$.

Turning finally to the initialization of the joint-estimation process, we see (for zero-prediction order)

$$e_0(n) = d(n)$$

where $d(n)$ is the desired response. Thus, to initiate this part of the computation, we generate $e_0(n)$ for each instant n. To complete the initialization of the recursive LSL algorithm for joint-process estimation, at time $n = 0$ we set

$$\pi_{m-1}(0) = 0 \qquad \text{for } m = 1, 2, \ldots, M + 1$$

Table 15.4 includes the initialization of the recursive LSL algorithm as described above.

15.12 RECURSIVE LSL FILTERS USING *A PRIORI* ESTIMATION ERRORS WITH ERROR FEEDBACK

To add further support to the statement made in Section 15.9 that the QRD–LSL algorithm is fundamental to the derivation of all other recursive LSL algorithms, we now pursue the derivation of another recursive LSL algorithm. Specifically, the algorithm considered in this section differs from the recursive LSL algorithm of the previous section in two respects. First, it is based on *a priori* estimation errors. Second, the reflection and regression coefficients of the algorithm are all derived *directly*.

Following the two-step procedures described previously (i.e., squaring and then comparing terms), but this time using the arrays of Eqs. (15.134), (15.137), and (15.140), we get the following three sets of results expressed in terms of *a priori* estimation errors:

1. *Adaptive Forward Prediction*:

$$\mathcal{B}_{m-1}(n-1) = \lambda \mathcal{B}_{m-1}(n-2) + \gamma_{m-1}(n-1)|\beta_{m-1}(n-1)|^2 \tag{15.160}$$

$$\gamma_m(n-1)\eta_m(n) = \gamma_{m-1}(n-1)\eta_{m-1}(n) + \gamma_{m-1}(n-1)\kappa_{f,m}^*(n)\beta_{m-1}(n-1) \tag{15.161}$$

$$\kappa_{f,m}(n) = \kappa_{f,m}(n-1) + k_{b,m-1}(n-1)\eta_m^*(n) \tag{15.162}$$

$$\gamma_m(n-1) = \gamma_{m-1}(n-1) - \frac{\gamma_{m-1}^2(n-1)}{\mathcal{B}_{m-1}(n-1)}|\beta_{m-1}(n-1)|^2 \tag{15.163}$$

where in the second line we have made use of the definition given in Eq. (15.108) for the forward reflection coefficient $\kappa_{f,m}(n)$, and in the third line we have made use of Eqs. (15.77) and (15.135). The order update of Eq. (15.161) is not quite in the right form yet. To put it in the right form, we substitute Eq. (15.163) in (15.161), cancel the common conversion factor $\gamma_{m-1}(n-1)$, and rearrange terms. We thus obtain

$$\eta_m(n) = \eta_{m-1}(n) + [\kappa_{f,m}^*(n) + \frac{\gamma_{m-1}(n-1)\beta_{m-1}^*(n-1)}{\mathcal{B}_{m-1}(n-1)}\eta_m(n)]\beta_{m-1}(n-1) \tag{15.164}$$

Define the gain factor $k_{b,m-1}(n-1)$ in terms of backward prediction variables as[3]

$$k_{b,m-1}(n-1) = -\frac{\gamma_{m-1}(n-1)\beta_{m-1}(n-1)}{\mathcal{B}_{m-1}(n-1)} \tag{15.165}$$

so that we may rewrite Eq. (15.162) as

$$\kappa_{f,m}(n) = \kappa_{f,m}(n-1) - \frac{\gamma_{m-1}(n-1)\beta_{m-1}(n-1)}{\mathcal{B}_{m-1}(n-1)}\eta_m^*(n) \tag{15.166}$$

Accordingly, we may recast the order update of Eq. (15.164) into the desired form:

$$\eta_m(n) = \eta_{m-1}(n) + \kappa_{f,m}^*(n-1)\beta_{m-1}(n-1) \tag{15.167}$$

where $\kappa_{f,m}(n-1)$ is the *past* value of the forward reflection coefficient for stage m of the lattice predictor; see Eq. (15.96).

2. *Adaptive Backward Prediction*:

$$\mathcal{F}_{m-1}(n) = \lambda\mathcal{F}_{m-1}(n-1) + \gamma_{m-1}(n-1)|\eta_{m-1}(n)|^2 \tag{15.168}$$

$$\gamma_m(n)\beta_m(n) = \gamma_{m-1}(n-1)\beta_{m-1}(n-1) + \gamma_{m-1}\kappa_{b,m}^*(n)\eta_{m-1}(n) \tag{15.169}$$

$$\kappa_{b,m}(n) = \kappa_{b,m}(n-1) + k_{f,m-1}(n)\beta_m^*(n) \tag{15.170}$$

$$\gamma_m(n) = \gamma_{m-1}(n-1) - \frac{\gamma_{m-1}^2(n-1)|\eta_{m-1}(n)|^2}{\mathcal{F}_{m-1}(n)} \tag{15.171}$$

where in the second line we have made use of the definition given in Eq. (15.119) for the backward reflection coefficient $\kappa_{b,m}(n)$ of stage m in the least-squares lattice predictor, and in the third line we have made use of Eqs. (15.78) and (15.138). Here again, we find that the order update of Eq. (15.169) is note quite in the right form. To put it in the right form, we substitute Eq. (15.171) in

[3]The formula of Eq. (15.165), defining the gain factor $k_{b,m-1}(n-1)$, is in perfect accord with that used in the traditional approach to the derivation of recursive least-squares lattice filters (Haykin, 1991).

(15.169), cancel the common conversion factor $\gamma_{m-1}(n-1)$, and rearrange terms. We may thus write

$$\beta_m(n) = \beta_{m-1}(n-1) + \left[\kappa_{b,m}^*(n) + \frac{\gamma_{m-1}(n-1)|\eta_{m-1}^*(n)|^2}{\mathcal{F}_{m-1}(n)}\beta_m(n)\right]\eta_{m-1}(n) \tag{15.172}$$

Define the gain factor $k_{f,m-1}(n)$ in terms of forward prediction variables as[4]

$$k_{f,m-1}(n) = -\frac{\gamma_{m-1}(n-1)\eta_{m-1}(n)}{\mathcal{F}_{m-1}(n)} \tag{15.173}$$

so that we may rewrite the order update of Eq. (15.170) as

$$k_{b,m}(n) = \kappa_{b,m}(n-1) - \frac{\gamma_{m-1}(n-1)\eta_{m-1}(n)}{\mathcal{F}_{m-1}(n)}\beta_m^*(n) \tag{15.174}$$

Now we may put the order update of Eq. (15.172) in the desired form:

$$\beta_m(n) = \beta_{m-1}(n-1) + \kappa_{b,m}^*(n-1)\eta_{m-1}(n) \tag{15.175}$$

where $\kappa_{b,m}(n-1)$ is the *past* value of the backward reflection coefficient for stage m of the least-squares lattice predictor; see Eq. (15.97).

3. *Adaptive Joint-process Estimation*:

$$\gamma_m(n)\xi_m(n) = \gamma_{m-1}(n)\xi_{m-1}(n) - \gamma_{m-1}(n)\kappa_{m-1}^*(n)\beta_{m-1}(n) \tag{15.176}$$

$$\kappa_{m-1}(n) = \kappa_{m-1}(n-1) + k_{b,m-1}(n)\xi_m^*(n) \tag{15.177}$$

where in the first line we have made use of the definition given in Eq. (15.128) for the joint-process regression coefficient $\kappa_{m-1}(n)$, and in the second line we have made use of Eqs. (15.79) and (15.135). Here also we find that the order update of Eq. (15.176) is not quite in the right form. To put it in the right form, we substitute the following time update [obtained by replacing $n-1$ with n in Eq. (15.163)]

$$\gamma_m(n) = \gamma_{m-1}(n) - \frac{\gamma_{m-1}^2(n)}{\mathcal{B}_{m-1}(n)}|\beta_{m-1}(n)|^2$$

in Eq. (15.176), cancel the common conversion factor $\gamma_{m-1}(n)$, and rearrange terms. We may thus write

$$\xi_m(n) = \xi_{m-1}(n) + \left(\frac{\gamma_{m-1}(n)\beta_{m-1}^*(n)}{\mathcal{B}_{m-1}(n)}\xi_m(n) - \kappa_{m-1}^*(n)\right)\beta_{m-1}(n) \tag{15.178}$$

[4]The definition given in Eq. (15.173) for the gain factor $k_{f,m-1}(n)$ is of exactly the same form as that used in the traditional approach to the derivation of recursive least-squares lattice filters (Haykin, 1991).

Using the definition of the gain factor $k_{b,m-1}(n)$, obtained by replacing $n-1$ in Eq. (15.165) with n, we may rewrite the time update of Eq. (15.177) as

$$\kappa_{m-1}(n) = \kappa_{m-1}(n-1) + \frac{\gamma_{m-1}(n)\beta_{m-1}(n)}{\mathcal{B}_{m-1}(n)}\,\xi_m^*(n) \tag{15.179}$$

Accordingly, we may recast Eq. (15.178) into the desired order update form:

$$\xi_m(n) = \xi_{m-1}(n) - \kappa_{m-1}^*(n-1)\beta_{m-1}(n) \tag{15.180}$$

where $\kappa_{m-1}(n-1)$ is the past value of the regression coefficient for prediction order $m-1$.

Summary of the Recursive LSL Algorithm Using *A Priori* Estimation Errors with Error Feedback

Eqs. (15.168), (15.160), (15.167), (15.175), (15.166), (15.174), and (15.163), in that order, define the computations involved in the forward and backward predictions of the recursive LSL algorithm using *a priori* estimation errors with error feedback. Equations (15.180) and (15.179) define the computations involved in the joint-process estimation part of the algorithm. A complete summary of the algorithm, including initial conditions of the soft-constraint form, is presented in Table 15.5.

Figure 15.13 presents a signal-flow graph of this new algorithm, emphasizing that order updating of the variables of interest (i.e., *a priori* forward prediction, backward prediction and joint-process estimation errors) at iteration n requires knowledge of the forward reflection coefficients, backward reflections coefficients, and regression coefficients at the previous iteration $n-1$.

An important difference between the two recursive LSL algorithms summarized in Tables 15.4 and 15.5 is the way in which the reflection coefficients and regression coefficients are updated. In the case of Table 15.4, the updating is performed *indirectly*. We first compute the cross-correlation between forward and delayed backward prediction errors and the cross-correlation between backward prediction errors and joint-process estimation errors. Next, we compute the sum of weighted forward prediction-error squares and the sum of weighted backward-error squares. Finally, we compute the reflection and regression coefficients by dividing a cross-correlation by a sum of weighted prediction-error squares. On the other hand, in Table 15.5, the updating of the reflection and regression coefficients is performed *directly*. The differences between indirect and direct forms of updating, as described herein, have an important bearing on the numerical behavior of these recursive LSL algorithms; this issue is discussed in detail in Chapter 17.

15.13 COMPUTATION OF THE LEAST-SQUARES WEIGHT VECTOR

In solving the joint-process estimation problem with an order-recursive adaptive filter, we have shown how the least-squares predictor can be expanded to include the estimation of a desired response. The solution to this problem encompasses the computation of a set of

TABLE 15.5 SUMMARY OF THE RECURSIVE LSL ALGORITHM USING *A PRIORI* ESTIMATION ERRORS WITH ERROR FEEDBACK

Predictions:
For $n = 1, 2, 3, \ldots$, compute the various order updates in the sequence $m = 1, 2, \ldots, M$, where M is the final order of the least-squares predictor:

$$\mathcal{F}_{m-1}(n) = \lambda \mathcal{F}_{m-1}(n-1) + \gamma_{m-1}(n-1)|\eta_{m-1}(n)|^2$$

$$\mathcal{B}_{m-1}(n-1) = \lambda \mathcal{B}_{m-1}(n-2) + \gamma_{m-1}(n-1)|\beta_{m-1}(n-1)|^2$$

$$\eta_m(n) = \eta_{m-1}(n) + \kappa_{f,m}^*(n-1)\beta_{m-1}(n-1)$$

$$\beta_m(n) = \beta_{m-1}(n-1) + \kappa_{b,m}^*(n-1)\eta_{m-1}(n)$$

$$\kappa_{f,m}(n) = \kappa_{f,m}(n-1) - \frac{\gamma_{m-1}(n-1)\beta_{m-1}(n-1)}{\mathcal{B}_{m-1}(n-1)}\,\eta_m^*(n)$$

$$\kappa_{b,m}(n) = \kappa_{b,m}(n-1) - \frac{\gamma_{m-1}(n-1)\eta_{m-1}(n)}{\mathcal{F}_{m-1}(n)}\,\beta_m^*(n)$$

$$\gamma_m(n-1) = \gamma_{m-1}(n-1) - \frac{\gamma_{m-1}^2(n-1)|\beta_{m-1}(n-1)|^2}{\mathcal{B}_{m-1}(n-1)}$$

Filtering:
For $n = 1, 2, 3, \ldots$, compute the various order updates in the sequence $m = 1, 2, \ldots, M+1$:

$$\xi_m(n) = \xi_{m-1}(n) - \kappa_{m-1}^*(n-1)\beta_{m-1}(n)$$

$$\kappa_{m-1}(n) = \kappa_{m-1}(n-1) + \frac{\gamma_{m-1}(n)\beta_{m-1}(n)}{\mathcal{B}_{m-1}(n)}\,\xi_m^*(n)$$

Initialization:
1. To initialize the algorithm, at time $n = 0$ set

$$\mathcal{F}_{m-1}(0) = \delta, \qquad \delta = \text{small positive constant}$$

$$\mathcal{B}_{m-1}(-1) = \delta$$

$$\kappa_{f,m}(0) = \kappa_{b,m}(0) = 0$$

$$\gamma_0(0) = 1$$

2. For each instant $n \geq 1$, generate the zeroth-order variables:

$$\eta_0(n) = \beta_0(n) = u(n)$$

$$\mathcal{F}_0(n) = \mathcal{B}_0(n) = \lambda \mathcal{F}_0(n-1) + |u(n)|^2$$

$$\gamma_0(n-1) = 1$$

3. For joint-process estimation, at time $n = 0$ set

$$\kappa_{m-1}(0) = 0$$

At each instant $n \geq 1$, generate the zeroth-order variable

$$\xi_0(n) = d(n)$$

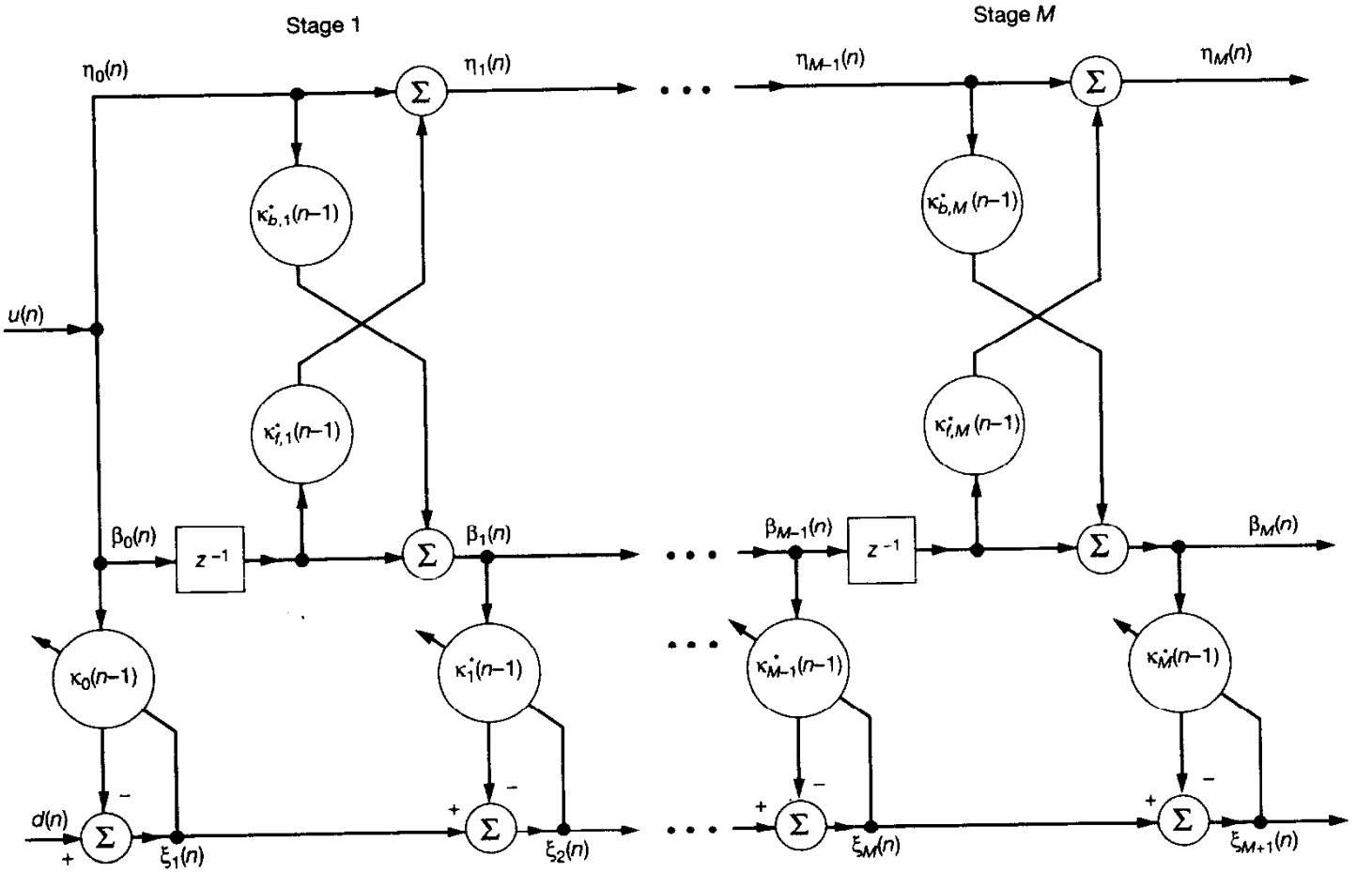


Figure 15.13 Joint process estimator using the recursive LSL algorithm based on *a priori* estimation errors.

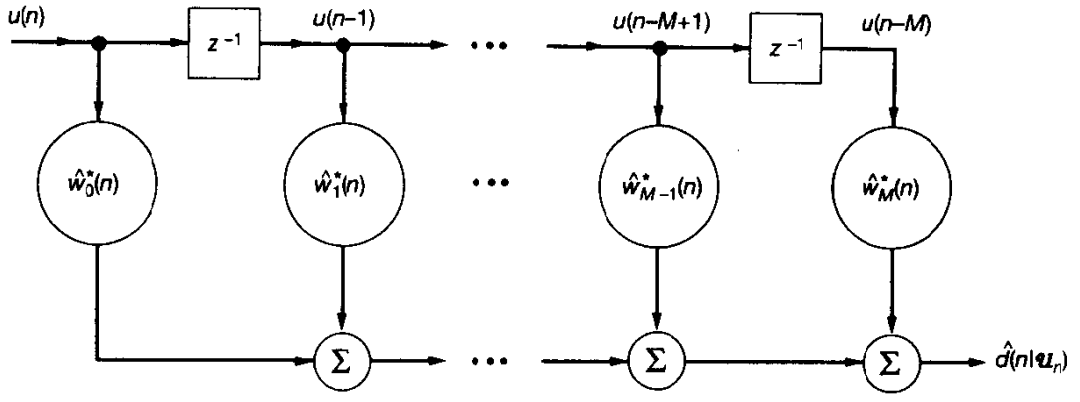


Figure 15.14 Conventional transversal fiter.

regression coefficient $\{\kappa_0(n), \kappa_1(n), \ldots, \kappa_M(n)\}$ that is fed with a corresponding set of inputs represented by the backward prediction errors $\{b_0(n), b_1, \ldots, b_M(n)\}$; see Fig. 15.7. Recognizing that there is a one-to-one correspondence between this set of backward prediction errors and the set of tap inputs $\{u(n), u(n-1), \ldots, (n-M)\}$, as shown in Eq. (15.70), we expect to find a corresponding relationship between the sequence of regression coefficients and the set of least-squares tap weights $\{\hat{w}_0(n), \hat{w}_1(n), \ldots, \hat{w}_M(n)\}$. The purpose of this section is to formally derive this relationship.

Consider the conventional tapped-delay-line or transversal filter structure shown in Fig. 15.14, where the tap inputs $u(n), u(n-1), \ldots, u(n-m)$ are derived directly from the process $u(n)$ and the tap weights $\hat{w}_0(n), \hat{w}_1(n), \ldots, \hat{w}_m(n)$ are used to form respective scalar inner products. From Chapter 11, we recall that the least-squares solution for the $(m-1)$-by-1 tap vector $\hat{\mathbf{w}}_m(n)$, consisting of the elements $\hat{w}_0(n), \hat{w}_1(n), \ldots, \hat{w}_m(n)$, is defined by

$$\boldsymbol{\Phi}_{m+1}(n)\,\hat{\mathbf{w}}_m(n) = \mathbf{z}_{m+1}(n) \tag{15.181}$$

where $\boldsymbol{\Phi}_{m-1}(n)$ is the $(m+1)$-by-$(m+1)$ correlation matrix of the tap inputs, and $\mathbf{z}_{m+1}(n)$ is the $(m+1)$-by-1 cross-correlation vector between the tap inputs and desired response. We modify Eq. (15.181) in two ways: (1) we premultiply both sides of the equation by the $(m+1)$-by-$(m+1)$ lower triangular transformation matrix $\mathbf{L}_m(n)$, and (2) we interject the $(m+1)$-by-$(m+1)$ identity matrix $\mathbf{I} = \mathbf{L}_m^H(n)\mathbf{L}_m^{-H}(n)$ between the matrix $\boldsymbol{\Phi}_{m+1}(n)$ and the vector $\hat{\mathbf{w}}_m(n)$ on the left-hand side of the equation. The matrix $\mathbf{L}_m(n)$ is defined in terms of the tap weights of backward prediction-error filters of orders 0, 1, 2, ..., m, as in Eq. (15.73). The symbol $\mathbf{L}_m^{-H}(n)$ denotes the Hermitian transpose of the inverse matrix $\mathbf{L}_m^{-1}(n)$. We may thus write

$$\mathbf{L}_m(n)\boldsymbol{\Phi}_{m-1}(n)\mathbf{L}_m^H(n)\mathbf{L}_m^{-H}(n)\,\hat{\mathbf{w}}_m(n) = \mathbf{L}_m(n)\mathbf{z}_{m+1}(n) \tag{15.182}$$

Let the product $\mathbf{L}_m(n)\boldsymbol{\Phi}_{m+1}(n)\mathbf{L}_m^H(n)$ on the left-hand side of Eq. (15.182) be denoted by

$$\mathbf{D}_{m+1}(n) = \mathbf{L}_m(n)\boldsymbol{\Phi}_{m+1}(n)\mathbf{L}_m^H(n) \tag{15.183}$$

Using the formula for the augmented normal equations for backward linear prediction, we may show that the product $\mathbf{\Phi}_{m+1}(n)\mathbf{L}_m^H(n)$ consists of a lower triangular matrix whose diagonal elements equal the various sums of weighted backward *a posteriori* prediction-error squares, that is, $\mathcal{B}_0(n), \mathcal{B}_1(n), \ldots, \mathcal{B}_m(n)$ (see Problem 10). The matrix $\mathbf{L}_m(n)$ is, by definition, a lower triangular matrix whose diagonal elements are all equal to unity. Hence, the product of $\mathbf{L}_m(n)$ and $\mathbf{\Phi}_{m+1}(n)\mathbf{L}_m^H(n)$ is a lower triangular matrix. We also know that $\mathbf{L}_m^H(n)$ is an upper triangular matrix, and so is the matrix product $\mathbf{L}_m(n)\mathbf{\Phi}_{m+1}(n)$. Hence, the product of $\mathbf{L}_m(n)\mathbf{\Phi}_{m+1}(n)$ and $\mathbf{L}_m^H(n)$ is an upper triangular matrix. In other words, the matrix $\mathbf{D}_{m+1}(n)$ is both *upper* and *lower* triangular, which can only be satisfied if it is *diagonal*. Accordingly, we may write

$$\begin{aligned}\mathbf{D}_{m+1}(n) &= \mathbf{L}_m(n)\mathbf{\Phi}_{m+1}(n)\mathbf{L}_m^H(n) \\ &= \operatorname{diag}[\mathcal{B}_0(n), \mathcal{B}_1(n), \ldots, \mathcal{B}_m(n)]\end{aligned} \tag{15.184}$$

Equation (15.184) is further proof that the backward *a posteriori* prediction errors $b_0(n)$, $b_1(n), \ldots, b_m(n)$ produced by the various stages of the least-squares lattice predictor are uncorrelated (in a time-averaged sense) at all instants of time.

The product $\mathbf{L}_m(n)\mathbf{z}_{m+1}(n)$ on the right-hand side of Eq. (15.182) equals the cross-correlation vector between the backward prediction errors and the desired response. Let $\mathbf{t}_{m+1}(n)$ denote this cross-correlation vector, as shown by

$$\mathbf{t}_{m-1}(n) = \sum_{i=1}^{n} \lambda^{n-i}\mathbf{b}_{m+1}(i)d^*(i) \tag{15.185}$$

where $d(i)$ is the desired response. Substituting Eq. (15.72) in (15.185), we thus get

$$\begin{aligned}\mathbf{t}_{m+1}(n) &= \sum_{i=1}^{n} \lambda^{n-i}\mathbf{L}_m(n)\mathbf{u}_{m+1}(i)d^*(i) \\ &= \mathbf{L}_m(n)\sum_{i=1}^{n} \lambda^{n-i}\mathbf{u}_{m+1}(i)d^*(i) \\ &= \mathbf{L}_m(n)\mathbf{z}_{m+1}(n)\end{aligned} \tag{15.186}$$

which is the desired result. Accordingly, the combined use of Eqs. (15.183) and (15.186) in Eq. (15.182) yields the *transformed RLS solution*:

$$\mathbf{D}_{m+1}(n)\mathbf{L}_m^{-H}(n)\hat{\mathbf{w}}_m(n) = \mathbf{t}_{m+1}(n) \tag{15.187}$$

Thus far we have considered how the application of lower triangular matrix $\mathbf{L}_m(n)$ transforms the RLS solution for the tap-weight vector of the conventional transversal structure shown in Fig. 15.14. We next wish to consider the RLS solution represented by the regression coefficient vector $\boldsymbol{\kappa}_m(n)$, which is denoted by

$$\boldsymbol{\kappa}_m(n) = [\kappa_0(n), \kappa_1(n), \ldots, \kappa_m(n)]^T \tag{15.188}$$

The regression coefficient vector $\boldsymbol{\kappa}_m(n)$ may be viewed as the solution that minimizes the index of performance

$$\sum_{i=1}^{n} \lambda^{n-i} |d(i) - \mathbf{b}_{m+1}^{T}(i)\boldsymbol{\kappa}_m^*(n)|^2$$

where $\boldsymbol{\kappa}_m(n)$ is held constant for $1 \leq i \leq n$. The resulting solution to this RLS problem is defined by

$$\mathbf{D}_{m+1}(n)\boldsymbol{\kappa}_m(n) = \mathbf{t}_{m+1}(n) \tag{15.189}$$

where, as defined before, $\mathbf{D}_{m+1}(n)$ is the $(m + 1)$-by-$(m + 1)$ correlation matrix of the backward *a posteriori* prediction errors used as inputs to the regression coefficients, and $\mathbf{t}_{m+1}(n)$ is the $(m + 1)$-by-1 cross-correlation vector between these tap inputs and the desired response.

By comparing the transformed RLS solution of Eq. (15.187) and the RLS solution of Eq. (15.189), we immediately deduce the following simple relationship between the tap-weight vector $\hat{\mathbf{w}}_m(n)$ in the structure of Fig. 15.14 and the corresponding regression coefficient vector $\boldsymbol{\kappa}_m(n)$ computed by a recursive LSL filter:

$$\boldsymbol{\kappa}_m(n) = \mathbf{L}_m^{-H}(n)\hat{\mathbf{w}}_m(n) \tag{15.190}$$

or, equivalently,

$$\hat{\mathbf{w}}_m(n) = \mathbf{L}_m^{H}(n)\boldsymbol{\kappa}_m(n) \tag{15.191}$$

We thus see that the lower triangular transformation matrix $\mathbf{L}_m(n)$ represents the connecting link between the regression coefficient vector $\boldsymbol{\kappa}_m(n)$ in Fig. 15.7 and the least-squares tap-weight vector $\hat{\mathbf{w}}_m(n)$ in Fig. 15.14.

15.14 COMPUTER EXPERIMENT ON ADAPTIVE PREDICTION

In this second computer experiment of the chapter, we use a first-order autoregressive (AR) process $u(n)$ to study *adaptive prediction*, with two objectives in mind:

- To evaluate the performance of the recursive LSL algorithm using *a posteriori* estimation errors
- To compare the performance of this algorithm with that of the LMS algorithm for a similar application

The evaluations are to be made for the same two sets of conditions described in Section 9.6:

1. AR parameter : $a = -0.99$
   variance of AR process $u(n)$: $\sigma_u^2 = 0.93627$
2. AR parameter : $a = +0.99$
   variance of AR process $u(n)$: $\sigma_u^2 = 0.995$

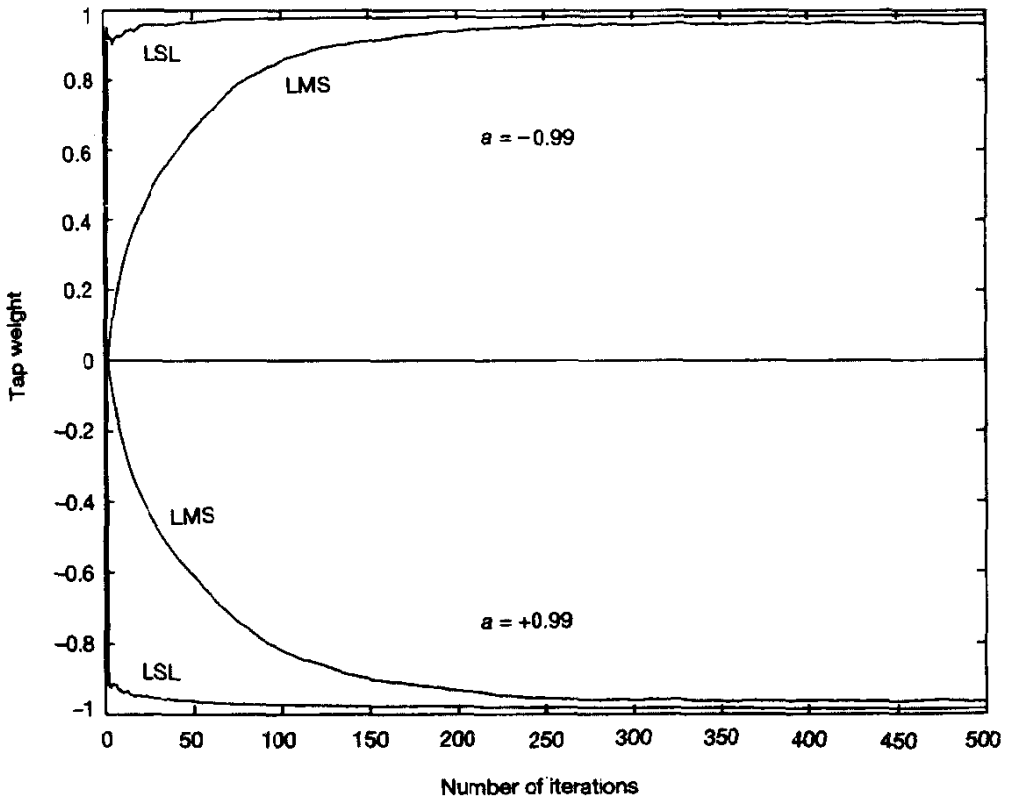


Figure 15.15 Comparison of the convergence behavior of the recursive LSL algorithm and LMS algorithm for autoregressive modeling of order 1.

Figure 15.15 shows the results of this experiment for the recursive LSL algorithm assuming that the exponential weighting factor $\lambda = 1$. This figure also includes the corresponding results of the experiment described in Section 9.6, using the LMS algorithm with step-size parameter $\mu = 0.05$. In both cases, the ensemble-averaged estimate of the AR parameter a is plotted versus the number of iterations n. The ensemble averaging was performed over 100 independent trials of the particular experiment.

The results of Fig. 15.15 show that:

- The recursive LSL algorithm converges to its steady-state condition much faster than the LMS algorithm
- After 500 iterations, the ensemble-averaged estimate of the AR parameter a produced by the recursive LSL algorithm is more accurate than that produced by the LMS algorithm.

15.15 OTHER VARIANTS OF LEAST-SQUARES LATTICE FILTERS

In Sections 15.11 and 15.12 we derived two recursive LSL filters (algorithms), one using *a posteriori* estimation errors and the other using *a priori* estimation errors with error feedback. The derivations were presented as special cases of the angle-normalized QRD–LSL algorithm, demonstrating the fundamental importance of this algorithm. There are, of course, many other recursive LSL algorithms that could also be derived from the angle-normalized QRD–LSL algorithm. Two other variants that immediately come to mind are the recursive LSL algorithm using *a posteriori* estimation errors with error feedback, and the recursive LSL algorithm using *a priori* estimation errors without error feedback. Hybrid combinations of the four recursive LSL algorithms mentioned here are obvious candidates that could be considered too.

The recursive LSL algorithm summarized in Table 15.4 is designed to propagate two reflection coefficients, one for solving the least-squares forward prediction problem and the other for solving the least-squares backward prediction problem. By employing a proper normalization of the forward and backward reflection coefficients, it is indeed possible to reformulate the recursive LSL algorithm of Table 15.4 so as to propagate a single reflection coefficient. The resulting algorithm is called the *normalized least-squares lattice algorithm* (Lee et al., 1981).

Yet another variant is the so-called *hybrid QR/lattice least-squares algorithm*, derived by Bellanger and Regalia (1991). This algorithm combines the good numerical properties of QR decomposition and the desirable order-recursive properties of least-squares lattice predictors. As shown by Sayed and Kailath (1994), the hybrid QR/lattice least-squares algorithm is essentially a rewriting of the angle-normalized QRD–LSL algorithm, whereby a certain collection of rows in the three arrays of the QRD–LSL algorithm is combined with the order updates:

$$\mathcal{F}_m(n) = \mathcal{F}_{m-1}(n) - \frac{|\Delta_{m-1}(n)|^2}{\mathcal{B}_{m-1}(n-1)} \tag{15.192}$$

$$\mathcal{B}_m(n) = \mathcal{B}_{m-1}(n-1) - \frac{|\Delta_{m-1}(n)|^2}{\mathcal{F}_{m-1}(n)} \tag{15.193}$$

These two order updates follow naturally from the augmented normal equations for least-squares estimation; see Problem 6.

The important point to note here is that by virtue of the following:

- The diversity of state-space models for characterizing the least-squares lattice filtering process
- The many variants of square-root Kalman filtering algorithms available for use
- The numerous ways in which *a priori* and *a posteriori* estimation errors can be hybridized

there is a large variety of recursive least-squares lattice algorithms that is essentially a matter of taste and patience, and that these algorithms may all be viewed as alternative rewritings of the angle-normalized QRD–LSL algorithm in exact arithmetic.

15.16 SUMMARY AND DISCUSSION

In this chapter we further consolidated the intimate relationship between Kalman filter theory and the family of adaptive linear filters that is rooted in least-squares estimation. In particular, we demonstrated how the square-root information filter, which is a variant of the Kalman filter, can be used to derive the QR-decomposition–based least-squares lattice (QRD–LSL) algorithm, which represents the most fundamental form of an order-recursive adaptive filter. We also demonstrated how other order-recursive adaptive filtering algorithms, such as the recursive LSL algorithm using *a posteriori* estimation errors and the recursive LSL algorithm using *a priori* estimation errors with error feedback, are in fact rewritings of the QRD–LSL algorithm.

The QRD–LSL algorithm combines highly desirable features of recursive least-squares estimation, QR-decomposition, and a lattice structure. Accordingly, it offers a unique set of operational and implementational advantages:

- The QRD–LSL algorithm has a *fast rate of convergence*, which is inherent in recursive least-squares estimation.
- The QRD–LSL algorithm can be implemented using a sequence of Givens rotations, which represent a form of QR decomposition. Moreover, the good numerical properties of the QR decomposition mean that the QRD–LSL algorithm is *numerically stable*.
- The QRD–LSL algorithm offers a *high level of computational efficiency*, in that its complexity is on the order of M, where M is the final prediction order (i.e., the number of available degrees of freedom).
- The lattice structure of the QRD–LSL algorithm is *modular* in nature, which means that the prediction order can be increased without having to recalculate all previous values. This property is particularly useful when there is no prior knowledge as to what the final value of the prediction order should be.
- Another implication of the modular structure of the QRD–LSL algorithm is that it lends itself to the use of *very large-scale integration (VLSI)* technology for its hardware implementation. Of course, the use of this sophisticated technology can only be justified if the application of interest calls for the use of VLSI chips in large numbers.
- The QRD–LSL algorithm includes an *integral set of desired variables and parameters* that are useful to have in signal-processing applications. Specifically, it offers the following three sets of useful by-products:
  - Angle-normalized forward and backward prediction errors

- Auxiliary parameters that can be used for the *indirect* computation of the forward and backward reflection coefficients and the regression coefficients (i.e., tap weights)

The recursive LSL algorithms enjoy many of the properties of the QRD–LSL algorithm, namely, fast convergence, modularity, and an integral set of useful parameters and variables for signal processing applications. However, the numerical properties of recursive LSL algorithms depend on whether error feedback is included or not in their composition; this issue is discussed in Chapter 17.

The order-recursive adaptive filters considered in this chapter have a computational advantage over the square-root adaptive filters considered in the previous chapter. In the former case, the computational cost increases linearly with the number of adjustable parameters, whereas in the latter case it increases as the square of the number of adjustable parameters. However, the use of order-recursive adaptive filters is limited to *temporal* signal processing applications that permit the exploitation of the time-shifting property of the input data. On the other hand, square-root adaptive filters can be used for both *temporal and spatial* signal-processing applications.

Traditionally, the derivations of these filters have been rather ad hoc, laborious, and certainly lacking a strong sense of unity. In contrast, the adoption of a Kalman filtering approach pioneered by Sayed and Kailath (1994), which we have followed in this book, not only overcomes these shortcomings of the traditional approach, but also offers the following advantages and possibilities:

- A *compact* representation of the adaptive filtering algorithms in the form of *arrays*, made up of prearrays, unitary rotations, and postarrays; the arrays propagate *all* the quantities needed to update the adaptive filtering algorithms.
- An opening to exploit the vast literature on Kalman filters, not so much to know how to develop new algorithms (as we already have enough of them) but rather to explore how to further improve the properties of adaptive filters.

In the context of the latter point, for example, we could mention that much has been written on the design of smoothing filters using Kalman filter theory (Gelb, 1974; Sorenson, 1985). It would therefore be enlightening to use this theory to derive latticelike realizations of smoothing filters based on recursive least-squares estimation. This would provide a framework for comparing notes with work done by Yuan and Stuller (1995), who have derived an algorithm for the design of order-recursive lattice smoothers that use past, present, and future data. Naturally, an adaptive smoother would outperform an adaptive filter, but would require the provision of an overall delay for physical realizeability in a real-time sense.

In closing this discussion, we should mention that the family of order-recursive adaptive filtering algorithms (including the QRD–LSL algorithm) is part of a larger class of adaptive filtering algorithms known collectively as *fast algorithms*. In the context of recursive least-squares estimation, an algorithm is said to be "fast" if its *computational*

complexity increases linearly with the number of adjustable parameters. A fast algorithm is therefore similar to the LMS algorithm in its computational requirement.

The class of order-recursive adaptive filtering algorithms, based on the use of reflection coefficients or their counterparts, is well suited for such adaptive signal-processing applications as predictive modeling, noise cancelation, and equalization. For fast RLS algorithms needed for other applications such as system identification and spectrum analysis where the emphasis is on the direct computation of adaptive transversal filter coefficients, we may look to the following alternatives:

- *Fast transversal filters (FTF) algorithm,* involving the combined use of four transversal filters for forward and backward predictions, gain vector computation, and joint-process estimation (Cioffi and Kailath, 1984). This is an elegant algorithm; unfortunately, it has a tendency to become numerically unstable when it is implemented in finite-precision arithmetic. To stabilize it, the algorithm has to be modified in a certain way, as explained in Chapter 17.
- *Fast QR-decomposition-based recursive least-squares estimation,* in which the triangularization of the data matrix and back-solving for the transversal filter's coefficient vector are *performed together* rather than separately (Liu, 1995). To account for this requirement, the Householder transformation used for the triangularization is modified in a special way. Unlike the FTF algorithm, the algorithm derived by Liu appears to be numerically stable.

PROBLEMS

1. Show that the parameter $\gamma_m(n)$ defined by

$$\gamma_m(n) = 1 - \mathbf{k}_m^H(n)\mathbf{u}_m(n)$$

equals the sum of weighted error squares resulting from use of the transversal filter in Fig. 15.3. The tap-weight vector of this filter equals the gain vector $\mathbf{k}_m(n)$, and the tap-input vector equals $\mathbf{u}_m(n)$. The filter is designed to produce the least-squares estimate of a desired response that equals the *first coordinate vector.*

2. The parameter $\Delta_{m-1}(n)$ is defined in Eq. (15.34). It is also defined in Eq. (15.51). Show that these two definitions are equivalent.

3. Let $\mathbf{\Phi}_m(n)$ denote the time-averaged correlation matrix of the tap-input vector $\mathbf{u}_m(n)$ at time n; likewise for $\mathbf{\Phi}_m(n-1)$. Show that the conversion factor $\gamma_m(n)$ is related to the determinants of these two matrices as follows:

$$\gamma_m(n) = \lambda \frac{\det[\mathbf{\Phi}_m(n-1)]}{\det[\mathbf{\Phi}_m(n)]}$$

where λ is the exponential weighting factor. *Hint*: Use the identity

$$\det(\mathbf{I}_1 + \mathbf{AB}) = \det(\mathbf{I}_2 + \mathbf{BA})$$

where $\mathbf{I}_1$ and $\mathbf{I}_2$ are identity matrices of appropriate dimensions, and $\mathbf{A}$ and $\mathbf{B}$ are matrices of compatible dimensions.

4. **(a)** Show that the inverse of the correlation matrix $\mathbf{\Phi}_{m-1}(n)$ may be expressed as follows:

$$\mathbf{\Phi}_{m+1}^{-1}(n) = \begin{bmatrix} 0 & \mathbf{0}_m^T \\ \mathbf{0}_m & \mathbf{\Phi}_m^{-1}(n-1) \end{bmatrix} + \frac{1}{\mathcal{F}_m(n)} \mathbf{a}_m(n)\mathbf{a}_m^H(n)$$

where $\mathbf{0}_m$ is the M-by-1 null vector, $\mathbf{0}_m^T$ is its transpose, $\mathcal{F}_m(n)$ is the minimum sum of weighted forward prediction-error squares, and $\mathbf{a}_m(n)$ is the tap-weight vector of forward prediction-error filter. Both $\mathbf{a}_m(n)$ and $\mathcal{F}_m(n)$ refer to prediction order m.

(b) Show that the inverse of $\mathbf{\Phi}_{m+1}(n)$ may also be expressed in the form

$$\mathbf{\Phi}_{m+1}^{-1}(n) = \begin{bmatrix} \mathbf{\Phi}_m^{-1}(n) & \mathbf{0}_m \\ \mathbf{0}_m^T & 0 \end{bmatrix} + \frac{1}{\mathcal{B}_m(n)} \mathbf{c}_m(n)\mathbf{c}_m^H(n)$$

where $\mathcal{B}_m(n)$ is the minimum sum of weighted backward *a posteriori* prediction-error squares, and $\mathbf{c}_m(n)$ is the tap-weight vector of the backward prediction-error filter. Both $\mathcal{F}_m(n)$ and $\mathbf{c}_m(n)$ refer to prediction order m.

5. Derive the following update formulas:

$$\gamma_{m+1}(n) = \gamma_m(n-1) - \frac{|f_m(n)|^2}{\mathcal{F}_m(n)}$$

$$\gamma_{m+1}(n) = \gamma_m(n) - \frac{|b_m(n)|^2}{\mathcal{B}_m(n)}$$

$$\gamma_{m+1}(n) = \lambda \frac{\mathcal{F}_m(n-1)}{\mathcal{F}_m(n)} \gamma_m(n-1)$$

$$\gamma_{m+1}(n) = \lambda \frac{\mathcal{B}_m(n-1)}{\mathcal{B}_m(n)} \gamma_m(n)$$

6. Using Eqs. (15.52) and (15.57), derive the following order-update recursions involving the sums of forward and backward prediction-error squares, respectively:

$$\mathcal{F}_m(n) = \mathcal{F}_{m-1}(n) - \frac{|\Delta_{m-1}(n)|^2}{\mathcal{B}_{m-1}(n-1)}$$

$$\mathcal{B}_m(n) = \mathcal{B}_{m-1}(n-1) - \frac{|\Delta_{m-1}(n)|^2}{\mathcal{F}_{m-1}(n)}$$

7. In this problem we show how the various quantities of the fast prediction equations relate to each other, and the parametric redundancy that they contain.[5]

(a) By combining parts (a) and (b) from Problem 4, show that

$$\begin{bmatrix} \mathbf{\Phi}_m^{-1}(n) & \mathbf{0}_m \\ \mathbf{0}_m^T & 0 \end{bmatrix} - \begin{bmatrix} 0 & \mathbf{0}_m^T \\ \mathbf{0}_m & \mathbf{\Phi}_m^{-1}(n-1) \end{bmatrix} = \frac{\mathbf{a}_m(n)\mathbf{a}_m^H(n)}{\mathcal{F}_m(n)} - \frac{\mathbf{c}_m(n)\mathbf{c}_m^H(n)}{\mathcal{B}_m(n)}$$

(b) From the recursive equations of Chap. 13, plus Eq. (15.26), show that the time update for $\mathbf{\Phi}_m^{-1}$ may be rewritten as

$$\mathbf{\Phi}_m^{-1}(n) = \lambda^{-1}\mathbf{\Phi}_m^{-1}(n-1) - \frac{\mathbf{k}_m(n)\mathbf{k}_m^H(n)}{\gamma_m(n)}$$

where $\mathbf{k}_m(n)$ is the gain vector and $\gamma_m(n)$ is the conversion factor.

[5]This problem was originally formulated by P. Regalia, private communication, 1995.

(c) By eliminating $\mathbf{\Phi}_m^{-1}(n-1)$ from the above two expressions, show that all the variables may be reconciled as

$$\begin{bmatrix} \mathbf{\Phi}_m^{-1}(n) & \mathbf{0}_m \\ \mathbf{0}_m^T & 0 \end{bmatrix} - \lambda \begin{bmatrix} 0 & \mathbf{0}_m^T \\ \mathbf{0}_m & \mathbf{\Phi}_m^{-1}(n) \end{bmatrix} = \frac{\mathbf{a}_m(n)\mathbf{a}_m^H(n)}{\mathcal{F}_m(n)} + \lambda \begin{bmatrix} 0 \\ \mathbf{k}_m(n) \end{bmatrix} \frac{[0,\ \mathbf{k}_m^H(n)]}{\gamma_m(n)} - \frac{\mathbf{c}_m(n)\mathbf{c}_m^H(n)}{\mathcal{B}_m(n)}$$

in which all the variables have a common time index n, and a common order index m. The left-hand side is called a *displacement residue* of $\mathbf{\Phi}_m^{-1}(n)$ and the right-hand side, being the sum and difference of three vector dyads, has rank not exceeding three. In matrix theory, $\mathbf{\Phi}_m^{-1}(n)$ is said to have *displacement rank* three as a result of the special structure of the data matrix exposed in Chapter 11. Note that this structure holds irrespective of the sequence $u(n)$ that builds the data matrix.

(d) Suppose we multiply the result of part (c) from the left by the row vector $[1, z/\sqrt{\lambda}, \ldots, (z/\sqrt{\lambda})^m]$, and from the right by the column vector $[1, w\sqrt{\lambda}, \ldots, (w/\sqrt{\lambda})^m]^H$, where z and w are two complex variables. Show that the result of part (c) is equivalent to the two-variable polynomial equation

$$(1 - zw^*)P(z,w^*) = A(z)A^*(w) + K(z)K^*(w) - C(z)C^*(w) \qquad \text{for all } z,\ w$$

provided that we make the correspondences

$$P(z, w^*) = [1, z/\sqrt{\lambda}, \ldots, (z/\sqrt{\lambda})^{m-1}]\mathbf{\Phi}_m^{-1}(n) \begin{bmatrix} 1 \\ w^*/\sqrt{\lambda} \\ \cdot \\ \cdot \\ \cdot \\ (w^*/\sqrt{\lambda})^{M-1} \end{bmatrix}$$

and

$$A(z) = [1, z/\sqrt{\lambda}, \ldots, (z/\sqrt{\lambda})^m] \frac{\mathbf{a}_m(n)}{\sqrt{\mathcal{F}_m(n)}}$$

$$K(z) = [1, z/\sqrt{\lambda}, \ldots, (z/\sqrt{\lambda})^m] \sqrt{\frac{\lambda}{\gamma_m(n)}} \begin{bmatrix} 0 \\ \mathbf{k}_m(n) \end{bmatrix}$$

$$C(z) = [1, z/\sqrt{\lambda}, \ldots, (z/\sqrt{\lambda})^m] \frac{\mathbf{c}_m(n)}{\sqrt{\mathcal{B}_m(n)}}$$

Similarly, $A^*(w) = [A(w)]^*$, and so on.

(e) Set $z = w = e^{j\omega}$ in the result of part (d) to show that

$$|A(e^{j\omega})|^2 + |K(e^{j\omega})|^2 = |C(e^{j\omega})|^2 \qquad \text{for all } \omega$$

that is, the three polynomials $A(z)$, $K(z)$, and $C(z)$ are power complementary along the unit circle $|z| = 1$.

(f) Show that, because $\mathbf{\Phi}_m^{-1}(n)$ is positive definite, the following system of inequalities necessarily results:

$$|A(z)|^2 + |K(z)|^2 - |C(z)|^2 = \begin{cases} <0, & |z| > 1 \\ =0, & |z| = 1 \\ >0, & |z| < 1 \end{cases}$$

Hint: Set $w^* = z^*$ in part (d) and note that, if $\mathbf{\Phi}_m^{-1}(n)$ is positive definite, the inequality

$$P(z,z^*) > 0 \qquad \text{for all } z$$

must result. Note that the center equality is equivalent to the result of part (e).

(g) Deduce from the first inequality of part (f) that $C(z)$ must be devoid of zeros in $|z| > 1$, and hence given $A(z)$ and $K(z)$, the polynomial $C(z)$ is uniquely determined from part (e) via spectral factorization. This shows that, once the forward prediction and gain quantities are known, the backward prediction variables contribute nothing further to the solution, and hence are theoretically redundant.

8. Justify the following relationships:

(a) Joint-process estimation errors:

$$|\epsilon_m(n)| = \sqrt{|e_m(n)| \cdot |\xi_m(n)|}$$

$$\text{ang}[\epsilon_m(n)] = \text{ang}[e_m(n)] + \text{ang}[\xi_m(n)]$$

(b) Backward prediction errors:

$$|\epsilon_{b,m}(n)| = \sqrt{|b_m(n)| \cdot |\beta_m(n)|}$$

$$\text{ang}[\epsilon_{b,m}(n)] = \text{ang}[b_m(n)] + \text{ang}[\beta_m(n)]$$

(c) Forward prediction errors:

$$|\epsilon_{f,m}(n)| = \sqrt{|f_m(n)| \cdot |\eta_m(n)|}$$

$$\text{ang}[\epsilon_{f,m}(n)] = \text{ang}[f_m(n)] + \text{ang}[\eta_m(n)]$$

9. Suppose that we have computed the cosine parameters $c_{f,m}(n)$ and $c_{b,m-1}(n-1)$ pertaining to the Givens transformations $\mathbf{\Theta}_{f,m}(n)$ and $\mathbf{\Theta}_{b,m-1}(n-1)$, respectively. Hence, show that the conversion factor $\gamma_{m+1}(n)$ may be updated in both time and order as follows:

$$\gamma_{m+1}^{1/2}(n) = c_{f,m}(n)c_{b,m-1}(n-1)\gamma_{m-1}^{1/2}(n-1)$$

10. The correlation matrix $\mathbf{\Phi}_{m+1}(n)$ is postmultiplied by the Hermitian transpose of the lower triangular matrix $\mathbf{L}_m(n)$, where $\mathbf{L}_m(n)$ is defined by Eq. (15.73). Show that the product $\mathbf{\Phi}_{m+1}(n)\mathbf{L}_m^H(n)$ consists of a lower triangular matrix whose diagonal elements equal the various sums of weighted backward prediction-error squares, $\mathcal{B}_0(n), \mathcal{B}_1(n), \ldots, \mathcal{B}_m(n)$. Hence, show that the product $\mathbf{L}_m(n)\mathbf{\Phi}_{m-1}(n)\mathbf{L}_m^H(n)$ is a diagonal matrix, as shown by

$$\mathbf{D}_{m+1}(n) = \text{diag}[\mathcal{B}_0(n), \mathcal{B}_1(n), \ldots, \mathcal{B}_m(n)]$$

11. Consider the case where the input samples $u(n), u(n-1), \ldots, u(n-M)$ have a *joint Gaussian distribution with zero mean*. Assume that, within a scaling factor, the ensemble-averaged correlation matrix $\mathbf{R}_{M+1}$ of the input signal is equal to its time-averaged correlation matrix $\mathbf{\Phi}_{M+1}(n)$ for time $n \geq M$. Show that the log-likelihood function for this input includes a term equal to the parameter $\gamma_M(n)$ associated with the recursive LSL algorithm. For this reason, the parameter $\gamma_M(n)$ is sometimes referred to as a *likelihood variable*.

12. Let $\hat{d}(n|\mathcal{U}_{n-m+1})$ denote the least-squares estimate of the desired response $d(n)$, given the inputs $u(n-m+1), \ldots, u(n)$ that span the space $\mathcal{U}_{n-m+1}$. Similarly, let $\hat{d}(n|\mathcal{U}_{n-m})$ denote the least-squares estimate of the desired response, given the inputs $u(n-m), u(n-m+1), \ldots, u(n)$ that span the space $\mathcal{U}_{n-m}$. In effect, the latter estimate exploits an additional piece of informa-

tion represented by the input $u(n - m)$. Show that this new information is represented by the corresponding backward prediction error $b_m(n)$. Also, show that the two estimates are related by the recursion

$$\hat{d}(n|\mathcal{U}_{n-m}) = \hat{d}(n|\mathcal{U}_{n-m+1}) + \kappa_m^*(n)b_m(n)$$

where $\kappa_m(n)$ denotes the pertinent regression coefficient in the joint-process estimator. Compare this result with that of Section 7.1 dealing with the concept of innovations.

13. Let $\Phi(n)$ denote the $(M + 1)$-by-$(M + 1)$ correlation matrix of the input data $u(n)$. Show that the change of variables to backward prediction errors brought about by using a lattice predictor achieves exactly the Cholesky decomposition of the matrix $\mathbf{P}(n) = \Phi^{-1}(n)$.

14. Expand the joint-process estimator of Fig. 15.7 so as to include (in modular form) the least-squares estimate of the desired response $d(n)$ for increasing prediction order m.

15. In Section 15.11 we discussed a modification of the *a priori* error LSL algorithm by using a form of error feedback. In this problem we consider the corresponding modified version of the *a posteriori* LSL algorithm. In particular, show that

$$\kappa_{f,m}(n) = \frac{\gamma_m(n-1)}{\gamma_{m-1}(n-1)}\left[\kappa_{f,m}(n-1) - \frac{1}{\lambda}\frac{b_{m-1}(n-1)f_{m-1}^*(n)}{\mathcal{B}_{m-1}(n-2)\gamma_{m-1}(n-1)}\right]$$

$$\kappa_{b,m} = \frac{\gamma_m(n)}{\gamma_{m-1}(n-1)}\left[\kappa_{b,m}(n-1) - \frac{1}{\lambda}\frac{f_{m-1}(n)b_{m-1}^*(n-1)}{\mathcal{F}_{m-1}(n-1)\gamma_{m-1}(n-1)}\right]$$

16. The accompanying table is a summary of the *normalized LSL algorithm*. The normalized parameters are defined by

$$\bar{f}_m(n) = \frac{f_m(n)}{\mathcal{F}_m^{1/2}(n)\gamma_m^{1/2}(n-1)}$$

$$\bar{b}_m(n) = \frac{b_m(n)}{\mathcal{B}_m^{1/2}(n)\gamma_m^{1/2}(n)}$$

$$\bar{\Delta}_m(n) = \frac{\Delta_m(n)}{\mathcal{F}_m^{1/2}(n)\mathcal{B}_m^{1/2}(n-1)}$$

Hence, derive the steps summarized in the table.

$$\bar{\Delta}_{m-1}(n) = \bar{\Delta}_{m-1}(n-1)[1 - |\bar{f}_{m-1}(n)|^2]^{1/2}[1 - |\bar{b}_{m-1}(n-1)|^2]^{1/2} + \bar{b}_{m-1}(n-1)\bar{f}_{m-1}^*(n)$$

$$\bar{b}_m(n) = \frac{\bar{b}_{m-1}(n-1) - \bar{\Delta}_{m-1}(n)\bar{f}_{m-1}(n)}{[1 - |\bar{\Delta}_{m-1}(n)|^2]^{1/2}[1 - |\bar{f}_{m-1}(n)|^2]^{1/2}}$$

$$\bar{f}_m(n) = \frac{\bar{f}_{m-1}(n) - \bar{\Delta}_{m-1}^*(n)\bar{b}_{m-1}(n-1)}{[1 - |\bar{\Delta}_{m-1}(n)|^2]^{1/2}[1 - |\bar{b}_{m-1}(n-1)|^2]^{1/2}}$$

CHAPTER

16

Tracking of Time-Varying Systems

In this second part of the book we have described two families of adaptive filtering algorithms, namely, the LMS family and the RLS family. Specifically, in Chapters 9 and 13 we considered the *average* behavior of the standard LMS and RLS algorithms operating in a stationary environment. For such an environment, the error-performance surface is fixed and the essential requirement is to seek, in a step-by-step fashion, the minimum point of that surface and thereby assure optimum or near optimum performance.

In this chapter we examine the operation of these two algorithms in a *nonstationary environment*, for which the optimum Wiener solution takes on a time-varying form. The net result is that the minimum point of the error-performance surface is no longer fixed. Consequently, the adaptive filtering algorithm now has the added task of *tracking* the minimum point of the error-performance surface. In other words, the algorithm is required to continuously track the statistical variations of the input, the occurrence of which is assumed to be "slow" enough for tracking to be feasible.

Tracking is a *steady-state phenomenon*. This is to be contrasted with convergence, which is a transient phenomenon. It follows therefore that for an adaptive filter to exercise its tracking capability, it must first pass from the transient mode to the steady-state mode of operation, and there must be provision for continuous adjustment of the free parameters of the filter. Moreover, we may state that, in general, the rate of convergence and tracking capability are two different properties of the algorithm. In particular, an adaptive filtering

algorithm with good convergence properties does *not* necessarily possess a fast tracking capability, and vice versa.

We begin the discussion by describing a particular time-varying model for system identification, which is subsequently used as the basis for evaluating the tracking performances of the standard LMS and RLS algorithms operating in a nonstationary environment.

16.1 MARKOV MODEL FOR SYSTEM IDENTIFICATION

Nonstationarity of an environment may arise in practice in one of two basic ways:

1. *The frame of reference provided by the desired response may be time varying.* Such a situation arises, for example, in system identification when an adaptive transversal filter is used to model a time-varying system. In this case, the correlation matrix of the tap inputs of the adaptive transversal filter remains fixed (as in a stationary environment), whereas the cross-correlation vector between the tap inputs and the desired response assumes a time-varying form.
2. *The stochastic process supplying the tap inputs of the adaptive filter is nonstationary.* This situation arises, for example, when an adaptive transversal filter is used to equalize a time-varying channel. In this second case, both the correlation matrix of the tap inputs in the adaptive transversal filter and the cross-correlation vector of the tap inputs and the desired response assume time-varying forms.

Thus, the tracking details of a time-varying system are not only dependent on the type of adaptive filter employed but are also *problem specific*.

In this chapter we will focus on a popular time-varying model for system identification, which is depicted in Fig. 16.1 (Widrow et al., 1976). The model is governed by three basic equations, as described next.

1. *First-order Markov process.* The unknown dynamic system is modeled as a transversal filter whose tap-weight vector $\mathbf{w}_o(n)$ (i.e., impulse response) undergoes a *first-order Markow process* written in vector form as follows [see Fig. 16.1(a)]:

$$\mathbf{w}_o(n+1) = a\mathbf{w}_o(n) + \boldsymbol{\omega}(n) \tag{16.1}$$

where a is a fixed parameter of the model, and $\boldsymbol{\omega}(n)$ is the *process noise vector* assumed to be of zero mean and correlation matrix $\mathbf{Q}$. In physical terms, the tap-weight vector $\mathbf{w}_o(n)$ may be viewed as originating from a source of random vector $\boldsymbol{\omega}(n)$, whose individual elements are applied to a bank of one-pole low-pass filters. Each such filter has a transfer function equal to $1/(1 - az^{-1})$, where z^{-1} is the unit-delay operator. It is assumed that the value of parameter a is very close to 1. The significance of this assumption is that the bandwidth of the low-pass filters is very much smaller than the incoming data rate. Equivalently, we may say that many iterations of the model in Fig. 16.1(a) are required to produce a significant change in the tap-weight vector $\mathbf{w}_o(n)$.

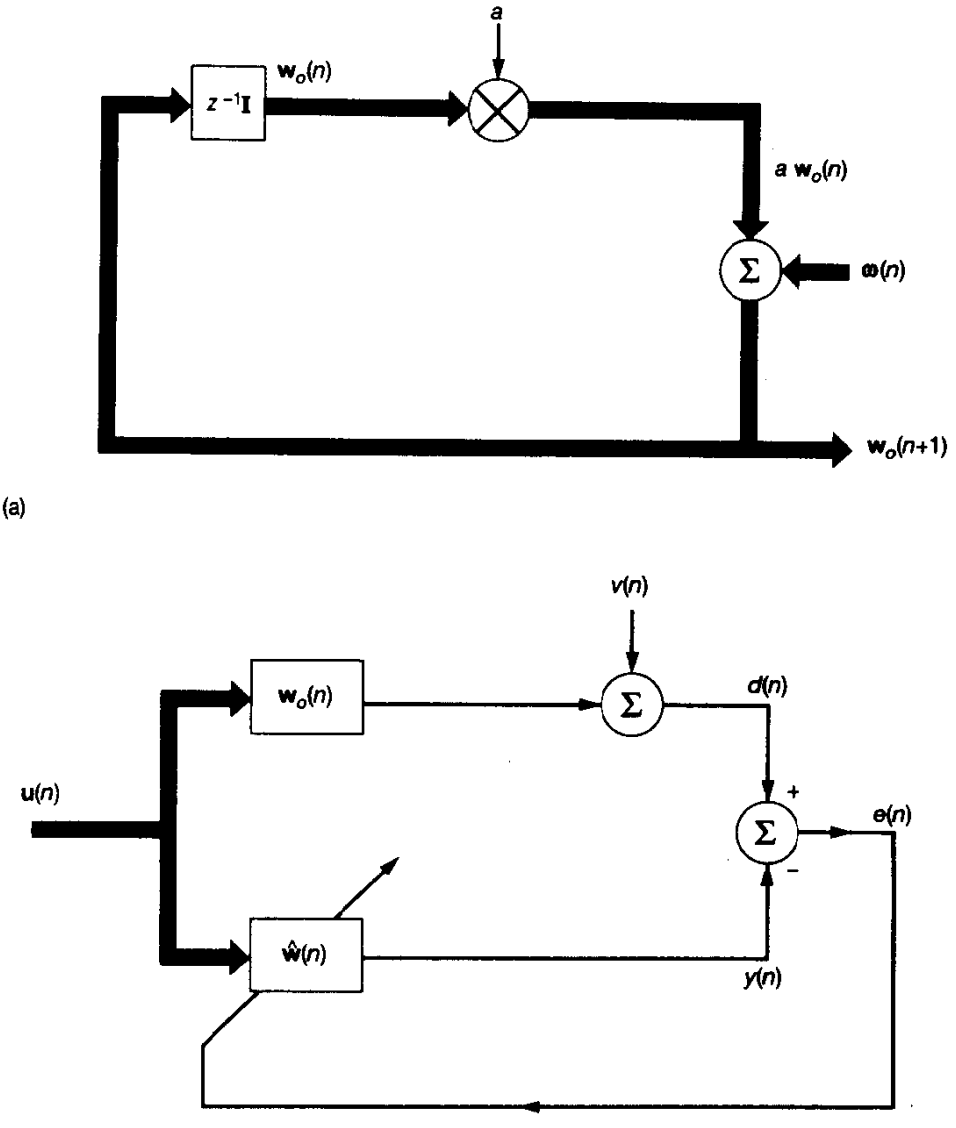


Figure 16.1 (a) Model of first-order Markov process. (b) System identification using adaptive filter.

2. *Desired response.* The desired response $d(n)$, providing a frame of reference for the adaptive filter, is defined by

$$d(n) = \mathbf{w}_o^H(n)\mathbf{u}(n) + v(n) \tag{16.2}$$

where $\mathbf{u}(n)$ is the input vector, which is common to both the unknown system and the adaptive filter; and $v(n)$ is the *measurement noise* assumed to be white

with zero mean and variance σ_v^2. Thus, even though both $\mathbf{u}(n)$ and $v(n)$ may be stationary random processes, the desired response $d(n)$ is a *nonstationary* random process by virtue of the fact that $\mathbf{w}_o(n)$ varies with time. Herein lies the challenge posed to the adaptive filter.

3. *Error signal.* The error signal $e(n)$, involved in the adaptive process, is defined by [see Fig. 16.1(b)]

$$\begin{aligned} e(n) &= d(n) - y(n) \\ &= \mathbf{w}_o^H(n)\mathbf{u}(n) + v(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n) \end{aligned} \tag{16.3}$$

where $\hat{\mathbf{w}}(n)$ is the tap-weight vector of the adaptive filter assumed to have a transversal structure. It is also assumed that the adaptive filter has the same number of taps as the unknown system represented by $\mathbf{w}_o$.

The tap-weight vector $\mathbf{w}_o(n)$ of the unknown system represents the "target" to be tracked by the adaptive filter. Whenever the tap-weight vector $\hat{\mathbf{w}}(n)$ of the adaptive filter equals $\mathbf{w}_o(n)$, the minimum mean-square error produced by the adaptive filter equals σ_v^2.

According to Eq. (16.2), the desired response $d(n)$ applied to the adaptive filter equals the overall output of the unknown system. Since this system is time-varying, the desired response is correspondingly nonstationary. Accordingly, with the correlation matrix of the tap inputs having the fixed value $\mathbf{R}$, we find that the adaptive filter has a quadratic bowl-shaped error-performance surface whose position is in a permanent state of motion.

Assumptions

Typically, the variations represented by the process noise vector $\boldsymbol{\omega}(n)$ in the Markov model of Fig. 16.1(a) are slow (i.e., bounded). This makes it possible for the adaptive transversal filter using the LMS or RLS algorithm to *track* the statistical variations in the dynamic behavior of the unknown system in Fig. 16.1(b).

To proceed with a tracking analysis of the LMS and RLS algorithms in the environment described in Fig. 16.1, the following conditions are assumed throughout the chapter (Macchi, 1995):

1. The process noise vector $\boldsymbol{\omega}(n)$ is independent of both the input vector $\mathbf{u}(n)$ and measurement noise $\mathbf{v}(n)$.
2. The input vector $\mathbf{u}(n)$ and measurement noise $v(n)$ are independent of each other.
3. The measurement noise $v(n)$ is white with zero mean and variance $\sigma_v^2 < \infty$.

Assumptions 2 and 3 have been made previously (see Chapters 9 and 13). Assumption 1, pertaining to the unknown system itself, is new to this chapter. Thus, there are three sources of randomness corresponding to physical phenomena to be considered: the input vector $\mathbf{u}(n)$, the measurement noise $v(n)$, and the process noise vector $\boldsymbol{\omega}(n)$. In practice,

they do *not* usually relate to each other in the unrealistic manner described under assumptions 1 to 3. Nevertheless, these assumptions are commonly made in the literature on adaptive filters for the sake of mathematical tractability. Collectively, they are referred to as the *independence assumption.*

16.2 DEGREE OF NONSTATIONARITY

In order to provide a clear definition of the rather ambiguous notion of what is meant by "slow" or "fast" statistical variations of the model, we may introduce the *degree of nonstationarity* (Macchi, 1986, 1995). In the context of the Markov model described in Fig. 16.1, the degree of nonstationarity, denoted by α, is formally defined as the square root of the average noise power due to the process noise vector $\boldsymbol{\omega}(n)$ to the average noise power due to the measurement noise $v(n)$, both of which refer to the output end of the time-varying system. That is,

$$\alpha = \left(\frac{E[|\boldsymbol{\omega}^H(n)\mathbf{u}(n)|^2]}{E[|v(n)|^2]} \right)^{1/2} \tag{16.4}$$

The degree of nonstationarity is therefore a characteristic of the time varying system alone; as such, it has nothing to do with the adaptive filter.

The numerator in Eq. (16.4) may be rewritten as follows [in light of the assumption that $\boldsymbol{\omega}(n)$ is independent of $\mathbf{u}(n)$]:

$$\begin{aligned} E[|\boldsymbol{\omega}^H(n)\mathbf{u}(n)|^2] &= E[\boldsymbol{\omega}^H(n)\mathbf{u}(n)\mathbf{u}^H(n)\boldsymbol{\omega}(n)] \\ &= \text{tr}\{E[\boldsymbol{\omega}^H(n)\mathbf{u}(n)\mathbf{u}^H(n)\boldsymbol{\omega}(n)]\} \\ &= E\{\text{tr}[\boldsymbol{\omega}^H(n)\mathbf{u}(n)\mathbf{u}^H(n)\boldsymbol{\omega}(n)]\} \\ &= E\{\text{tr}[\boldsymbol{\omega}(n)\boldsymbol{\omega}^H(n)\mathbf{u}(n)\mathbf{u}^H(n)]\} \\ &= \text{tr}\{E[\boldsymbol{\omega}(n)\boldsymbol{\omega}^H(n)\mathbf{u}(n)\mathbf{u}^H(n)]\} \\ &= \text{tr}\{E[\boldsymbol{\omega}(n)\boldsymbol{\omega}^H(n)]E[\mathbf{u}(n)\mathbf{u}^H(n)]\} \\ &= \text{tr}\,[\mathbf{QR}] \end{aligned} \tag{16.5}$$

where tr[•] denotes the *trace* of the matrix enclosed inside the square brackets, $\mathbf{R}$ is the correlation matrix of the input vector $\mathbf{u}(n)$, and $\mathbf{Q}$ is the correlation matrix of the process noise vector $\boldsymbol{\omega}(n)$. The denominator in Eq. (16.4) is simply the variance σ_v^2 of the zero-mean measurement noise $v(n)$. Accordingly, we may reformulate the degree of nonstationary for the Markov model of Fig. 16.1 simply as

$$\alpha = \frac{1}{\sigma_v}(\text{tr}[\mathbf{RQ}])^{1/2} = \frac{1}{\sigma_v}(\text{tr}[\mathbf{QR}])^{1/2} \tag{16.6}$$

The degree of nonstationarity α bears a useful relation to the misadjustment $\mathcal{M}$ of the adaptive filter, as explained here. First, we note that the minimum mean-squared error $J_{\min}$

that the adaptive filter in Fig. 16.1(b) can ever attain is equal to the variance σ_v^2 of the measurement noise $v(n)$. Next, we note that the best that the adaptive filter can ever do in tracking the time-varying system of Fig. 16.1(a) is to produce a *weight-error vector* $\boldsymbol{\epsilon}(n)$ that is equal to the process noise vector $\boldsymbol{\omega}(n)$. Then, following the terminology introduced in Section 9.4, we may set the weight-error correlation matrix $\mathbf{K}(n)$ equal to the correlation matrix $\mathbf{Q}$ of $\boldsymbol{\omega}(n)$. Hence, recalling from that chapter that the excess mean-squared error $J_{ex}(n)$ is equal to $\text{tr}[\mathbf{RK}(n)]$, we may state that the excess mean-squared error attained by the adaptive filter can never be less than $\text{tr}[\mathbf{RQ}]$. Thus, in light of the definition of misadjustment as the ratio of the excess mean-squared error to the minimum mean-squared error [see Eq. (9.67)], we may write

$$\mathcal{M} = \frac{J_{ex}}{J_{min}} \geq \frac{\text{tr}[\mathbf{RQ}]}{\sigma_v^2} = \alpha^2 \tag{16.7}$$

In other words, the square root of the misadjustment $\mathcal{M}$ places an upper bound on the degree of nonstationarity.

We will have more to say on misadjustment as a measure of tracking performance in the next section. For now, we may use Eq. (16.7) to make two noteworthy remarks:

1. For slow statistical variations, α is small. This, in turn, means that it should be possible to build an adaptive filter that can track the time-varying system.
2. When the statistical variations of the environment are too fast, α may be greater than 1. In such a case, the misadjustment produced by the adaptive filter exceeds 100 percent, which means that there is no advantage to be gained in building an adaptive filter to solve the tracking problem.

16.3 CRITERIA FOR TRACKING ASSESSMENT

With the unknown dynamic system in Fig. 16.1 modeled as a transversal filter whose tap-weight vector is denoted by $\mathbf{w}_o(n)$, and with the tap-weight vector of the adaptive transversal filter denoted by $\hat{\mathbf{w}}(n)$, we may define the *weight-error vector* as

$$\boldsymbol{\epsilon}(n) = \hat{\mathbf{w}}(n) - \mathbf{w}_o(n) \tag{16.8}$$

On the basis of $\boldsymbol{\epsilon}(n)$, we may go on to define two figures of merit for assessing the tracking capability of an adaptive filter.

1. *Mean-square Deviation*

A commonly used figure of merit for tracking assessment is the *mean-square deviation* (MSD) between the actual weight vector $\mathbf{w}_o(n)$ of the unknown dynamic system and the adjusted weight vector $\hat{\mathbf{w}}(n)$ of the adaptive filter, defined by (Benveniste and Ruget, 1982; Macchi, 1986; Benveniste, 1987):

$$\begin{aligned} \mathcal{D}(n) &= E[\|\hat{\mathbf{w}}(n) - \mathbf{w}_o(n)\|^2] \\ &= E[\|\boldsymbol{\epsilon}(n)\|^2] \end{aligned} \tag{16.9}$$

where the number of iterations n is assumed to be large enough for the adaptive filter's transient mode of operation to have finished. Equation (16.9) may be reformulated in the equivalent form (following steps similar to those presented in Eq. (16.5)]:

$$\mathcal{D}(n) = \text{tr}[\mathbf{K}(n)] \tag{16.10}$$

where $\mathbf{K}(n)$ is the correlation matrix of the error vector $\boldsymbol{\epsilon}(n)$:

$$\mathbf{K}(n) = E[\boldsymbol{\epsilon}(n)\boldsymbol{\epsilon}^H(n)] \tag{16.11}$$

Clearly, the mean-square deviation $\mathcal{D}(n)$ should be small for a good tracking performance.

The weight-error vector $\boldsymbol{\epsilon}(n)$ may be expressed as the sum of two components, as shown by

$$\boldsymbol{\epsilon}(n) = \boldsymbol{\epsilon}_1(n) + \boldsymbol{\epsilon}_2(n) \tag{16.12}$$

where $\boldsymbol{\epsilon}_1(n)$ is *weight vector noise* defined by

$$\boldsymbol{\epsilon}_1(n) = \hat{\mathbf{w}}(n) - E[\hat{\mathbf{w}}(n)] \tag{16.13}$$

where $E[\hat{\mathbf{w}}(n)]$ is the ensemble-averaged value of the tap-weight vector; and $\boldsymbol{\epsilon}_2(n)$ is the *weight vector lag* defined by

$$\boldsymbol{\epsilon}_2(n) = E[\hat{\mathbf{w}}(n)] - \mathbf{w}_o(n) \tag{16.14}$$

Invoking the assumptions made in Section 16.1, we have

$$E[\boldsymbol{\epsilon}_1^H(n)\,\boldsymbol{\epsilon}_2(n)] = E[\boldsymbol{\epsilon}_2^H(n)\boldsymbol{\epsilon}_1(n)] = 0 \tag{16.15}$$

Accordingly, we may express the mean-square deviation $\mathcal{D}(n)$ as the sum of two components, as shown by

$$\mathcal{D}(n) = \mathcal{D}_1(n) + \mathcal{D}_2(n) \tag{16.16}$$

The first term $\mathcal{D}_1(n)$ is called the *estimation variance*, which is due to the weight vector noise $\boldsymbol{\epsilon}_1(n)$; it is defined by

$$\mathcal{D}_1(n) = E[\|\boldsymbol{\epsilon}_1(n)\|^2] \tag{16.17}$$

The estimation variance $\mathcal{D}_1(n)$ is always present, even in the stationary case. The second term $\mathcal{D}_2(n)$ is called the *lag variance*, which is due to the weight vector lag; it is defined by

$$\mathcal{D}_2(n) = E[\|\boldsymbol{\epsilon}_2(n)\|^2] \tag{16.18}$$

The presence of $\mathcal{D}_2(n)$ is testimony to the nonstationary nature of the environment. The decomposition of the mean-square deviation $\mathcal{D}(n)$ into estimation variance $\mathcal{D}_1(n)$ and lag variance $\mathcal{D}_2(n)$, as described in Eq. (16.16), is called the *decoupling property* (Macchi, 1986a,b).

2. *Misadjustment*

Another commonly used figure of merit for assessing the tracking capability of an adaptive filter is the *misadjustment*, which is defined by (Widrow et al., 1976)

$$\mathcal{M}(n) = \frac{J_{ex}(n)}{\sigma_v^2} \tag{16.19}$$

where $J_{ex}(n)$ is the excess (residual) mean-squared error of the adaptive filter measured with respect to the variance σ_v^2 of the white noise component $v(n)$ at the output of the transversal model in Fig. 16.1(b). Here again it is assumed that the number of iterations n is large enough for the transient period to have ended. In light of Eq. (9.65), justified under the independence assumption, we may express the excess mean-squared error in terms of $\mathbf{K}(n)$, the correlation matrix of the weight error vector $\boldsymbol{\epsilon}(n)$, as follows:

$$J_{ex}(n) = \text{tr}[\mathbf{RK}(n)] \tag{16.20}$$

where $\mathbf{R}$ is the correlation matrix of the input vector $\mathbf{u}(n)$. Accordingly, we may reformulate Eq. (16.19) in the equivalent form

$$\mathcal{M}(n) = \frac{\text{tr}[\mathbf{RK}(n)]}{\sigma_v^2} \tag{16.21}$$

For a good tracking performance, it is apparent that the misadjustment $\mathcal{M}(n)$ should be small compared to unity.

As with the mean-square deviation, the excess mean-squared error $J_{ex}(n)$ may be expressed as the sum of two components, $J_{ex1}(n)$ and $J_{ex2}(n)$, by virtue of the assumptions made in Section 16.1. The first component $J_{ex1}(n)$ is due to the weight vector noise $\boldsymbol{\epsilon}_1(n)$; it is called the *estimation noise*. The second component $J_{ex2}(n)$ is due to the weight vector lag $\boldsymbol{\epsilon}_2(n)$; it is called the *lag noise*. The presence of the latter term is attributed directly to the nonstationary nature of the environment. Correspondingly, we may express the misadjustment $\mathcal{M}(n)$ as

$$\mathcal{M}(n) = \mathcal{M}_1(n) + \mathcal{M}_2(n) \tag{16.22}$$

where $\mathcal{M}_1(n) = J_{ex1}(n)/\sigma_v^2$ and $\mathcal{M}_2(n) = J_{ex2}(n)/\sigma_v^2$. The first term $\mathcal{M}_1(n)$ is called the *noise misadjustment* and the second term $\mathcal{M}_2(n)$ is called the *lag misadjustment*. Thus, the decoupling property is true for the misadjustment too, in that the estimation noise and lag noise are decoupled in power.

In general, both figures of merit, $\mathcal{D}(n)$ and $\mathcal{M}(n)$, depend on the number if iterations (time) n. Moreover, they highlight different aspects of the tracking problem in a complementary way, as subsequent analysis will reveal.

16.4 TRACKING PERFORMANCE OF THE LMS ALGORITHM

To proceed with a study of the tracking problem, consider the system model of Fig. 16.1 in which the adaptive (transversal) filter is implemented using the LMS algorithm. According to this algorithm, the tap-weight vector of the adaptive filter is updated as follows:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \mu\mathbf{u}(n)e^*(n) \tag{16.23}$$

where μ is the step-size parameter. Substituting Eq. (16.3) for the error signal $e(n)$ in Eq. (16.23), we may reformulate the LMS algorithm in the expanded form:

$$\hat{\mathbf{w}}(n+1) = [\mathbf{I} - \mu\mathbf{u}(n)\mathbf{u}^H(n)]\hat{\mathbf{w}}(n) + \mathbf{u}(n)\mathbf{u}^H(n)\mathbf{w}_o(n) + \mu\mathbf{u}(n)v^*(n) \tag{16.24}$$

Next, using the definition of Eq. (16.7) for the weight-error vector $\boldsymbol{\epsilon}(n)$ and the description of a first-order Markov model given in Eq. (16.1), we may write (after combining terms)

$$\begin{aligned}\boldsymbol{\epsilon}(n+1) &= \hat{\mathbf{w}}(n+1) - \mathbf{w}_o(n+1) \\ &= [\mathbf{I} - \mu\mathbf{u}(n)\mathbf{u}^H(n)]\boldsymbol{\epsilon}(n) + (1-a)\,\mathbf{w}_o(n) + \mu\mathbf{u}(n)v^*(n) - \boldsymbol{\omega}(n)\end{aligned} \tag{16.25}$$

where $\mathbf{I}$ is the identity matrix. The linear stochastic difference equation (16.25) provides a complete description of the LMS algorithm embedded in the system model of Fig. 16.1. A general tracking theory of the Markov model based on Eq. (16.25) is yet to be developed. The approach usually taken is to assume that the model parameter a is very close to 1, so that we may ignore the term $(1 - a)\mathbf{w}_o$. In so doing, we are in fact developing a tracking theory for the *random walk model* (Macchi and Turki, 1992; Macchi, 1995). Thus, with $a = 1$, Eq. (16.25) reduces to

$$\boldsymbol{\epsilon}(n+1) = [\mathbf{I} - \mu\mathbf{u}(n)\mathbf{u}^H(n)]\boldsymbol{\epsilon}(n) + \mu\mathbf{u}(n)v^*(n) - \boldsymbol{\omega}(n) \tag{16.26}$$

Typically, the step-size parameter μ is assigned a small value in order to realize a good tracking performance.[1] Then under this condition, we may solve Eq. (16.26) for the weight-error vector $\boldsymbol{\epsilon}(n)$ by invoking the *direct-averaging method* (Kushner, 1984), which was discussed previously in Chapter 9. Specifically, we may state that for a small μ, the solution $\boldsymbol{\epsilon}(n)$ to the linear stochastic difference equation (16.26) is "close" to the solution of another linear stochastic difference equation that is obtained by replacing the system matrix $[\mathbf{I} - \mu\mathbf{u}(n)\mathbf{u}^H(n)]$ with its ensemble average $(\mathbf{I} - \mu\mathbf{R})$, where $\mathbf{R}$ is the correlation matrix of the input vector $\mathbf{u}(n)$. We may thus write the new stochastic difference equation as follows:

$$\boldsymbol{\epsilon}(n+1) = (\mathbf{I} - \mu\mathbf{R})\boldsymbol{\epsilon}(n) + \mu\mathbf{u}(n)v^*(n) - \boldsymbol{\omega}(n) \tag{16.27}$$

By right, we should have used a different notation for the weight-error vector in Eq. (16.27) to distinguish it from that used in Eq. (16.26). We have opted not to do so merely for convenience of presentation. To evaluate the correlation matrix of $\boldsymbol{\epsilon}(n+1)$ given in Eq. (16.27), we invoke the independence assumption described earlier. Under this assump-

[1] In practical situations, we often find that μ is assigned a large value that may lie outside the scope of stochastic approximation theory; this is usually done in order to obtain a fast rate of convergence. To deal with this dilemma, Perrier et al. (1994) use the perturbation expansion method, first proposed by Solo (1992), to investigate the steady-state performance of the LMS algorithm. In particular, it is shown how to explicitly integrate the correlation coefficients of the input signal up to the second order in μ.

tion, the correlation matrix of the weight-error vector $\boldsymbol{\epsilon}(n)$ is readily determined from Eq. (16.27) to be

$$\begin{aligned}\mathbf{K}(n+1) &= E[\boldsymbol{\epsilon}(n+1)\boldsymbol{\epsilon}^H(n+1)] \\ &= (\mathbf{I} - \mu\mathbf{R})\mathbf{K}(n)(\mathbf{I} - \mu\mathbf{R}) + \mu\sigma_v^2\mathbf{R} + \mathbf{Q}\end{aligned} \tag{16.28}$$

For a *steady-state solution* of the difference equation (16.28), for which n is large, we may legitimately set $\mathbf{K}(n+1) = \mathbf{K}(n)$. Furthermore, assuming that the step-size parameter μ is small enough to justify ignoring the term $\mu^2\mathbf{R}\mathbf{K}(n)\mathbf{R}$ in comparison with the identity matrix $\mathbf{I}$, we may approximate Eq. (16.28) as follows (after rearranging terms):

$$\mathbf{R}\mathbf{K}(n) + \mathbf{K}(n)\mathbf{R} \simeq \mu\sigma_v^2\mathbf{R} + \frac{1}{\mu}\mathbf{Q} \tag{16.29}$$

This is the equation for assessing the tracking capability of the LMS algorithm applied to the system model of Fig. 16.1, under the assumption that a is close to unity.

Mean-square Deviation of the LMS Algorithm

To proceed with the evaluation of the mean-square deviation of the LMS algorithm, we premultiply both sides of Eq. (16.29) by the inverse matrix $\mathbf{R}^{-1}$, and then take the trace of the resultant matrices. We thus obtain

$$\text{tr}[\mathbf{K}(n)] + \text{tr}[\mathbf{R}^{-1}\mathbf{K}(n)\mathbf{R}] \simeq \mu M\sigma_v^2 + \frac{1}{\mu}\text{tr}[\mathbf{R}^{-1}\mathbf{Q}] \tag{16.30}$$

Next, we recognize that since $\mathbf{K}(n)$ and $\mathbf{R}$ have the same dimensions, then

$$\begin{aligned}\text{tr}[\mathbf{R}^{-1}\mathbf{K}(n)\mathbf{R}] &= \text{tr}[\mathbf{K}(n)\mathbf{R}\mathbf{R}^{-1}] \\ &= \text{tr}[\mathbf{K}(n)]\end{aligned}$$

Accordingly, we may use Eqs. (16.10) and (16.30) to evaluate the mean-square deviation of the LMS algorithm, as shown by

$$\mathcal{D}(n) \simeq \frac{\mu}{2}M\sigma_v^2 + \frac{1}{2\mu}\text{tr}[\mathbf{R}^{-1}\mathbf{Q}], \qquad n \text{ large} \tag{16.31}$$

The first term, $\mu M\sigma_v^2/2$, is the estimation variance due to the measurement noise $v(n)$; it varies *linearly* with the step-size parameter μ. The second term, $\text{tr}[\mathbf{R}^{-1}\mathbf{Q}]/2\mu$, is the lag variance due to the process noise vector $\boldsymbol{\omega}(n)$; it varies *inversely* with the step-size parameter μ, thereby permitting a faster tracking speed.

Let μ_{opt} denote the optimum value of the step-size parameter for which the mean-square deviation attains its minimum value $\mathcal{D}_{\min}$. This optimum condition is realized when the estimation variance and lag variance contribute equally to the mean-square deviation. From Eq. (16.31) we thus readily find that

$$\mu_{\text{opt}} \simeq \frac{1}{\sigma_v\sqrt{M}}(\text{tr}\mathbf{R}^{-1}\mathbf{Q}])^{1/2} \tag{16.32}$$

and

$$D_{\min} \simeq \sigma_v \sqrt{M}\,(\mathrm{tr}[\mathbf{R}^{-1}\mathbf{Q}])^{1/2} \tag{16.33}$$

Misadjustment of the LMS Algorithm

To evaluate the misadjustment of the LMS algorithm for the system identification scenario described in Fig. 16.1, we take the trace of the matrix quantities on both sides of Eq. (16.29), and so write

$$\mathrm{tr}[\mathbf{RK}(n)] + \mathrm{tr}[\mathbf{K}(n)\mathbf{R}] \simeq \mu\sigma_v^2\,\mathrm{tr}[\mathbf{R}] + \frac{1}{\mu}\mathrm{tr}[\mathbf{Q}] \tag{16.34}$$

Next, recognizing that the traces of $\mathbf{RK}(n)$ and $\mathbf{K}(n)\mathbf{R}$ are equal, we may apply the formula of Eq. (16.21) to the problem at hand, and so express the misadjustment of the LMS algorithm as

$$\mathcal{M}(n) \simeq \frac{\mu}{2}\mathrm{tr}[\mathbf{R}] + \frac{1}{2\mu\sigma_v^2}\mathrm{tr}[\mathbf{Q}], \qquad n \text{ large} \tag{16.35}$$

The first term, $\mu\mathrm{tr}[\mathbf{R}]/2$, is the noise misadjustment caused by the measurement noise $v(n)$; this term is of the same form as in a stationary environment, which is not surprising. The second term, $\mathrm{tr}[\mathbf{Q}]/2\mu\sigma_v^2$, is the lag misadjustment caused by the process noise vector $\boldsymbol{\omega}(n)$, which is representative of nonstationarity in the environment.

The noise misadjustment varies *linearly* with the step-size parameter μ, whereas the lag misadjustment varies *inversely* with μ. The optimum value of the step-size parameter, μ_{opt}, for which the misadjustment attains its minimum value, $\mathcal{M}_{\min}$, occurs when the estimation noise and lag noise are equal. We thus readily find from Eq. (16.35) that

$$\mu_{\text{opt}} \simeq \frac{1}{\sigma_v}\left(\frac{\mathrm{tr}[\mathbf{Q}]}{\mathrm{tr}[\mathbf{R}]}\right)^{1/2} \tag{16.36}$$

and

$$\mathcal{M}_{\min} \simeq \frac{1}{\sigma_v}(\mathrm{tr}[\mathbf{R}]\mathrm{tr}[\mathbf{Q}])^{1/2} \tag{16.37}$$

Equations (16.33) and (16.37) indicate that, in general, optimization of the two figures of merit, the mean-square deviation and misadjustment, leads to different values for the optimum setting of the step-size parameter μ. This should not be surprising since these two figures of merit emphasize different aspects of the tracking problem. However the choice is made, it is presumed that the optimum μ satisfies the condition for convergence of the LMS algorithm in the mean square.

16.5 TRACKING PERFORMANCE OF THE RLS ALGORITHM

Consider next the RLS algorithm used to implement the adaptive filter in the system model of Fig. 16.1. From Chapter 13 we recall that the corresponding update equation for the weight vector $\hat{\mathbf{w}}(n)$ of the adaptive transversal filter may be written in the form

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \boldsymbol{\Phi}^{-1}(n)\mathbf{u}(n)\xi^*(n) \tag{16.38}$$

where $\mathbf{\Phi}(n)$ is the correlation matrix of the input vector $\mathbf{u}(n)$:

$$\mathbf{\Phi}(n) = \sum_{i=1}^{n} \lambda^{n-i}\mathbf{u}(i)\mathbf{u}^H(i) \tag{16.39}$$

and $\xi(n)$ is the *a priori* estimation error:

$$\xi(n) = d(n) - \hat{\mathbf{w}}^H(n-1)\mathbf{u}(n) \tag{16.40}$$

To accommodate the slight change in the notation for the weight vector in Eq. (16.38) compared to that in Eq. (16.23), we modify the first-order Markov model of Eq. (16.1) and the desired response $d(n)$ of Eq. (16.2) as follows, respectively:

$$\mathbf{w}_o(n) = a\mathbf{w}_o(n-1) + \boldsymbol{\omega}(n) \tag{16.41}$$

and

$$d(n) = \mathbf{w}_o^H(n-1)\mathbf{u}(n) + v(n) \tag{16.42}$$

Accordingly, using Eqs. (16.38), (16.41), and (16.42), we may express the update equation for the weight-error vector in the RLS algorithm as shown by

$$\boldsymbol{\epsilon}(n) = [\mathbf{I} - \mathbf{\Phi}^{-1}(n)\mathbf{u}(n)\mathbf{u}^H(n)]\boldsymbol{\epsilon}(n-1) + \mathbf{\Phi}^{-1}(n)\mathbf{u}(n)v^*(n) + (1-a)\mathbf{w}_o(n-1) - \boldsymbol{\omega}(n) \tag{16.43}$$

where $\mathbf{I}$ is the identity matrix. The linear stochastic difference equation (16.43) provides a complete description of the RLS algorithm embedded in the system model of Fig. 16.1, bearing in mind the aforementioned minor change in notation. As with the LMS algorithm, we assume that the model parameter a is very close to 1, so that we may ignore the term $(1-a)\mathbf{w}_o(n-1)$. That is, the process equation is described essentially by a random-walk model, for which Eq. (16.43) reduces to

$$\boldsymbol{\epsilon}(n) = [\mathbf{I} - \mathbf{\Phi}^{-1}(n)\mathbf{u}(n)\mathbf{u}^H(n)]\boldsymbol{\epsilon}(n-1) + \mathbf{\Phi}^{-1}(n)\mathbf{u}(n)v^*(n) - \boldsymbol{\omega}(n) \tag{16.44}$$

Before proceeding further, it is instructive to find an approximation for the inverse matrix $\mathbf{\Phi}^{-1}(n)$ that makes the tracking analysis of the RLS algorithm mathematically trackable in a meaningful manner. To do so, we first take the expectation of both sides of Eq. (16.39), obtaining

$$\begin{aligned} E[\mathbf{\Phi}(n)] &= \sum_{i=1}^{n} \lambda^{n-i} E[\mathbf{u}(i)\mathbf{u}^H(i)] \\ &= \sum_{i=1}^{n} \lambda^{n-i}\mathbf{R} \\ &= \mathbf{R}(1 + \lambda + \lambda^2 + \ldots + \lambda^{n-1}) \end{aligned} \tag{16.45}$$

where $\mathbf{R}$ is the ensemble-averaged correlation matrix of the input vector $\mathbf{u}(n)$. The series inside the parentheses on the right-hand side of Eq. (16.45) represents a geometric series with the following description: a first term equal to unity, geometric ratio equal to λ, and

length equal to n. Assuming that n is large enough for us to treat the geometric series to be essentially of infinite length, we may use the formula for the sum of such a series to rewrite Eq. (16.45) in the compact form

$$E[\mathbf{\Phi}(n)] = \frac{\mathbf{R}}{1-\lambda}, \qquad n \text{ large} \tag{16.46}$$

Equation (16.46) defines the expectation (ensemble average) of $\mathbf{\Phi}(n)$, on the basis of which we may express $\mathbf{\Phi}(n)$ itself as follows (Eleftheriou and Falconer, 1986):

$$\mathbf{\Phi}(n) = \frac{\mathbf{R}}{1-\lambda} + \tilde{\mathbf{\Phi}}(n), \qquad n \text{ large} \tag{16.47}$$

where $\tilde{\mathbf{\Phi}}(n)$ is a Hermitian *perturbation matrix* whose individual entries are represented by zero-mean random variables that are statistically independent from the input vector $\mathbf{u}(n)$. Assuming a slow adaptive process (i.e., the exponential weighting factor λ is close to unity), we may view the $\mathbf{\Phi}(n)$ in Eq. (16.47) as a *quasi-deterministic* matrix, in the sense that for large n we have[2]

$$E[\|\tilde{\mathbf{\Phi}}(n)\|^2] << E[\|\mathbf{\Phi}(n)\|^2]$$

where $\|\cdot\|$ denotes matrix norm. Under this condition, we may go one step further by ignoring the perturbation matrix $\tilde{\mathbf{\Phi}}(n)$, and so approximate the correlation matrix $\mathbf{\Phi}(n)$ as

$$\mathbf{\Phi}(n) \simeq \frac{\mathbf{R}}{1-\lambda}, \qquad n \text{ large} \tag{16.48}$$

This approximation is crucial to the tracking analysis of the RLS algorithm presented herein. In a corresponding way to Eq. (16.48), we may express the inverse matrix $\mathbf{\Phi}^{-1}(n)$ as

$$\mathbf{\Phi}^{-1}(n) \simeq (1-\lambda)\mathbf{R}^{-1}, \qquad n \text{ large} \tag{16.49}$$

where $\mathbf{R}^{-1}$ is the inverse of the ensemble-averaged correlation matrix $\mathbf{R}$.

Returning to Eq. (16.44) and using the approximation of (16.49) for $\mathbf{\Phi}^{-1}(n)$, we may now write

$$\begin{aligned}\boldsymbol{\epsilon}(n) \simeq\ & [\mathbf{I} - (1-\lambda)\mathbf{R}^{-1}\mathbf{u}(n)\mathbf{u}^H(n)]\boldsymbol{\epsilon}(n-1) \\ & + (1-\lambda)\mathbf{R}^{-1}\mathbf{u}(n)v^*(n) - \boldsymbol{\omega}(n), \qquad n \text{ large}\end{aligned} \tag{16.50}$$

[2]A completely general proof that the correlation matrix $\mathbf{\Phi}(n)$ is quasi-deterministic is yet to be presented in the literature. This issue was apparently first discussed in Eleftheriou and Falconer (1986) using heuristic arguments. It is also discussed in Macchi and Bershad (1991), where in Appendix II of that paper a proof is presented for the case of a nonstationary signal, namely, a noisy chirped sinusoid. This signal includes the commonly encountered example of a pure sinusoid in additive white Gaussian noise as a special case, which validates the proof for a stationary environment, too. However, a limitation of the proof presented by Macchi and Bershad is that it hinges on the unrealistic assumption that successive input vectors are statistically independent.

Typically, the exponential weighting factor λ is close to unity so that $1 - \lambda$ has a small value. Then, invoking the direct-averaging method, we may state that the solution $\boldsymbol{\epsilon}(n)$ is "close" to the solution of the new stochastic difference equation:

$$\boldsymbol{\epsilon}(n) \simeq \lambda\, \boldsymbol{\epsilon}(n-1) + (1-\lambda)\mathbf{R}^{-1}\mathbf{u}(n)v^*(n) - \boldsymbol{\omega}(n), \qquad n \text{ large} \tag{16.51}$$

which is obtained by replacing the system matrix $[\mathbf{I} - (1-\lambda)\mathbf{R}^{-1}\mathbf{u}(n)\mathbf{u}^H(n)]$ in Eq. (16.50) by its ensemble average $\lambda\mathbf{I}$. For convenience of presentation, we have again retained the same notation for the weight-error vector in Eq. (16.51) as that used previously. Finally, evaluating the correlation matrix of $\boldsymbol{\epsilon}(n)$ in Eq. (16.51) and invoking the independence assumption, we obtain

$$\mathbf{K}(n) \simeq \lambda^2\, \mathbf{K}(n-1) + (1-\lambda)^2\sigma_v^2\mathbf{R}^{-1} + \mathbf{Q}, \qquad n \text{ large} \tag{16.52}$$

Equation (16.52) for the RLS algorithm has a form that is dramatically different from that of Eq. (16.28) for the LMS algorithm, which, of course, is to be expected.

For a *steady-state solution* of the difference equation (16.52), for which n is large, we may legitimately set $\mathbf{K}(n-1) = \mathbf{K}(n)$. Under this condition, Eq. (16.52) takes on the simplified form

$$(1-\lambda^2)\mathbf{K}(n) \simeq (1-\lambda)^2\sigma_v^2\mathbf{R}^{-1} + \mathbf{Q}, \qquad n \text{ large} \tag{16.53}$$

For λ close to unity, we may approximate $1-\lambda^2$ as follows:

$$\begin{aligned} 1 - \lambda^2 &= (1-\lambda)(1+\lambda) \\ &\simeq 2(1-\lambda) \end{aligned} \tag{16.54}$$

Accordingly, we may further simplify the correlation matrix $\mathbf{K}(n)$ for the RLS algorithm as follows:

$$\mathbf{K}(n) \simeq \frac{1-\lambda}{2}\sigma_v^2\mathbf{R}^{-1} + \frac{1}{2(1-\lambda)}\mathbf{Q}, \qquad n \text{ large} \tag{16.55}$$

This is the equation for evaluating the tracking capability of the RLS algorithm for the system identification problem described in Fig. 16.1, subject to the condition that a is close to unity.

Mean-square Deviation of the RLS Algorithm

Applying the formula of Eq. (16.10) to (16.55), we readily find that the mean-square deviation of the RLS algorithm is defined by

$$\mathcal{D}(n) \simeq \frac{1-\lambda}{2}\sigma_v^2\,\mathrm{tr}[\mathbf{R}^{-1}] + \frac{1}{2(1-\lambda)}\,\mathrm{tr}[\mathbf{Q}], \qquad n \text{ large} \tag{16.56}$$

The first term, $(1-\lambda)\sigma_v^2\mathrm{tr}[\mathbf{R}^{-1}]/2$, is the estimation variance due to the measurement noise $v(n)$. The second term, $\mathrm{tr}[\mathbf{Q}]/2(1-\lambda)$, is the lag variance due to the process noise vector $\boldsymbol{\omega}$. These two contributions vary in proportion to $(1-\lambda)$ and $(1-\lambda)^{-1}$, respec-

tively. The optimum value of the forgetting factor, λ_{opt}, occurs when these two contributions are equal. Thus, from Eq. (16.56) we readily find that

$$\lambda_{opt} \simeq 1 - \frac{1}{\sigma_v}\left(\frac{\text{tr}[\mathbf{Q}]}{\text{tr}[\mathbf{R}^{-1}]}\right)^{1/2} \tag{16.57}$$

Correspondingly, the minimum mean-square deviation of the RLS algorithm is given by

$$\mathcal{D}_{min} \simeq \sigma_v\,(\text{tr}[\mathbf{R}^{-1}]\text{tr}[\mathbf{Q}])^{1/2} \tag{16.58}$$

Misadjustment of the RLS Algorithm

Multiplying both sides of Eq. (16.55) by the correlation matrix $\mathbf{R}$, we get

$$\mathbf{RK}(n) \simeq \frac{1-\lambda}{2}\sigma_v^2\mathbf{I} + \frac{1}{2(1-\lambda)}\mathbf{RQ}, \qquad n \text{ large} \tag{16.59}$$

The identity matrix $\mathbf{I}$ is of size M-by-M, where M is the number of taps in the adaptive transversal filter. Hence, taking the trace of the two sides of Eq. (16.59) yields

$$\text{tr}[\mathbf{RK}(n)] \simeq \frac{1-\lambda}{2}\sigma_v^2 M + \frac{1}{2(1-\lambda)}\text{tr}[\mathbf{RQ}], \qquad n \text{ large} \tag{16.60}$$

Finally, using the formula of Eq. (16.21), we readily find that the misadjustment of the RLS algorithm is given by

$$\mathcal{M}(n) \simeq \frac{1-\lambda}{2}M + \frac{1}{2(1-\lambda)\sigma_v^2}\text{tr}[\mathbf{RQ}], \qquad n \text{ large} \tag{16.61}$$

The first term on the right-hand side of Eq. (16.61) represents the noise misadjustment of the RLS algorithm due to the measurement noise $v(n)$. It varies linearly with $1 - \lambda$; note also that it depends on the number of taps M in the adaptive transversal filter. The second term represents the lag misadjustment of the RLS algorithm due to the process noise vector $\boldsymbol{\omega}(n)$; it varies inversely with $1 - \lambda$. The optimum value of the forgetting factor, λ_{opt}, occurs when these two contributions are equal. We thus find from Eq. (16.61) that

$$\lambda_{opt} \simeq 1 - \frac{1}{\sigma_v}\left(\frac{1}{M}\text{tr}[\mathbf{RQ}]\right)^{1/2} \tag{16.62}$$

Correspondingly, the minimum value of the misadjustment produced by the RLS algorithm is given by

$$\mathcal{M}_{min} \simeq \frac{1}{\sigma_v}\,(M\,\text{tr}[\mathbf{RQ}])^{1/2} \tag{16.63}$$

Here also we find that the two criteria, minimum misadjustment and minimum mean-square deviation, lead to different values for λ_{opt}. For these values to be meaningful, we must have $0 < \lambda_{opt} < 1$.

We now have all the tools we need to make a quantitative comparison between the LMS and RLS algorithms in the context of the system model depicted in Fig. 16.1.

the stochastic difference equations (16.26) and (16.50). In the RLS algorithm, the input vector $\mathbf{u}(n)$ is premultiplied by the inverse matrix $\mathbf{R}^{-1}$, wherein lies the fundamental difference between the LMS and RLS algorithms. Moreover, comparing Eqs. (16.26) and (16.50) on which the tracking analysis presented in the previous two sections is based, we see that $1 - \lambda$ in the RLS algorithm plays an analogous role to that of μ in the LMS algorithm. In making this analogy, however, we should try to be more precise. In particular, the exponential weighting factor λ is dimensionless, whereas the step-size parameter μ has the inverse dimension of power. To correct for this dimensional discrepancy, we do the following:

- For the LMS algorithm, we define the *normalized step-size parameter*

$$\nu = \mu\sigma_u^2 \tag{16.64}$$

  where σ_μ^2 is the variance of the zero-mean tap input $u(n)$.

- For the RLS algorithm, we define the *forgetting rate*

$$\beta = 1 - \lambda \tag{16.65}$$

Moving into the main issue of interest, we may use Eqs. (16.33) and (16.58), and Eqs. (16.37) and (16.63) to formulate a corresponding pair of ratios for comparing the "optimum" tracking performance of the LMS and RLS algorithms for the system identification problem at hand; one ratio is based on the mean-square deviation and the other is based on misadjustment as the figure of merit. Specifically, we may write

$$\frac{\mathcal{D}_{\min}^{\text{LMS}}}{\mathcal{D}_{\min}^{\text{RLS}}} \simeq \left(\frac{M\text{tr}[\mathbf{R}^{-1}\mathbf{Q}]}{\text{tr}[\mathbf{R}^{-1}]\text{tr}[\mathbf{Q}]}\right)^{1/2} \tag{16.66}$$

and

$$\frac{\mathcal{M}_{\min}^{\text{LMS}}}{\mathcal{M}_{\min}^{\text{RLS}}} \simeq \left(\frac{\text{tr}[\mathbf{R}]\text{tr}[\mathbf{Q}]}{M\text{tr}[\mathbf{RQ}]}\right)^{1/2} \tag{16.67}$$

where $\mathbf{R}$ is the correlation matrix of the input vector $\mathbf{u}(n)$, $\mathbf{Q}$ is the correlation matrix of the process noise vector $\boldsymbol{\omega}(n)$, and M is the number of taps in the adaptive transversal filter of Fig. 16.1. Clearly, whatever comparison we make between the LMS and RLS algorithms on the basis of Eqs. (16.66) and (16.67), the result depends on the prevalent environmental conditions and, in particular, on how the correlation matrices $\mathbf{Q}$ and $\mathbf{R}$ are defined. In what follows, we consider three specific examples.[3]

[3]Example 1 is discussed in Widrow and Walach (1984) and Eleftheriou and Falconer (1986); Examples 2 and 3 are discussed in Benveniste et al. (1987) and Slock and Kailath (1993).

Example 1: $\mathbf{Q} = \sigma_\omega^2 \mathbf{I}$

Consider first the case of process noise vector $\boldsymbol{\omega}(n)$ in the first-order Markov model of Eq. (16.1) originating from a white noise source of zero mean and variance σ_ω^2. We may thus express the correlation matrix $\mathbf{Q}$ of $\boldsymbol{\omega}(n)$ as

$$\mathbf{Q} = \sigma_\omega^2 \mathbf{I} \tag{16.68}$$

where $\mathbf{I}$ is the M-by-M identity matrix. Then, using Eq. (16.68) in (16.66) and (16.67), we get the following respective results (after canceling common terms):

$$\mathcal{D}_{\text{min}}^{\text{LMS}} \simeq \mathcal{D}_{\text{min}}^{\text{RLS}}, \qquad \mathbf{Q} = \sigma_\omega^2 \mathbf{I} \tag{16.69}$$

and

$$\mathcal{M}_{\text{min}}^{\text{LMS}} \simeq \mathcal{M}_{\text{min}}^{\text{RLS}}, \qquad \mathbf{Q} = \sigma_\omega^2 \mathbf{I} \tag{16.70}$$

Accordingly, we may state that the LMS and RLS algorithms produce essentially the same minimum levels of misadjustment and mean-square deviation for the case of a process noise vector $\boldsymbol{\omega}(n)$ drawn from a white-noise source.

Example 2: $\mathbf{Q} = c_1 \mathbf{R}$

Consider next the example when the correlation matrix $\mathbf{Q}$ of the process noise vector $\boldsymbol{\omega}(n)$ in the first-order Markov model of Eq. (16.1) equals a constant c_1 times the correlation matrix $\mathbf{R}$ of the input vector $\mathbf{u}(n)$. The scaling factor c_1 is introduced here for two reasons:

1. To account for the fact that the process noise vector $\boldsymbol{\omega}(n)$ and the input vector $\mathbf{u}(n)$ are ordinarily measured in different units.
2. To ensure that the optimum μ for the LMS algorithm in Eq. (16.32) or (16.36), and the optimum λ for the RLS algorithm in Eq. (16.57) or (16.62) assume meaningful values.

Thus, putting $\mathbf{Q} = c_1 \mathbf{R}$ in Eqs. (16.66) and (16.67) and canceling the scaling factor c_1, we get the two comparative yardsticks listed under $\mathbf{Q} = c_1 \mathbf{R}$ in Table 16.1. Before commenting on these results, it is instructive to go on and consider the complementary example described next.

TABLE 16.1 COMPARATIVE YARDSTICKS FOR LMS AND RLS ALGORITHMS FOR EXAMPLES 2 AND 3

	$\mathbf{Q} = c_1 \mathbf{R}$	$\mathbf{Q} = c_2 \mathbf{R}^{-1}$
$\dfrac{\mathcal{D}_{\text{min}}^{\text{LMS}}}{\mathcal{D}_{\text{min}}^{\text{RLS}}}$	$\dfrac{M}{(\text{tr}[\mathbf{R}^{-1}]\ \text{tr}[\mathbf{R}])^{1/2}}$	$\dfrac{(M\ \text{tr}[\mathbf{R}^{-2}])^{1/2}}{\text{tr}[\mathbf{R}^{-1}]}$
$\dfrac{\mathcal{M}_{\text{min}}^{\text{LMS}}}{\mathcal{M}_{\text{min}}^{\text{RLS}}}$	$\dfrac{\text{tr}[\mathbf{R}]}{(M\ \text{tr}[\mathbf{R}^2])^{1/2}}$	$\dfrac{1}{M}(\text{tr}[\mathbf{R}]\text{tr}[\mathbf{R}^{-1}])^{1/2}$

Example 3. $\mathbf{Q} = c_2\mathbf{R}^{-1}$

In this final example, the correlation matrix $\mathbf{Q}$ of the process noise vector $\boldsymbol{\omega}(n)$ is equal to a constant c_2 times the inverse of the correlation matrix $\mathbf{R}$ of the input vector $\mathbf{u}(n)$. The scaling factor c_2 is used here for exactly the same reasons explained in Example 2. Thus, putting $\mathbf{Q} = c_2\mathbf{R}^{-1}$ in Eqs. (16.66) and (16.67) and again canceling the scaling factor c_2, we get the remaining two comparative yardsticks listed under $\mathbf{Q} = c_2\mathbf{R}^{-1}$ in Table 16.1.

The 2-by-2 array of entries shown in this table exhibits a useful property, namely, that of *reciprocal symmetry*. The significance of this property in the context of tracking will become apparent presently.

The cross-diagonal terms in the array of Table 16.1 lend themselves to the application of the *Cauchy–Schwarz inequality*, which results in the following two useful bounds (see Problem 5):

$$\mathcal{M}_{\min}^{\text{LMS}} \leq \mathcal{M}_{\min}^{\text{RLS}} \qquad \text{for } \mathbf{Q} = c_1\mathbf{R} \tag{16.71}$$

and

$$\mathcal{D}_{\min}^{\text{RLS}} \leq \mathcal{D}_{\min}^{\text{LMS}} \qquad \text{for } \mathbf{Q} = c_2\mathbf{R}^{-1} \tag{16.72}$$

These two bounds are indeed manifestations of the property of reciprocal symmetry, on the basis of which we may make the following two statements:

1. For $\mathbf{Q} = c_1\mathbf{R}$, the LMS algorithm performs better than the RLS algorithm, in that it produces a minimum level of misadjustment $\mathcal{M}_{\min}$ that is smaller than the corresponding value produced by the RLS algorithm.
2. For $\mathbf{Q} = c_2\mathbf{R}^{-1}$, the RLS algorithm performs better than the LMS algorithm, in that it produces a minimum mean-square deviation $\mathcal{D}_{\min}$ that is smaller than the corresponding value produced by the LMS algorithm.

However, in general, we cannot be as conclusive on the implications of the diagonal entries in Table 16.1. Nevertheless, in light of the aforementioned property of reciprocal symmetry, we can say the following. If, for $\mathbf{Q} = c_1\mathbf{R}$, the minimum mean-square deviation $\mathcal{D}_{\min}$ produced by the LMS algorithm is smaller than the corresponding value produced by the RLS algorithm, then it is true that for $\mathbf{Q} = c_2\mathbf{R}^{-1}$ the minimum misadjustment $\mathcal{M}_{\min}$ produced by the RLS algorithm is smaller than the corresponding value produced by the LMS algorithm.

To illustrate the validity of this latter statement, consider the special case of an adaptive filter with $M = 2$, for which the 2-by-2 correlation matrix of the input vector $\mathbf{u}(n)$ is denoted by

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{21} \\ r_{21} & r_{22} \end{bmatrix}$$

For this specification of $\mathbf{R}$, the 2-by-2 array of Table 16.1 takes on the particular form presented in Table 16.2. Next, recognizing that since any 2-by-2 correlation matrix satisfies the condition

$$(r_{11} - r_{22})^2 + (2r_{21})^2 \geq 0,$$

TABLE 16.2 COMPARATIVE YARDSTICKS FOR LMS AND RLS ALGORITHMS FOR EXAMPLES 2 AND 3, ASSUMING $M = 2$

	$\mathbf{Q} = c_1\mathbf{R}$	$\mathbf{Q} = c_2\mathbf{R}^{-1}$
$\dfrac{\mathcal{D}_{\min}^{\text{LMS}}}{\mathcal{D}_{\min}^{\text{RLS}}}$	$\dfrac{2\sqrt{r_{11}r_{22}-r_{21}^2}}{r_{11}+r_{22}}$	$\dfrac{\sqrt{2(r_{11}^2+2r_{21}^2+r_{22}^2)}}{r_{11}+r_{22}}$
$\dfrac{\mathcal{M}_{\min}^{\text{LMS}}}{\mathcal{M}_{\min}^{\text{RLS}}}$	$\dfrac{r_{11}+r_{22}}{\sqrt{2(r_{11}^2+2r_{21}^2+r_{22}^2)}}$	$\dfrac{r_{11}+r_{22}}{2\sqrt{r_{11}r_{22}-r_{21}^2}}$

then Table 16.2 leads us to make the following statements encompassing all the four entries of the array:

1. For $\mathbf{Q} = c_1\mathbf{R}$, the LMS algorithm performs better than the RLS algorithm, in that it yields smaller values for both $\mathcal{D}_{\min}$ and $\mathcal{M}_{\min}$.
2. For $\mathbf{Q} = c_2\mathbf{R}^{-1}$, the RLS algorithm performs better than the LMS algorithm, in that it yields smaller values for both $\mathcal{D}_{\min}$ and $\mathcal{M}_{\min}$.

Examples 2 and 3 clearly illustrate that neither the LMS algorithm nor the RLS algorithm has a complete monopoly over a good tracking behavior. Rather, we find that one or the other of these two adaptive filtering algorithms is the preferred algorithm for tracking a nonstationary environment, depending on the prevalent environmental conditions.

16.7 ADAPTIVE RECOVERY OF A CHIRPED SINUSOID IN NOISE

Up to this point in our discussion of using an adaptive filter to track a time-varying system, we have focused our attention on the performance of LMS and RLS algorithms in the context of system identification. As mentioned previously in Section 16.1, in such a scenario, only the cross-correlation vector between the input vector $\mathbf{u}(n)$ and the desired response $d(n)$ is time varying. In this section we briefly consider a more difficult problem: the *adaptive recovery of a chirped sinusoid* (tone) buried in additive white Gaussian noise (Macchi and Bershad, 1991; Bershad and Macchi, 1991; Macchi, 1995).

To perform such a task, we may use an *adaptive line enhancer* (ALE) that consists of a one-step predictor, configured as shown in Fig. 16.2. The received (input) signal $u(n)$ consists of two components:

$$u(n) = s(n) + v(n) \tag{16.73}$$

where $s(n)$ is the desired signal and $v(n)$ is the additive noise component. Typically, the desired signal $s(n)$ has a much narrower bandwidth than the noise $v(n)$. This property is exploited by the ALE to produce an output signal $\hat{s}(n)$ that represents an estimate of the desired signal $s(n)$. The difference between the input signal $u(n)$ and the output signal $\hat{s}(n)$ defines the error signal $e(n)$, which is used to adjust the tap weights of the adaptive filter.

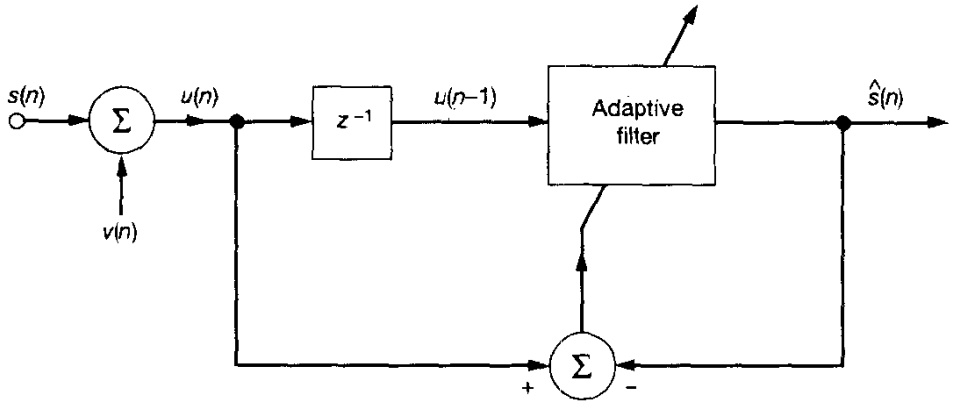


Figure 16.2 Adaptive line enhancer.

The adaptive recovery of a chirped sinusoid in noise is of special interest for several reasons:

- The chirped sinusoid represents a well-defined form of nonstationarity that is of a deterministic nature.
- The chirped sinusoid, characterized by a linear shift in frequency, may be used as a first-order model of the Doppler effect encountered in mobile communications.
- In adaptive prediction, both the correlation matrix of the input vector $\mathbf{u}(n)$ and the cross-correlation vector between $\mathbf{u}(n)$ and the desired response represented by the first element of $\mathbf{u}(n)$ are time varying. Accordingly, a mathematical analysis of tracking a chirped sinusoid in noise is more difficult than the system identification problem.

The chirped sinusoid, denoted by $s(n)$, is defined by the complex exponential

$$s(n) = \sqrt{P_s}\exp\left(j2\pi f_c n + j\frac{\psi}{2}n^2 + \varphi\right) \tag{16.74}$$

where $\sqrt{P_s}$ is the *signal amplitude* assumed to be constant, f_c is the *center frequency*, ψ is the chirp rate, and φ is an arbitrary phase shift. The signal $s(n)$ is deterministic but nonstationary because of the chirping. The instantaneous angular frequency of the chirped sinusoid $s(n)$ is defined by the derivative

$$\frac{d}{dn}\left(2\pi f_c n + \frac{\psi}{2}n^2 + \varphi\right) = 2\pi f_c + \psi n$$

The angular frequency deviation, measured inside a time interval τ, is $\psi\tau$. Equation (16.74) may therefore be viewed as a narrow-band model, provided that we have

$$\psi\tau << 2\pi f_c$$

The complex chirped sinusoid $s(n)$ is corrupted by additive white Gaussian noise $v(n)$ of zero mean and variance σ_v^2, as indicated in Eq. (16.73). The signal-to-noise ratio at the ALE input is thus defined by

$$\rho = \frac{P_s}{\sigma_v^2} \tag{16.75}$$

A detailed mathematical analysis of recovery of the chirped sinusoid $s(n)$ from the noisy received signal $u(n)$ using an ALE based on the LMS algorithm, is presented in (Bershad and Macchi, 1991). The corresponding analysis for the case of an ALE based on the RLS algorithm is presented in (Macchi and Bershad, 1991). In order to obtain functionally similar parameters for both the LMS and RLS algorithms, and for reasons discussed in Section 16.6, the following *normalized adaptation constants* are introduced for these two algorithms (Macchi et al., 1991):

$$\nu = \begin{cases} \mu\sigma_u^2(1 + \rho) & \text{for LMS} \\ \beta = 1 - \lambda & \text{for RLS} \end{cases} \tag{16.76}$$

The definition of ν for the RLS algorithm is the same as before; but for the LMS algorithm, it now incorporates the factor $(1 + \rho)$, where ρ is the signal-to-noise ratio. For both the LMS and RLS algorithms, highlights of the findings reported in (Macchi and Bershad, 1991; Bershad and Macchi, 1991) may be summarized as follows:

- The noise misadjustment due to the weight vector noise $\boldsymbol{\epsilon}_1(n)$ is dominated by a term of order ν, as shown by

$$\mathcal{M}_1 = \frac{MJ_{\min}}{2}\nu \tag{16.77}$$

  where $J_{\min}$ is the minimum mean-squared error produced by the Wiener filter; for the chirp input it is given by

$$J_{\min} = \frac{M + 1}{M}\sigma_v^2 \tag{16.78}$$

- The lag misadjustment due to the mean lag vector $\boldsymbol{\epsilon}_2(n)$ is dominated by a term of order ν^{-2}, as shown by

$$\mathcal{M}_2 = \frac{\psi^2}{\nu^2} C(\rho, M)\,\sigma_v^2 \tag{16.79}$$

  where $C(\rho,M)$ is a proportionality factor depending on the algorithm used. Specifically, we have

$$C(\rho, M) = \begin{cases} \dfrac{1}{12}(M + 1)\left(1 - \dfrac{1}{M}\right)(1 + \rho)^2 & \text{for LMS} \\ \left(\dfrac{M + 1}{2}\right)^2 \rho & \text{for RLS} \end{cases} \tag{16.80}$$

The overall misadjustment is equal to the sum of $\mathcal{M}_1$ and $\mathcal{M}_2$ for both algorithms:

$$\mathcal{M} = \frac{MJ_{\min}}{2}\nu + \frac{\psi^2}{\nu^2} C(\rho, M)\sigma_v^2 \tag{16.81}$$

The overall misadjustment $\mathcal{M}$ is minimized by setting the normalized adaptation constant ν equal to the optimum value:

$$\nu_{\text{opt}} = \begin{cases} \left(\frac{\psi^2}{3}\left(1 - \frac{1}{M}\right)(1 + \rho)^2\right)^{1/3} & \text{for LMS} \\ (\rho\psi^2(M + 1))^{1/3} & \text{for RLS} \end{cases} \tag{16.82}$$

For both algorithms, the minimum misadjustment is

$$\mathcal{M}_{\min} = \frac{3}{4} M J_{\min} \nu_{\text{opt}} \tag{16.83}$$

The relative tracking performance of the LMS and RLS algorithms for the recovery of a chirped sinusoid in noise may be determined by considering the ratio:

$$\begin{aligned} \frac{\mathcal{M}_{\min}^{\text{LMS}}}{\mathcal{M}_{\min}^{\text{RLS}}} &= \frac{\nu_{\text{opt}}^{\text{LMS}}}{\nu_{\text{opt}}^{\text{RLS}}} \\ &\simeq \left(\frac{(1 + \rho)^2}{3\rho M}\right)^{1/3}, \qquad M \text{ large} \end{aligned} \tag{16.84}$$

where, in the last line, the approximation is taken for large M. Based on the result of Eq. (16.84), we readily find that for $\rho < 3M$ the LMS algorithm has a smaller misadjustment than the RLS algorithm, and for $\rho > 3M$ the RLS algorithm has a smaller misadjustment than the LMS algorithm. In other words, for small signal-to-noise ratios the LMS algorithm has a better tracking performance than the RLS algorithm, and for large signal-to-noise ratios the reverse is true.

Sensitivity to the Choice of Adaptation Rate

The minimum misadjustment of Eq. (16.83) for the chirped sinusoid in noise is only 50 percent greater than the misadjustment of Eq. (16.77) for the stationary case of an ordinary sinusoid in noise, providing that the optimum adaptation constant of Eq. (16.82) is utilized. The main difference between the stationary and chirped sinusoidal cases is that, for the stationary case, the misadjustment can be made arbitrarily small by decreasing ν as indicated by Eq. (16.77). For the chirped sinusoidal case, on the other hand, the penalty incurred by setting ν too small is an unbounded increase in the lag misadjustment $\mathcal{M}_2$, as indicated by Eq. (16.79). By using the optimum value of ν specified in Eq. (16.82), the best compromise between the weight vector noise $\boldsymbol{\epsilon}_1(n)$ and the mean lag vector $\boldsymbol{\epsilon}_2(n)$ is achieved. This is illustrated further in what follows for the RLS algorithm.[4]

The sensitivity in tracking performance to the choice of the adaptation constant $\nu = (1 - \beta) = 1 - \lambda$ for the RLS algorithm increases with increasing chirp rate ψ. This can be illustrated by evaluating the reflection coefficients of the recursive least-squares lattice (LSL) algorithm (i.e., lattice implementation of the RLS algorithm). For stationary

[4]The material presented in this subsection is based on Zeidler, J.R., private communication, 1995.

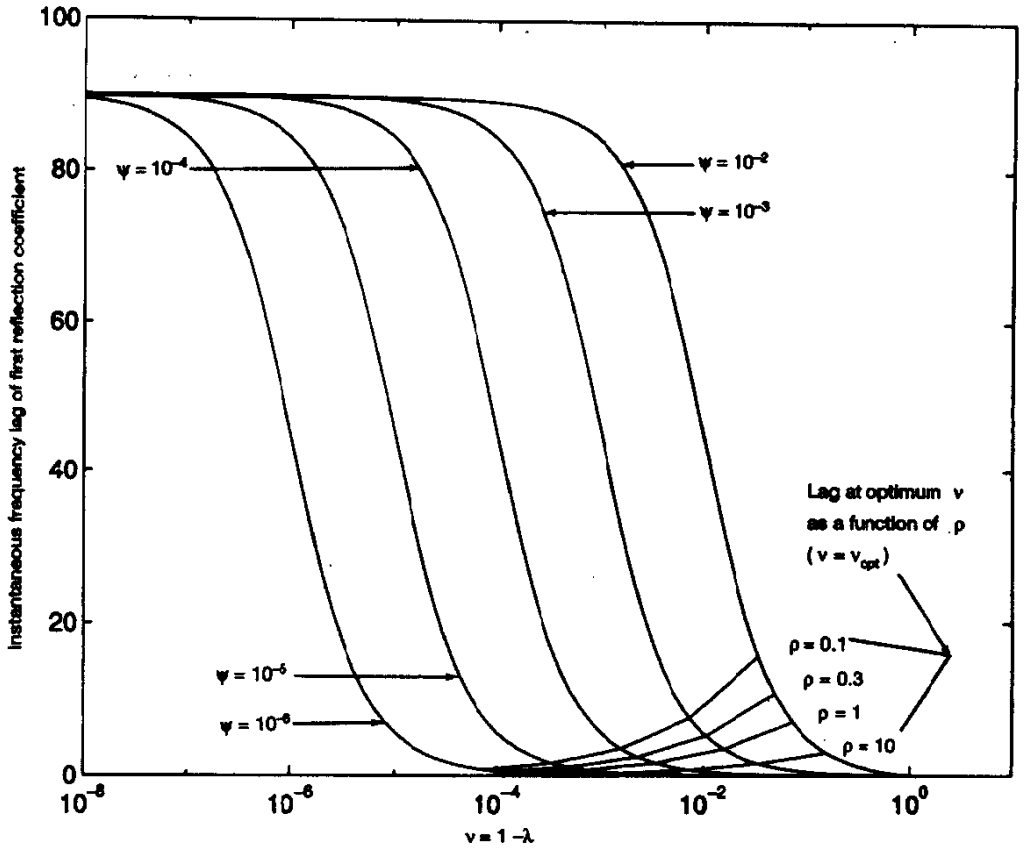


Figure 16.3 The instantaneous frequency lag as a function of the adaptation constant ν. The frequency lag for $\nu = \nu_{opt}$ is indicated for $\rho = 0.1$ to $\rho = 10$ on the curves.

inputs, the derivations of these reflection coefficients were presented in Chapter 15. The analysis of that chapter has been extended to the nonstationary case of a chirped sinusoid in noise in (Soni et al., 1995). Figure 16.3 shows the expected value of the frequency lag derived by Soni et al. for the first reflection coefficient of a lattice predictor of order $M = 3$, plotted as a function of ν for a wide range of chirp rates between $\psi = 10^{-2}$ and $\psi = 10^{-6}$. The *frequency lag* is defined as the steady-state difference in phase lag between the reflection coefficient of a particular stage in the lattice predictor and the corresponding reflection coefficient of the "optimum" lattice predictor that *works just as well with a chirped sinusoid as with a pure sinusoid.*[5] It is thus a good measure for assessing the tracking performance of the lattice predictor.

[5]This optimum time-varying predictor is in perfect accord with an idea first described in (Macchi and Bershad, 1991) for a transversal filter implementation of the RLS algorithm.

The magnitude of the optimum ν specified by Eq. (16.82) is also overlaid on the frequency lag versus ν plots for values of the signal-to-noise ratio between $\rho = 0.1$ and $\rho = 10$ in order to illustrate the frequency lag associated with the optimum adaptation constant ν_{opt}. Based on the results presented in Fig. 16.3, we note the following:

- For slow chirp rates, the instantaneous frequency lag associated with the optimum adaptation constant ν_{opt} is less than one degree; and there is a wide range of values for ν in the vicinity of ν_{opt} that would provide a negligible frequency lag.
- For a fixed signal-to-noise ratio ρ, the tracking performance of the predictor deteriorates with increasing chirp rate ψ. For example, for the extreme case of $\psi = 10^{-2}$ and $\rho = 0.1$ shown in Fig. 16.3, the frequency lag associated with the optimum adaptation constant ν_{opt} increases to about 20 degrees. Moreover, at this point, the slope of the frequency lag curve versus ν is high, with the result that the frequency lag changes rapidly with variations in ν.
- For a fixed chirp rate ψ, the tracking performance of the predictor improves with increasing signal-to-noise ratio ρ. For example, for the fast chirp rate $\psi = 10^{-2}$ and high signal-to-noise ratio $\rho = 10$, the frequency lag associated with the optimum adaptation constant ν_{opt} decreases to less than 5 degrees. Also, at this point, the slope of the frequency lag versus ν curve decreases considerably relative to its value at $\rho = 0.1$; in other words, the sensitivity of the tracking performance to variations in ν is considerably reduced with increasing ρ.

The above results clearly illustrate that by selecting an adaptation constant ν close to its optimum value ν_{opt} defined by Eq. (16.82), the lattice predictor can effectively track a chirped sinusoidal signal in noise over a wide range of chirp rates and with an acceptably small frequency lag, provided that the signal-to-noise ratio is high. The results also indicate that the sensitivity in tracking performance to the selection of ν increases nonlinearly with increasing chirp rate ψ and decreasing signal-to-noise ratio, which is intuitively satisfying.

Tracking of a Chirped Nonzero-bandwidth Signal in Noise

The desired signal in the study reported by Macchi and Bershad (1991) consists of a chirped sinusoid (tone), which is deterministic and therefore has no information content. In a subsequent study reported by Wei et al. (1994), the previous theory of Macchi and Bershad is extended to the more *general* case of a chirped signal buried in noise. In effect, the desired signal assumes a *finite bandwidth*, which makes it resemble a communication signal more closely.

To compare the tracking performances of the LMS and RLS algorithms for a chirped nonzero-bandwidth signal, Wei et al. (1994) consider an autoregressive (AR) process of order one that may be used to model many narrow-band signals. Specifically, the baseband signal of interest is modeled by the recursive equation:

$$s(n) = s(n-1) + v(n) \tag{16.85}$$

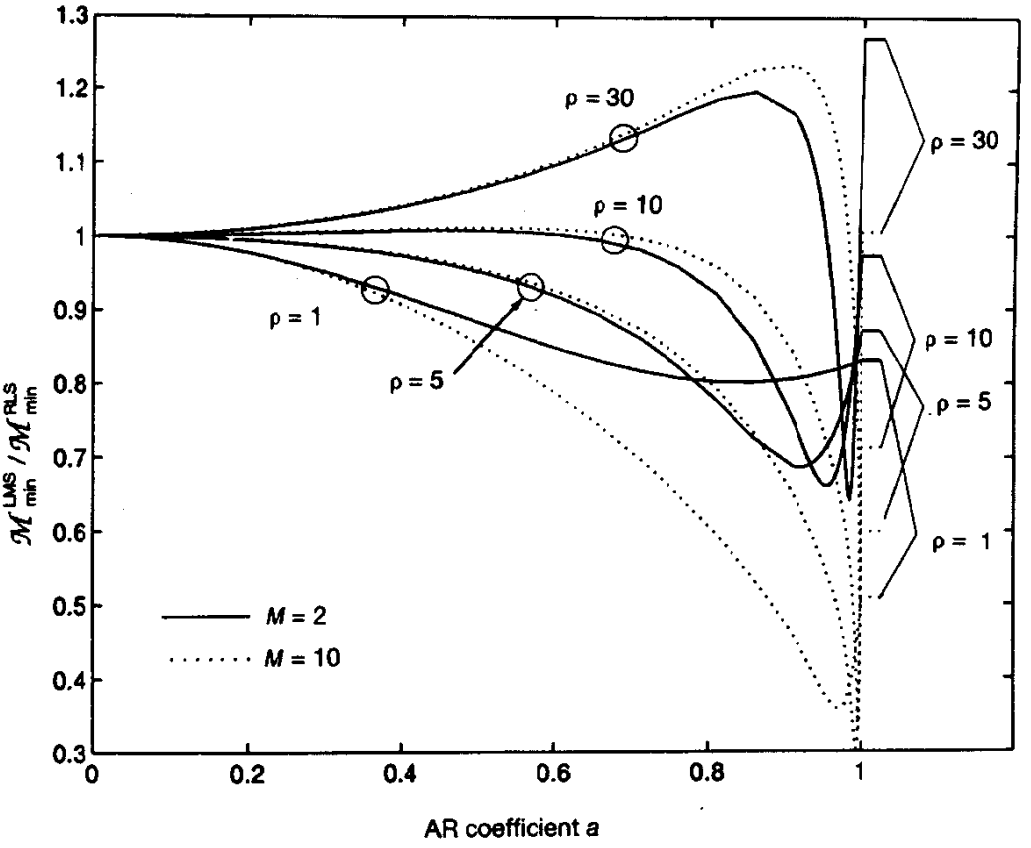


Figure 16.4 Ratio of LMS misadjustment versus RLS misadjustment for an AR process of order one.

where a is the AR coefficient, and $\nu(n)$ is a white noise process of zero mean and variance

$$\sigma_\nu^2 = \sigma_s^2 (1 - a^2) \tag{16.86}$$

where σ_s^2 is the variance of $s(n)$. In a mobile communications environment, for example, the baseband signal $u(n)$ is modulated onto a carrier wave for transmission over the channel, in the course of which it may also be Doppler-shifted by some relative motion between the transmitter and receiver. Accordingly, the baseband form of the received signal may be modeled as a chirped AR process of order one, as shown by (Wei et al., 1994)

$$s(n) = a\Omega\psi^{-1/2}\psi^n s(n-1) + \nu(n) \tag{16.87}$$

where $\nu(n)$ is a white-noise process of variance $\sigma_s^2(1 - a^2)$.

Figure 16.4 plots the ratio $\mathcal{M}_{min}^{LMS}/\mathcal{M}_{min}^{RLS}$ versus the AR coefficient a for varying signal-to-noise ratio ρ and for two different values of filter length M. Based on this figure, we may make the following observations (Wei et al., 1994):

- At low signal-to-noise ratios ($\rho \leq 10$ dB), the LMS algorithm tracks the chirped AR process better than the RLS algorithm.

- For narrow-band signals ($a \simeq 0.95$), the LMS algorithm also performs better than the RLS algorithm.
- For high signal-to-noise ratios ($\rho \geq 10$ dB), the RLS algorithm performs better than the LMS algorithm.

These observations reinforce the earlier conclusions reached by Macchi and Bershad, in that the LMS algorithm or the RLS algorithm has a better tracking performance, depending on the signal-to-noise ratio.

16.8 HOW TO IMPROVE THE TRACKING BEHAVIOR OF THE RLS ALGORITHM

For a given "linear" model of a time-varying system, it is well known that the Kalman filter is the optimum tracking algorithm, assuming Gaussian statistics. In light of the material presented in Chapters 13 to 15, we know that there are one-to-one correspondences between the Kalman filter and the RLS algorithm. Simply put, the RLS algorithm is a special case of the Kalman filter. Given this intimate relationship between the RLS algorithm and Kalman filter, how then is it that the RLS algorithm and its variants have not fully inherited the good tracking properties of the Kalman filter? Before proceeding to offer a possible explanation for this dilemma, it is informative to recall the underlying state-space model of the standard RLS algorithm, which has the form (see Chapter 13)

$$\mathbf{x}(n+1) = \lambda^{-1/2}\mathbf{x}(n) \tag{16.88}$$

$$y(n) = \mathbf{u}^H(n)\mathbf{x}(n) + v(n) \tag{16.89}$$

Comparing this state-space model with the first-order Markov model described in Eqs. (16.1) and (16.2), we immediately see that there is a serious *model mismatch* between these two situations. Specifically, the state equation (16.88) has zero process noise, which is in direct violation of what the Markov model described in Eq. (16.1) would imply. Herein lies the root of the problem.

The message that we wish to convey here is that if the system designer has prior knowledge of the underlying physical model of a particular task, then that knowledge should be exploited in formulating an appropriate state-space model for the RLS algorithm. In the sequel, we describe two examples illustrating how this objective can be accomplished.

Model I Incorporating a Process (State) Noise Vector

In the system identification problem described by Eqs. (16.1) and (16.2), the underlying Markov model has a nonzero process noise vector $\boldsymbol{\omega}(n)$. The state-space model for the RLS algorithm is therefore formulated as follows:

$$\mathbf{x}(n+1) = a\mathbf{x}(n) + \mathbf{v}_1(n) \tag{16.90}$$

$$y(n) = \mathbf{u}^H(n)\mathbf{x}(n) + v_2(n) \tag{16.91}$$

The process (state) noise vector $\mathbf{v}_1(n)$ in Eq. (16.90) is modeled as having zero mean and correlation matrix $\mathbf{Q}(n)$, the same as that of the process noise vector $\boldsymbol{\omega}(n)$ in the Markov model of Eq. (16.2). For example, we may have $\mathbf{Q}(n) = q\mathbf{I}$, where $\mathbf{I}$ is the identity matrix. In such a case, the elements of $\mathbf{v}(n)$ constitute a set of independent white noise sources, each having zero mean and variance q. The implication of using the state equation (16.90) in place of that of (16.88) is that, in the language of Kalman filter theory, the filtered state-error correlation matrix $\mathbf{K}(n)$ and the predicted state-error correlation matrix $\mathbf{K}(n + 1, n)$ are no longer equal. In particular, in view of Eq. (16.90), and assuming that $\mathbf{Q}(n) = q\mathbf{I}$, the RLS algorithm is modified as follows (Haykin et al., 1995):

Starting with $\hat{\mathbf{w}}(0|-1) = E[\mathbf{w}_o(0)]$ and $\mathbf{P}(0,-1) = \Pi_0$, compute for $n \geq 0$:

$$\mathbf{k}(n) = a\mathbf{P}(n, n-1)\mathbf{u}(n)\,[\mathbf{u}^H(n)\mathbf{P}(n, n-1)\mathbf{u}(n) + 1]^{-1}$$

$$\xi(n) = y(n) - \hat{\mathbf{w}}^H(n|n-1)\mathbf{u}(n)$$

$$\hat{\mathbf{w}}(n+1|n) = a\hat{\mathbf{w}}(n|n-1) + \mathbf{k}(n)\xi^*(n)$$

$$\mathbf{P}(n) = \mathbf{P}(n, n-1) - \frac{\mathbf{P}(n, n-1)\mathbf{u}(n)\mathbf{u}^H(n)\mathbf{P}(n, n-1)}{\mathbf{u}^H(n)\mathbf{P}(n, n-1)\mathbf{u}(n) + 1}$$

$$\mathbf{P}(n+1, n) = a^2\mathbf{P}(n) + q\mathbf{I}$$

This algorithm is hereafter referred to as the *extended RLS algorithm—version* 1 (ERLS-1).

In effect, the last computation step in the summary of the standard RLS algorithm presented in Table 13.1 has been replaced by the last two steps of ERLS-1. Note also that putting $q = 0$ and $a = 1$ reduces ERLS 1 to its standard form with $\lambda = 1$.

Model II Incorporating a Nonconstant Transition Matrix

The presence of a process noise vector, as in Eq. (16.90), describes one way in which nonstationarity can be accounted for. Another way in which nonstationarity can be represented is to have a transition matrix that is *not* a constant, as described here:

$$\mathbf{x}(n+1) = \mathbf{F}(n+1, n)\mathbf{x}(n) \tag{16.92}$$

$$y(n) = \mathbf{u}^H(n)\mathbf{x}(n) + v(n) \tag{16.93}$$

For example, in tracking a chirped signal in noise, the transition matrix $\mathbf{F}(n + 1, n)$ may consist of a diagonal matrix whose entries depend on the chirp rate and are unknown. Pursuing this example further, let $p_1, p_2, \ldots, p_M$ denote the *unknown* diagonal elements of the transition matrix, as shown by

$$\mathbf{F}(n+1, n) = \text{diag}[p_1, p_2, \ldots, p_M] \tag{16.94}$$

We may then define a new state vector:

$$\mathbf{x}'(n) = \begin{bmatrix} \mathbf{x}(n) \\ \mathbf{p} \end{bmatrix} \tag{16.95}$$

where

$$\mathbf{p} = [p_1, p_2, \ldots, p_M]^T \tag{16.96}$$

The original state-space model of Eqs. (16.92) and (16.93) may now be rewritten in the nonlinear form:

$$\begin{aligned}\mathbf{x}'(n+1) &= \begin{bmatrix}\mathbf{x}(n+1)\\ \mathbf{p}\end{bmatrix}\\ &= \begin{bmatrix}\text{diag}[p_1, p_2, \ldots, p_M] & \mathbf{O}\\ \mathbf{O} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{x}(n)\\ \mathbf{p}\end{bmatrix} \qquad (16.97)\\ &= \begin{bmatrix}\text{diag}[p_1, p_2, \ldots, p_M] & \mathbf{O}\\ \mathbf{O} & \mathbf{I}\end{bmatrix}\mathbf{x}'(n)\end{aligned}$$

$$\begin{aligned}y(n) &= [\mathbf{u}^H(n), \mathbf{O}]\begin{bmatrix}\mathbf{x}(n)\\ \mathbf{p}\end{bmatrix} + v(n)\\ &= [\mathbf{u}^H(n), \mathbf{O}]\,\mathbf{x}'(n) + v(n) \qquad (16.98)\end{aligned}$$

In other words, we have a nonlinear state-space model on our hands, which may be described in words as follows:

- The modified state vector $\mathbf{x}'(n+1)$ at time $n+1$ is a nonlinear function of its value $\mathbf{x}'(n)$ at time n.
- The observation $y(n)$ consists of another nonlinear function of $\mathbf{x}'(n)$ plus a noise component $v(n)$.

The important point to note, however, is that the mathematical forms of both nonlinear functions, and the ways in which they depend on the unknown parameters $p_1, p_2, \ldots, p_M$, are *known*. This, in turn, means that we may compute the gradients of $\mathbf{x}'(n)$ and $y(n)$ with respect to these unknown parameters, and thereby set the stage for applying the RLS version of the extended Kalman filter. The *extended Kalman filter* provides a device for tracking a nonstationary system whose underlying state-space model is nonlinear. Its derivation rests on a "linearization" procedure applied to the nonlinear state-space model of the system, as discussed in Chapter 7. Basically, this procedure results in a linear "approximation" of the model, whereafter we may proceed with the application of Kalman filter theory (its RLS version in our case) to estimate both the unknown state vector $\mathbf{x}(n)$ and the unknown parameter vector $\mathbf{p}$ in the usual way. The resulting algorithm, referred to as the *extended RLS algorithm–version 2* (ERLS-2), proceeds as follows[6]

Starting with $\hat{\mathbf{w}}(0|-1) = E[\mathbf{w}_o(0)]$, $\hat{\boldsymbol{\psi}}_{|-1} = E[\boldsymbol{\psi}]$, and

$$\mathbf{P}(0, -1) = \begin{bmatrix}\boldsymbol{\Pi}_0 & \mathbf{0}\\ \mathbf{0} & \pi_0\end{bmatrix},$$

[6]The algorithm described here is due to A. H. Sayed; for more details, see Haykin et al. (1995).

compute for $n \geq 0$:

$$\mathbf{k}(n) = \mathbf{P}(n, n-1)\begin{bmatrix}\mathbf{u}(n)\\ 0\end{bmatrix}\left([\mathbf{u}^H(n)\ 0]\,\mathbf{P}(n, n-1)\begin{bmatrix}\mathbf{u}(n)\\ 0\end{bmatrix} + \sigma^2(n)\right)^{-1}$$

$$\xi(n) = y(n) - \hat{\mathbf{w}}^H(n|n-1)\mathbf{u}(n)$$

$$\begin{bmatrix}\hat{\mathbf{w}}(n|n)\\ \hat{\boldsymbol{\psi}}_{|n}\end{bmatrix} = \begin{bmatrix}\hat{\mathbf{w}}(n|n-1)\\ \hat{\boldsymbol{\psi}}_{|n-1}\end{bmatrix} + \mathbf{k}(n)\xi^*(n),$$

$$\hat{\mathbf{w}}(n+1|n) = \mathbf{F}(\hat{\boldsymbol{\psi}}_{|n})\hat{\mathbf{w}}(n|n),$$

$$\mathbf{P}(n+1, n) = \mathbf{F}(n+1, n)\mathbf{P}(n, n)\mathbf{F}^H(n+1, n),$$

$$\mathbf{P}(n, n) = (\mathbf{I} - \mathbf{k}(n)\,[\mathbf{u}^H(n)\ 0])\,\mathbf{P}(n, n-1).$$

The important point that we are trying to make here, in presenting the ERLS-1 and ERLS-2 algorithms, is that by virtue of the one-to-one correspondences between RLS variables and Kalman variables, we have the vast literature on Kalman filter theory to build improved versions of the RLS algorithm so as to properly handle the tracking of nonstationary systems.

16.9 COMPUTER EXPERIMENT ON SYSTEM IDENTIFICATION

The subject matter of the computer experiment on system identification described in this section builds on the results of Examples 2 and 3 of Section 16.6 and material presented in Section 16.8 for an adaptive transversal filter with $M = 2$ taps. The purpose of the experiment is twofold:

- To demonstrate that it is possible for the LMS algorithm to outperform the standard RLS algorithm, and vice versa.
- To demonstrate the tracking optimality of the ERLS-1 algorithm compared with the standard RLS and LMS algorithms.

In the experiment, there are three sets of parameters to be considered:

1. *Basic parameters*, made up of the following:
   - The parameter a and the variance σ_v^2 of the zero-mean measurement noise $v(n)$ in the first-order Markov model of Eq. (16.1).
   - The elements r_{11}, r_{21}, and r_{22} of the correlation matrix $\mathbf{R}$ of the zero-mean input vector $\mathbf{u}(n)$
2. *Auxiliary parameters*, made up of the following:
   - The scaling factor c_1 for the case of $\mathbf{Q} = c_1\mathbf{R}$ (Example 2)
   - The scaling factor c_2 for the case of $\mathbf{Q} = c_2\mathbf{R}^{-1}$ (Example 3)

where $\mathbf{Q}$ is the correlation matrix of the zero-mean process noise vector $\boldsymbol{\omega}(n)$ in the first-order Markov model.

3. *Frame of reference*, chosen arbitrarily as

$$\delta = \mathcal{D}_{\min}^{\text{RLS}}$$

The input vector $\mathbf{u}(n)$ is drawn from a two-dimensional Gaussian process of zero mean and correlation matrix

$$\mathbf{R} = 10^{-4}\begin{bmatrix} 1 & -0.75 \\ -0.75 & 1 \end{bmatrix}$$

The model parameters are

$$a = 0.9998$$

$$\sigma_v^2 = 0.04$$

This completes the specification of the basic parameters of the experiment. For the auxiliary parameters, we have

$$c_1 = 2.7344 \times 10^{-4}$$

$$c_2 = 0.160 \times 10^{-4}$$

Finally, the frame of reference is

$$\delta = 0.01$$

To initialize the experiments, we set

$$E[\mathbf{w}_o(0)] = \mathbf{0}$$

Under the assumption of ergodicity, the instantaneous weight-error vector

$$\boldsymbol{\epsilon}(n) = \hat{\mathbf{w}}(n|n-1) - \mathbf{w}_o(n)$$

is measured by time-averaging over one simulation run of $N = 50{,}000$ iterations in the steady state (i.e., after all transients have essentially dissipated). In the simulations, this is taken to occur at the iteration index $n = 50{,}000$. The values of n and N so chosen can be justified by noting that plots of the simulated quantities as a function of N show no discernible change by that point.

Table 16.3 presents the simulation results for the two cases:

Case 1: $\mathbf{Q} = c_1\mathbf{R}$
Case 2: $\mathbf{Q} = c_2\mathbf{R}^{-1}$

In each case, the experimental values included in the table pertain to the following:

- Minimum mean-square deviation $\mathcal{D}_{\min}$ and minimum misadjustment $\mathcal{M}_{\min}$ produced by the standard RLS algorithm

TABLE 16.3 SIMULATION RESULTS FOR TWO DIFFERENT CASES

	Case 1 $\mathbf{Q} = c_1 \mathbf{R}$	Case 2 $\mathbf{Q} = c_2 \mathbf{R}^{-1}$
$\mathcal{D}_{\min}^{\text{RLS}}$	0.0105	0.0103
$\mathcal{D}_{\min}^{\text{LMS}}$	0.0071	0.0135
$\mathcal{D}^{\text{ERLS-1}}$	0.0069	0.0102
$\mathcal{M}_{\min}^{\text{RLS}}$	0.0842	0.0423
$\mathcal{M}_{\min}^{\text{LMS}}$	0.0698	0.0666
$\mathcal{M}^{\text{ERLS-1}}$	0.0673	0.0419

- Minimum mean-square deviation $\mathcal{D}_{\min}$ and minimum misadjustment $\mathcal{M}_{\min}$ produced by the LMS algorithm
- Mean-square deviation $\mathcal{D}$ and misadjustment $\mathcal{M}$ produced by the ERLS-1 algorithm.

The simulation results clearly demonstrate the superiority of the LMS algorithm over the standard RLS algorithm for Case 1, and vice versa for Case 2. Moreover, they show that the ERLS-1 algorithm performs better (though only marginally) than the optimal LMS/RLS algorithm in each case. Most likely, the marginal improvement is an artifact of the choice of experimental parameters; the choice makes both the relative mean-square weight deviation and relative mean-square misadjustment sufficiently small, so that differences between the performances of the algorithms are not easily discernible over what passes as normal simulation variance and numerical noise.

16.10 AUTOMATIC TUNING OF THE ADAPTATION CONSTANTS

Returning to the model of a time-varying system described in Fig. 16.1, by now we know how to calculate optimum values for the step-size parameter μ in the LMS algorithm using Eq. (16.32) or (16.36), and how to calculate optimum values for the exponential weighting factor λ in the RLS algorithm using Eq. (16.57) or (16.62). Irrespective of the optimality criterion used to assess the tracking performance of the adaptive filter employed to track the system, these calculations require knowledge of the correlation matrix $\mathbf{Q}$ of the process noise vector $\boldsymbol{\omega}(n)$ in the model of Fig. 16.1(a) and the correlation matrix $\mathbf{R}$ of the input vector $\mathbf{u}(n)$ in the model of Fig. 16.1(b). This observation prompts us to raise a basic question that lies at the heart of what adaptive filtering is all about:

- How is the optimum value of the step-size parameter μ in the LMS algorithm or that of the exponential weighting factor λ in the RLS algorithm to be chosen, when details of the underlying physical model of the system and its variability with time are not known?

In Benveniste et al. (1990), certain modifications to the LMS and RLS algorithms are proposed by superimposing adaptive schemes on their respective formulations for the purpose of tuning μ in the LMS algorithm and λ in the RLS algorithm. The practical validity of the idea described therein has been fully supported by (1) signal-processing applications on adaptive equalization and phase-locked loop presented in Brossier (1992), and (2) proof of convergence based on a fairly strong result rooted in stochastic approximation theory that is presented in Kushner and Yang (1995).

LMS Algorithm with Adaptive Gain

The purpose of the adaptive scheme suggested in Benveniste et al. (1990) and elaborated on in Kushner and Yang (1995) is to find an estimate for the particular value of the step-size parameter μ that minimizes the ensemble-averaged cost function

$$J(n) = \frac{1}{2} E[|e(n)|^2] \tag{16.99}$$

where $e(n)$ is the estimation error:

$$e(n) = d(n) - \mathbf{w}^H(n)\mathbf{u}(n) \tag{16.100}$$

Differentiating the cost function $J(n)$ with respect to the step-size parameter μ yields the scalar gradient

$$\begin{aligned} \nabla_\mu(n) &= \frac{\partial J(n)}{\partial \mu} \\ &= \frac{1}{2} E\left[\frac{\partial e(n)}{\partial \mu} e^*(n) + \frac{\partial e^*(n)}{\partial \mu} e(n)\right] \end{aligned} \tag{16.101}$$

From Eq. (16.100) we readily find that

$$\frac{\partial e(n)}{\partial \mu} = -\boldsymbol{\psi}^H(n)\,\mathbf{u}(n) \tag{16.102}$$

where the vector $\boldsymbol{\psi}(n)$ denotes the gradient of the tap-weight vector $\mathbf{w}(n)$ with respect to the step-size parameter μ:

$$\boldsymbol{\psi}(n) = \frac{\partial \mathbf{w}(n)}{\partial \mu} \tag{16.103}$$

Accordingly, we may redefine the scalar gradient $\nabla_\mu(n)$ as

$$\nabla_\mu(n) = -\frac{1}{2} E[\boldsymbol{\psi}^H(n)\mathbf{u}(n)e^*(n) + \mathbf{u}^H(n)\boldsymbol{\psi}(n)e(n)] \tag{16.104}$$

Let $\mu(n)$ and $\hat{\mathbf{w}}(n)$ denote the actual sequences of step-sizes and estimates of $\mathbf{w}(n)$, respectively, resulting from the operation of the adaptive scheme. The adaptation proceeds in an iterative manner. Computation of the estimate $\hat{\mathbf{w}}(n)$ follows the LMS algorithm described in Chapter 9. In a manner similar to that computation, we may formulate the recursion for updating $\mu(n)$ as follows:

$$\mu(n+1) = \mu(n) - \alpha\hat{\nabla}_{\mu}(n) \tag{16.105}$$

where α is a small, positive *learning-rate parameter*, and $\hat{\nabla}_{\mu}(n)$ is an estimate of the scalar gradient $\nabla_{\mu}(n)$.

Define $\hat{\boldsymbol{\psi}}(n) = \partial\hat{\mathbf{w}}(n)/\partial\mu(n)$ as the estimate of the derivative $\boldsymbol{\psi}(n)$. Then, on the basis of Eq. (16.104) we may formulate the *instantaneous estimate*

$$\begin{aligned}\hat{\nabla}_{\mu}(n) &= -\frac{1}{2}[\hat{\boldsymbol{\psi}}^H(n)\mathbf{u}(n)e^*(n) + \mathbf{u}^H(n)\hat{\boldsymbol{\psi}}(n)e(n)] \\ &= -\operatorname{Re}[\hat{\boldsymbol{\psi}}^H(n)\mathbf{u}(n)e^*(n)]\end{aligned} \tag{16.106}$$

where Re signifies the real-part operator, and $e(n)$ is defined by Eq. (16.100) with the estimate $\hat{\mathbf{w}}(n)$ used in place of $\mathbf{w}(n)$.

We are now ready to describe the two-step adaptive scheme for tuning the step-size parameter in the LMS algorithm:

1. Given the old value $\mu(n)$ of the step-size parameter, its updated value is computed using the recursion:

$$\mu(n+1) = \mu(n) + \alpha\operatorname{Re}[\hat{\boldsymbol{\psi}}^H(n)\mathbf{u}(n)e^*(n)] \tag{16.107}$$

2. Starting with the usual recursion for updating the tap-weight vector:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \mu(n)\mathbf{u}(n)e^*(n) \tag{16.108}$$

and differentiating it with respect to $\mu(n)$, we get the recursion for updating the estimate $\hat{\boldsymbol{\psi}}(n)$, as described here

$$\begin{aligned}\hat{\boldsymbol{\psi}}(n+1) &= \hat{\boldsymbol{\psi}}(n) + \mathbf{u}(n)e^*(n) + \mu(n)\mathbf{u}(n)\frac{\partial e^*(n)}{\partial\mu(n)} \\ &= \hat{\boldsymbol{\psi}}(n) + \mathbf{u}(n)e^*(n) - \mu(n)\mathbf{u}(n)\mathbf{u}^H(n)\hat{\boldsymbol{\psi}}(n)\end{aligned} \tag{16.109}$$

In the last line of Eq. (16.109), we have adapted the use of Eq. (16.102) for the problem at hand, with $\hat{\boldsymbol{\psi}}(n)$ used in place of $\boldsymbol{\psi}(n)$.

We may now summarize the *LMS algorithm with adaptive gain* as follows:

Starting with some initial values $\hat{\mathbf{w}}(0)$, $\mu(0)$, and $\hat{\boldsymbol{\psi}}(0)$, compute for $n > 0$:

$$\begin{aligned}e(n) &= d(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n) \\ \hat{\mathbf{w}}(n+1) &= \hat{\mathbf{w}}(n) + \mu(n)\mathbf{u}(n)e^*(n) \\ \mu(n+1) &= [\mu(n) + \alpha\operatorname{Re}[\hat{\boldsymbol{\psi}}^H(n)\mathbf{u}(n)e^*(n)]]_{\mu^-}^{\mu^+} \\ \hat{\boldsymbol{\psi}}(n+1) &= [\mathbf{I} - \mu(n)\mathbf{u}(n)\mathbf{u}^H(n)]\,\boldsymbol{\psi}(n) + \mathbf{u}(n)e^*(n)\end{aligned}$$

In the third line of this summary, the bracket with μ_- and μ_+ indicates *truncation*. According to simulations reported in Kushner and Yang (1995), the lower level of truncation, μ_-, appears to play a relatively insignificant role; it may be set equal to zero or some small number. On the other hand, the upper level of truncation, μ_+, is highly crucial for good behavior of the algorithm. Typically, the optimum value of the step-size parameter for a good tracking behavior is near the point of instability, so the assignment of too large a value to μ_+ may cause the LMS algorithm to become unstable. In any application, the user usually develops enough experience to see how to set μ_+.

RLS Algorithm with Adaptive Memory

Consider next the RLS algorithm equipped with an adaptive scheme for tuning the exponential weighting factor λ. In this case, the objective is to find the particular value of λ that optimizes the cost function

$$J'(n) = \frac{1}{2} E[|\xi(n)|^2] \tag{16.110}$$

where $\xi(n)$ is the *a priori* estimation error defined by

$$\xi(n) = d(n) - \hat{\mathbf{w}}^H(n-1)\mathbf{u}(n) \tag{16.111}$$

Differentiating the cost function $J'(n)$ with respect to λ yields

$$\begin{aligned} \nabla_\lambda(n) &= \frac{\partial J'(n)}{\partial \lambda} \\ &= \frac{1}{2} E\left[\frac{\partial \xi(n)}{\partial \lambda}\xi^*(n) + \frac{\partial \xi^*(n)}{\partial \lambda}\xi(n)\right] \end{aligned} \tag{16.112}$$

Define

$$\boldsymbol{\psi}(n) = \frac{\partial \mathbf{w}(n)}{\partial \lambda} \tag{16.113}$$

We may then, with the aid of Eq. (16.111), redefine the scalar gradient $\nabla_\lambda(n)$ as

$$\nabla_\lambda(n) = -\frac{1}{2} E[\boldsymbol{\psi}^H(n-1)\,\mathbf{u}(n)\xi^*(n) + \mathbf{u}^H(n)\,\boldsymbol{\psi}(n-1)\xi(n)] \tag{16.114}$$

The updating of the tap-weight vector in the RLS algorithm involves the gain vector $\mathbf{k}(n) = \mathbf{P}(n)\mathbf{u}(n)$, as shown by [see Eq. (13.25)]

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{P}(n)\mathbf{u}(n)\xi^*(n) \tag{16.115}$$

Let $\mathbf{S}(n)$ denote the derivative of the inverse correlation matrix $\mathbf{P}(n)$ with respect to λ:

$$\mathbf{S}(n) = \frac{\partial \mathbf{P}(n)}{\partial \lambda} \tag{16.116}$$

Then, using Eqs. (16.111), (16.115) and (16.116) in Eq. (16.113) yields

$$\boldsymbol{\psi}(n) = [\mathbf{I} - \mathbf{k}(n)\mathbf{u}^H(n)]\,\boldsymbol{\psi}(n-1) + \mathbf{S}(n)\mathbf{u}(n)\xi^*(n) \tag{16.117}$$

For the recursion to compute $\mathbf{S}$, we first use Eqs. (13.18) and (13.19) to write

$$\mathbf{P}(n) = \lambda^{-1}\,\mathbf{P}(n-1) - \frac{\lambda^{-2}\,\mathbf{P}(n-1)\,\mathbf{u}(n)\mathbf{u}^H(n)\mathbf{P}(n-1)}{1 + \lambda^{-1}\,\mathbf{u}^H(n)\,\mathbf{P}(n-1)\mathbf{u}(n)} \tag{16.118}$$

Hence, differentiating Eq. (16.118) with respect to λ and then collecting terms, we get

$$\begin{aligned}\mathbf{S}(n) = {} & \lambda^{-1}\,[\mathbf{I} - \mathbf{k}(n)\mathbf{u}^H(n)]\,\mathbf{S}(n-1)\,[\mathbf{I} - \mathbf{u}(n)\,\mathbf{k}^H(n)] \\ & + \lambda^{-1}\mathbf{k}(n)\mathbf{k}^H(n) - \lambda^{-1}\mathbf{P}(n)\end{aligned} \tag{16.119}$$

We are now ready to formulate the *RLS algorithm with adaptive memory*. Let $\lambda(n)$, $\hat{\mathbf{w}}(n)$, and $\hat{\boldsymbol{\psi}}(n)$ denote the actual values of the exponential weighting factor, tap-weight vector $\mathbf{w}(n)$, and gradient $\boldsymbol{\psi}(n)$ computed by the algorithm at iteration n. Then, using the instantaneous estimate $-\mathrm{Re}[\boldsymbol{\psi}^H(n-1)\mathbf{u}(n)\xi^*(n)]$ for the scalar gradient $\nabla_\lambda(n)$, based on Eq. (16.114) we may adaptively compute the exponential weighting factor using the recursion:

$$\begin{aligned}\lambda(n) &= \lambda(n-1) - \alpha\hat{\nabla}_\lambda(n) \\ &= \lambda(n-1) + \alpha\,\mathrm{Re}[\boldsymbol{\psi}^H(n-1)\mathbf{u}(n)\xi^*(n)]\end{aligned} \tag{16.120}$$

where α is a small, positive learning-rate parameter. Thus, incorporating this recursion into the standard RLS algorithm, we may summarize the RLS algorithm with adaptive memory as follows:

Starting with the initial values $\hat{\mathbf{w}}(0)$, $\mathbf{P}(0)$, $\lambda(0)$, $\mathbf{S}(0)$, and $\hat{\boldsymbol{\psi}}(0)$, compute for $n > 0$:

$$\mathbf{k}(n) = \frac{\lambda^{-1}(n-1)\mathbf{P}(n-1)\mathbf{u}(n)}{1 + \lambda^{-1}(n-1)\mathbf{u}^H(n)\mathbf{P}(n-1)\mathbf{u}(n)}$$

$$\xi(n) = d(n) - \hat{\mathbf{w}}^H(n-1)\mathbf{u}(n)$$

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{k}(n)\xi^*(n)$$

$$\mathbf{P}(n) = \lambda^{-1}(n-1)\mathbf{P}(n-1) - \lambda^{-1}(n-1)\mathbf{k}(n)\mathbf{u}^H(n)\mathbf{P}(n-1)$$

$$\lambda(n) = [\lambda(n-1) + \alpha\,\mathrm{Re}[\hat{\boldsymbol{\psi}}^H(n-1)\mathbf{u}(n)\xi^*(n)]]_{\lambda_-}^{\lambda_+}$$

$$\begin{aligned}\mathbf{S}(n) = {} & \lambda^{-1}(n)[\mathbf{I} - \mathbf{k}(n)\mathbf{u}^H(n)]\,\mathbf{S}(n-1)\,[\mathbf{I} - \mathbf{u}(n)\mathbf{k}^H(n)] \\ & + \lambda^{-1}(n)\mathbf{k}(n)\mathbf{k}^H(n) - \lambda^{-1}(n)\mathbf{P}(n)\end{aligned}$$

$$\hat{\boldsymbol{\psi}}(n) = [\mathbf{I} - \mathbf{k}(n)\mathbf{u}^H(n)]\,\hat{\boldsymbol{\psi}}(n-1) + \mathbf{S}(n)\mathbf{u}(n)\xi^*(n)$$

As with the LMS algorithm with adaptive gain, the bracket with λ_- and λ_+ in the fifth line of this summary indicates truncation. The upper level of truncation, λ_+, may be set close to (but less than) unity. The lower level of truncation, λ_-, plays a more crucial role, and its value may be determined by the user through experimentation.

16.11 SUMMARY AND DISCUSSION

In this chapter we studied the tracking performance of adaptive filters when operating in a nonstationary environment. First and foremost, tracking is *problem-specific*. Thus, the tracking performance of an adaptive filter, used in system identification is quite different from that in channel equalization or the adaptive recovery of a desired signal in additive noise.

To assess the tracking capability of an adaptive filter, we may use the mean-square deviation $\mathcal{D}(n)$ or the misadjustment $\mathcal{M}(n)$. These two figures of merit highlight the tracking performance of the adaptive filter in their own individual ways. The important point to note is that in either case, we may identify two contributions: one being representative of the stationary case, and the other being attributed to nonstationarity of the environment.

Based on the mean-square deviation and misadjustment, we find that, in general, the LMS algorithm exhibits a more robust tracking behavior than the RLS algorithm. Therefore, it should not be surprising to find that, in data transmission over time-varying communication channels, the LMS algorithm is preferred over the RLS algorithm, not only because of its simplicity but also because of its better tracking capability.

This latter remark pertains to the conventional formulation of the RLS algorithm, as described in Chapter 13. However, we are not restricted exclusively to this version of the RLS algorithm. Rather, in light of the one-to-one correspondences that exist between RLS variables and Kalman variables, we may draw upon the vast literature on Kalman filter theory to build other versions of the RLS algorithm to suit the application. In particular, we may mention two possible routes:

1. The underlying state-space model of the RLS algorithm is formulated as in Eqs. (16.90) and (16.91), with the process noise vector $\mathbf{v}_1(n)$ included in the model to account for the nonstationary behavior of the environment. Such an approach is well suited for a system identification problem with a Markovian description, as in Eq. (16.1).
2. The nonstationarity is accounted for by including in the state-space model of the RLS algorithm a transition matrix $\mathbf{F}(n+1, n)$ that is not a constant or even unknown. The state-space model of the RLS algorithm is then formulated in nonlinear terms, and the application of extended Kalman filter formalism is invoked. This latter approach is, for example, appropriate for the tracking of a chirped signal in additive noise.

The conclusion to be drawn from this discussion is that whatever prior knowledge is available about the task at hand, it should be exploited, so as to *minimize the mismatch* between the state-space model of the RLS algorithm and the mathematical model for the problem of interest, and thereby improve the tracking performance of the RLS algorithm in a nonstationary environment.

PROBLEMS

1. In qualitative terms, describe how the error-performance surface of an adaptive equalizer for a time-varying communication channel differs from that of an adaptive equalizer for a communication channel of fixed characteristics.
2. In prediction, the present value of a signal constitutes the desired response and a finite set of its past values constitutes the input vector. Describe how the error-performance surface of an adaptive predictor for a nonstationary process differs from that for a stationary process.
3. The weight-error vector $\boldsymbol{\epsilon}(n)$ may be expressed as the sum of the weight vector noise $\boldsymbol{\epsilon}_1(n)$ and weight vector lag $\boldsymbol{\epsilon}_2(n)$. Show that

$$E[\boldsymbol{\epsilon}_1^H(n)\boldsymbol{\epsilon}_2(n)] = E[\boldsymbol{\epsilon}_2^H(n)\boldsymbol{\epsilon}_1(n)] = 0$$

   Under the assumption that $\hat{\mathbf{w}}(n)$ and $\mathbf{w}_o(n)$ are statistically independent in Fig. 16.1, show that

$$E[\|\boldsymbol{\epsilon}(n)\|^2] = E[\|\boldsymbol{\epsilon}_1(n)\|^2] + E[\|\boldsymbol{\epsilon}_2(n)\|^2]$$

4. Continuing with Problem 3, use the independence assumption to show that

$$E[\boldsymbol{\epsilon}_1^H(n)\mathbf{u}(n)\mathbf{u}^H(n)\boldsymbol{\epsilon}_1(n)] = \text{tr}[\mathbf{R}\mathbf{K}_1(n)]$$

$$E[\boldsymbol{\epsilon}_2^H(n)\mathbf{u}(n)\mathbf{u}^H(n)\boldsymbol{\epsilon}_2(n)] = \text{tr}[\mathbf{R}\mathbf{K}_2(n)]$$

$$E[\boldsymbol{\epsilon}_1^H(n)\mathbf{u}(n)\mathbf{u}^H(n)\boldsymbol{\epsilon}_2^H(n)] = E[\boldsymbol{\epsilon}_2^H(n)\mathbf{u}(n)\mathbf{u}^H(n)\boldsymbol{\epsilon}_2(n)] = 0$$

   where $\mathbf{u}(n)$ is the input vector assumed to be of zero mean, $\mathbf{R}$ is the correlation matrix of $\mathbf{u}(n)$, and $\mathbf{K}_1(n)$ and $\mathbf{K}_2(n)$ are the correlation matrices of $\boldsymbol{\epsilon}_1(n)$ and $\boldsymbol{\epsilon}_2(n)$, respectively. How is the correlation matrix $\mathbf{K}(n)$ of $\boldsymbol{\epsilon}(n)$ related to $\mathbf{K}_1(n)$ and $\mathbf{K}_2(n)$?
5. Given the vectors $\mathbf{x}$ and $\mathbf{y}$, both assumed to have the same dimension, the *Cauchy–Schwarz inequality* states that

$$|\mathbf{x}^H\mathbf{y}|^2 \leq \|\mathbf{x}\|^2 \|\mathbf{y}\|^2$$

   Applying this inequality to the cross-diagonal terms of the 2-by-2 array in Table 16.1, derive the results presented in Eqs. (16.71) and (16.72).
6. Continuing with the results presented in Table 16.2 for an adaptive filter with $M = 2$, do the following:
   (a) For the LMS algorithm, determine the minimum mean-square deviation $\mathcal{D}_{\text{min}}$ and the minimum misadjustment $\mathcal{M}_{\text{min}}$, and the corresponding optimum values of the step-size parameter μ.
   (b) For the RLS algorithm, determine the minimum mean-square deviation $\mathcal{D}_{\text{min}}$ and the minimum misadjustment $\mathcal{M}_{\text{min}}$, and the corresponding optimum values of the exponential weighting factor λ.

   Hence, verify the entries presented in Table 16.2.

CHAPTER

17

Finite-Precision Effects

A study of adaptive filters would be incomplete without some discussion of the effects of *quantization* or *round-off errors* that arise when they are implemented digitally.

The theory of adaptive filtering developed in previous chapters assumes the use of an *analog model* (i.e., infinite precision) for the samples of input data as well as the internal algorithmic calculations. This assumption is made in order to take advantage of well-understood continuous mathematics. Adaptive filter theory, however, cannot be applied to the construction of an adaptive filter directly; rather it provides an *idealized framework* for such a construction. In particular, in a *digital* implementation of an adaptive filtering algorithm as encountered in practice, the input data and internal calculations are all quantized to a *finite precision* that is determined by design and cost considerations. Consequently, the quantization process has the effect of causing the performance of a digital implementation of the algorithm to deviate from its theoretical value. The nature of this deviation is influenced by a combination of factors:

- The type of design details of the adaptive filtering algorithm employed
- The degree of ill-conditioning (i.e., the eigenvalue spread) in the underlying correlation matrix that characterizes the input data
- The form of numerical computation (fixed-point or floating-point) employed

It is important for us to understand the numerical properties of adaptive filtering algorithms, as it would obviously help us in meeting design specifications. Moreover, the

cost of a digital implementation of an algorithm is influenced by the *number of bits* (i.e., precision) available for performing the numerical computations associated with the algorithm. Generally speaking, the cost of implementation increases with the number of bits employed. There is therefore practical motivation for using the minimum number of bits possible.

We begin our study of the numerical properties of adaptive filtering algorithms by examining the sources of quantization error and the related issues of numerical stability and accuracy.

17.1 QUANTIZATION ERRORS

In a digital implementation of an adaptive filter, there are essentially two sources of quantization error to be considered as described here.

1. *Analog-to-digital conversion.* Given that the input data are in analog form, we may use an analog-to-digital converter for their numerical representation. For our present discussion, we assume a quantization process with a *uniform step size* δ and a set of *quantizing levels* positioned at $0, \pm\delta \pm 2\delta, \cdots$. Figure 17.1 illustrates the input-output characteristic of a typical uniform quantizer. Consider a particular sample at the quantizer input, with an amplitude that lies in the range $i\delta - (\delta/2)$ to $i\delta + (\delta/2)$, where i is an integer (positive or negative, including zero) and $i\delta$ defines the *quantizer output*. The quantization process thus described introduces a region of uncertainty of width δ, centered on $i\delta$. Let η denote the quantization error. Correspondingly, the quantizer input is $i\delta + \eta$, where η is bounded as $-(\delta/2) \leq \eta \leq (\delta/2)$. When the quantization is fine enough (say, the number of quantizing levels is 64 or more), and the signal spectrum is sufficiently rich, the distortion produced by the quantizing process may be modeled as an additive independent source of white noise with zero mean and variance determined by the quantizer step size δ (Gray, 1990). It is customary to assume that the quantization error η is *uniformly distributed* over the range $-\delta/2$ to $\delta/2$. The variance of the quantization error is therefore given by

$$\begin{aligned}\sigma^2 &= \int_{-\delta/2}^{\delta/2} \frac{1}{\delta}\eta^2 \, d\eta \\ &= \frac{\delta^2}{12}\end{aligned} \tag{17.1}$$

We assume that the quantizer input is properly scaled, so that it lies inside the interval $(-1, +1]$. With each quantizing level represented by B bits plus sign, the quantizer step size is

$$\delta = 2^{-B} \tag{17.2}$$

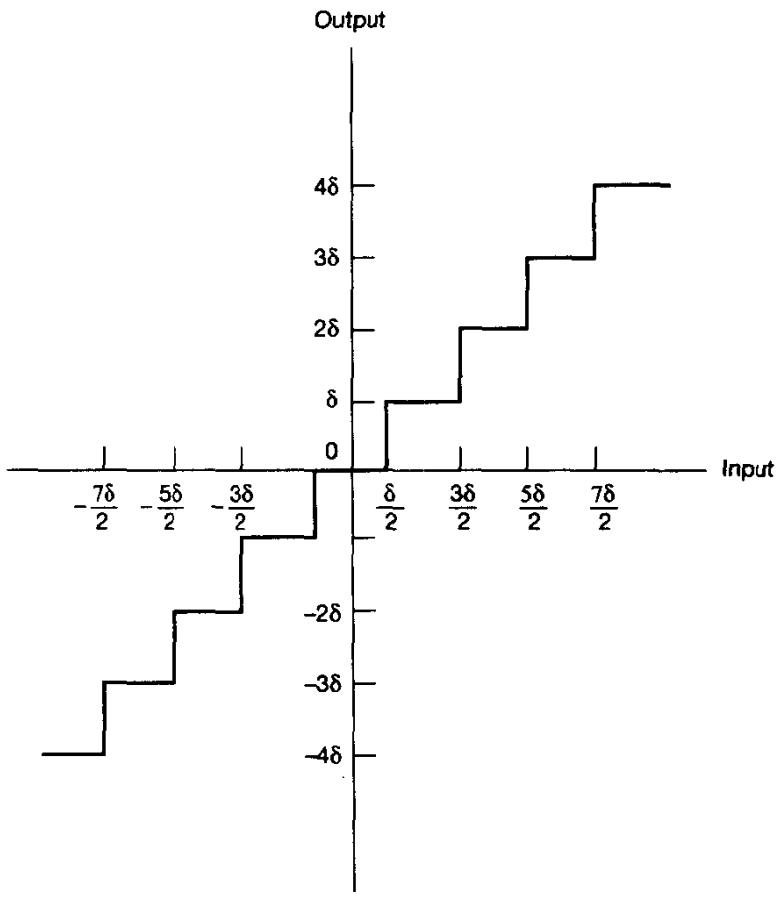


Figure 17.1 Input–output characteristic of a uniform quantizer.

Substituting Eq. (17.2) in (17.1), we find that the quantization error resulting from the digital representation of input analog data has the variance

$$\sigma^2 = \frac{2^{-2B}}{12} \tag{17.3}$$

2. *Finite word-length arithmetic.* In a digital machine, a finite word length is commonly used to store the result of internal arithmetic calculations. Assuming that *no* overflow takes place during the course of computation, additions do not introduce error (if fixed-point arithmetic is used), whereas each multiplication introduces an error after the product is quantized. The statistical characterization of finite word-length arithmetic errors may be quite different from that of analog-to-digital conversion errors. Finite word-length arithmetic errors may have a *nonzero mean*, which results from either rounding off or truncating the output of a multiplier so as to match the prescribed word length.

The presence of finite word-length arithmetic raises serious concern in the digital implementation of an adaptive filter, particularly when the tap weights (coefficients) of the filter are updated on a continuous basis. The digital version of the filter exhibits a specific *response* or *propagation* to such errors, causing its performance to deviate from the ideal (i.e., infinite-precision) form of the filter. Indeed, it is possible for the deviation to be of a catastrophic nature in the sense that the errors resulting from the use of finite-precision arithmetic may accumulate without bound. If such a situation is allowed to persist, the filter is ultimately driven into an overflow condition, and the algorithm is said to be *numerically unstable*. Clearly, for an adaptive filter to be of practical value, it has to be numerically stable. An adaptive filter is said to be *numerically stable* if the use of finite-precision arithmetic results in deviations from the infinite-precision form of the filter that are *bounded*. It is important to recognize that numerical stability is an inherent characteristic of an adaptive filter. In other words, if an adaptive filter is numerically unstable, then increasing the number of bits used in a digital implementation of the filter will not change the stability condition of that implementation.

Another issue that requires attention in a digital implementation of an adaptive filter is that of *numerical accuracy*. Unlike numerical stability, however, the numerical accuracy of an adaptive filter is determined by the number of bits used to implement the internal calculations of the filter. The larger the number of bits used, the smaller the deviation from ideal performance, and the more accurate therefore would be the digital implementation of the filter. In practical terms, it is only meaningful to speak of the numerical accuracy of an adaptive filter if it is numerically stable.

For the remainder of this chapter, we discuss the numerical properties of adaptive filtering algorithms and related issues. We begin with the LMS algorithm and then move on to RLS adaptive filtering algorithms, presented in the same order as in previous chapters of the book.

17.2 LEAST-MEAN-SQUARE ALGORITHM

In order to simplify the discussion of finite-precision effects on the performance of the LMS algorithm,[1] we will depart from the practice followed in previous chapters, and assume that the input data and therefore the filter coefficients are all *real valued*. This

[1]The first treatment of finite-precision effects in the LMS algorithm was presented by Gitlin et al. (1973). Subsequently, more detailed treatments of these effects were presented by Weiss and Mitra (1979), Caraiscos and Liu (1984), and Alexander (1987). The paper by Caraiscos and Liu considers steady-state conditions, whereas the paper by Alexander is broader in scope, in that it considers transient conditions. The problem of finite-precision effects in the LMS algorithm is also discussed in Cioffi (1987) and Sherwood and Bershad (1987). Another problem encountered in the practical use of the LMS algorithm is that of parameter drift, which is discussed in detail in Sethares et al. (1986). The material presented in this section is very much influenced by the contents of these papers. In our presentation, we assume the use of *fixed-point arithmetic*. Error analysis of the LMS algorithm for *floating-point arithmetic* is discussed in Caraiscos and Liu (1984).

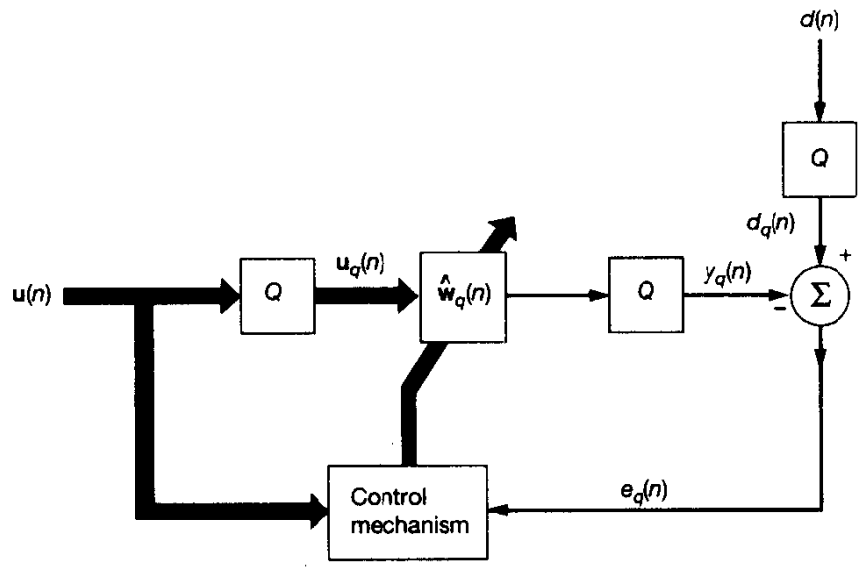


Figure 17.2 Block diagram representation of the finite-precision form of the LMS algorithm.

assumption, made merely for convenience of presentation, will in no way affect the validity of the findings presented in this section.

A block diagram of the *finite-precision least-mean-square (LMS) algorithm* is depicted in Fig. 17.2. Each of the blocks (operators) labeled Q represents a *quantizer*. Each one introduces a *quantization*, or *round-off error* of its own. Specifically, we may describe the input-output relations of the quantizers operating in Fig. 17.2 as follows:

1. For the input quantizer connected to $\mathbf{u}(n)$ we have

$$\begin{aligned} \mathbf{u}_q(n) &= Q[\mathbf{u}(n)] \\ &= \mathbf{u}(n) + \boldsymbol{\eta}_u(n) \end{aligned} \tag{17.4}$$

   where $\boldsymbol{\eta}_u(n)$ is the *input quantization error vector.*

2. For the quantizer connected to the desired response $d(n)$, we have

$$\begin{aligned} d_q(n) &= Q[d(n)] \\ &= d(n) + \eta_d(n) \end{aligned} \tag{17.5}$$

   where $\eta_d(n)$ is the *desired response quantization error.*

3. For the quantized tap-weight vector $\hat{\mathbf{w}}_q(n)$, we write

$$\begin{aligned} \hat{\mathbf{w}}_q(n) &= Q[\hat{\mathbf{w}}(n)] \\ &= \hat{\mathbf{w}}(n) + \Delta\hat{\mathbf{w}}(n) \end{aligned} \tag{17.6}$$

   where $\hat{\mathbf{w}}(n)$ is the tap-weight vector in the infinite-precision LMS algorithm, and $\Delta\hat{\mathbf{w}}(n)$ is the *tap-weight error vector* resulting from quantization.

4. For the quantizer connected to the output of the transversal filter represented by the quantized tap-weight vector $\hat{\mathbf{w}}_q(n)$, we write

$$\begin{aligned} y_q(n) &= Q[\mathbf{u}_q^T(n)\hat{\mathbf{w}}_q(n)] \\ &= \mathbf{u}_q^T(n)\hat{\mathbf{w}}_q(n) + \eta_y(n) \end{aligned} \tag{17.7}$$

where $\eta_y(n)$ is the *filtered output quantization error*.

The finite-precision LMS algorithm is described by the following pair of relations:

$$e_q(n) = d_q(n) - y_q(n) \tag{17.8}$$

$$\hat{\mathbf{w}}_q(n+1) = \hat{\mathbf{w}}_q(n) + Q[\mu e_q(n)\mathbf{u}_q(n)] \tag{17.9}$$

where $y_q(n)$ is itself defined in Eq. (17.7). The quantizing operation indicated on the right-hand side of Eq. (17.9) is not shown explicitly in Fig. 17.2; nevertheless, it is basic to the operation of the finite-precision LMS algorithm. The use of Eq. (17.9) has the following practical implication. The product $\mu e_q(n)\mathbf{u}_q(n)$, representing a scaled version of the gradient vector estimate, is quantized *before* addition to the contents of the *tap-weight accumulator*. Because of hardware constraints, this form of digital implementation is preferred to the alternative method of operating the tap-weight accumulator in double precision and then quantizing the tap weight to single precision at the accumulator output.

In a statistical analysis of the finite-precision LMS algorithm, it is customary to make the following assumptions:

1. The input data are properly scaled so as to *prevent overflow* of the elements of the quantized tap-weight vector $\hat{\mathbf{w}}_q(n)$ and the quantized output $y_q(n)$ during the filtering operation.
2. Each data sample is represented by B_D bits plus sign, and each tap weight is represented by B_W bits plus sign. Thus, the quantization error associated with a B_D-plus-sign bit number (i.e., data sample) has the variance

$$\sigma_D^2 = \frac{2^{-2B_D}}{12} \tag{17.10}$$

Similarly, the quantization error associated with a B_W-plus-sign bit number (i.e., tap weight) has the variance

$$\sigma_W^2 = \frac{2^{-2B_W}}{12} \tag{17.11}$$

3. The elements of the input quantization error vector $\boldsymbol{\eta}_u(n)$ and the desired response quantization error $\eta_d(n)$ are *white-noise* sequences, independent of the signals and from each other. Moreover, they have zero mean and variance σ_D^2.
4. The output quantization error $\eta_y(n)$ is a white-noise sequence, independent of the input signals and other quantization errors. It has a mean of zero and a variance

equal to $c\sigma_D^2$, where c is a constant that depends on the way in which the inner product $\mathbf{u}_q^T(n)\hat{\mathbf{w}}_q(n)$ is computed. If the individual scalar products in $\mathbf{u}_q^T(n)\hat{\mathbf{w}}_q(n)$ are all computed without quantization, then summed, and the final result is quantized in B_D bits plus sign, the constant c is unity and the variance of $\eta_y(n)$ is σ_D^2 as defined in Eq. (17.10). If, on the other hand, the individual scalar products in $\mathbf{u}^T(n)\hat{\mathbf{w}}_q(n)$ are quantized and then summed, the constant c is M and the variance of $\eta_y(n)$ is $M\sigma_D^2$, where M is the number of taps in the transversal filter implementation of the LMS algorithm.

5. The independence theory of Section 9.4, dealing with the infinite-precision LMS algorithm, is invoked.

Total Output Mean-squared Error

The filtered output $y_q(n)$, produced by the finite-precision LMS algorithm, presents a *quantized estimate* of the desired response. The *total output error* is therefore equal to the difference $d(n) - y_q(n)$. Using Eq. (17.7), we may therefore express this error as

$$\begin{aligned} e_{\text{total}}(n) &= d(n) - y_q(n) \\ &= d(n) - \mathbf{u}_q^T(n)\hat{\mathbf{w}}_q(n) - \eta_y(n) \end{aligned} \tag{17.12}$$

Substituting Eqs. (17.4) and (17.6) in Eq. (17.12), and ignoring all quantization error terms higher than first order, we get

$$e_{\text{total}}(n) = [d(n) - \mathbf{u}^T(n)\hat{\mathbf{w}}(n)] - [\Delta\hat{\mathbf{w}}^T(n)\mathbf{u}(n) + \boldsymbol{\eta}_u^T(n)\hat{\mathbf{w}}(n) + \eta_y(n)] \tag{17.13}$$

The term inside the first set of square brackets on the right-hand side of Eq. (17.13) is the estimation error $e(n)$ in the infinite-precision LMS algorithm. The term inside the second set of square brackets is entirely due to quantization errors in the finite-precision LMS algorithm. Because of assumptions 3 and 4 (i.e., the quantization errors $\boldsymbol{\eta}_u$ and η_y are independent of the input signals and of each other), the quantization error-related terms $\Delta\hat{\mathbf{w}}^T(n)\mathbf{u}(n)$, and $\eta_y(n)$ are uncorrelated with each other. Basically, for the same reason, the infinite-precision estimation error $e(n)$ is uncorrelated with both $\boldsymbol{\eta}_u^T(n)\hat{\mathbf{w}}(n)$ and $\eta_y(n)$. By invoking the independence assumption of Chapter 9, we may write

$$E[e(n)\Delta\hat{\mathbf{w}}^T(n)\mathbf{u}(n)] = E[\Delta\hat{\mathbf{w}}^T(n)]E[e(n)\mathbf{u}(n)]$$

Moreover, by invoking this same independence assumption, we may show that the expectation $E[\Delta\hat{\mathbf{w}}(n)]$ is zero (see Problem 2). Hence, $e(n)$ and $\Delta\hat{\mathbf{w}}^T(n)u(n)$ are also uncorrelated. In other words, the infinite-precision estimation error $e(n)$ is uncorrelated with all the three quantization-error-related terms, $\Delta\hat{\mathbf{w}}^T(n)\mathbf{u}(n)$, $\boldsymbol{\eta}_u^T(n)\hat{\mathbf{w}}(n)$, and $\eta_y(n)$ in Eq. (17.13).

Using these observations, and assuming that the step-size parameter μ is small, it is shown in Caraiscos and Liu (1984) that the *total output mean-squared error* produced in the finite-precision algorithm has the following *steady-state* structure:

$$E[e_{\text{total}}^2(n)] = J_{\min}(1 + \mathcal{M}) + \xi_1(\sigma_w^2, \mu) + \xi_2(\sigma_D^2) \tag{17.14}$$

The first term $J_{\min}(1 + \mathcal{M})$ on the right-hand side of Eq. (17.14) is the mean-squared error of the infinite-precision LMS algorithm. In particular, $J_{\min}$ is the *minimum mean-squared*

error of the optimum Wiener filter, and $\mathcal{M}$ is the *misadjustment* of the infinite-precision LMS algorithm. The second term $\xi_1(\sigma_w^2, \mu)$ arises because of the error $\Delta\hat{\mathbf{w}}(n)$ in the quantized tap-weight vector $\hat{\mathbf{w}}_q(n)$. This contribution to the total output mean-squared error is *inversely proportional to the step-size parameter* μ. The third term $\xi_2(\sigma_D^2)$ arises because of two quantization errors: the error $\boldsymbol{\eta}_u(n)$ in the quantized input vector $\mathbf{u}_q(n)$ and the error $\eta_y(n)$ in the quantized filter output $y_q(n)$. However, unlike $\xi_1(\sigma_w^2, \mu)$, this final contribution to the total output mean-squared error is, to a first order of approximation, independent of the step-size parameter μ.

From the infinite-precision theory of the LMS algorithm presented in Chapter 9, we know that decreasing μ reduces the misadjustment $\mathcal{M}$ and thus leads to an improved performance of the algorithm. In contrast, the inverse dependence of the contribution $\xi_1(\sigma_w^2, \mu)$ on μ in Eq. (17.14) indicates that decreasing μ has the effect of increasing the deviation from infinite-precision performance. In practice, therefore, the step-size parameter μ may only be decreased to a level at which the degrading effects of quantization errors in the tap weights of the finite-precision LMS algorithm become significant.

Since the misadjustment $\mathcal{M}$ decreases with μ, and the contribution $\xi_1(\sigma_w^2, \mu)$ increases with reduced μ, we may (in theory) find an optimum value of μ for which the total output mean-squared error in Eq. (17.14) is minimized. However, it turns out that this minimization results in an optimum value μ_o for the step-size parameter μ that is too small to be of practical value. In other words, it does not permit the LMS algorithm to converge completely. Indeed, Eq. (17.14) for calculating the total output mean-squared error is valid only for a μ that is well in excess of μ_o. Such a choice of μ is necessary so as to prevent the occurrence of a phenomenon known as stalling, described later in the section.

Deviations during the Convergence Period

Equation (17.14) describes the general structure of the total output mean-squared error of the finite-precision LMS algorithm, assuming that the algorithm has reached steady state. During the convergence period of the algorithm, however, the situation is more complicated.

A detailed treatment of the transient adaptation properties of the finite-precision LMS algorithm is presented in Alexander (1987). In particular, a general formula is derived for the tap-weight misadjustment, or *perturbation*, of the finite-precision LMS algorithm, which is measured with respect to the tap-weight solution computed from the infinite-precision form of the algorithm. The tap-weight misadjustment is defined by

$$\mathcal{W}(n) = E[\Delta\hat{\mathbf{w}}^T(n)\Delta\hat{\mathbf{w}}(n)] \tag{17.15}$$

where the *tap-weight error vector* $\Delta\hat{\mathbf{w}}(n)$ is itself defined by [see Eq. (17.6)]

$$\Delta\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}_q(n) - \hat{\mathbf{w}}(n) \tag{17.16}$$

The tap-weight vectors $\hat{\mathbf{w}}_q(n)$ and $\hat{\mathbf{w}}(n)$ refer to the finite-precision and infinite-precision forms of the LMS algorithm, respectively. To determine $\mathcal{W}(n)$, the weight up-date equation (17.9) is written as

$$\hat{\mathbf{w}}_q(n+1) = \hat{\mathbf{w}}_q(n) + \mu e_q(n)\mathbf{u}_q(n) + \boldsymbol{\eta}_w(n) \tag{17.17}$$

where $\boldsymbol{\eta}_w(n)$ is the *gradient quantization error vector*; it results from quantizing the product $\mu e_q(n)\mathbf{u}_q(n)$ that represents a scaled version of the gradient vector estimate. The individual elements of $\boldsymbol{\eta}_w(n)$ are assumed to be uncorrelated in time and with each other, and assumed to have a common variance σ_w^2. For this assumption to be valid, the step-size parameter μ must be large enough to prevent the stalling phenomenon from occurring; this phenomenon is described later in the section.

Applying an orthogonal transformation to $\hat{\mathbf{w}}_q(n)$ in Eq. (17.17) in a manner similar to that described in Section 5.6, the propagation characteristics of the tap-weight misadjustment $\mathcal{W}(n)$ during adaptation and steady state may be studied. Using such an approach, Alexander (1987) has derived some important theoretical results, supported by computer simulation. These results are summarized here.

1. The tap weights in the LMS algorithm are the *most sensitive* of all parameters to quantization. For the case of *uncorrelated input data*, the variance σ_w^2 [that enters the statistical characterization of the tap-weight update equation (17.17)] is proportional to the reciprocal of the product $r(0)\mu$, where $r(0)$ is the average input power and μ is the step-size parameter. For the case of *correlated input data*, the variance σ_w^2 is proportional to the reciprocal of $\mu\lambda_{\min}$, where $\lambda_{\min}$ is the smallest eigenvalue of the correlation matrix $\mathbf{R}$ of the input data vector $\mathbf{u}(n)$.
2. For uncorrelated input data, the adaptation time constants of the tap-weight misadjustment $\mathcal{W}(n)$ are heavily dependent on the step-size parameter μ.
3. For correlated input data, the adaptation time constants of $\mathcal{W}(n)$ are heavily dependent on the interaction between μ and the minimum eigenvalue $\lambda_{\min}$.

From a design point of view, it is thus important to recognize that the step-size parameter μ cannot be chosen too small; this is in spite of the infinite-precision theory of the LMS algorithm that advocates a small value for μ. Moreover, the more ill-conditioned the input process $u(n)$ is, the more pronounced the finite-precision effects in a digital implementation of the LMS algorithm will be.

Leaky LMS Algorithm

To further stabilize the digital implementation of the LMS algorithm, we may use a technique known as *leakage*.[2] Basically, leakage prevents the occurrence of overflow in a limited-precision environment by providing a compromise between minimizing the mean-squared error and containing the energy in the impulse response of the adaptive filter. However, the prevention of overflow is attained at the expense of an increase in hardware cost and at the expense of a degradation in performance compared to the infinite-precision form of the conventional LMS algorithm.

[2]Leakage may be viewed as a technique for increasing algorithm robustness (Ioannou and Kokotovic, 1983; Ioannou, 1990). For a historical account of the leakage technique in the context of adaptive filtering, see Cioffi (1987). For discussions of the leakage LMS algorithm, see Widrow and Stearns (1985) and Cioffi (1987).

In the leaky LMS algorithm, the cost function

$$J(n) = e^2(n) + \alpha\|\hat{\mathbf{w}}(n)\|^2 \tag{17.18}$$

is minimized with respect to the tap-weight vector $\hat{\mathbf{w}}(n)$, where α is a positive control parameter. The first term on the right-hand side of Eq. (17.18) is the squared estimation error, and the second term is the energy in the tap-weight vector $\hat{\mathbf{w}}(n)$. The minimization described herein (for real data) yields the following time update for the tap-weight vector (see Problem 7, Chapter 9)

$$\hat{\mathbf{w}}(n+1) = (1 - \mu\alpha)\hat{\mathbf{w}}(n) + \mu e(n)\mathbf{u}(n) \tag{17.19}$$

where α is a constant that satisfies the condition

$$0 \leq \alpha < \frac{1}{\mu}$$

Except for the *leakage factor* $(1 - \mu\alpha)$ associated with the first term on the right-hand side of Eq. (17.19), the algorithm is of the same mathematical form as the conventional LMS algorithm.

Note that the inclusion of the leakage factor $(1 - \mu\alpha)$ in Eq. (17.19) has the equivalent effect of adding a white-noise sequence of zero mean and variance α to the input process $u(n)$. This suggests another method for stabilizing a digital implementation of the LMS algorithm. Specifically, a relatively weak white-noise sequence (of variance α), known as *dither*, is added to the input process $u(n)$, and samples of the combination are then used as tap inputs (Werner, 1983).

Stalling Phenomenon

There is another phenomenon, known as the *stalling* or *lock-up phenomenon*, not evident from Eq. (17.14), which may arise in a digital implementation of the LMS algorithm. This phenomenon occurs when the gradient estimate is *not* sufficiently noisy. To be specific, a digital implementation of the LMS algorithm *stops adapting* or *stalls*, whenever the correction term $\mu e_q(n)u_q(n - i)$ for the ith tap weight in the update equation (17.9) is smaller in magnitude than the *least significant bit (LSB)* of the tap weight, as shown by (Gitlin et al., 1973)

$$|\mu e_q(n_0)u_q(n_0 - i)| \leq \text{LSB} \tag{17.20}$$

Here, n_0 is the time at which the ith tap weight stops adapting. Suppose that the condition of Eq. (17.20) is first satisfied for the ith tap eight. To a first order of approximation, we may replace $u_q(n_0 - i)$ by its *root-mean-square* (rms) value, A_{rms}. Accordingly, using this value in Eq. (17.20), we get the following relation for the rms value of the quantized estimation error when adaptation in the digitally implemented LMS algorithm stops:

$$|e_q(n)| \leq \frac{\text{LSB}}{\mu A_{\text{rms}}} = e_D(\mu) \tag{17.21}$$

The quantity $e_D(\mu)$, defined on the right-hand side of (17.21), is called the *digital residual error*.

To prevent the algorithm-stalling phenomenon due to digital effects, the digital residual error $e_D(\mu)$ must be made as small as possible. According to the definition of Eq. (17.21), this requirement may be satisfied in one of two ways:

1. The least significant bit (LSB) is reduced by picking a sufficiently large number of bits for the digital representation of each tap weight.
2. The step-size parameter μ is made as large as possible, while still guaranteeing convergence of the algorithm.

Another method of preventing the stalling phenomenon is to insert *dither* at the input of the quantizer that feeds the tap-weight accumulator (Sherwood and Bershad, 1987). Dither is a random sequence that essentially "linearizes" the quantizer. In other words, the addition of dither guarantees that the quantizer input is *noisy* enough for the gradient quantization error vector $\boldsymbol{\eta}_w$ to be again modeled as white noise (i.e., the elements of $\boldsymbol{\eta}_w$ are uncorrelated in time and with each other, and have a common variance σ_w^2). When dither is used in the manner described here, it is desirable to minimize its effect on the overall operation of the LMS algorithm. This is commonly achieved by shaping the power spectrum of the dither so that it is effectively rejected by the algorithm at its output.

Parameter Drift

In addition to the numerical problems associated with the LMS algorithm, there is one other rather subtle problem that is encountered in practical applications of the algorithm. Specifically, certain classes of input excitation can lead to *parameter drift*; that is, parameter estimates or tap weights in the LMS algorithm attain arbitrarily large values despite bounded inputs, bounded disturbances, and bounded estimation errors (Sethares et al., 1986). Although such an unbounded behavior may be unexpected, it is possible for the parameter estimates to drift to infinity while all the signals observable in the algorithm converge to zero. Parameter drift in the LMS algorithm may be viewed as a *hidden form of instability*, since the tap weights represent "internal" variables of the algorithm. As such, it may result in new numerical problems, increased sensitivity to unmodeled disturbances, and degraded long-term performance.

In order to appreciate the subtleties of the parameter drift problem, we need to introduce some new concepts relating to the parameter space. We therefore digress briefly from the issue at hand to do so.

A sequence of information-bearing tap-input vectors $\mathbf{u}(n)$ for varying time n may be used to partition the *real M-dimensional parameter space* $\mathbb{R}^M$ into orthogonal subspaces, where M is the number of tap weights (i.e., the available number of degrees of freedom). The aim of this partitioning is to convert the stability analysis of an adaptive filtering algo-

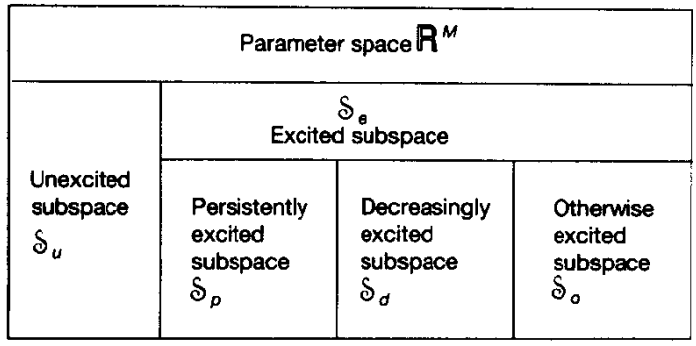


Figure 17.3 Decomposition of parameter space $\mathbb{R}^M$, based on excitation.

rithm (e.g., the LMS algorithm) into simpler subsystems and thereby provide a closer linkage between the transient behavior of the parameter estimates and the filter excitations. The partitioning we have in mind is depicted in Fig. 17.3. In particular, we may identify the following subspaces of $\mathbb{R}^M$.

1. *The unexcited subspace.* Let the M-by-1 vector $\mathbf{z}$ be any element of the parameter space $\mathbb{R}^M$, which satisfies two conditions:
   - The Euclidean norm of the vector $\mathbf{z}$ is 1; that is,
   $$\|\mathbf{z}\| = 1$$
   - The vector $\mathbf{z}$ is orthogonal to the tap-input vector $\mathbf{u}(n)$ for all but a finite number of n; that is,
   $$\mathbf{z}^T\mathbf{u}(n) \neq 0, \qquad \text{only finitely often} \tag{17.22}$$

   Let $\mathcal{S}_u$ denote the subspace of $\mathbb{R}^M$ that is spanned by the set of all such vectors $\mathbf{z}$. The subspace $\mathcal{S}_u$ is called the *unexcited subspace* in the sense that it spans those *directions* in the parameter space $\mathbb{R}^M$ that are excited only *finitely often.*
2. *The excited subspace.* Let $\mathcal{S}_e$ denote the *orthogonal complement* of the unexcited subspace $\mathcal{S}_u$. Clearly, $\mathcal{S}_e$ is also a subspace of the parameter space $\mathbb{R}^M$. It contains those directions in the parameter space $\mathbb{R}^M$ that are excited *infinitely often.* Thus, except for the null vector, every element $\mathbf{z}$ *belonging to the subspace* $\mathcal{S}_e$ satisfies the condition
   $$\mathbf{z}^T\mathbf{u}(n) \neq 0, \qquad \text{infinitely often} \tag{17.23}$$

   The subspace $\mathcal{S}_e$ is called the *excited subspace.*

   The subspace $\mathcal{S}_e$ may itself be decomposed into three orthogonal subspaces of its own, depending on the effects of different types of excitation on the behavior of the adaptive filtering algorithm. Specifically, three subspaces of $\mathcal{S}_e$ may be identified as follows (Sethares et al., 1986):

- *The persistently excited subspace.* Let $\mathbf{z}$ be any vector of unit norm that lies in the excited subspace $\mathcal{S}_e$. For any positive integer m and any $\alpha > 0$, choose the vector $\mathbf{z}$ such that we have

$$\mathbf{z}^T\mathbf{u}(i) > \alpha \qquad \text{for } n \le i \le n + m \text{ and for all but a finite number of } n \tag{17.24}$$

Given the integer m and the constant α, let $\mathcal{S}_p(m, \alpha)$ be the subspace spanned by all such vectors $\mathbf{z}$ that satisfy the condition of (17.24). There exist a finite m_0 and a positive α_0 for which *the subspace* $\mathcal{S}_p(m_0, \alpha_0)$ is *maximal*. In other words, $\mathcal{S}_p(m_0, \alpha_0)$ contains $\mathcal{S}_p(m, \alpha)$ for all $m > 0$ and for all $\alpha > 0$. The subspace $\mathcal{S}_p \equiv \mathcal{S}_p(m_0, \alpha_0)$ is called the *persistently excited subspace*; and m_0 is called the *interval of excitation*. For every "direction" $\mathbf{z}$ that lies in the persistently excited subspace $\mathcal{S}_p$, there is an excitation of level α_0 at least once in all but a finite number of intervals of length m_0. In the persistently excited subspace, we are therefore able to find a tap-input vector $\mathbf{u}(n)$ *rich* enough to excite all the internal modes that govern the transient behavior of the adaptive filtering algorithm being probed (Narendra and Annaswamy, 1989).

- *The subspace of decreasing excitation.* Consider a sequence $u(i)$ for which we have

$$\left(\sum_{i=1}^{\infty} |u(i)|^p\right)^{1/p} < \infty \tag{17.25}$$

Such a sequence is said to be an element of the *normed linear space* l^p for $1 < p < \infty$. The norm of this new space is defined by

$$\|\mathbf{u}\|_p = \left(\sum_{i=1}^{\infty} |u(i)|^p\right)^{1/p} \tag{17.26}$$

Note that if the sequence $u(i)$ is an element of the normed linear space l^p for $1 < p < \infty$, then

$$\lim_{n\to\infty} u(n) = 0 \tag{17.27}$$

Let $\mathbf{z}$ be any unit-norm vector $\mathbf{z}$ that lies in the excited subspace $\mathcal{S}_e$ such that for $1 < p < \infty$, the sequence $\mathbf{z}^T\mathbf{u}(n)$ lies in the normed linear space l^p. Let $\mathcal{S}_d$ be the subspace that is spanned by all such vectors $\mathbf{z}$. The subspace $\mathcal{S}_d$ is called the *subspace of decreasing excitation* in the sense that each direction of $\mathcal{S}_d$ is decreasingly excited. For any vector $\mathbf{z} \neq \mathbf{0}$, the two conditions

$$|\mathbf{z}^T\mathbf{u}(n)| = \alpha > 0, \qquad \text{infinitely often}$$

and

$$\lim_{n\to\infty} \mathbf{z}^T\mathbf{u}(n) = 0$$

cannot be satisfied simultaneously. In actual fact, we find that the subspace of decreasing excitation $\mathcal{S}_d$ is orthogonal to the subspace of persistent excitation $\mathcal{S}_p$.

- *The otherwise excited subspace.* Let $\mathcal{S}_p \cup \mathcal{S}_d$ denote the *union* of the persistently excited subspace $\mathcal{S}_p$ and the subspace of decreasing excitation $\mathcal{S}_d$. Let $\mathcal{S}_o$ denote the orthogonal complement of $\mathcal{S}_p \cup \mathcal{S}_d$ that lies in the excited subspace $\mathcal{S}_e$. The subspace $\mathcal{S}_o$ is called the *otherwise excited subspace.* Any vector that lies in the subspace $\mathcal{S}_o$ is not unexciting, not persistently exciting, and not in the normal linear space l^p for any finite p. An example of such a signal is the sequence

$$\mathbf{z}^T\mathbf{u}(n) = \frac{1}{\ln(1+n)}, \qquad n = 1, 2, \ldots \tag{17.28}$$

Returning to our discussion of the parameter drift problem in the LMS algorithm, we find that for bounded excitations and bounded disturbances, in the case of unexcited and persistently exciting subspaces the parameter estimates resulting from the application of the LMS algorithm are indeed bounded. However, in the decreasing and otherwise excited cases, parameter drift may occur (Sethares et al., 1986). A common method of counteracting the parameter drift problem in the LMS algorithm is to introduce leakage into the tap-weight update equation of the algorithm. Here is another reason for using the leaky LMS algorithm that was described previously.

17.3 RECURSIVE LEAST-SQUARES ALGORITHM

The recursive least-squares (RLS) algorithm offers an alternative to the LMS algorithm as a tool for the solution of adaptive filtering problems. From the discussion presented in Chapter 13, we know that the RLS algorithm is characterized by a fast rate of convergence that is relatively insensitive to the eigenvalue spread of the underlying correlation matrix of the input data, and a negligible misadjustment (zero for a stationary environment without disturbances). Moreover, although it is computationally demanding (in the sense that its computational complexity is on the order of M^2, where M is the dimension of the tap-weight vector), the mathematical formulation and therefore implementation of the RLS algorithm is relatively simple. However, there is a numerical instability problem to be considered when the RLS algorithm is implemented in finite-precision arithmetic.

Basically, *numerical instability* or *explosive divergence* of the RLS algorithm is of a similar nature to that experienced in Kalman filtering, of which the RLS algorithm is a special case. Indeed, the problem may be traced to the fact that the time-updated matrix $\mathbf{P}(n)$ in the Riccati equation is computed as the difference between two nonnegative definite matrices, as indicated in Eq. (13.19). Accordingly, explosive divergence of the algorithm occurs when the matrix $\mathbf{P}(n)$ loses the property of positive definiteness or Hermitian sym-

TABLE 17.1 SUMMARY OF A COMPUTATIONALLY EFFICIENT SYMMETRY-PRESERVING VERSION OF THE RLS ALGORITHM

Initialize the algorithm by setting

$$\mathbf{P}(0) = \delta^{-1}\mathbf{I}, \qquad \delta = \text{small positive constant}$$

$$\hat{\mathbf{w}}(0) = \mathbf{0}$$

For each instant of time, $n = 1, 2, \ldots$, compute

$$\boldsymbol{\pi}(n) = \mathbf{P}(n-1)\mathbf{u}(n)$$

$$r(n) = \frac{1}{\lambda + \mathbf{u}^H(n)\boldsymbol{\pi}(n)}$$

$$\mathbf{k}(n) = r(n)\boldsymbol{\pi}(n)$$

$$\xi(n) = d(n) - \hat{\mathbf{w}}^H(n-1)\mathbf{u}(n)$$

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{k}(n)\xi^*(n)$$

$$\mathbf{P}(n) = \text{Tri}\{\lambda^{-1}[\mathbf{P}(n-1) - \mathbf{k}(n)\boldsymbol{\pi}^H(n)]\}$$

metry. This is precisely what happens in the usual formulation of the RLS algorithm described in Table 13.1 (Verhaegen, 1989).

How then can the RLS algorithm be formulated so that the Hermitian symmetry of the matrix $\mathbf{P}(n)$ is preserved despite the presence of numerical errors? For obvious practical reasons, it would also be satisfying if the solution to this fundamental question can be attained in a computationally efficient manner. With these issues in mind, we present in Table 17.1 a particular version of the RLS algorithm from Yang (1994), which describes a computationally efficient procedure[3] for preserving the Hermitian symmetry of $\mathbf{P}(n)$ by design. The improved computational efficiency of this algorithm is achieved because it computes simply the upper/lower triangular part of the matrix $\mathbf{P}(n)$, as signified by the operator Tri{ }, and then fills in the rest of the matrix to preserve Hermitian symmetry. Moreover, division by λ is replaced by multiplication with the precomputed value of λ^{-1}.

Error Propagation Model

According to the algorithm of Table 17.1, the recursions involved in the computation of $\mathbf{P}(n)$ proceed as follows:

$$\boldsymbol{\pi}(n) = \mathbf{P}(n-1)\mathbf{u}(n) \tag{17.29}$$

$$r(n) = \frac{1}{\lambda + \mathbf{u}^H(n)\boldsymbol{\pi}(n)} \tag{17.30}$$

[3]Verhaegen (1989) describes another symmetry-preserving version of the RLS algorithm; Verhaegen's version is less efficient than Yang's version in computational terms. However, both versions exhibit the same numerical behavior.

$$\mathbf{k}(n) = r(n)\boldsymbol{\pi}(n) \tag{17.31}$$

$$\mathbf{P}(n) = \text{Tri}\{\lambda^{-1}[\mathbf{P}(n-1) - \mathbf{k}(n)\boldsymbol{\pi}^H(n)]\} \tag{17.32}$$

where λ is the exponential weighting factor. Consider the propagation of a *single* quantization error at time $n-1$ to subsequent recursions, under the assumption that no other quantization errors are made. In particular, let

$$\mathbf{P}_q(n-1) = \mathbf{P}(n-1) + \boldsymbol{\eta}_p(n-1) \tag{17.33}$$

where the *error matrix* $\boldsymbol{\eta}_p(n-1)$ arises from the quantization of $\mathbf{P}(n-1)$. The corresponding quantized value of $\boldsymbol{\pi}(n)$ is

$$\boldsymbol{\pi}_q(n) = \boldsymbol{\pi}(n) + \boldsymbol{\eta}_p(n-1)\mathbf{u}(n) \tag{17.34}$$

Let $r_q(n)$ denote the quantized value of $r(n)$. Using the defining equation (17.30), we may write

$$\begin{aligned} r_q(n) &= \frac{1}{\lambda + \mathbf{u}^H(n)\boldsymbol{\pi}_q(n)} \\ &= \frac{1}{\lambda + \mathbf{u}^H(n)\boldsymbol{\pi}(n) + \mathbf{u}^H(n)\,\boldsymbol{\eta}_p(n-1)\mathbf{u}(n)} \\ &= \frac{1}{\lambda + \mathbf{u}^H(n)\boldsymbol{\pi}(n)}\left(1 + \frac{\mathbf{u}^H(n)\boldsymbol{\eta}_p(n-1)\mathbf{u}(n)}{\lambda + \mathbf{u}^H(n)\boldsymbol{\pi}(n)}\right)^{-1} \\ &= \frac{1}{\lambda + \mathbf{u}^H(n)\boldsymbol{\pi}(n)} - \frac{\mathbf{u}^H(n)\boldsymbol{\eta}_p(n-1)\mathbf{u}(n)}{(\lambda + \mathbf{u}^H(n)\boldsymbol{\pi}(n))^2} + O(\boldsymbol{\eta}_p^2) \\ &= r(n) - \frac{\mathbf{u}^H(n)\boldsymbol{\eta}_p(n-1)\mathbf{u}(n)}{(\lambda + \mathbf{u}^H(n)\boldsymbol{\pi}(n))^2} + O(\boldsymbol{\eta}_p^2) \end{aligned} \tag{17.35}$$

where $O(\boldsymbol{\eta}_p^2)$ denotes the order of magnitude $\|\boldsymbol{\eta}_p\|^2$.

In an ideal situation, the infinite-precision scalar quantity $r(n)$ is *nonnegative*, taking on values between zero and $1/\lambda$. On the other hand, if $\mathbf{u}^H(n)\boldsymbol{\pi}(n)$ is small compared to λ and λ itself is small enough compared to 1, then according to Eq. (17.35), in a finite-precision environment it is possible for the quantized quality $r_q(n)$ to take on a *negative* value larger in magnitude than $1/\lambda$. When this happens, the RLS algorithm exhibits explosive divergence (Bottomley and Alexander, 1989).[4]

The quantized value of the gain vector $\mathbf{k}(n)$ is written as

$$\begin{aligned} \mathbf{k}_q(n) &= r_q(n)\boldsymbol{\pi}_q(n) \\ &= \mathbf{k}(n) + \boldsymbol{\eta}_k(n) \end{aligned} \tag{17.36}$$

where $\boldsymbol{\eta}_k(n)$ is the *gain vector quantization error*, defined by

$$\boldsymbol{\eta}_k(n) = r(n)\,(\mathbf{I} - \mathbf{k}(n)\mathbf{u}^H(n))\,\boldsymbol{\eta}_p(n-1)\mathbf{u}(n) + O(\boldsymbol{\eta}_p^2) \tag{17.37}$$

[4]According to Bottomley and Alexander (1989), the evolution of the factor $r(n)$ provides a good indication of explosive divergence as this factor grows large, then suddenly becomes negative.

Finally, using Eq. (17.32), we find that the quantization error incurred in computing the updated inverse-correlation matrix $\mathbf{P}(n)$ is

$$\boldsymbol{\eta}_p(n) = \lambda^{-1}(\mathbf{I} - \mathbf{k}(n)\mathbf{u}^H(n))\,\boldsymbol{\eta}_p(n-1)(\mathbf{I} - \mathbf{k}(n)\mathbf{u}^H(n))^H \tag{17.38}$$

where the term $O(\boldsymbol{\eta}_p^2)$ has been ignored.

On the basis of Eq. (17.38), it would be tempting to conclude that $\boldsymbol{\eta}_p^H(n) = \boldsymbol{\eta}_p(n)$ and therefore the RLS algorithm of Table 17.1 is *Hermitian-symmetry preserving*, if we can assume that the condition $\boldsymbol{\eta}_p^H(n-1) = \boldsymbol{\eta}_p(n-1)$ holds at the previous iteration. We are justified in making this assertion by virtue of the fact there is *no* blow-up in this formulation of the RLS algorithm, as demonstrated in what follows (it is also assumed that there is no stalling).

Equation (17.38) defines the *error propagation mechanism* for the RLS algorithm summarized in Table 17.1 on the basis of a single quantization error in $\mathbf{P}(n-1)$. The matrix $\mathbf{I} - \mathbf{k}(n)\mathbf{u}^H(n)$ plays a crucial role in the way in which the single quantization error $\boldsymbol{\eta}_p(n-1)$ propagates through the algorithm. Using the original definition given in Eq. (13.22) for the gain vector, namely,

$$\mathbf{k}(n) = \boldsymbol{\Phi}^{-1}(n)\mathbf{u}(n) \tag{17.39}$$

we may write

$$\mathbf{I} - \mathbf{k}(n)\mathbf{u}^H(n) = \mathbf{I} - \boldsymbol{\Phi}^{-1}(n)\mathbf{u}(n)\mathbf{u}^H(n) \tag{17.40}$$

Next, from Eq. (13.12) we have

$$\boldsymbol{\Phi}(n) = \lambda\boldsymbol{\Phi}(n-1) + \mathbf{u}(n)\mathbf{u}^H(n) \tag{17.41}$$

Multiplying both sides of Eq. (17.41) by the inverse matrix $\boldsymbol{\Phi}^{-1}(n)$ and rearranging terms, we get

$$\mathbf{I} - \boldsymbol{\Phi}^{-1}(n)\mathbf{u}(n)\mathbf{u}^H(n) = \lambda\boldsymbol{\Phi}^{-1}(n)\boldsymbol{\Phi}(n-1) \tag{17.42}$$

Comparing Eqs. (17.40) and (17.42), we readily deduce that

$$\mathbf{I} - \mathbf{k}(n)\mathbf{u}^H(n) = \lambda\boldsymbol{\Phi}^{-1}(n)\boldsymbol{\Phi}(n-1) \tag{17.43}$$

Suppose now we consider the effect of the quantization error $\boldsymbol{\eta}_p(n_0)$ induced at time $n_0 \leq n$. When the RLS algorithm of Table 17.1 is used and the matrix $\mathbf{P}(n)$ remains Hermitian, then according to the error-propagation model of Eq. (17.38), the effect of the quantization error $\boldsymbol{\eta}_p(n_0)$ becomes modified at time n as follows:

$$\boldsymbol{\eta}_p(n) = \lambda^{-(n-n_0)}\,\boldsymbol{\varphi}(n, n_0)\boldsymbol{\eta}_p(n_0)\boldsymbol{\varphi}^H(n, n_0),\quad n \geq n_0 \tag{17.44}$$

where $\boldsymbol{\varphi}(n, n_0)$ is a *transition matrix* defined by

$$\boldsymbol{\varphi}(n, n_0) = (\mathbf{I} - \mathbf{k}(n)\mathbf{u}^H(n)) \cdots (\mathbf{I} - \mathbf{k}(n_0 + 1)\,\mathbf{u}^H(n_0 + 1)) \tag{17.45}$$

The repeated use of Eq. (17.43) in (17.45) leads us to express the transition matrix in the equivalent form

$$\varphi(n, n_0) = \lambda^{n-n_0} \mathbf{\Phi}^{-1}(n)\mathbf{\Phi}(n_0) \tag{17.46}$$

The correlation matrix $\mathbf{\Phi}(n)$ is defined by [see Eq. (13.9)]

$$\mathbf{\Phi}(n) = \sum_{i=1}^{n} \lambda^{n-i}\mathbf{u}(i)\mathbf{u}^H(i) \tag{17.47}$$

On the basis of this definition, the tap-input vector $\mathbf{u}(n)$ is said to be *uniformly persistently exciting* for sufficiently large n if there exist some $a > 0$ and $n > 0$ such that the following condition is satisfied (Ljung and Ljung, 1985):

$$\mathbf{\Phi}(n) \geq a\mathbf{I} \qquad \text{for } n \geq N \tag{17.48}$$

The notation used in Eq. (17.48) is shorthand for saying that the matrix $\mathbf{\Phi}(n)$ is positive definite. The condition for persistent excitation not only guarantees the positive definiteness of $\mathbf{\Phi}(n)$, but also guarantees its matrix norm to be uniformly bounded for $n \geq N$, as shown by

$$\|\mathbf{\Phi}^{-1}(n)\| \leq \frac{1}{a} \qquad \text{for } n \geq N \tag{17.49}$$

Returning to the transition matrix $\varphi(n, n_0)$ of Eq. (17.46) and invoking the *mutual consistency*[5] property of a matrix norm, we may write

$$\|\varphi(n, n_0)\| \leq \lambda^{n-n_0} \|\mathbf{\Phi}^{-1}(n)\| \cdot \|\mathbf{\Phi}(n_0)\| \tag{17.50}$$

Next, invoking the inequality of (17.49), we may rewrite that of Eq. (17.50) as

$$\|\varphi(n, n_0)\| \leq \frac{\lambda^{n-n_0}}{a} \|\mathbf{\Phi}(n_0)\| \tag{17.51}$$

Finally, we may use the error propagation equation (17.44) to express the vector norm of $\boldsymbol{\eta}_p(n)$ as

$$\|\boldsymbol{\eta}_p(n)\| \leq \lambda^{-(n-n_0)} \|\varphi(n, n_0)\| \cdot \|\boldsymbol{\eta}_p(n-1)\| \cdot \|\varphi^H(n, n_0)\|$$

which, in light of (17.51), may be rewritten as

$$\|\boldsymbol{\eta}_p(n)\| \leq \lambda^{n-n_0} M, \qquad n \geq n_0 \tag{17.52}$$

where M is a positive number defined by

$$M = \frac{1}{a^2} \|\mathbf{\Phi}(n_0)\|^2 \|\boldsymbol{\eta}_p(n-1)\| \tag{17.53}$$

[5]Consider two matrices **A** and **B** of compatible dimensions. The mutual consistency property states that (see page 168)'

$$\|\mathbf{AB}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{B}\|$$

Equation (17.53) states that the RLS algorithm of Table 17.1 is *exponentially stable* in the sense that a single quantization error $\eta_p(n_0)$ occurring in the inverse correlation matrix $\mathbf{P}(n_0)$ at time n_0 decays exponentially provided that $\lambda < 1$ (i.e., the algorithm has *finite memory*).[6] In other words, the propagation of a single error through this formulation of the standard RLS algorithm with finite memory is *contractive*. Computer simulations validating this result are presented in Verhaegen (1989).

However, the single-error propagation for the case of *growing memory* (i.e., $\lambda = 1$) is *not* contractive. The reason for saying so is that when $\lambda = 1$, neither $\boldsymbol{\varphi}(n, n_0) \leq \mathbf{I}$ nor $\|\boldsymbol{\varphi}(n, n_0)\| \leq 1$ holds, even if the input vector $\mathbf{u}(n)$ is persistently exciting. Consequently, the accumulation of numerical errors may cause the algorithm to be divergent (Yang, 1994). In an independent study, Slock and Kailath (1991) also point out that the error propagation mechanism in the RLS algorithm with $\lambda = 1$ is unstable and of a random walk type. Moreover, there is experimental evidence for this numerical divergence, which is reported in (Ardalan and Alexander, 1987).

Stalling Phenomenon

As with the LMS algorithm, a second form of divergence, referred to as the *stalling phenomenon*, occurs when the tap weights in the RLS algorithm stop adapting. In particular, this phenomenon occurs when the quantized elements of the matrix $\mathbf{P}(n)$ become very small, such that multiplication by $\mathbf{P}(n)$ is equivalent to multiplication by a zero matrix (Bottomley and Alexander, 1989). Clearly, the stalling phenomenon may arise no matter how the RLS algorithm is implemented.

The stalling phenomenon is directly linked to the exponential weighting factor λ and the variance σ_u^2 of the input data $u(n)$. Assuming that λ is close to unity, we find from the definition of the correlation matrix $\boldsymbol{\Phi}(n)$ that the expectation of $\boldsymbol{\Phi}(n)$ is given by [see Eq. 16.46)]

$$E[\boldsymbol{\Phi}(n)] \simeq \frac{\mathbf{R}}{1 - \lambda}, \qquad \text{large } n \tag{17.54}$$

For λ close to unity, we have

$$E[\mathbf{P}(n)] = E[\boldsymbol{\Phi}^{-1}(n)] \simeq (E[\boldsymbol{\Phi}(n)])^{-1} \tag{17.55}$$

Hence, using Eq. (17.54) in Eq. (17.55), we get

$$E[\mathbf{P}(n)] \simeq (1 - \lambda)\mathbf{R}^{-1}, \qquad \text{large } n \tag{17.56}$$

[6]The first rigorous proof that single-error propagation in the RLS algorithm is exponentially stable for $\lambda < 1$ was presented in (Ljung and Ljung, 1984). This was followed by a more detailed investigation in (Verhaegen, 1989). Reconfirmation that the error propagation mechanism in the RLS algorithm is exponentially stable was subsequently presented in (Slock and Kailath, 1991; Yang, 1994).

where $\mathbf{R}^{-1}$ is the inverse of matrix $\mathbf{R}$. Assuming that the tap-input vector $\mathbf{u}(n)$ is drawn from a wide-sense stationary process with zero mean, we may write

$$\mathcal{R} = \frac{1}{\sigma_u^2}\mathbf{R} \tag{17.57}$$

where $\mathcal{R}$ is a *normalized correlation matrix* with diagonal elements equal to 1 and off-diagonal elements less than or equal to 1 in magnitude, and σ_u^2 is the variance of an input data sample $u(n)$. We may therefore rewrite Eq. (17.56) as

$$E[\mathbf{P}(n)] \simeq \left(\frac{1-\lambda}{\sigma_u^2}\right)\mathcal{R}^{-1} \qquad \text{for large } n \tag{17.58}$$

Equation (17.58) reveals that the RLS algorithm may stall if the exponential weighting factor λ is close to 1 and/or the input data variance σ_u^2 is large. Accordingly, we may prevent stalling of the standard RLS algorithm by using a sufficiently large number of accumulator bits in the computation of the inverse correlation matrix $\mathbf{P}(n)$.

17.4 SQUARE-ROOT ADAPTIVE FILTERS

In Chapter 14 we discussed three particular forms of square-root adaptive filters, namely, the *QR-decomposition–based recursive least-squares (QR–RLS)* algorithm, the *extended QR–RLS algorithm*, and the *inverse QR–RLS algorithm*. The QR–RLS and extended QR–RLS algorithms are special cases of the square-root information filter, whereas the inverse QR–RLS algorithm is a special case of the square-root covariance (Kalman) filter.

QR–RLS Algorithm

It is generally agreed that the QR decomposition is one of the best numerical procedures for solving the recursive least-squares estimation problem because of two important properties:

1. The QR decomposition operates on the input data directly.
2. The QR decomposition involves the use of only numerically well-behaved unitary rotations (e.g., Givens rotations).

In particular, the QR–RLS algorithm propagates the square root of the correlation matrix $\Phi(n)$ rather than $\Phi(n)$ itself. Hence, the condition number of $\Phi^{1/2}(n)$ equals the square root of the condition number of $\Phi(n)$. This, in turn, results in a significant reduction in the dynamic range of data handled by QR-decomposition–based algorithms, and therefore a more accurate computation than the standard RLS algorithm that propagates $\Phi(n)$. Moreover, the finite-precision form of the QR–RLS algorithm is *stable in a bounded input–bounded output (BIBO) sense* (Leung and Haykin, 1989; Liu et al., 1991). However,

it must be stressed that the BIBO stability of the QR–RLS algorithm does not guarantee that the various quantities computed by the algorithm remain meaningful in any sense when operating in a finite-precision environment (Yang and Böhme, 1992). In particular, a unitary rotation (e.g., sequence of Givens rotations) is used to annihilate a certain vector in the prearray, and the unitary rotation then operates on other related entries in the prearray. A perturbation in internal computations may produce a corresponding perturbation in rotation angles, which introduces yet another source of numerical error in the rotated entries of the postarray. These errors, in turn, produce further perturbations of their own in subsequent computations of the rotation angles, and the process goes on. The net result is that we have a *complicated parametric feedback system*, and it is not entirely clear whether this feedback system is in fact numerically stable.

Yang and Böhme (1992) present experimental results that demonstrate the numerical stability of the QR–RLS algorithm for $\lambda < 1$; they used this algorithm to perform adaptive prediction of an autoregressive (AR) process. All the computer simulations reported in that paper were performed on a personal computer (PC) using floating-point arithmetic. In order to observe finite-precision effects in a reasonable simulation time, the effective number of mantissa bits in the floating-point representation was reduced. This was done by truncating the mantissa at a predefined position without affecting the exponent. In the experiments reported by Yang and Böhme, the mantissa length took on the values 52, 12, and 5 bits; the resulting word-length changes were found to have only a minor effect on the convergence behavior of the QR–RLS algorithm. Yang and Böhme (1992) also show that the QR–RLS algorithm diverges when $\lambda = 1$.

The numerical stability of the QR–RLS algorithm for $\lambda < 1$ has also been demonstrated experimentally by Ward et al. (1986) in the context of adaptive beamforming. In particular, they show that for the same number of bits of arithmetic precision, the QR–RLS algorithm offers a significantly better performance than the sample matrix inversion algorithm described in Reed et al. (1974).

Extended QR–RLS Algorithm

In comparing the QR–RLS and extended QR–RLS algorithms summarized in Table 14.2, we see that these two algorithms differ in the quantities that they propagate. In particular, the QR–RLS algorithm propagates $\boldsymbol{\Phi}^{1/2}(n)$, whereas the extended QR–RLS algorithm propagates both $\boldsymbol{\Phi}^{1/2}(n)$ and its Hermitian inverse $\boldsymbol{\Phi}^{-H/2}(n)$, independently from each other. Accordingly, in a finite-precision environment, the extended QR–RLS algorithm behaves in a profoundly different way from the QR–RLS algorithm.

Let $\mathbf{X}(n-1)$ denote the quantized version of the Hermitian inverse matrix $\boldsymbol{\Phi}^{-H/2}(n-1)$, stored at time $n-1$, as shown by

$$\mathbf{X}(n-1) = \boldsymbol{\Phi}^{-H/2}(n-1) + \boldsymbol{\eta}_X(n-1) \tag{17.59}$$

where the component $\boldsymbol{\eta}_X(n-1)$ represents the effect of round-off errors. Then, assuming that there are no additional local errors introducted at time n, the recursion pertaining to the bottom parts of the prearray and postarray of Eq. (14.61) takes on the following form:

$$[\lambda^{-1/2}\mathbf{X}(n-1) \quad \mathbf{0}]\boldsymbol{\Theta}(n) = [\mathbf{X}(n) \quad -\mathbf{y}(n)] \tag{17.60}$$

where $\boldsymbol{\Theta}(n)$ is a unitary rotation. The vector $\mathbf{Y}(n)$ is the quantized version of $\mathbf{k}(n)\gamma^{-1/2}(n)$ as shown by

$$\mathbf{y}(n) = \mathbf{k}(n)\,\gamma^{-1/2}(n) + \boldsymbol{\eta}_y(n) \tag{17.61}$$

where $\mathbf{k}(n)$ is the gain vector, and $\gamma(n)$ is the conversion factor. In a corresponding way to Eq. (17.59), the updated matrix $\mathbf{X}(n)$ may be expressed as

$$\mathbf{X}(n) = \boldsymbol{\Phi}^{-H/2}(n) + \boldsymbol{\eta}_X(n) \tag{17.62}$$

Hence, substituting Eqs. (17.59), (17.61), and (17.62) in Eq. (17.60), we get

$$\begin{aligned}[\lambda^{-1/2}\boldsymbol{\Phi}^{-H/2}(n-1) + \lambda^{-1/2}\boldsymbol{\eta}_X(n-1) \quad \mathbf{0}]\boldsymbol{\Theta}(n)& \\ = [\boldsymbol{\Phi}^{-H/2}(n) + \boldsymbol{\eta}_X(n) \quad - \mathbf{k}(n)\gamma^{-1/2}(n) - \boldsymbol{\eta}_y(n)]&\end{aligned} \tag{17.63}$$

But, under infinite-precision arithmetic, we have

$$[\lambda^{-1/2}\boldsymbol{\Phi}^{-H/2}(n-1) \quad \mathbf{0}]\boldsymbol{\Theta}(n) = [\boldsymbol{\Phi}^{-H/2}(n) \quad - \mathbf{k}(n)\gamma^{-1/2}(n)]$$

Accordingly, we may simplify Eq. (17.63) to

$$[\lambda^{-1/2}\boldsymbol{\eta}_X(n-1) \quad \mathbf{0}]\boldsymbol{\Theta}(n) = [\boldsymbol{\eta}_x(n) \quad - \boldsymbol{\eta}_y(n)] \tag{17.64}$$

Equation (17.64) reveals that the error propagation due to $\boldsymbol{\eta}_X(n-1)$ is *not* necessarily stable, in that local errors tend to grow unboundedly (Moonen and Vandewalle, 1990). The unlimited error growth is due to (1) the amplification produced by the factor $\lambda^{-1/2}$ for $\lambda < 1$, and (2) the fact that the unitary rotation $\boldsymbol{\Theta}(n)$ is independent of the error $\boldsymbol{\eta}_X(n)$. Consequently, as the recursion progresses, the stored values of $\boldsymbol{\Phi}^{1/2}$ and $\boldsymbol{\Phi}^{-H/2}$ deviate more and more from each other's Hermitian inverse, thereby contradicting the very hypothesis on which the recursion in the extended QR–RLS algorithm is based. This is indeed another example of *numerical inconsistency*, to which all forms of numerical divergence in RLS algorithms can be traced. Another manifestation of this phenomenon is that the quantity $-\mathbf{k}(n)\gamma^{-1/2}(n)$, which is the by-product of updating $\boldsymbol{\Phi}^{-H/2}(n-1)$, deviates more and more from its infinite-precision value; this, in turn, causes the weight updating to produce unreliable results.

To combat this problem in the extended QR–RLS algorithm, Moonen and Vandewalle (1990) describe a procedure that forces $\boldsymbol{\Phi}^{-H/2}(n)$ to be the exact Hermitian inverse of $\boldsymbol{\Phi}^{1/2}(n)$ at every iteration of the algorithm by using Jacobi-type rotations. However, the price paid for achieving this requirement is increased computational complexity.

Inverse QR–RLS Algorithm

The inverse QR–RLS algorithm propagates $\mathbf{P}^{1/2}(n)$, which is the square root of the inverse correlation matrix $\mathbf{P}(n) = \boldsymbol{\Phi}^{-1}(n)$. Thus, although the inverse QR–RLS algorithm differs from the QR–RLS algorithm that propagates $\boldsymbol{\Phi}^{1/2}(n)$, the two algorithms do share a common feature: they both avoid the propagation of the Hermitian inverse of their respective

matrix quantities. Accordingly, the inverse QR–RLS algorithm is able to exploit the good numerical properties of the QR decomposition in a manner similar to the QR–RLS algorithm.

For $\lambda < 1$, the propagation of a single error in the inverse QR–RLS algorithm (and, for that matter, in the QR–RLS algorithm) is *exponentially stable*. The rationale for this statement follows the numerical stability analysis of the RLS algorithm presented in Section 17.3.

However, for $\lambda = 1$, the single-error propagation is *not* contractive. It may therefore be conjectured that the accumulation of quantization errors can cause the inverse QR–RLS algorithm to be numerically divergent; this phenomenon has been confirmed experimentally, using computer simulations.[7] A similar remark applies to the QR–RLS algorithm, for which experimental validation is presented in Yang and Böhme (1992).

Summarizing Comments

In summary, we may say the following on the requirement to operate a square-root adaptive filter in a finite-precision environment[8]:

- Use of the extended QR–RLS algorithm is not recommended.
- If only the estimation error $e(n)$ is required, the QR–RLS algorithm is the preferred choice.
- If the weight vector $\hat{\mathbf{w}}(n)$ is required, both the QR–RLS algorithm (in conjunction with back-substitution) and the inverse QR–RLS algorithms are good candidates.
- If, in addition, a systolic implementation of the filter is desired, the inverse QR–RLS algorithm is the best choice.

17.5 ORDER-RECURSIVE ADAPTIVE FILTERS

In order-recursive adaptive filtering algorithms (and, for that matter, in all fast RLS algorithms known to date), the particular section responsible for joint-process estimation is subordinate to the section responsible for performing the forward and backward linear predictions. Accordingly, the numerical stability of this class of adaptive filtering algorithms is critically dependent on how the prediction section performs its computations, as discussed in the sequel.

QRD–LSL Algorithm

In the *QR-decomposition–based least-squares lattice (QRD–LSL) algorithm* summarized in Table 15.3, the prediction section consists of $M-1$ lattice stages, where $M-1$ is the pre-

[7]Yang, B., private communication, 1995.

[8]The summarizing comments presented herein are based on Yang, B., private communication, 1995.

diction order. Each stage of the prediction section uses QR decomposition in the form of Givens rotations to perform its computations. The net result is that the sequence of input data $u(n)$, $u(n-1)$, . . . , $u(n-M+1)$ is transformed into a corresponding sequence of angle-normalized backward prediction errors $\epsilon_{b,0}(n)$, $\epsilon_{b,1}(n)$, . . . , $\epsilon_{b,m-1}(n)$. Given this latter sequence, the joint-process estimation section also uses QR decomposition, on a stage-by-stage basis, to perform its computations; the final product is a least-squares estimate of some desired response $d(n)$. In other words, all the computations throughout the algorithm are performed using QR decomposition.

From a numerical point of view, the QRD–LSL algorithm has some desirable properties:

1. The sines and cosines involved in applying the Givens rotations are all numerically *well behaved.*
2. The algorithm is *numerically consistent* in that, from one iteration to the next, each section of the algorithm propagates the *minimum* possible number of parameters needed for a satisfactory operation; that is, the propagation of related parameters is avoided. In particular, Table 15.3 shows that the parameters propagated by the three sections of the algorithm are as follows:

Section	Parameters propagated
Forward prediction	$\mathcal{R}_{m-1}^{1/2}(n-2)$, $p_{f,m-1}(n-1)$, $\gamma_{m-1}^{1/2}(n-1)$
Backward prediction	$\mathcal{F}_{m-1}^{1/2}(n-1)$, $p_{b,m-1}(n-1)$
Joint-process estimation	$p_{m-1}(n-1)$

3. The auxiliary parameters $p_{f,m-1}(n-1)$, $p_{b,m-1}(n-1)$, and $p_{m-1}(n-1)$, which are involved in the order updating of the angle-normalized estimation errors $\epsilon_{f,m-1}(n)$, $\epsilon_{b,m-1}(n)$, and $\epsilon_m(n)$, respectively, are all computed *directly*. That is, *local error feedback* is involved in the time-update recursions used to compute each of these auxiliary parameters. This form of feedback is another factor in assuring numerical stability of the algorithm.

There is experimental evidence for numerical stability of the QRD–LSL algorithm. In this context, we may mention computer simulation studies reported by Ling (1989), Yang and Böhme (1992), McWhirter and Proudler (1993), and Levin and Cowan (1994), all of which have demonstrated the numerical robustness of variants of the QRD–LSL algorithm. Of particular interest is the paper by Levin and Cowan (1994), in which the per-

formance of eight different adaptive filtering algorithms of the RLS family was evaluated in a finite-precision environment. The results presented therein demonstrate the superior performance of algorithms belonging to the square-root information domain (exemplified by the QRD–LSL algorithm) over those belonging to the covariance domain. Moreover, of the eight algorithms considered in the study, the QRD–LSL algorithm appeared to be the least affected by numerical imprecision.

Further evidence for numerical robustness of the QRD–LSL is presented in Capman et al. (1995). An acoustic echo canceler is described in that paper by combining a multirate scheme with a variant of the QRD–LSL algorithm. Simulation results presented therein demonstrate the numerical robustness of this solution to the echo cancelation problem.

Recursive LSL Algorithms

Turning next to *recursive least-squares lattice (LSL) algorithms*, we note from the discussion presented in Chapter 15 that these algorithms are special cases of the QRD–LSL algorithm. Indeed, they are derived by squaring the arrays of the QRD–LSL algorithm and then comparing terms. Recognizing that, in the context of numerical behavior of an algorithm, "squaring" has an opposite effect to that of "square-rooting," we may state that the performance of recursive LSL algorithms in a limited-precision environment is always *inferior* to that of the QRD–LSL algorithm from which they are derived.

A recursive LSL algorithm provides a "fast" solution to the recursive least-squares problem by employing a multistage lattice predictor for transforming the input data into a corresponding sequence of backward prediction errors. This transformation may be viewed as a form of the classical Gram–Schmidt orthogonalization procedure. The Gram–Schmidt orthogonalization is known to be *numerically inaccurate* (Stewart, 1973). Correspondingly, a conventional form of the recursive LSL algorithm (be it based on *a posteriori* or *a priori* prediction errors) has poor numerical behavior. The key to a practical method of overcoming the numerical accuracy problem in a recursive LSL algorithm is to update the forward and backward reflection coefficients *directly*, rather than first computing the individual sums of weighted forward and backward prediction errors and their cross-correlations and then taking ratios of the appropriate quantities (as in a conventional LSL algorithm). This is precisely what is done in a recursive LSL algorithm *with error feedback* (Ling and Proakis, 1984), exemplified by the algorithm summarized in Table 15.5. For a prescribed fixed-point representation, a recursive LSL algorithm with error feedback works with much more accurate values of the forward and backward reflection coefficients; these two coefficients are the key parameters in any recursive LSL algorithm. The direct computation of the forward and backward reflection coefficients therefore has the overall effect of preserving the positive definiteness of the underlying inverse correlation matrix of the input data, despite the presence of quantization errors due to finite-precision effects. Therefore, insofar as numerical performance is

concerned, recursive LSL algorithms with error feedback are preferred to their conventional forms.[9]

Gradient Adaptive Lattice Algorithm

Order-recursive adaptive filtering algorithms include the (inexact) *gradient adaptive lattice (GAL) algorithm*, in which the reflection coefficient (one per stage of the algorithm) is also updated directly (see Appendix G). In other words, there is a form of error feedback built into the operation of the GAL algorithm. In light of what has been said above, we expect the GAL algorithm to exhibit a good numerical behavior too. This is indeed borne out by the results of a computer simulation reported in (Satorius et al., 1983).

The GAL algorithm has the advantage of simplicity over recursive LSL algorithms (with or without error feedback). On the other hand, recursive LSL algorithms have an advantage over the GAL algorithm in that they provide a faster rate of convergence, because *no* approximations are made in their derivations.

17.6 FAST TRANSVERSAL FILTERS

We conclude our discussion of the numerical behavior of adaptive filtering algorithms by considering the *fast transversal filters (FTF) algorithm*. As with order-recursive adaptive filters, the FTF algorithm solves the recursive least-squares problem by exploiting the time-shift invariance property of the input data.

The FTF algorithm uses four separate transversal filters that share a common input, as indicated in the block diagram of Fig. 17.4. The four filters have distinct tasks:

- recursive forward linear prediction
- recursive backward linear prediction
- recursive computation of the gain vector
- recursive estimation of the desired response.

A summary of the FTF algorithm[10] is presented in Table 17.2; the notations used herein are the same as those described in Chapter 15. An attractive feature of this algorithm is that it permits direct computation of the coefficients of a transversal filter model.

[9]North et al. (1993) present computer simulations (using floating-point arithmetic), comparing the numerical behavior of a 32-bit directly updated recursive LSL algorithm (i.e., with error feedback) with a 32-bit indirectly updated recursive LSL algorithm. The study involved adaptive interference cancelation. In the recursive LSL algorithm with indirect updating, it was found that after about 10^5 iterations the accumulation of numerical errors resulted in a degradation of approximately 20 dB in interference cancelation, compared to the directly updated recursive LSL algorithm.

[10]For a derivation of the FTF algorithm, see Cioffi and Kailath (1984), Haykin (1991), Slock and Kailath (1993), and Sayed and Kailath (1994).

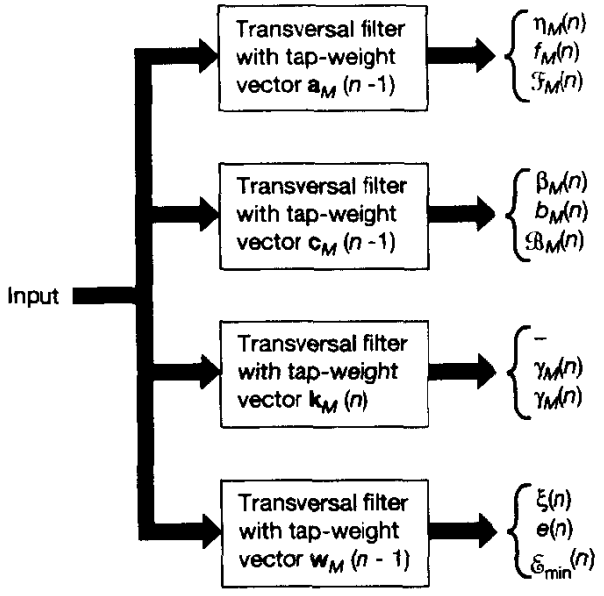


Figure 17.4 Block diagram highlighting the computations involved in the FTF algorithm, assuming a prediction order M.

Unfortunately, when the FTF algorithm is implemented in finite-precision arithmetic, numerical errors may cause the algorithm to diverge. The numerical divergence is necessarily preceded by the algorithm losing its least-squares character. In particular, the FTF algorithm contains unstable modes of propagation that are never excited in exact arithmetic, but do manifest themselves in finite precision arithmetic (Slock, 1992; Regalia, 1992).

Rescue Variable

Experimentation with the FTF algorithm indicates that a certain, positive quantity in the algorithm becomes negative due to the accumulated effect of numerical errors just before divergence of the algorithm occurs (Lin, 1984; Cioffi and Kailath, 1984). The quantity in question equals the ratio of the conversion factors for prediction errors M and $M + 1$, as shown by

$$\zeta_M(n) = \frac{\gamma_{M+1}(n)}{\gamma_M(n)} \tag{17.65}$$

We refer to $\zeta_M(n)$ as the *rescue variable*. For the ideal case of infinite prediction, we have $0 \leq \zeta_M(n) \leq 1$. A violation of this restriction on the permissible value of $\zeta_M(n)$ is due to finite-precision effects.

The rescue variable $\zeta_M(n)$ may be expressed in the equivalent form (see Problem 8)

$$\zeta_M(n) = 1 - \beta_M^*(n)\,\gamma_{M+1}(n)\,\bar{k}_{M+1,M+1}(n) \tag{17.66}$$

TABLE 17.2 SUMMARY OF THE FTF ALGORITHM

Predictions

$$\eta_M(n) = \mathbf{a}_M^H(n-1)\mathbf{u}_{M+1}(n)$$

$$f_M(n) = \gamma_M(n-1)\eta_M(n)$$

$$\mathcal{F}_M(n) = \lambda\mathcal{F}_M(n-1) + \eta_M(n)f_M^*(n)$$

$$\gamma_{M+1}(n) = \lambda\frac{\mathcal{F}_M(n-1)}{\mathcal{F}_M(n)}\gamma_M(n-1)$$

$$\bar{\mathbf{k}}_{M+1}(n) = \begin{bmatrix} 0 \\ \bar{\mathbf{k}}_M(n-1) \end{bmatrix} + \lambda^{-1}\frac{\eta_M(n)}{\mathcal{F}_M(n-1)}\mathbf{a}_M(n-1)$$

$$\mathbf{a}_M(n) = \mathbf{a}_M(n-1) - f_M^*(n)\begin{bmatrix} 0 \\ \bar{\mathbf{k}}_M(n-1) \end{bmatrix}$$

$$\beta_M(n) = \lambda\mathcal{B}_M(n-1)\bar{k}_{M+1,M+1}(n)$$

$$\gamma_M(n) = [1 - \beta_M^*(n)\gamma_{M+1}(n)\bar{k}_{M+1,M+1}(n)]^{-1}\gamma_{M+1}(n)$$

Rescue[a] variable $= [1 - \beta_M^*(n)\gamma_{M+1}(n)\bar{k}_{M+1,M+1}(n)]$

$$b_M(n) = \gamma_M(n)\beta_M(n)$$

$$\mathcal{B}_M(n) = \lambda\mathcal{B}_M(n-1) + \beta_M(n)b_M^*(n)$$

$$\begin{bmatrix} \bar{\mathbf{k}}_M(n) \\ 0 \end{bmatrix} = \bar{\mathbf{k}}_{M+1}(n) - \bar{k}_{M+1,M+1}(n)\mathbf{c}_M(n-1)$$

$$\mathbf{c}_M(n) = \mathbf{c}_M(n-1) - b_M^*(n)\begin{bmatrix} \bar{\mathbf{k}}_M(n) \\ 0 \end{bmatrix}$$

Filtering

$$\xi_M(n) = d(n) - \hat{\mathbf{w}}_M^H(n-1)\mathbf{u}_M(n)$$

$$e_M(n) = \gamma_M(n)\xi_M(n)$$

$$\hat{\mathbf{w}}_M(n) = \hat{\mathbf{w}}_M(n-1) + \bar{\mathbf{k}}_M(n)e_M^*(n)$$

Note: $\bar{k}_{M+1,M+1}(n)$ is the last element of the normalized gain vector $\bar{\mathbf{k}}_{M+1}(n)$.

[a]Rescue if variable is negative: Save $\hat{\mathbf{w}}_M(n)$ as initial condition and reuse in the augmented cost function:

$$\mathcal{E}'(n) = \mu\lambda^n\|\mathbf{w}_M(n) - \hat{\mathbf{w}}_M(n)\|^2 + \sum_{i=1}^{n}\lambda^{n-i}|e(i)|^2$$

where the factor $\mu > 0$ ensures a quadratic cost function.

where $\beta_M(n)$ is the backward *a priori* prediction error, and $\bar{k}_{M+1,M+1}(n)$ is the last element of the normalized gain vector $\bar{\mathbf{k}}_{M+1}(n)$. Equation (17.66) indicates that tight control has to be exercised on the computation of $\beta_M(n)$, $\gamma_{M+1}(n)$ [or, equivalently, $\gamma_M^{-1}(n)$], and $\bar{k}_{M+1,M+1}(n)$ in order to ensure numerical stability of the FTF algorithm.

Error Feedback

An important characteristic of the FTF algorithm is that the three quantities just mentioned [i.e., $\beta_M(n)$, $\gamma_M^{-1}(n)$, and $\bar{k}_{M+1,M+1}(n)$] may each be computed in two different ways:

- The transversal filtering of an input data vector
- The manipulation of scalar quantities

To be specific, consider the computation of the backward *a priori* prediction error $\beta_M(n)$. Obviously, we may compute $\beta_M(n)$ as

$$\beta_M^{(f)}(n) = \mathbf{c}_M^H(n-1)\mathbf{u}_{M+1}(n) \tag{17.67}$$

where $\mathbf{c}_M(n-1)$ is the tap-weight vector of the backward prediction-error filter at iteration $n-1$, and $\mathbf{u}_{M+1}(n)$ is the tap-input vector. The superscript (f) in the symbol $\beta_M^{(f)}(n)$ denotes "filtering." From the FTF algorithm summarized in Table 17.2, we see that $\beta_M(n)$ may also be computed as

$$\beta_M^{(s)}(n) = \lambda \mathcal{B}_M(n-1)\bar{k}_{M+1,M+1}(n) \tag{17.68}$$

where λ is the exponential weighting factor, $\mathcal{B}_M(n-1)$ is the minimum sum of backward *a posteriori* prediction-error squares, and $\bar{k}_{M+1,M+1}(n)$ is the last element of the normalized gain vector $\bar{\mathbf{k}}_{M+1}(n)$. The superscript (s) in the symbol $\beta_M^{(s)}(n)$ denotes "scalar manipulation."

When infinite-precision arithmetic is used, the two different ways of computing $\beta_M(n)$ described in Eqs. (17.67) and (17.68) will yield identical results. However, the results will differ from each other when finite-precision arithmetic is used. Similar remarks apply to $\gamma_M^{-1}(n)$ and $\bar{k}_{M+1,M+1}(n)$.

The difference signals resulting from the different methods of computing $\beta_M(n)$, $\gamma_M^{-1}(n)$, and $\bar{k}_{M+1,M+1}(n)$ may be viewed as output signals of the error propagation mechanism. Most importantly, they may be used in a *feedback* scheme designed for the purpose of influencing the error propagation responsible for their generation in the first place. This is indeed the idea behind *error feedback* as a method for stabilizing the FTF algorithm (Slock and Kailath, 1988, 1991; Botto and Moustakides, 1989). In particular, the difference signals may be fed back to any point in the FTF algorithm without altering the true RLS character of the algorithm, since they would vanish when exact arithmetic is used. In the error feedback scheme proposed by Slock and Kailath, the difference signals are fed back into the computation of the specific quantities they are actually associated with.

Consider, for example, the feedback stabilization of $\beta_M(n)$. Given the two finite-precision values $\beta_M^{(f)}(n)$ and $\beta_M^{(s)}(n)$ computed in accordance with Eqs. (17.67) and (17.68), respectively, we use a convex combination of these two values to define the final value of $\beta_M(n)$ as described in Slock and Kailath (1991):

$$\begin{aligned}\beta_M(n) &= \beta_M^{(s)}(n) + K(\beta_M^{(f)}(n) - \beta_M^{(s)}(n)) \\ &= K\beta_M^{(f)}(n) + (1-K)\,\beta_M^{(s)}(n)\end{aligned} \tag{17.69}$$

where K is a *feedback constant*. A similar approach may be used to stabilize the other two variables of concern: $\gamma_M^{-1}(n)$ and $\bar{k}_{M+1,M+1}(n)$.

The quantity $\beta_M(n)$ is used in several places in the FTF algorithm. This motivates the use of different values for the feedback constant K in those different plans, thereby providing more freedom in affecting the error propagation in the FTF algorithm (Slock and Kailath, 1991). Such a choice of feedback mechanism is intuitively appealing; it will make it possible to stabilize all unstable modes in error propagation, which is the ultimate goal of error feedback. The cost of stabilization is a modest increase in computational complexity, from $7M$ to $(7M + M)$, where M is the filter length.

The operation of the *stabilized FTF algorithm* may thus be summarized as follows (Slock and Kailath, 1991, 1993; Slock, 1992):

- Forward and backward linear predictions are used to make the computational complexity of the algorithm linear in the filter length M. In so doing, however, the algorithm becomes potentially unstable in a finite-precision environment.
- Error feedback is used to reintroduce redundancy of order M into the algorithm, the purpose of which is to fortify the algorithm against numerical errors incurred in the recursions.
- The error feedback mechanism is, however, only able to stabilize the FTF algorithm for a restricted range of the exponential weighting factor λ, defined by:

$$1 - \frac{1}{2M} < \lambda < 1 \tag{17.70}$$

where M is the length of the transversal filter. The permissible range of values of λ described in (17.70) covers some useful choices of interest in practice.

17.7 SUMMARY AND DISCUSSION

In this chapter we discussed the numerical stability of the LMS and RLS families of adaptive filtering algorithms.

The LMS algorithm is *numerically robust*. When operating in a limited-precision environment, the point to note is that the step-size parameter μ may only be decreased to a level at which the degrading effects of round-off errors in the tap weights of the finite-precision LMS algorithm become significant. Moreover, a finite-precision implementation of the LMS algorithm may be improved by incorporating *leakage* into the algorithm.

As for the RLS family, the use of square-root filtering provides a powerful technique for realizing a robust numerical behavior. For adaptive beamforming, which is spatial in nature, a good choice is the QR–RLS algorithm, which is the RLS version of the square-root (Kalman) information filter. On the other hand, for temporal applications of adaptive filters where we can exploit the time-shift invariance property of the input data, a good choice is the order-recursive QRD–LSL algorithm, which is also rooted in square-root

information filtering. In light of the numerical stability analysis of the RLS algorithm, we may say the following in the context of the QR–RLS, inverse QR–RLS, and QRD–LSL algorithms:

- The propagation of a single error is exponentially stable, provided that the exponential weighting factor λ is less than unity. This behavior appears to guarantee a "small" overall accumulation of error in the algorithm.
- When $\lambda = 1$, the error propagation is not contractive, with the result that the accumulation of numerical errors may lead to divergence of the algorithm.

Having said this, however, we lack a unified treatment of finite-precision effects in the RLS family of adaptive filtering algorithms that goes beyond a single error propagation.

In the literature, it is sometimes argued that square-root filters are (1) expensive and (2) awkward to calculate, constituting a bottleneck for overall performance. For these reasons, square-root free versions of the QR–RLS and QRD–LSL algorithms have been formulated, using special methods for performing Givens rotations without square roots, the idea for which dates back to Gentleman (1973). In the case of Kalman filtering, the method of UD-factorization (Bierman, 1977) was developed specifically for avoiding the actual use of square roots. It now appears that square-root free algorithms actually introduce a number of problems (Stewart and Chapman, 1990):

- Square-root free algorithms may become numerically unstable, and potentially suffer from serious overflow/underflow problems.
- Based on the knowledge that square roots are simpler (or even equally as complex) as divider arrays, the reformulation of standard adaptive filtering algorithms in square-root free form may actually increase arithmetic complexity.

Another noteworthy point is that RLS adaptive filtering algorithms requiring the use of square roots can be programmed very efficiently on *CORDIC processors*, in which square root operations make no explicit appearance. Here we recognize that most of these algorithms are decomposable in the two basic operations of rotation and vectoring. These two operations are indeed fundamental to a CORDIC processor (Volder, 1959):

1. *Rotation.* In the rotation mode, the CORDIC processor is given the coordinates of a two-element vector and an angle of rotation. The processor then computes the coordinate components of the original vector after rotation through the desired angle.
2. *Vectoring.* In the vectoring mode, the coordinates of a two-element vector are given. The CORDIC processor then rotates the vector until the angular argument is zero. The angle of rotation in this second mode is therefore the negative of the original angular argument. In other words, the vectoring operation is equivalent to the annihilation of an element of a two-element vector.

Apparently, a CORDIC processor yields the fastest implementation of these two basic operations. Moreover, its integral circuit implementation in the form of silicon chips makes its use for the implementation of RLS algorithms involving square roots all the more attractive.[11]

PROBLEMS

1. Consider the digital implementation of the LMS algorithm using fixed-point arithmetic, as discussed in Section 17.2. Show that the M-by-1 error vector $\Delta\hat{\mathbf{w}}(n)$, incurred by quantizing the tap-weight vector, may be updated as follows:

$$\Delta\hat{\mathbf{w}}(n+1) = \mathbf{F}(n)\Delta\mathbf{w}(n) + \mathbf{t}(n), \qquad n = 0, 1, 2, \ldots$$

   where $\mathbf{F}(n)$ is an M-by-M matrix and $\mathbf{t}(n)$ is an M-by-1 vector. Hence, define $\mathbf{F}(n)$ and $\mathbf{t}(n)$. Base your analysis on real-valued data.

2. Using the results of Problem 1, and invoking the independence assumption of Chapter 9, show that

$$E[\Delta\hat{\mathbf{w}}(n)] = \mathbf{0}$$

3. Consider two transversal filters I and II, both of length M. Filter I has all of its tap inputs as well as tap weights represented in infinite-precision form. Filter II is identical to filter I, except for the fact that its tap weights are represented in finite-precision form. Let $y_{\mathrm{I}}(n)$ and $y_{\mathrm{II}}(n)$ denote the respective filter outputs for the tap-inputs $u(n), u(n-1), \ldots, u(n-M-1)$. Define the error

$$\epsilon(n) = y_{\mathrm{I}}(n) - y_{\mathrm{II}}(n)$$

   Assuming that the inputs $u(n)$ are independent random variables with a common rms value equal to A_{rms}, show that the mean-square value of the error $\epsilon(n)$ is

$$E[\epsilon^2(n)] = A_{\mathrm{rms}}^2 \sum_{i=0}^{M-1} (w_i - w_{iq})^2$$

   where w_{iq} is the quantized version of the tap weight w_i.

4. Consider an LMS algorithm with 17 taps and a step-size parameter $\mu = 0.07$. The input data stream has an rms value of unity.
   (a) Given the use of a quantization process with 12-bit accuracy, calculate the corresponding value of the digital residual error.
   (b) Suppose the only source of output error is that due to quan ization of the tap weights. Using the result of Problem 3, calculate the rms value of the resulting measurement error in the output. Compare this error with the digital residual error calculated in part (a).

5. Demonstrate the Hermitian symmetry-preserving property of the RLS algorithm summarized in Table 17.1. Assume that a single quantization error is made at time $n-1$, as shown by

$$\mathbf{P}_q(n-1) = \mathbf{P}(n-1) + \boldsymbol{\eta}_p(n-1)$$

[11] Rader (1990) describes a linear systolic array for adaptive beamforming based on the Cholesky factorization, which uses the CORDIC processor for its hardware implementations in VSLI form.

where $\mathbf{P}_q(n-1)$ is the quantized value of the matrix $\mathbf{P}(n-1)$ and $\boldsymbol{\eta}_p(n-1)$ is the quantization error matrix.

6. In Eqs. (17.24) and (17.48) we presented two different ways of defining the condition for a tap-output vector $\mathbf{u}(n)$ to be persistently exciting. Reconcile these two conditions.
7. It may be argued that the extended QRD–LSL algorithm described in Section 15.10 is potentially unstable in a limited-precision environment. Yet the recursive LSL algorithm with error feedback derived from the extended QRD–LSL algorithm in Section 15.12 has good numerical properties. Explain the rationale for these two statements.
8. Equation (17.66) defines a formula for the rescue variable $\zeta_M(n)$ that arises in the FTF algorithm. Derive the formula.