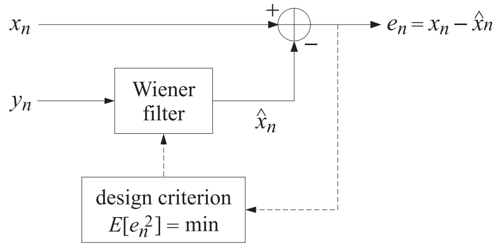
16 Adaptive Filters

16.1 Adaptive Implementation of Wiener Filters

We review briefly the solution of the Wiener filtering problem.



The general solution does not place any a priori restriction on the order of the Wiener filter. In general, an infinite number of weights is required to achieve the lowest estimation error. However, in adaptive implementations we must insist in advance that the number of filter weights be finite. This is so because the adaptation algorithm adapts each weight individually. Obviously, we cannot adapt an infinite number of weights. We will assume then, that the optimal Wiener filter is an FIR filter, say with $M + 1$ weights

$$\mathbf{h} = [h_0, h_1, h_2, \dots, h_M]^T, \quad H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + \cdots + h_M z^{-M}$$

This filter processes the available observations y_n to produce the estimate

$$\hat{x}_n = \sum_{m=0}^{M} h_m y_{n-m} = h_0 y_n + h_1 y_{n-1} + h_2 y_{n-2} + \cdots + h_M y_{n-M}$$

The weights h_m are chosen optimally so that the mean-square estimation error is minimized; that is,

$$\mathcal{E} = E[e_n^2] = \min, \quad e_n = x_n - \hat{x}_n$$

This minimization criterion leads to the orthogonality equations, which are the determining equations for the optimal weights. Writing the estimate in vector notation

$$\hat{x}_n = [h_0, h_1, \dots, h_M] \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-M} \end{bmatrix} = \mathbf{h}^T \mathbf{y}(n)$$

we may write the orthogonality equations as

$$E[e_n y_{n-m}] = 0, \quad 0 \le m \le M$$

or, equivalently,

$$E[e_n \mathbf{y}(n)] = 0$$

These give the normal equations

$$E[(x_n - \hat{x}_n)\mathbf{y}(n)] = E\big[(x_n - \mathbf{h}^T\mathbf{y}(n))\mathbf{y}(n)\big] = 0, \quad \text{or,}$$

$$E[\mathbf{y}(n)\mathbf{y}(n)^T]\mathbf{h} = E[x_n \mathbf{y}(n)], \quad \text{or,}$$

$$R\,\mathbf{h} = \mathbf{r}, \quad R = E[\mathbf{y}(n)\mathbf{y}(n)^T], \quad \mathbf{r} = E[x_n\mathbf{y}(n)]$$

The optimal weights are obtained then by

$$\mathbf{h} = R^{-1}\mathbf{r} \tag{16.1.1}$$

The corresponding minimized value of the estimation error is computed by

$$\begin{aligned}\mathcal{E} &= E[e_n^2] = E[e_n(x_n - \mathbf{h}^T\mathbf{y}(n))] = E[e_n x_n] = E\big[(x_n - \mathbf{h}^T\mathbf{y}(n))x_n\big] \\ &= E[x_n^2] - \mathbf{h}^T E[\mathbf{y}(n)x_n] = E[x_n^2] - \mathbf{h}^T\mathbf{r} = E[x_n^2] - \mathbf{r}^T R^{-1}\mathbf{r}\end{aligned}$$

The normal equations, and especially the orthogonality equations, have their usual *correlation canceling* interpretations. The signal x_n being estimated can be written as

$$x_n = e_n + \hat{x}_n = e_n + \mathbf{h}^T\mathbf{y}(n)$$

It is composed of two parts, the term e_n which because of the orthogonality equations is entirely uncorrelated with $\mathbf{y}(n)$, and the second term, which is correlated with $\mathbf{y}(n)$. In effect, the filter removes from x_n any part of it that is correlated with the secondary input $\mathbf{y}(n)$; what is left, e_n, is uncorrelated with $\mathbf{y}(n)$. The Wiener filter acts as a correlation canceler. If the primary signal x_n and the secondary signal $\mathbf{y}(n)$ are in any way correlated, the filter will cancel from the output e_n any such correlations.

One difficulty with the above solution is that the statistical quantities R and $\mathbf{r}$ must be known, or at least estimated, in advance. This can be done either by block processing or adaptive processing methods. The principal advantages of block processing methods are that the design is based on a single, fixed, data record and that the length of the data record may be very short. Thus, such methods are most appropriate in applications where the *availability* of data is limited, as for example, in parametric spectrum estimation based on a single block of data, or in deconvolution applications where the data to be deconvolved are already available, for example, a still distorted picture or a recorded segment of a seismic response.

Availability of data, however, is not the only consideration. In a *changing environment*, even if more data could be collected, it may not be correct to use them in the design because stationarity may not be valid for the longer data block. Block processing methods can still be used in such cases, but the optimum filters must be *redesigned* every time the environment changes, so that the filter is always matched to the data being processed by it. This is, for example, what is done in speech processing. The input speech signal is divided into fairly short segments, with each segment assumed to arise from a stationary process, then the statistical correlations are estimated by sample correlations and the optimal prediction coefficients corresponding to each segment are computed. In a sense, this procedure is data-adaptive, but more precisely, it is block-by-block adaptive.

In other applications, however, we do not know how often to redesign and must use *adaptive* implementations that provide an automatic way of redesigning the optimum processors to continually track the environment. For example, communications and radar antennas are vulnerable to jamming through their sidelobes. Adaptive sidelobe cancelers continuously adjust themselves to steer nulls toward the jammers even when the jammers may be changing positions or new jammers may be coming into play. Another example is the equalization of unknown or changing channels, or both. In switched telephone lines the exact transmission channel is not known in advance but is established at the moment the connection is made. Similarly, in fading communications channels the channel is continuously changing. To undo the effects of the channel, such as amplitude and phase distortions, an equalizer filter must be used at the receiving end that effectively acts as an inverse to the channel. Adaptive equalizers determine automatically the characteristics of the channel and provide the required inverse response. Other applications, well-suited to adaptive implementations, are noise canceling, echo canceling, linear prediction and spectrum estimation, and system identification and control.

In this chapter we discuss several adaptation algorithms, such as the Widrow-Hoff least mean square (LMS) algorithm, the conventional recursive least squares (RLS) algorithm, the fast RLS algorithms, and the adaptive lattice algorithms and present some of their applications [1341–1349]. A typical adaptive implementation of a Wiener filter is depicted in Fig. 16.1.1.

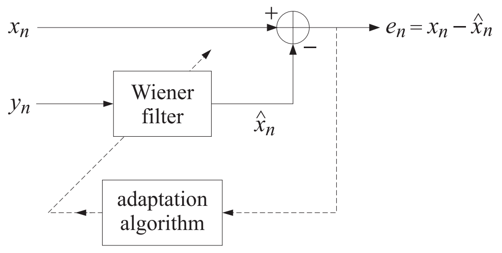


Fig. 16.1.1 Adaptive Wiener filter.

The adaptation algorithm continuously monitors the output error signal e_n and attempts to minimize the output power $E[e_n^2]$, or, equivalently tries to decorrelate e_n from the secondary input y_n. At each time instant n, the current values of the weights are used to perform the filtering operation. The computed output e_n is then used by the adaptation part of the algorithm to change the weights in the direction of their optimum values. As processing of the input signals x_n and y_n takes place and the filter gradually learns the statistics of these inputs, its weights gradually converge to their optimum values given by the Wiener solution (16.1.1). Clearly, the input statistics must remain unchanged for at least as long as it takes the filter to learn it and converge to its optimum configuration. If, after convergence, the input statistics should change, the filter will respond by readjusting its weights to their new optimum values, and so on. In other words, the adaptive filter will track the non-stationary changes of the input statistics as long as such changes occur slowly enough for the filter to converge between changes. The three basic issues in any adaptive implementation are:

1. The learning or convergence *speed* of the algorithm.
2. The computational *complexity* of the algorithm.
3. The numerical *accuracy and stability* of the algorithm.

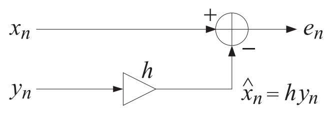
The convergence speed is an important factor because it determines the maximum rate of change of the input non-stationarities that can be usefully tracked by the filter. The computational complexity refers to the number of operations required to update the filter from one time instant to the next. The table below shows how various adaptive algorithms fare under these requirements.

algorithm	speed	complexity	stability
LMS	slow	simple	stable
RLS	fast	complex	stable
Fast RLS	fast	simple	unstable
Lattice	fast	simple	stable

Only adaptive lattice algorithms satisfy all three requirements. We will discuss these algorithms in detail later on. In the next section we begin with the LMS algorithm because it is the simplest and most widely used. We finish this section with the obvious remark that adaptive or block processing optimal filter designs, regardless of type, cannot do any better than the theoretical Wiener solution. The optimal filter, therefore, should be first analyzed theoretically to determine if it is worth using it in the application at hand.

16.2 Correlation Canceler Loop (CCL)

To illustrate the basic principles behind adaptive filters, consider the simplest possible filter, that is, a filter with only one weight



The weight h must be selected optimally so as to produce the best possible estimate of x_n:

$$\hat{x}_n = hy_n$$

The estimation error is expressed as

$$\begin{aligned}\mathcal{E} &= E[e_n^2] = E\big[(x_n - hy_n)^2\big] = E[x_n^2] - 2hE[x_n y_n] + E[y_n^2]h^2 \\ &= E[x_n^2] - 2hr + Rh^2\end{aligned} \tag{16.2.1}$$

The minimization condition is

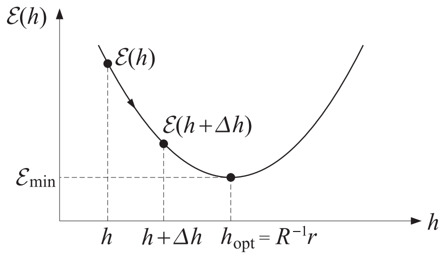
$$\frac{\partial\mathcal{E}}{\partial h} = 2E\left[e_n\frac{\partial e_n}{\partial h}\right] = -2E[e_n y_n] = -2r + 2Rh = 0 \tag{16.2.2}$$

which gives the optimum solution $h_{\rm opt} = R^{-1}r$, and also shows the correlation cancellation condition $E[e_n y_n] = 0$. The adaptive implementation is based on solving the equation

$$\frac{\partial\mathcal{E}}{\partial h} = 0 \tag{16.2.3}$$

iteratively, using a gradient-descent method. The dependence of the error $\mathcal{E}$ on the filter parameter h is parabolic, with an absolute minimum occurring at the above optimal value $h_{\rm opt} = R^{-1}r$.

This is shown below



In the adaptive version, the filter parameter h is made *time-dependent*, $h(n)$, and is updated from one time instant to the next as follows

$$h(n+1) = h(n) + \Delta h(n) \tag{16.2.4}$$

where $\Delta h(n)$ is a correction term that must be chosen properly in order to ensure that eventually the time-varying weight $h(n)$ will converge to the optimal value:

$$h(n) \to h_{\rm opt} = R^{-1}r \quad \text{as} \quad n \to \infty$$

The filtering operation is now given by the still *linear* but *time non-invariant* form

$$\hat{x}_n = h(n)y_n \tag{16.2.5}$$

The computation of the estimate at the next time instant should be made with the new weight, that is,

$$\hat{x}_{n+1} = h(n+1)y_{n+1}$$

and so on. The simplest way to choose the correction term $\Delta h(n)$ is the gradient-descent, or steepest-descent, method. The essence of the method is this: It is required that the change $h \to h + \Delta h$ must move the performance index closer to its minimum than before, that is, Δh must be such that

$$\mathcal{E}(h + \Delta h) \le \mathcal{E}(h)$$

Therefore, if we always demand this, the repetition of the procedure will lead to smaller and smaller values of $\mathcal{E}$ until the smallest value has been attained. Assuming that Δh is sufficiently small, we may expand to first order and obtain the condition

$$\mathcal{E}(h) + \Delta h \frac{\partial \mathcal{E}(h)}{\partial h} \le \mathcal{E}(h)$$

If Δh is selected as the *negative gradient* $-\mu(\partial\mathcal{E}/\partial h)$ then this inequality will be guaranteed, that is, if we choose

$$\Delta h = -\mu \frac{\partial \mathcal{E}(h)}{\partial h} \tag{16.2.6}$$

then the inequality is indeed satisfied:

$$\mathcal{E}(h) + \Delta h \frac{\partial \mathcal{E}(h)}{\partial h} = \mathcal{E}(h) - \mu \left| \frac{\partial \mathcal{E}(h)}{\partial h} \right|^2 \le \mathcal{E}(h)$$

The adaptation parameter μ must be small enough to justify keeping only the first-order terms in the above Taylor expansion. Applying this idea to our little adaptive filter, we choose the correction $\Delta h(n)$ according to Eq. (16.2.6), so that

$$h(n+1) = h(n) + \Delta h(n) = h(n) - \mu \frac{\partial \mathcal{E}(h(n))}{\partial h} \tag{16.2.7}$$

Using the expression for the gradient $\dfrac{\partial \mathcal{E}(h)}{\partial h} = -2r + 2Rh$, we find

$$\begin{aligned} h(n+1) &= h(n) - \mu\big[-2r + 2Rh(n)\big] \\ &= (1 - 2\mu R)h(n) + 2\mu r \end{aligned}$$

This difference equation may be solved in closed form. For example, using z-transforms with any initial conditions $h(0)$, we find

$$h(n) = h_{\rm opt} + (1 - 2\mu R)^n \big(h(0) - h_{\rm opt}\big) \tag{16.2.8}$$

where $h_{\rm opt} = R^{-1}r$. The coefficient $h(n)$ will converge to its optimal value $h_{\rm opt}$, regardless of the starting value $h(0)$, provided μ is selected such that

$$|1 - 2\mu R| < 1$$

or, $-1 < 1 - 2\mu R < 1$, or since μ must be positive (to be in the negative direction of the gradient), μ must satisfy

$$0 < \mu < \frac{1}{R} \tag{16.2.9}$$

To select μ, one must have some a priori knowledge of the magnitude of the input variance $R = E[y_n^2]$. Such choice for μ will guarantee convergence, but the speed of convergence is controlled by how close the number $1 - 2\mu R$ is to one. The closer it is to unity, the slower the speed of convergence. As μ is selected closer to zero, the closer $1 - 2\mu R$ moves towards one, and thus the slower the convergence rate. Thus, the adaptation parameter μ must be selected to be small enough to guarantee convergence but not too small to cause a very slow convergence.

16.3 The Widrow-Hoff LMS Adaptation Algorithm

The purpose of the discussion in Sec. 16.2 was to show how the original Wiener filtering problem could be recast in an iterative form. From the practical point of view, this reformulation is still not computable since the adaptation of the weights requires a priori knowledge of the correlations R and r. In the Widrow-Hoff algorithm the above adaptation algorithm is replaced with one that is computable [1341,1342]. The gradient that appears in Eq. (16.2.7)

$$h(n+1) = h(n) - \mu \frac{\partial \mathcal{E}(h(n))}{\partial h}$$

is replaced by an *instantaneous* gradient by *ignoring* the expectation instructions, that is, the theoretical gradient

$$\frac{\partial \mathcal{E}(h(n))}{\partial h} = -2E[e_n y_n] = -2r + 2Rh(n) = -2E[x_n y_n] + 2E[y_n^2]h(n)$$

is replaced by

$$\frac{\partial \mathcal{E}}{\partial h} = -2e_n y_n = -2\big(x_n - h(n)y_n\big)y_n = -2x_n y_n + 2y_n^2 h(n) \tag{16.3.1}$$

so that the weight-adjustment algorithm becomes

$$h(n+1) = h(n) + 2\mu e_n y_n \tag{16.3.2}$$

In summary, the required computations are done in the following order:

1. At time n, the filter weight $h(n)$ is available.

2. Compute the filter output $\hat{x}_n = h(n)y_n$.
3. Compute the estimation error $e_n = x_n - \hat{x}_n$.
4. Compute the next filter weight $h(n+1) = h(n) + 2\mu e_n y_n$.
5. Go to next time instant $n \to n+1$.

The following remarks are in order:

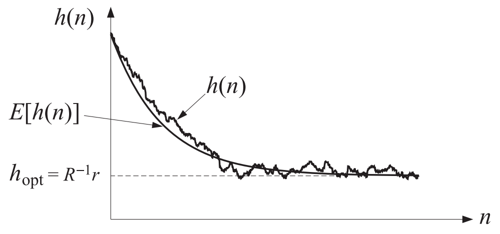
1. The output error e_n is fed back and used to control the adaptation of the filter weight $h(n)$.
2. The filter tries to decorrelate the secondary signal from the output e_n. This, is easily seen as follows: If the weight $h(n)$ has more or less reached its optimum value, then $h(n+1) \simeq h(n)$, and the adaptation equation implies also approximately that $e_n y_n \simeq 0$.
3. Actually, the weight $h(n)$ never really reaches the theoretical limiting value $h_{\text{opt}} = R^{-1}r$. Instead, it stabilizes about this value, and continuously fluctuates about it.
4. The approximation of ignoring the expectation instruction in the gradient is known as the *stochastic approximation*. It complicates the mathematical aspects of the problem considerably. Indeed, the difference equation

$$h(n+1) = h(n) + 2\mu e_n y_n = h(n) + 2\mu\big(x_n - h(n)y_n\big)y_n$$

makes $h(n)$ depend on the random variable y_n in highly nonlinear fashion, and it is very difficult to discuss even the average behavior of $h(n)$.
5. In discussing the average behavior of the weight $h(n)$, the following approximation is typically (almost invariably) made in the literature

$$\begin{aligned} E\big[h(n+1)\big] &= E\big[h(n)\big] + 2\mu E[x_n y_n] - 2\mu E\big[h(n)y_n^2\big] \\ &= E\big[h(n)\big] + 2\mu E[x_n y_n] - 2\mu E\big[h(n)\big]E\big[y_n^2\big] \\ &= E\big[h(n)\big] + 2\mu r - 2\mu E\big[h(n)\big]R \end{aligned}$$

where in the last term, the expectation $E\big[h(n)\big]$ was factored out, as though $h(n)$ were independent of y_n. With this approximation, the average $E\big[h(n)\big]$ satisfies the same difference equation as before with solution given by Eq. (16.2.8). Typically, the weight $h(n)$ will be fluctuating about the theoretical convergence curve as it converges to the optimal value, as shown below



After convergence, the adaptive weight $h(n)$ continuously fluctuates about the Wiener solution h_{opt}. A measure of these fluctuations is the mean-square deviation of $h(n)$ from h_{opt}, that is, $E\big[(h(n) - h_{\text{opt}})^2\big]$. Under some restrictive conditions, this quantity has been calculated [1350] to be

$$E\big[(h(n) - h_{\text{opt}})^2\big] \to \mu\mathcal{E}_{\min} \quad \text{(for large } n\text{)}$$

where $\mathcal{E}_{\min}$ is the minimized value of the performance index (16.2.1). Thus, the adaptation parameter μ controls the size of these fluctuations. This gives rise to the basic trade-off of the LMS algorithm: to obtain high accuracy in the converged weights (small fluctuations), a small value of μ is required, but this will slow down the convergence rate.

A realization of the CCL is shown in Fig. 16.3.1. The filtering part of the realization must be clearly distinguished from the feedback control loop that performs the adaptation of the filter weight.

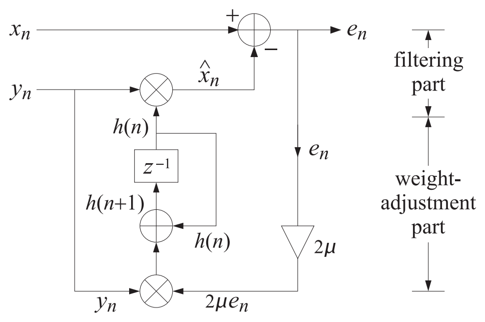


Fig. 16.3.1 Correlation canceler loop.

Historically, the correlation canceler loop was introduced in adaptive antennas as a *sidelobe canceler* [1351–1356] The CCL is the simplest possible adaptive filter, and forms the elementary *building block* of more complicated, higher-order adaptive filters.

We finish this section by presenting a simulation example of the CCL loop. The primary signal x_n was defined by

$$x_n = -0.8y_n + u_n$$

where the first term represents that part of x_n which is correlated with y_n. The part u_n is not correlated with y_n. The theoretical value of the CCL weight is found as follows:

$$r = E[x_n y_n] = -0.8E[y_n y_n] + E[u_n y_n] = -0.8R + 0 \quad \Rightarrow \quad h_{\text{opt}} = R^{-1}r = -0.8$$

The corresponding output of the CCL will be $\hat{x}_n = h_{\text{opt}} y_n = -0.8y_n$, and therefore it will completely cancel the first term of x_n leaving at the output $e_n = x_n - \hat{x}_n = u_n$.

In the simulation we generated 1000 samples of a zero-mean white-noise signal y_n of variance 0.1, and another independent set of 1000 samples of a zero-mean white-noise signal u_n also of variance 0.1, and computed x_n. The adaptation algorithm was initialized, as is usually done, to zero initial weight $h(0) = 0$. Fig. 16.3.2 shows the transient behavior of the adaptive weight $h(n)$, as well as the theoretical weight $E\big[h(n)\big]$, as a function of the number of iterations n, for the two values of μ, $\mu = 0.03$ and $\mu = 0.01$.

Note that in both cases, the adaptive weight converges to the theoretical value $h_{\text{opt}} = -0.8$, and that the smaller μ is slower but the fluctuations are also smaller. After the adaptive weight has reached its asymptotic value, the CCL begins to operate optimally, removing the correlated part of x_n from the output e_n.

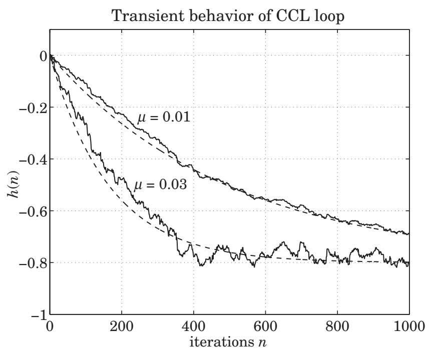
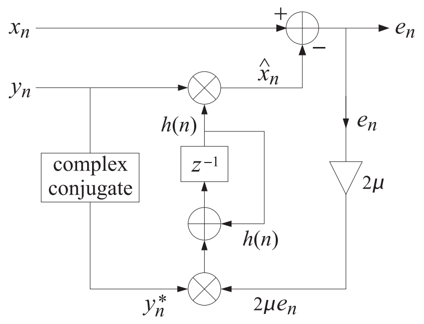


Fig. 16.3.2 Transient behavior of theoretical (dashed) and adaptive weights $h(n)$.

Later on we will consider the complex-valued version of adaptive Wiener filters. Their elementary building block is the complex CCL shown below



The performance index is now

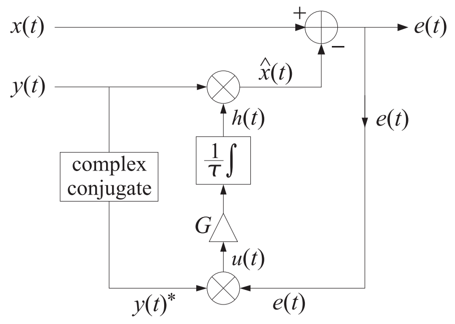
$$\mathcal{E} = E\big[|e_n|^2\big] = E\big[|x_n - hy_n|^2\big] = \min$$

with optimum solution

$$h_{\rm opt} = R^{-1}r\,,\quad R = E[y_n^* y_n]\,,\quad r = E[x_n y_n^*]$$

Analog implementations of the CCL are used in adaptive antennas. An analog CCL is shown

below



where a high gain amplifier G and an ordinary RC-type integrator are used. If τ denotes the RC time constant of the integrator, the weight updating part of the CCL is

$$\tau\dot{h}(t)+h(t)= Gu(t)= Ge(t)y^*(t)$$

The performance of the analog CCL can be analyzed by replacing the adaptive weight $h(t)$ by its statistical average, satisfying

$$\tau\dot{h}(t)+h(t)= GE\big[e(t)y^*(t)\big] = GE\big[\big(x(t)-h(t)y(t)\big)y^*(t)\big]$$

or, defining $R = E\big[y(t)y^*(t)\big]$ and $r = E\big[x(t)y^*(t)\big]$,

$$\tau\dot{h}(t)+h(t)= Gr - GRh(t)$$

with solution for $t \geq 0$:

$$h(t)= h_{\rm opt} + \big(h(0)-h_{\rm opt}\big)e^{-at}$$

where $h_{\rm opt}$ is the asymptotic value

$$h_{\rm opt} = (1 + GR)^{-1}Gr$$

Thus, a high gain G is needed to produce an asymptotic value close to the theoretical Wiener solution $R^{-1}r$. The time constant of adaptation is given by

$$\frac{1}{a} = \frac{\tau}{1 + GR}$$

Note that this particular implementation always converges and the speed of convergence is still inversely dependent on R.

16.4 Adaptive Linear Combiner

A straightforward generalization of the correlation canceler loop is the adaptive linear combiner, where one has available a main signal x_n and a number of secondary signals $y_m(n)$, $m = 0, 1, \dots, M$. These $(M + 1)$ secondary signals are to be linearly combined with appropriate weights $h_0, h_1, \dots, h_M$ to form an estimate of x_n:

$$\hat{x}_n = h_0 y_0(n)+h_1 y_1(n)+\cdots + h_M y_M(n)= [h_0, h_1, \dots, h_M]\begin{bmatrix} y_0(n) \\ y_1(n) \\ \vdots \\ y_M(n)\end{bmatrix} = \mathbf{h}^T\mathbf{y}(n)$$

A realization of this is shown in Fig. 16.4.1. The adaptive linear combiner is used in adaptive radar and sonar arrays [1351–1355]. It also encompasses the case of the ordinary FIR, or transversal, Wiener filter [1342].

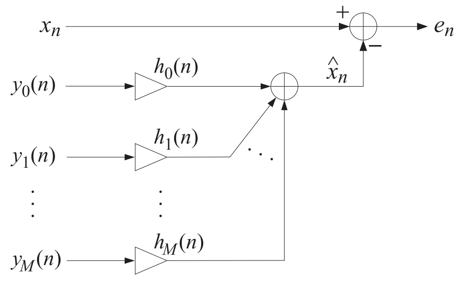


Fig. 16.4.1 Linear combiner.

The optimal weights h_m minimize the estimation error squared

$$\mathcal{E} = E[e_n^2] = \min\,, \quad e_n = x_n - \hat{x}_n$$

The corresponding orthogonality equations state that the estimation error be orthogonal (decorrelated) to each secondary signal $y_m(n)$:

$$\frac{\partial\mathcal{E}}{\partial h_m} = 2E\left[e_n\frac{\partial e_n}{\partial h_m}\right] = -2E[e_n y_m(n)] = 0\,, \quad 0 \le m \le M$$

or, in vector form

$$E[e_n\mathbf{y}(n)] = 0 \quad\Rightarrow\quad E[x_n\mathbf{y}(n)] - E[\mathbf{y}(n)\mathbf{y}^T(n)]\mathbf{h} = \mathbf{r} - R\mathbf{h} = 0$$

with optimum solution $\mathbf{h}_{\rm opt} = R^{-1}\mathbf{r}$.

The adaptive implementation is easily obtained by allowing the weights to become time-dependent, $\mathbf{h}(n)$, and updating them in time according to the gradient-descent algorithm

$$\mathbf{h}(n+1) = \mathbf{h}(n) - \mu\,\frac{\partial\mathcal{E}(\mathbf{h}(n))}{\partial\mathbf{h}}$$

with instantaneous gradient

$$\frac{\partial\mathcal{E}}{\partial\mathbf{h}} = -2E[e_n\mathbf{y}(n)] \to -2e_n\mathbf{y}(n)$$

so that

$$\mathbf{h}(n+1) = \mathbf{h}(n) + 2\mu e_n\mathbf{y}(n)$$

or, component-wise

$$h_m(n+1) = h_m(n) + 2\mu e_n y_m(n)\,, \quad 0 \le m \le M \tag{16.4.1}$$

The computational algorithm is summarized below:

1. $\hat{x}_n = h_0(n)y_0(n) + h_1(n)y_1(n) + \cdots + h_M(n)y_M(n)$
2. $e_n = x_n - \hat{x}_n$
3. $h_m(n+1) = h_m(n) + 2\mu e_n y_m(n)\,, \quad 0 \le m \le M$

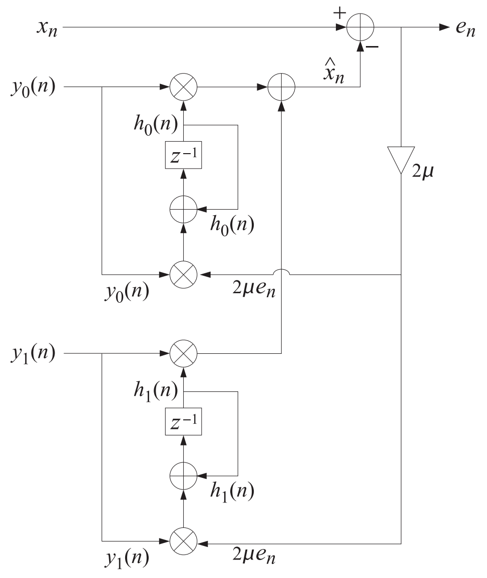


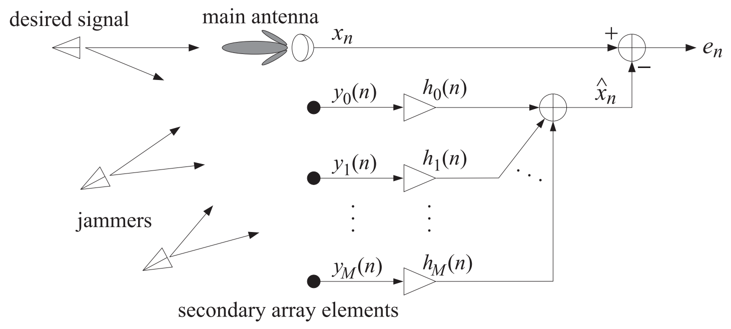
Fig. 16.4.2 Adaptive linear combiner.

It is evident that each weight $h_m(n)$ is being adapted by its own correlation canceler loop, while all weights use the same feedback error e_n to control their loops. The case of two weights $(M = 1)$ is shown in Fig. 16.4.2.

The adaptive linear combiner has two major applications:

1. Adaptive sidelobe canceler.
2. Adaptive FIR Wiener filter.

The two cases differ only in the way the inputs to the linear combiner are supplied. The linear combiner part, performing the optimum processing, is the same in both cases. The time series case is discussed in the next section. The array problem is depicted below.



It consists of a main and a number of secondary antennas. The main antenna is highly directional and oriented toward the desired signal. Jammers picked up by the sidelobes of the main antenna and by the secondary antennas will tend to be canceled because the adaptive linear combiner, acting as a correlation canceler, will adjust itself to cancel that part of the main signal that is correlated with the secondary ones. The desired signal may also be canceled partially if it is picked up by the secondary antennas. Strong jammers, however, will generally dominate and as a result the canceler will configure itself to cancel them. The cancellation of the desired signal can also be prevented by imposing additional constraints on the filter weights that can sustain the beam in the desired look-direction.

The adaptation speed of the adaptive canceler is affected by the relative power levels of the jammers. If there are jammers with greatly differing powers, the overall adaptation speed may be slow. The stronger jammers tend to be canceled faster; the weaker ones more slowly. Qualitatively this may be understood by inspecting, for example, expression (14.2.32). The power levels P_i of the plane waves act as *penalty* factors in the performance index, that is, the minimization of the performance index will tend to favor first the largest terms in the sum. This limitation of the LMS algorithm has led to the development of alternative algorithms, such as adaptive Gram-Schmidt preprocessors or RLS, in which all jammers get canceled equally fast.

16.5 Adaptive FIR Wiener Filter

The adaptive FIR or transversal filter is a special case of the adaptive linear combiner. In this case, there is only one secondary signal y_n. The required $M+1$ signals $y_m(n)$ are provided as delayed replicas of y_n, that is,

$$y_m(n)=y_{n-m} \tag{16.5.1}$$

A realization is shown in Fig. 16.5.1. The estimate of x_n is

$$\hat{x}_n=\sum_{m=0}^{M}h_m(n)y_{n-m}=h_0(n)y_n+h_1(n)y_{n-1}+\cdots+h_M(n)y_{n-M}$$

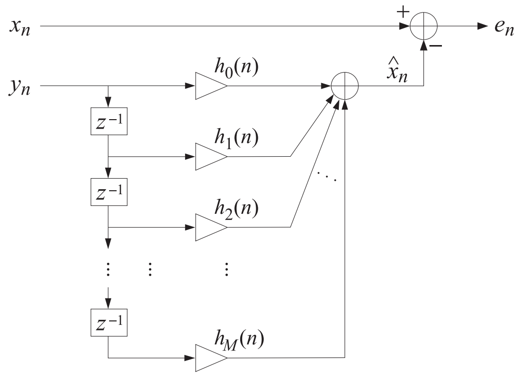


Fig. 16.5.1 Adaptive FIR Wiener filter.

The time-varying filter weights $h_m(n)$ are continuously updated according to the gradient-descent LMS algorithm

$$h_m(n+1)=h_m(n)+2\mu e_n y_m(n)\,,\quad\text{or,}$$
$$h_m(n+1)=h_m(n)+2\mu e_n y_{n-m}\,,\quad 0\le m\le M \tag{16.5.2}$$

Each weight is therefore updated by its own CCL. Again, we summarize the computational steps:

1. Compute the estimate $\hat{x}_n=\sum_{m=0}^{M}h_m(n)y_{n-m}$
2. Compute the error signal $e_n=x_n-\hat{x}_n$
3. Adjust the weights $h_m(n+1)=h_m(n)+2\mu e_n y_{n-m}\,,\quad 0\le m\le M$

The function **lms** is an implementation of the algorithm. With a minor modification it can also be used for the more general adaptive linear combiner. Each call to the function reads a pair of input samples $\{x_n,y_n\}$, performs the filtering operation to produce the output pair $\{\hat{x}_n,e_n\}$, updates the filter coefficients $h_m(n)$ to their new values $h_m(n+1)$ to be used by the next call, and updates the internal state of the filter. It is essentially the function **dwf** with the weight adaptation part added to it.

Next, we present the same simulation example as that given in Section Sec. 16.3, but it is now approached with a two-tap adaptive filter $(M=1)$. The filtering equation is in this case

$$\hat{x}_n=h_0(n)y_n+h_1(n)y_{n-1}$$

The theoretical Wiener solution is found as follows: First note that

$$\begin{aligned}R_{xy}(k)&=E[x_{n+k}y_n]=E\big[(-0.8y_{n+k}+u_{n+k})y_n\big]=-0.8E[y_{n+k}y_n]\\&=-0.8R_{yy}(k)=-0.8R(k)\end{aligned}$$

Thus, the cross correlation vector is

$$\mathbf{r}=\begin{bmatrix}R_{xy}(0)\\R_{xy}(1)\end{bmatrix}=-0.8\begin{bmatrix}R(0)\\R(1)\end{bmatrix}$$

and the Wiener solution becomes:

$$\begin{aligned}\mathbf{h}=R^{-1}\mathbf{r}&=\begin{bmatrix}R(0)&R(1)\\R(1)&R(0)\end{bmatrix}^{-1}\begin{bmatrix}-0.8R(0)\\-0.8R(1)\end{bmatrix}\\&=\frac{-0.8}{R(0)^2-R(1)^2}\begin{bmatrix}R(0)&-R(1)\\-R(1)&R(0)\end{bmatrix}\begin{bmatrix}R(0)\\R(1)\end{bmatrix}=\begin{bmatrix}-0.8\\0\end{bmatrix}\end{aligned}$$

We could have expected that h_1 is zero, since the signal x_n does not depend on y_{n-1}, but only on y_n. The adaptive weights were both initialized to the (arbitrary) value of $h_0(0)=h_1(0)=-0.4$, and the value of μ was 0.03. Fig. 16.5.2 shows the two adaptive weights $h_0(n)$ and $h_1(n)$ as a function of n, converging to their optimal values of $h_0=-0.8$ and $h_1=0$.

How does one select the filter order M? The rule is that the filter must have at least as many delays as that part of x_n which is correlated with y_n. To see this, suppose x_n is related to y_n by

$$x_n=c_0y_n+c_1y_{n-1}+\cdots+c_Ly_{n-L}+u_n \tag{16.5.3}$$

where u_n is uncorrelated with y_n. Then, the filter order must be at least L. If $M\ge L$, we can write:

$$x_n=c_0y_n+c_1y_{n-1}+\cdots+c_My_{n-M}+u_n=\mathbf{c}^T\mathbf{y}(n)+u_n$$

where $\mathbf{c}$ is the extended vector having $c_i=0$ for $L+1\le i\le M$. The cross-correlation between x_n and $\mathbf{y}(n)$ is

$$\mathbf{r}=E[x_n\mathbf{y}(n)]=E\big[(\mathbf{y}^T(n)\mathbf{c})\mathbf{y}(n)\big]=E[\mathbf{y}(n)\mathbf{y}^T(n)]\mathbf{c}=R\mathbf{c}$$

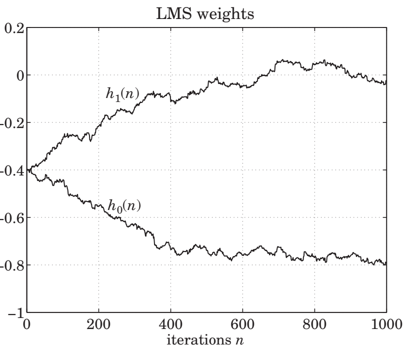


Fig. 16.5.2 Transient behavior of FIR adaptive filter.

Thus, the Wiener solution will be $\mathbf{h} = R^{-1}\mathbf{r} = \mathbf{c}$. This, in turn, implies the complete cancellation of the y-dependent part of x_n. Indeed, $\hat{x}_n = \mathbf{h}^T\mathbf{y}(n) = \mathbf{c}^T\mathbf{y}(n)$ and

$$e_n = x_n - \hat{x}_n = \big(\mathbf{c}^T\mathbf{y}(n) + u_n\big) - \mathbf{c}^T\mathbf{y}(n) = u_n$$

What happens if we underestimate the filter order and choose $M < L$? In this case, we expect to cancel completely the first M terms of Eq. (16.5.3) and to cancel the remaining terms as much as possible. To see this, we separate out the first M terms writing

$$x_n = [c_0,\dots,c_M]\begin{bmatrix} y_n \\ \vdots \\ y_{n-M}\end{bmatrix} + [c_{M+1},\dots,c_L]\begin{bmatrix} y_{n-M-1} \\ \vdots \\ y_{n-L}\end{bmatrix} + u_n \equiv \mathbf{c}_1^T\mathbf{y}_1(n) + \mathbf{c}_2^T\mathbf{y}_2(n) + u_n$$

The problem of estimating x_n using an Mth order filter is equivalent to the problem of estimating x_n from $\mathbf{y}_1(n)$. The cross-correlation between x_n and $\mathbf{y}_1(n)$ is

$$E[x_n\mathbf{y}_1(n)] = E[\mathbf{y}_1(n)\mathbf{y}_1^T(n)]\mathbf{c}_1 + E[\mathbf{y}_1(n)\mathbf{y}_2^T(n)]\mathbf{c}_2$$

It follows that the optimum estimate of x_n is

$$\begin{aligned}
\hat{x}_n &= E[x_n\mathbf{y}_1^T(n)]E[\mathbf{y}_1(n)\mathbf{y}_1^T(n)]^{-1}\mathbf{y}_1(n) \\
&= \big(\mathbf{c}_1^T E[\mathbf{y}_1(n)\mathbf{y}_1^T(n)] + \mathbf{c}_2^T E[\mathbf{y}_2(n)\mathbf{y}_1^T(n)]\big)E[\mathbf{y}_1(n)\mathbf{y}_1^T(n)]^{-1}\mathbf{y}_1(n) \\
&= \big(\mathbf{c}_1^T + \mathbf{c}_2^T E[\mathbf{y}_2(n)\mathbf{y}_1^T(n)]E[\mathbf{y}_1(n)\mathbf{y}_1^T(n)]^{-1}\big)\mathbf{y}_1(n) \\
&= \mathbf{c}_1^T\mathbf{y}_1(n) + \mathbf{c}_2^T\hat{\mathbf{y}}_{2/1}(n)
\end{aligned}$$

where $\hat{\mathbf{y}}_{2/1}(n) = E[\mathbf{y}_2(n)\mathbf{y}_1^T(n)]E[\mathbf{y}_1(n)\mathbf{y}_1^T(n)]^{-1}\mathbf{y}_1(n)$ is recognized as the optimum estimate of $\mathbf{y}_2(n)$ based on $\mathbf{y}_1(n)$. Thus, the estimation error will be

$$\begin{aligned}
e_n = x_n - \hat{x}_n &= [\mathbf{c}_1^T\mathbf{y}_1(n) + \mathbf{c}_2^T\mathbf{y}_2(n) + u_n] - [\mathbf{c}_1\mathbf{y}_1(n) + \mathbf{c}_2^T\hat{\mathbf{y}}_{2/1}(n)] \\
&= \mathbf{c}_2^T[\mathbf{y}_2(n) - \hat{\mathbf{y}}_{2/1}(n)] + u_n
\end{aligned}$$

which shows that the $\mathbf{y}_1(n)$ part is removed completely, and the $\mathbf{y}_2(n)$ part is removed as much as possible.

16.6 Speed of Convergence

The convergence properties of the LMS algorithm [1342,1350,1356] may be discussed by restoring the expectation values where they should be, that is

$$\frac{\partial\mathcal{E}}{\partial\mathbf{h}} = -2E[e_n\mathbf{y}(n)], \quad \mathbf{y}(n) = \begin{bmatrix} y_0(n) \\ y_1(n) \\ \vdots \\ y_M(n)\end{bmatrix} = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-M}\end{bmatrix}$$

resulting in the difference equation for the weight vector

$$\begin{aligned}
\mathbf{h}(n+1) &= \mathbf{h}(n) - \mu\frac{\partial\mathcal{E}}{\partial\mathbf{h}} \\
&= \mathbf{h}(n) + 2\mu E[e_n\mathbf{y}(n)] \\
&= \mathbf{h}(n) + 2\mu\{E[x_n\mathbf{y}(n)] - E[\mathbf{y}(n)\mathbf{y}^T(n)]\mathbf{h}(n)\} \\
&= \mathbf{h}(n) + 2\mu\mathbf{r} - 2\mu R\mathbf{h}(n)
\end{aligned}$$

or,

$$\mathbf{h}(n+1) = (I - 2\mu R)\mathbf{h}(n) + 2\mu\mathbf{r} \tag{16.6.1}$$

where $\mathbf{r} = E[x_n\mathbf{y}(n)]$ and $R = E[\mathbf{y}(n)\mathbf{y}^T(n)]$. The difference equation (16.6.1) has the following solution, where $\mathbf{h}_{\text{opt}} = R^{-1}\mathbf{r}$

$$\mathbf{h}(n) = \mathbf{h}_{\text{opt}} + (I - 2\mu R)^n(\mathbf{h}(0) - \mathbf{h}_{\text{opt}})$$

Convergence to $\mathbf{h}_{\text{opt}}$ requires that the quantity $(1 - 2\mu\lambda)$, for every eigenvalue λ of R, have magnitude less than one (we assume that R has full rank and therefore all its eigenvalues are positive):

$$|1 - 2\mu\lambda| < 1 \quad\Leftrightarrow\quad -1 < 1 - 2\mu\lambda < 1 \quad\Leftrightarrow\quad 0 < \mu < \frac{1}{\lambda}$$

This condition will be guaranteed if we require this inequality for $\lambda_{\max}$, the maximum eigenvalue:

$$0 < \mu < \frac{1}{\lambda_{\max}} \tag{16.6.2}$$

Note that $\lambda_{\max}$ can be bounded from above by

$$\lambda_{\max} < \text{tr}(R) = \sum_{i=0}^{M} R_{ii} = \sum_{i=0}^{M} R(0) = (M+1)R(0)$$

and one may require instead $\mu < 1/\big((M+1)R(0)\big)$. As for the speed of convergence, suppose that μ is selected half-way within its range (16.6.2), near $0.5/\lambda_{\max}$, then the rate of convergence will depend on the slowest converging term of the form $(1 - 2\mu\lambda)^n$ that is, the term having $|1 - 2\mu\lambda|$ as close to one as possible. This occurs for the smallest eigenvalue $\lambda = \lambda_{\min}$. Thus, the slowest converging term is effectively given by $(1 - 2\mu\lambda_{\min})^n = (1 - \lambda_{\min}/\lambda_{\max})^n$. The effective time constant in seconds is obtained by writing $t = nT$, where T is the sampling period, and using the approximation

$$\left(1 - \frac{\lambda_{\min}}{\lambda_{\max}}\right)^n \simeq \exp\left(-\frac{\lambda_{\min}}{\lambda_{\max}}n\right) = e^{-t/\tau}$$

where

$$\tau = T\frac{\lambda_{\max}}{\lambda_{\min}}$$

The *eigenvalue spread* $\lambda_{\max}/\lambda_{\min}$ controls, therefore, the *speed of convergence.* The convergence can be as fast as one sampling instant T if the eigenvalue spread is small, i.e., $\lambda_{\max}/\lambda_{\min} \simeq 1$. But, the convergence will be slow if the eigenvalue spread is large. As we shall see shortly, a large spread in the eigenvalues of the covariance matrix R corresponds to a highly self-correlated signal y_n.

Thus, we obtain the general qualitative result that in situations where the secondary signal is strongly self-correlated, the convergence of the gradient-based LMS algorithm will be slow. In many applications, such as channel equalization, the convergence must be as quick as possible. Alternative adaptation schemes exist that combine the computational simplicity of the LMS algorithm with a fast speed of convergence. Examples are the fast RLS and the adaptive lattice algorithms.

The possibility of accelerating the convergence rate may be seen by considering a more general version of the gradient-descent algorithm in which the time update for the weight vector is chosen as

$$\Delta\mathbf{h} = -\mathcal{M}\,\frac{\partial\mathcal{E}}{\partial\mathbf{h}} \tag{16.6.3}$$

where $\mathcal{M}$ is a *positive definite and symmetric* matrix. The LMS steepest descent case is obtained as a special case of this when $\mathcal{M}$ is proportional to the unit matrix I, $\mathcal{M} = \mu I$. This choice guarantees convergence towards the minimum of the performance index $\mathcal{E}(\mathbf{h})$, indeed,

$$\mathcal{E}(\mathbf{h}+\Delta\mathbf{h}) \simeq \mathcal{E}(\mathbf{h}) + \Delta\mathbf{h}^T\left(\frac{\partial\mathcal{E}}{\partial\mathbf{h}}\right) = \mathcal{E}(\mathbf{h}) - \left(\frac{\partial\mathcal{E}}{\partial\mathbf{h}}\right)^T\mathcal{M}\left(\frac{\partial\mathcal{E}}{\partial\mathbf{h}}\right) \le \mathcal{E}(\mathbf{h})$$

Since the performance index is

$$\mathcal{E} = E[e_n^2] = E\big[\big(x_n - \mathbf{h}^T\mathbf{y}(n)\big)^2\big] = E[x_n^2] - 2\mathbf{h}^T\mathbf{r} + \mathbf{h}^T R\mathbf{h}$$

it follows that $\partial\mathcal{E}/\partial\mathbf{h} = -2(\mathbf{r} - R\mathbf{h})$, and the difference equation for the adaptive weights becomes

$$\mathbf{h}(n+1) = \mathbf{h}(n) + \Delta\mathbf{h}(n) = \mathbf{h}(n) + 2\mathcal{M}\big(\mathbf{r} - R\mathbf{h}(n)\big)$$

or,

$$\mathbf{h}(n+1) = (I - 2\mathcal{M}R)\mathbf{h}(n) + 2\mathcal{M}\mathbf{r} \tag{16.6.4}$$

with solution for $n \ge 0$

$$\mathbf{h}(n) = \mathbf{h}_{\text{opt}} + (I - 2\mathcal{M}R)^n\big(\mathbf{h}(0) - \mathbf{h}_{\text{opt}}\big) \tag{16.6.5}$$

where $\mathbf{h}_{\text{opt}} = R^{-1}\mathbf{r}$ is the asymptotic value, and $\mathbf{h}(0)$, the initial value. It is evident from Eq. (16.6.4) or (16.6.5) that the choice of $\mathcal{M}$ can drastically affect the speed of convergence. For example, if $\mathcal{M}$ is chosen as

$$\mathcal{M} = (2R)^{-1} \tag{16.6.6}$$

then $I - 2\mathcal{M}R = 0$, and the convergence occurs in just one step! This choice of $\mathcal{M}$ is equivalent to *Newton's method* of solving the system of equations

$$\mathbf{f}(\mathbf{h}) = \frac{\partial\mathcal{E}}{\partial\mathbf{h}} = 0$$

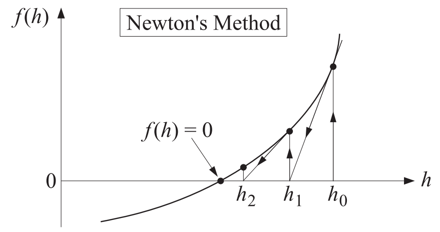
for the optimal weights. Indeed, Newton's method linearizes about each point $\mathbf{h}$ to get the next point, that is, $\Delta\mathbf{h}$ is selected such that

$$\mathbf{f}(\mathbf{h}+\Delta\mathbf{h}) \simeq \mathbf{f}(\mathbf{h}) + \left(\frac{\partial\mathbf{f}}{\partial\mathbf{h}}\right)\Delta\mathbf{h} = 0$$

where we expanded to first order in $\Delta\mathbf{h}$. Solving for $\Delta\mathbf{h}$, we obtain

$$\Delta\mathbf{h} = -\left(\frac{\partial\mathbf{f}}{\partial\mathbf{h}}\right)^{-1}\mathbf{f}(\mathbf{h})$$

But since $\mathbf{f}(\mathbf{h}) = -2(\mathbf{r} - R\mathbf{h})$, we have $\partial\mathbf{f}/\partial\mathbf{h} = 2R$. Therefore, the choice $\mathcal{M} = (2R)^{-1}$ corresponds precisely to Newton's update. Newton's method is depicted below for the one-dimensional case.



Note that the property that Newton's method converges in one step is a well-known property valid for *quadratic* performance indices (in such cases, the gradient $\mathbf{f}(\mathbf{h})$ is already linear in $\mathbf{h}$ and therefore Newton's local linearization is exact). The important property about the choice $\mathcal{M} = (2R)^{-1}$ is that $\mathcal{M}$ is proportional to the inverse of R. An alternative choice could have been $\mathcal{M} = \alpha R^{-1}$. In this case $I - 2\mathcal{M}R$ becomes proportional to the identity matrix:

$$I - 2\mathcal{M}R = (1 - 2\alpha)I$$

having equal eigenvalues. Stability requires that $|1 - 2\alpha| < 1$, or equivalently, $0 < \alpha < 1$, with Newton's choice corresponding exactly to the middle of this interval, $\alpha = 1/2$. Therefore, the disparity between the eigenvalues that could slow down the convergence rate is eliminated, and all eigenmodes converge at the same rate (which is faster the more $\mathcal{M}$ resembles $(2R)^{-1}$).

The implementation of such Newton-like methods requires knowledge of R, which we do not have (if we did, we would simply compute the Wiener solution $\mathbf{h}_{\text{opt}} = R^{-1}\mathbf{r}$.) However, as we shall see later, the so-called *recursive least-squares algorithms* effectively provide an implementation of Newton-type methods, and that is the reason for their extremely fast convergence. Adaptive *lattice* filters also have very fast convergence properties. In that case, because of the orthogonalization of the successive lattice stages of the filter, the matrix R is diagonal (in the decorrelated basis) and the matrix $\mathcal{M}$ can also be chosen to be diagonal so as to equalize and speed up the convergence rate of all the filter coefficients. Recursive least-squares and adaptive lattice filters are discussed in Sections Sec. 16.16 and 16.18, respectively.

Finally, we would like to demonstrate the previous statement that a strongly correlated signal y_n has a large spread in the eigenvalue spectrum of its covariance matrix. For simplicity, consider the 2×2 case

$$R = E[\mathbf{y}(n)\mathbf{y}^T(n)] = E\left[\begin{bmatrix} y_n \\ y_{n-1}\end{bmatrix}[y_n,\, y_{n-1}]\right] = \begin{bmatrix} R(0) & R(1) \\ R(1) & R(0)\end{bmatrix}$$

The two eigenvalues are easily found to be

$$\lambda_{\min} = R(0) - |R(1)|$$

$$\lambda_{\max} = R(0) + |R(1)|$$

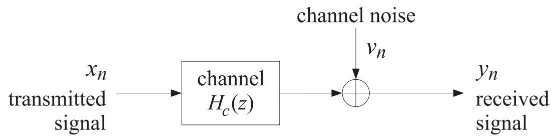
and therefore, the ratio $\lambda_{\min}/\lambda_{\max}$ is given by

$$\frac{\lambda_{\min}}{\lambda_{\max}} = \frac{R(0) - |R(1)|}{R(0) + |R(1)|}$$

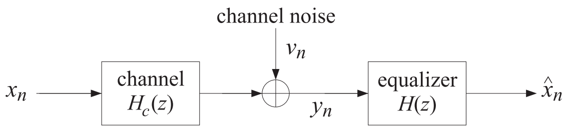
Since for an autocorrelation function we always have $|R(1)| \le R(0)$, it follows that the largest value of $R(1)$ is $\pm R(0)$, implying that for highly correlated signals the ratio $\lambda_{\min}/\lambda_{\max}$ will be very close to zero.

16.7 Adaptive Channel Equalizers

Channels used in digital data transmissions can be modeled very often by linear time-invariant systems. The standard model for such a channel including channel noise is shown here.

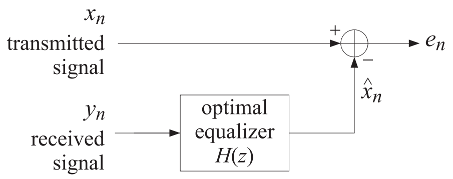


In the Figure, $H_c(z)$ is the transfer function for the channel and v_n, the channel noise, assumed to be additive white gaussian noise. The transfer function $H_c(z)$ incorporates the effects of the modulator and demodulator filters, as well as the channel distortions. The purpose of a channel equalizer is to undo the distorting effects of the channel and recover, from the received waveform y_n, the signal x_n that was transmitted. Typically, a channel equalizer will be an FIR filter with enough taps to approximate the inverse transfer function of the channel. A basic equalizer system is shown below.



In this figure, $H(z)$ is the desired transfer function of the equalizer. In many situations, such in the telephone network, the channel is not known in advance, or it may be time-varying as in the case of multipath channels. Therefore, it is desirable to design equalizers adaptively [1357–1359].

A channel equalizer, adaptive or not, is an optimal filter since it tries to produce as good an estimate $\hat{x}_n$ of the transmitted signal x_n as possible. The Wiener filtering concepts that we developed thus far are ideally suited to this problem. This is shown below.



The design of the optimal filter requires two things: first, the autocorrelation of the received signal y_n, and second, the cross-correlation of the transmitted signal x_n with the received signal. Since the transmitted signal is not available at the receiver, the following procedure is used. After the channel connection is established, a "training" sequence x_n, which is also known to the receiver, is transmitted over the channel. Then, the equalizer may be designed, and then the actual message transmitted. To appreciate the equalizer's action as an inverse filter, suppose that the training sequence x_n is a white-noise sequence of variance σ_x^2. According to the theory developed in Chap. 11, the optimal filter estimating x_n on the basis of y_n is given by

$$H(z)=\frac{1}{\sigma_\epsilon^2 B(z)}\left[\frac{S_{xy}(z)}{B(z^{-1})}\right]_+$$

where $B(z)$ is the spectral factor of $S_{yy}(z)=\sigma_\epsilon^2 B(z)B(z^{-1})$. To simplify the discussion, let us ignore the causal instruction:

$$H(z)=\frac{S_{xy}(z)}{\sigma_\epsilon^2 B(z)B(z^{-1})}=\frac{S_{xy}(z)}{S_{yy}(z)}$$

Since we have $Y(z)=H_c(z)X(z)+V(z)$, we find

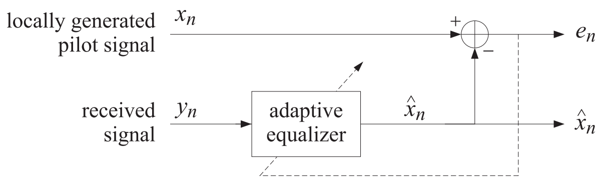
$$\begin{aligned}
S_{xy}(z) &= S_{xx}(z)H_c(z^{-1})+S_{xv}(z)=S_{xx}(z)H_c(z^{-1})=\sigma_x^2 H_c(z^{-1})\\
S_{yy}(z) &= H_c(z)H_c(z^{-1})S_{xx}(z)+S_{vv}(z)=\sigma_x^2 H_c(z)H_c(z^{-1})+\sigma_v^2
\end{aligned}$$

the equalizer's transfer function is then

$$H(z)=\frac{S_{xy}(z)}{S_{yy}(z)}=\frac{\sigma_x^2 H_c(z^{-1})}{\sigma_x^2 H_c(z)H_c(z^{-1})+\sigma_v^2}$$

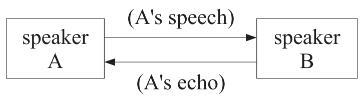
It is seen that when the channel noise is weak (small σ_v^2), the equalizer essentially behaves as the inverse filter $1/H_c(z)$ of the channel.

In an adaptive implementation, we must use a filter with a finite number of weights. These weights are adjusted adaptively until they converge to their optimal values. Again, during this "training mode" a known pilot signal is sent over the channel and is received as y_n. At the receiving end, the pilot signal is locally generated and used in the adaptation algorithm. This implementation is shown below.

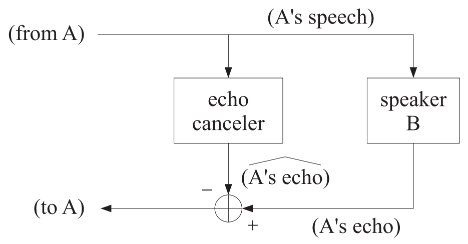


16.8 Adaptive Echo Cancelers

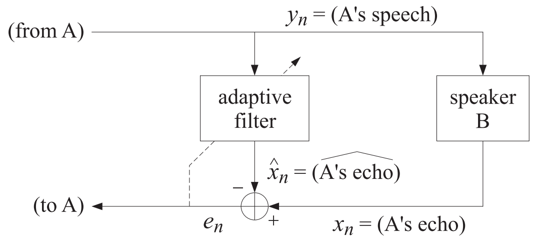
Consider two speakers A and B connected to each other by the telephone network. As a result of various impedance mismatches, when A's speech reaches B, it manages to "leak" through and echoes back to speaker A, as though it were B's speech.



An echo canceler may be placed near B's end, as shown.



It produces an (optimum) estimate of A's echo through B's circuits, and then proceeds to cancel it from the signal returning to speaker A. Again, this is another case for which optimal filtering ideas are ideally suited. An adaptive echo canceler is an adaptive FIR filter placed as shown [1360–1365].



As always, the adaptive filter will adjust itself to cancel any correlations that might exist between the secondary signal y_n (A's speech) and the primary signal x_n (A's echo).

16.9 Adaptive Noise Canceling

In many applications, two signals are available; one is composed of a desired signal plus undesired noise interference, and the other is composed only of noise interference which, if not identical with the noise part of the first signal, is correlated with it. This is shown in Fig. 16.9.1. An adaptive noise canceler [1350] is an adaptive filter as shown in the Figure. It acts as a correlation canceler. If the signals x_n and y_n are in any way correlated (i.e., the noise component of x_n with y_n), then the filter will respond by adapting its weights until such correlations are canceled from the output e_n. It does so by producing the best possible replica of the noise component of x_n and proceeding to cancel it. The output e_n will now consist mainly of the desired signal.

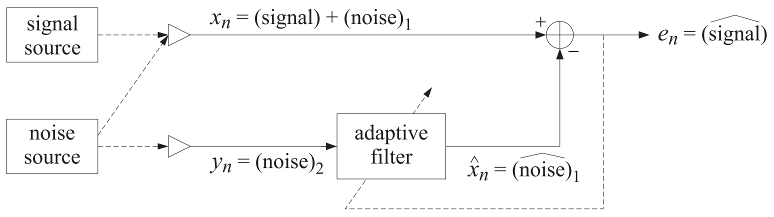

signal source
noise source
x_n = (signal) + (noise)$_1$
+
−
$e_n = \widehat{\text{(signal)}}$
adaptive filter
y_n = (noise)$_2$
$\hat{x}_n = \widehat{\text{(noise)}}_1$

Fig. 16.9.1 Adaptive noise canceler.

There are many applications of adaptive noise canceling, such as adaptive sidelobe cancellation, acoustic noise cancellation [1368–1370], canceling 60 Hz interference in EKG recordings, plasma estimation [1371], and ghost cancellation in television [1372].

An interesting property of the adaptive noise canceler is that when the secondary signal y_n is purely sinusoidal at some frequency ω_0, the adaptive filter behaves as a *notch filter* [1350,1373] at the sinusoid's frequency, that is, the transfer relationship between the primary input x_n and the output e_n becomes the time-invariant transfer function of a notch filter. This is a surprising property since the adaptation equations for the weights and the filtering I/O equation are in general time-noninvariant. To understand this effect, it proves convenient to work with complex-valued signals using a complex-valued reformulation of the LMS algorithm [1374]. We make a short digression on this, first. We assume that x_n, y_n and the weights $\mathbf{h}(n)$ are complex-valued. The performance index is replaced by

$$\mathcal{E} = E[e_n^* e_n]$$

where the I/O filtering equation is still given by

$$\hat{x}_n = \sum_{m=0}^{M} h_m y_{n-m} = \mathbf{h}^T \mathbf{y}(n)$$

Since the weights $\mathbf{h}$ are complex, the index $\mathcal{E}$ depends on both the real and the imaginary parts of $\mathbf{h}$. Equivalently, we may think of $\mathcal{E}$ as a function of the two independent variables $\mathbf{h}$ and $\mathbf{h}^*$. A complex change in the weights $\Delta\mathbf{h}$ will change the index to

$$\mathcal{E}(\mathbf{h}+\Delta\mathbf{h}, \mathbf{h}^* + \Delta\mathbf{h}^*) = \mathcal{E}(\mathbf{h},\mathbf{h}^*) + \Delta\mathbf{h}^T \frac{\partial\mathcal{E}}{\partial\mathbf{h}} + \Delta\mathbf{h}^\dagger \frac{\partial\mathcal{E}}{\partial\mathbf{h}^*}$$

Choosing $\Delta\mathbf{h}$ to be proportional to the complex conjugate of the negative gradient, that is,

$$\Delta\mathbf{h} = -2\mu \frac{\partial\mathcal{E}}{\partial\mathbf{h}^*} = 2\mu E[e_n \mathbf{y}(n)^*]$$

will move the index $\mathcal{E}$ towards its minimum value; indeed,

$$\mathcal{E}(\mathbf{h}+\Delta\mathbf{h}, \mathbf{h}^* + \Delta\mathbf{h}^*) = \mathcal{E}(\mathbf{h},\mathbf{h}^*) - 4\mu \left(\frac{\partial\mathcal{E}}{\partial\mathbf{h}}\right)^\dagger \left(\frac{\partial\mathcal{E}}{\partial\mathbf{h}}\right) \leq \mathcal{E}(\mathbf{h},\mathbf{h}^*)$$

Thus, the complex version of the LMS algorithm consists simply of replacing the instantaneous gradient by its complex conjugate [1374]. We summarize the algorithm as follows:

1. Compute $\hat{x}_n = \mathbf{h}(n)^T \mathbf{y}(n)$.
2. Compute $e_n = x_n - \hat{x}_n$.
3. Update weights $\mathbf{h}(n+1) = \mathbf{h}(n) + 2\mu e_n \mathbf{y}(n)^*$.

Using this complex version, we now discuss the notching behavior of the adaptive filter. Suppose y_n is sinusoidal

$$y_n = Ae^{j\omega_0 n}$$

at some frequency ω_0. Then, the weight-update equation becomes:

$$h_m(n+1) = h_m(n) + 2\mu e_n y_{n-m}^* = h_m(n) + 2\mu A^* e^{-j\omega_0(n-m)}$$

for $m = 0, 1, \dots, M$. The factor $e^{-j\omega_0(n-m)}$ suggests that we look for a solution of the form

$$h_m(n) = f_m(n) e^{-j\omega_0(n-m)}$$

Then, $f_m(n)$ must satisfy the difference equation

$$e^{-j\omega_0} f_m(n+1) = f_m(n) + 2\mu A^* e_n$$

As a difference equation in n, this equation has constant coefficients, and, therefore, may be solved by z-transform techniques. Taking z-transforms of both sides we find

$$e^{-j\omega_0} z F_m(z) = F_m(z) + 2\mu A^* E(z)$$

which may be solved for $F_m(z)$ in terms of $E(z)$ to give

$$F_m(z) = E(z)\, \frac{2\mu A^* e^{j\omega_0}}{z - e^{j\omega_0}}$$

On the other hand, the I/O filtering equation from y_n to the output $\hat{x}_n$ is

$$\hat{x}_n = \sum_{m=0}^{M} h_m(n)y_{n-m} = \sum_{m=0}^{M} f_m(n)e^{-j\omega_0(n-m)}Ae^{j\omega_0(n-m)} = \sum_{m=0}^{M} f_m(n)A$$

or, in the z-domain

$$\hat{X}(z) = \sum_{m=0}^{M} F_m(z)A = E(z)\frac{2\mu(M+1)|A|^2e^{j\omega_0}}{z-e^{j\omega_0}}$$

Finally, the I/O equation from x_n to e_n becomes

$$e_n = x_n - \hat{x}_n$$

and, in the z-domain

$$E(z) = X(z) - \hat{X}(z) = X(z) - E(z)\frac{2\mu(M+1)|A|^2e^{j\omega_0}}{z-e^{j\omega_0}}$$

which may be solved for the transfer function $E(z)/X(z)$

$$\frac{E(z)}{X(z)} = \frac{z-e^{j\omega_0}}{z-Re^{j\omega_0}}\,, \quad R \equiv 1 - 2\mu(M+1)|A|^2 \tag{16.9.1}$$

This filter has a zero at $z = e^{j\omega_0}$ which corresponds to the notch at the frequency ω_0. For sufficiently small values of μ and A, the filter is stable; its pole is at $z = Re^{j\omega_0}$ and can be made to lie inside the unit circle $(0 < R < 1)$. If the primary input x_n happens to have a sinusoidal component at frequency ω_0, this component will be completely notched away from the output. This will take place even when the sinusoidal reference signal is very weak (i.e., when A is small). The implications of this property for *jamming by signal cancellation* in adaptive array processing have been discussed in [1375]. The notching behavior of the adaptive noise canceler when the reference signal consists of a sinusoid plus noise has been discussed in [1376].

A related result is that the adaptive noise canceler behaves as a time-invariant *comb filter* whenever its secondary input y_n is a periodic train of impulses separated by some period [1377]. This property can be used to cancel periodic interference. Because the method of signal averaging can be thought of as comb filtering, the above property may also be used as an alternative method to perform signal averaging for pulling weak periodic signals from background noise, such as evoked potentials [1378].

16.10 Adaptive Line Enhancer

A special case of adaptive noise canceling is when there is only one signal x_n available which is contaminated by noise. In such a case, the signal x_n provides its own reference signal y_n, which is taken to be a delayed replica of x_n, that is, $y_n = x_{n-\Delta}$, as shown in Fig. 16.10.1, referred to as the adaptive line enhancer (ALE) [1350,1379–1381].

Will such arrangement be successful? The adaptive filter will respond by canceling any components of the main signal x_n that are in any way correlated with the secondary signal $y_n = x_{n-\Delta}$. Suppose the signal x_n consists of two parts: a narrowband component that has long-range correlations such as a sinusoid, and a broadband component which will tend to have short-range

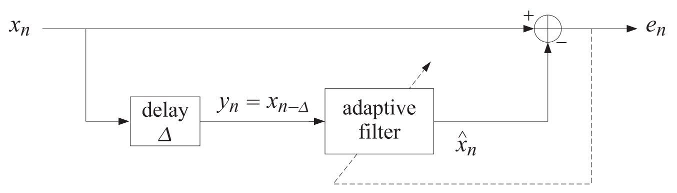
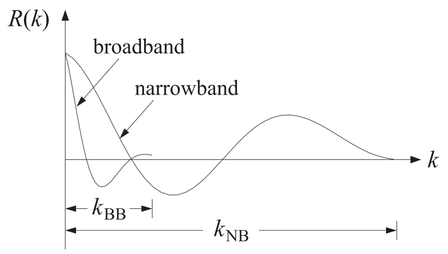


Fig. 16.10.1 Adaptive line enhancer.

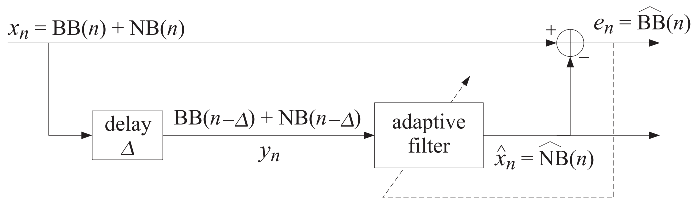
correlations. One of these could represent the desired signal and the other an undesired interfering noise. Pictorially the autocorrelations of the two components could look as follows.



where $k_{\rm NB}$ and $k_{\rm BB}$ are effectively the self-correlation lengths of the narrowband and broadband components, respectively. Beyond these lags, the respective correlations die out quickly. Suppose the delay Δ is selected so that

$$k_{\rm BB} \le \Delta \le k_{\rm NB}$$

Since Δ is longer than the effective correlation length of the BB component, the delayed replica $\text{BB}(n-\Delta)$ will be entirely uncorrelated with the BB part of the main signal. The adaptive filter will not be able to respond to this component. On the other hand, since Δ is shorter than the correlation length of the NB component, the delayed replica $\text{NB}(n-\Delta)$ that appears in the secondary input will still be correlated with the NB part of the main signal, and the filter will respond to cancel it. Thus, the filter outputs will be as shown.


$x_n = \text{BB}(n) + \text{NB}(n)$
+
−
$e_n = \widehat{\text{BB}}(n)$
delay Δ
$\text{BB}(n-\Delta) + \text{NB}(n-\Delta)$
y_n
adaptive filter
$\hat{x}_n = \widehat{\text{NB}}(n)$

Note that if Δ is selected to be longer than both correlation lengths, the secondary input will become uncorrelated with the primary input, and the adaptive filter will turn itself off. In the opposite case, when the delay Δ is selected to be less than both correlation lengths, then both components of the secondary signal will be correlated with the primary signal, and therefore, the adaptive filter will respond to cancel the primary x_n completely. The computational algorithm for the ALE is as follows

1. $\hat{x}_n = \sum_{m=0}^{M} h_m(n)y(n-m) = \sum_{m=0}^{M} h_m(n)x(n-m-\Delta)$

2. $e_n = x_n - \hat{x}_n$
3. $h_m(n+1) = h_m(n) + 2\mu e_n x(n-m-\Delta)\,, \quad m = 0, 1, \dots, M$

The Wiener solution for the steady-state weights is $\mathbf{h} = R^{-1}\mathbf{r}$, where R and $\mathbf{r}$ are both expressible in terms of the autocorrelation of the signal x_n, as follows:

$$R_{ij} = E[y_{n-i}y_{n-j}] = E[x_{n-\Delta-i}\,x_{n-\Delta-j}] = R_{xx}(i-j)$$

$$r_i = E[x_n y_{n-i}] = E[x_n x_{n-\Delta-i}] = R_{xx}(i+\Delta)$$

for $i, j = 0, 1, \dots, M$. When the input signal consists of multiple sinusoids in additive white noise, the inverse R^{-1} may be obtained using the methods of Sec. 14.2, thus resulting in a closed form expression for the steady-state optimal weights [1381].

16.11 Adaptive Linear Prediction

A linear predictor is a special case of the ALE with the delay $\Delta = 1$. It is shown in Fig. 16.11.1, where to be consistent with our past notation on linear predictors we have denoted the main signal by y_n. The secondary signal, the input to the adaptive filter, is then y_{n-1}. Due to the special sign convention used for linear predictors, the adaptation algorithm now reads

1. $\hat{y}_n = -[a_1(n)y_{n-1} + a_2(n)y_{n-2} + \cdots + a_M(n)y_{n-M}]$
2. $e_n = y_n - \hat{y}_n = y_n + a_1(n)y_{n-1} + \cdots + a_M(n)y_{n-M}$
3. $a_m(n+1) = a_m(n) - 2\mu e_n y_{n-m}\,, \quad m = 1, 2\dots, M$

The realization of Fig. 16.11.1 can be redrawn more explicitly as in Fig. 16.11.2. The **lmsap** is an implementation of the LMS adaptive predictor. At each call, the function reads a sample y_n, computes the filter output e_n, updates the filter coefficients $a_m(n)$ to their new values $a_m(n+1)$ to be used by the next call, and updates the registers of the tapped delay line. With a small modification it can be used in the adaptive array problem (see below).

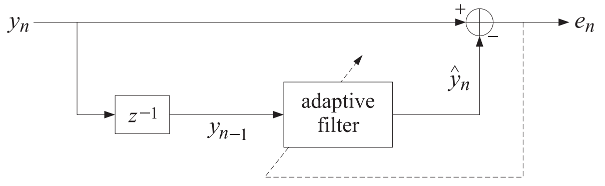


Fig. 16.11.1 Adaptive linear predictor.

Because of the importance of the adaptive predictor, we present a direct derivation of the LMS algorithm as it applies to this case. The weights a_m are chosen optimally to minimize the mean output power of the filter, that is, the mean-square prediction error:

$$\mathcal{E} = E[e_n^2] = \mathbf{a}^T R\mathbf{a} = \min \tag{16.11.1}$$

where $\mathbf{a} = [1, a_1, a_2, \dots, a_M]^T$ is the prediction error filter. The performance index (16.11.1) is minimized with respect to the M weights a_m. The gradient with respect to a_m is the mth component of the vector $2R\mathbf{a}$, namely,

$$\frac{\partial\mathcal{E}}{\partial a_m} = 2(R\mathbf{a})_m = 2\big(E[\mathbf{y}(n)\mathbf{y}(n)^T]\mathbf{a}\big)_m = 2\big(E[\mathbf{y}(n)\mathbf{y}(n)^T\mathbf{a}]\big)_m$$

$$= 2\big(E[\mathbf{y}(n)e_n]\big)_m = 2E[e_n y_{n-m}]$$

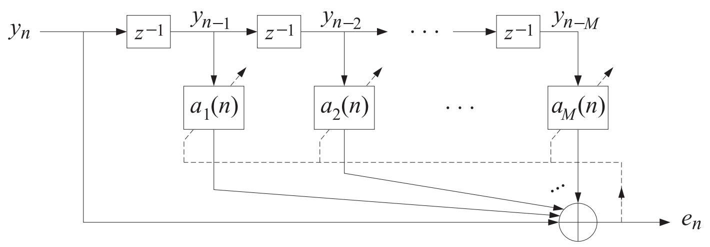


Fig. 16.11.2 Direct-form realization of adaptive predictor.

The instantaneous gradient is obtained by *ignoring* the expectation instruction. This gives for the LMS time-update of the mth weight

$$\Delta a_m(n) = -\mu\frac{\partial\mathcal{E}}{\partial a_m} = -2\mu e_n y_{n-m}\,, \quad m = 1, 2, \dots, M \tag{16.11.2}$$

The adaptive predictor may be thought of as an *adaptive whitening filter*, or an analysis filter which determines the LPC model parameters adaptively. As processing of the signal y_n takes place, the autoregressive model parameters a_m are extracted *on-line*. This is but one example of on-line system identification methods [1384–1392].

The extracted model parameters may be used in any desired way—for example, to provide the *autoregressive spectrum estimate* of the signal y_n. One of the advantages of the adaptive implementation is that it offers the possibility of tracking slow changes in the spectra of non-stationary signals. The only requirement for obtaining meaningful spectrum estimates is that the non-stationary changes of the spectrum be slow enough for the adaptive filter to have a chance to converge between changes. Typical applications are the tracking of sinusoids in noise whose frequencies may be slowly changing [1382,1383,1393], or tracking the time development of the spectra of non-stationary EEG signals [1028]. At each time instant n, the adaptive weights $a_m(n)$, $m = 1, 2, \dots, M$ may be used to obtain an instantaneous autoregressive estimate of the spectrum of y_n in the form

$$S_n(\omega) = \frac{1}{\big|1 + a_1(n)e^{-j\omega} + a_2(n)e^{-2j\omega} + \cdots + a_M(n)e^{-Mj\omega}\big|^2}$$

This is the adaptive implementation of the LP spectrum estimate discussed in Sec. 14.2. The same adaptive approach to LP spectrum estimation may also be used in the problem of multiple source location, discussed in Sec. 14.3. The only difference in the algorithm is to replace y_{n-m} by $y_m(n)$—that is, by the signal recorded at the mth sensor at time n—and to use the complex-valued version of the LMS algorithm. For completeness, we summarize the computational steps in this case, following the notation of Sec. 14.3.

1. $e(n) = y_0(n) + a_1(n)y_1(n) + a_2(n)y_2(n) + \cdots + a_M(n)y_M(n)$
2. $a_m(n+1) = a_m(n) - 2\mu e(n)y_m^*(n)\,, \quad m = 1, 2, \dots, M$

At each time instant n, the corresponding spatial spectrum estimate may be computed by

$$S_n(k) = \frac{1}{\big|1 + a_1(n)e^{-jk} + a_2(n)e^{-2jk} + \cdots + a_M(n)e^{-Mjk}\big|^2}$$

where the wavenumber k and its relationship to the angle of bearing was defined in Sec. 14.3. Fig. 16.11.3 shows the corresponding adaptive array processing configuration.

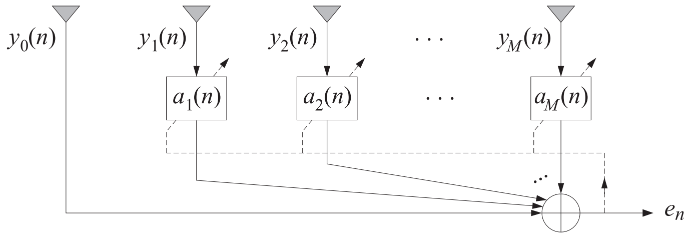


Fig. 16.11.3 Adaptive array processor.

The time-adaptive as well as the block-data adaptive methods of superresolution array processing have been reviewed in [1099,1129]. The above LMS algorithm for the array weights is effectively equivalent to the Howells-Applebaum algorithm [1351–1355]. Adaptive predictors may also be used to improve the performance of spread-spectrum systems [1396–1402].

16.12 Adaptive Implementation of Pisarenko's Method

In Sec. 14.2, we noted that the Pisarenko eigenvalue problem was equivalent to the minimization of the performance index

$$\mathcal{E} = E[e_n^* e_n] = \mathbf{a}^\dagger R\mathbf{a} = \min \tag{16.12.1}$$

subject to the quadratic constraint

$$\mathbf{a}^\dagger \mathbf{a} = 1 \tag{16.12.2}$$

where

$$e_n = \sum_{m=0}^{M} a_m y_{n-m} = [a_0, a_1, \dots, a_M]\begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-M}\end{bmatrix} = \mathbf{a}^T \mathbf{y}(n)$$

The solution of the minimization problem shown in Eqs. (16.12.1) and (16.12.2) is the eigenvector $\mathbf{a}$ belonging to the minimum eigenvalue of the covariance matrix R. If there are L sinusoids of frequencies ω_i, $i = 1, 2, \dots, L$, and we use a filter of order M, such that $M \geq L$, then the eigenpolynomial $A(z)$ corresponding to the minimum eigenvector $\mathbf{a}$ will have L zeros on the unit circle at precisely the desired set of frequencies, that is,

$$A(z_i) = 0\,, \quad \text{where} \quad z_i = e^{j\omega_i}\,, \quad i = 1, 2, \dots, L$$

The adaptive implementation [1403] of the Pisarenko eigenvalue problem is based on the above minimization criterion. The LMS gradient-descent algorithm can be used to update the weights, but some care must be taken to satisfy the essential quadratic constraint (16.12.2) at each iteration of the algorithm. Any infinitesimal change $d\mathbf{a}$ of the weights must respect the constraint. This means the $d\mathbf{a}$ cannot be arbitrary but must satisfy the condition

$$d(\mathbf{a}^\dagger \mathbf{a}) = \mathbf{a}^\dagger (d\mathbf{a}) + (d\mathbf{a})^\dagger \mathbf{a} = 0 \tag{16.12.3}$$

so that the new weight $\mathbf{a} + d\mathbf{a}$ still lies on the quadratic surface $\mathbf{a}^\dagger \mathbf{a} = 1$. The ordinary gradient of the performance index $\mathcal{E}$ is

$$\frac{\partial \mathcal{E}}{\partial \mathbf{a}^*} = R\mathbf{a}$$

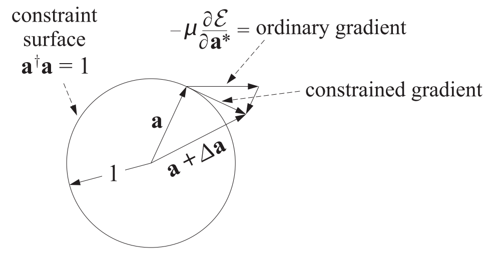
Projecting this onto the surface $\mathbf{a}^\dagger \mathbf{a} = 1$ by the projection matrix $\mathcal{P} = I - \mathbf{a}\mathbf{a}^\dagger$, where I is the $(M+1)$-dimensional unit matrix, we obtain the "constrained" gradient

$$\left(\frac{\partial \mathcal{E}}{\partial \mathbf{a}^*}\right)_c = \mathcal{P}\,\frac{\partial \mathcal{E}}{\partial \mathbf{a}^*} = (I - \mathbf{a}\mathbf{a}^\dagger)(R\mathbf{a}) = R\mathbf{a} - \mathcal{E}\mathbf{a} \tag{16.12.4}$$

which is *tangent* to the constraint surface at the point $\mathbf{a}$. The vanishing of the constrained gradient is equivalent to the Pisarenko eigenvalue problem. The weight update can now be chosen to be proportional to the constrained gradient

$$\Delta\mathbf{a} = -\mu\left(\frac{\partial \mathcal{E}}{\partial \mathbf{a}^*}\right)_c = -\mu(R\mathbf{a} - \mathcal{E}\mathbf{a})$$

The projection of the gradient onto the constraint surface is shown below.


constraint surface $\mathbf{a}^\dagger\mathbf{a} = 1$
$-\mu\dfrac{\partial\mathcal{E}}{\partial\mathbf{a}^*}$ = ordinary gradient
constrained gradient
$\mathbf{a}$
$\mathbf{a}+\Delta\mathbf{a}$
1

This choice guarantees that $\Delta\mathbf{a}$ satisfies Eq. (16.12.3); indeed, because of the projection matrix in front of the gradient, it follows that $\mathbf{a}^\dagger \Delta\mathbf{a} = 0$. Actually, since $\Delta\mathbf{a}$ is not infinitesimal, it will correspond to a *finite* motion along the tangent to the surface at the point $\mathbf{a}$. Thus, the new point $\mathbf{a} + \Delta\mathbf{a}$ will be slightly off the surface and must be renormalized to have unit norm. Using the properties,

$$R\,\mathbf{a} = E[\mathbf{y}(n)^*\mathbf{y}(n)^T]\mathbf{a} = E[\mathbf{y}(n)^* e_n] \quad \text{and} \quad \mathcal{E} = E[e_n^* e_n]$$

we write the update as

$$\Delta\mathbf{a} = -\mu\big(E[e_n \mathbf{y}(n)^*] - E[e_n^* e_n]\mathbf{a}\big)$$

The LMS algorithm is obtained by ignoring the indicated ensemble expectation values. The weight adjustment procedure consists of two steps: first, shift the old weight $\mathbf{a}(n)$ by $\Delta\mathbf{a}(n)$, and then renormalize it to unit norm:

$$\mathbf{a}(n+1) = \frac{\mathbf{a}(n) + \Delta\mathbf{a}(n)}{\|\mathbf{a}(n) + \Delta\mathbf{a}(n)\|} \tag{16.12.5}$$

where the weight update is computed by

$$\Delta\mathbf{a}(n) = -\mu\big[e_n \mathbf{y}(n)^* - e_n^* e_n \mathbf{a}(n)\big] \tag{16.12.6}$$

In summary, the computational steps are as follows:

1. At time n, $\mathbf{a}(n)$ is available and normalized to unit norm.
2. Compute the output $e_n = \sum_{m=0}^{M} a_m(n) y_{n-m} = \mathbf{a}(n)^T \mathbf{y}(n)$.
3. Update the filter weights using Eq. (16.12.5) and (16.12.6).
4. Go to the next time instant, $n \to n+1$.

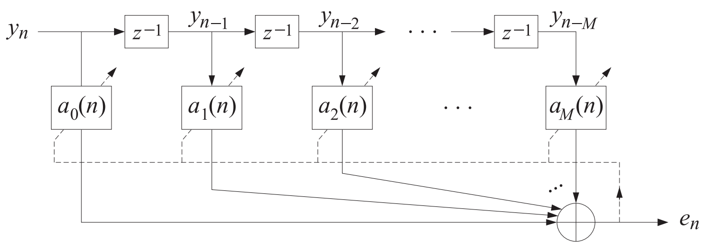
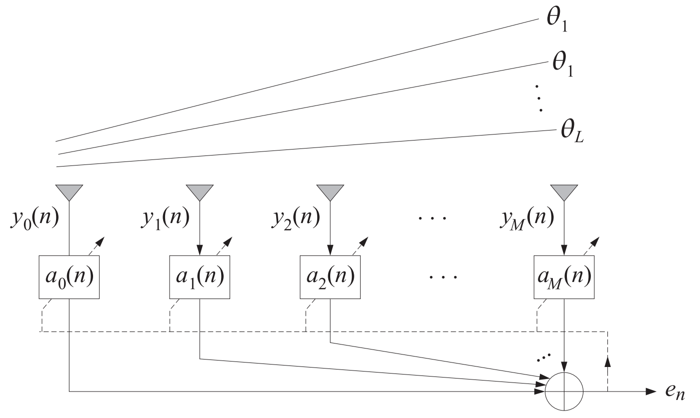


Fig. 16.12.1 Adaptive implementation of Pisarenko's method.

A realization of the adaptive filter is shown in Fig. 16.12.1. After a number of iterations, the algorithm may be stopped and the Pisarenko spectrum estimate computed:

$$S_n(\omega)=\frac{1}{\big|a_0(n)+a_1(n)e^{-j\omega}+a_2(n)e^{-2j\omega}+\cdots+a_M(n)e^{-Mj\omega}\big|^2}$$

After convergence, $S_n(\omega)$ should exhibit very sharp peaks at the sought frequencies ω_i, $i=1,2\dots,L$. The convergence properties of this algorithm have been studied in [1404]. Alternative adaptation schemes for the weights have been proposed in [1406]. The algorithm may also be applied to the array problem of *multiple source location* [1407]. Again, the only change is to replace y_{n-m} by $y_m(n)$, depicted below.



Both the adaptive prediction and the Pisarenko approaches to the two problems of extracting sinusoids in noise and multiple emitter location have a common aim, namely, to produce an adaptive filter $A(z)$ with zeros very near or on the unit circle at the desired frequency angles. Taking the inverse magnitude response as an estimate of the spectrum of the signal,

$$S(\omega)=\frac{1}{|A(\omega)|^2}$$

is a simple device to obtain a curve that exhibits sharp spectral peaks at the desired frequencies.

A satisfactory alternative approach would be simply to find the roots of the polynomial $A(z)$ and pick those that are closest to the unit circle. The phase angles of these roots are precisely the desired frequencies. In other words, the frequency information we are attempting to extract by means of the adaptive filter is more directly represented by the zeros of the filter than by its weights.

It would be desirable then to develop methods by which these zeros can be estimated directly without having to submit the filter $A(z)$ to root-finding algorithms. In implementing this idea adaptively, we would like to adapt and track the zeros of the adaptive filter as they move about on the complex z-plane, converging to their final destinations which are the desired zeros. In this way, the frequency information can be extracted directly. Such "zero-tracking" adaptation algorithms have been proposed recently [1408,1409].

Even though the representations of the filter in terms of its zeros and in terms of its weights are mathematically equivalent, the zero representation may be more appropriate in some applications in the sense that a better insight into the nature of the underlying processes may be gained from it than from the weight representation.

As an example, we mention the problem of predicting epileptic seizures by LPC modeling of the EEG signal where it was found [1410] that the trajectories of the zeros of the prediction-error filter on the z-plane exhibited an unexpected behavior, namely, prior to the onset of a seizure, one of the zeros became the "most mobile" and moved towards the unit circle, whereas the other zeros did not move much. The trajectory of the most mobile zero could be used as a signature for the onset of the oncoming seizure. Such behavior could not be easily discerned by the frequency response or by the final zero locations.

Next, we describe briefly the *zero-tracking algorithm* as it applies to the Pisarenko problem and present a simulation example. Its application to adaptive prediction and to emitter location has been discussed in [1409]. For simplicity, we assume that the number of sinusoids that are present is the same as the order of the filter $\mathbf{a}$, that is, $L=M$. The case $L<M$ will be discussed later on. The eigenpolynomial of the minimum eigenvector $\mathbf{a}$ may be factored into its zeros as follows:

$$\begin{aligned}A(z)&=a_0+a_1z^{-1}+a_2z^{-2}+\cdots+a_Mz^{-M}\\&=a_0(1-z_1z^{-1})(1-z_2z^{-1})\cdots(1-z_Mz^{-1})\end{aligned}\tag{16.12.7}$$

where a_0 may be thought of as a normalization factor which guarantees the unit norm constraint (16.12.2), and $z_i=e^{j\omega_i}$, $i=1,2,\dots,M$ are the desired sinusoid zeros on the unit circle.

In the adaptive implementation, the weights a_m become time-dependent $a_m(n)$ and are adapted from each time instant to the next until they converge to the asymptotic values defined by Eq. (16.12.7). At each n, the corresponding polynomial can be factored into its zeros as follows:

$$\begin{aligned}&a_0(n)+a_1(n)z^{-1}+a_2(n)z^{-2}+\cdots+a_M(n)z^{-M}\\&=a_0(n)\big(1-z_1(n)z^{-1}\big)\big(1-z_2(n)z^{-1}\big)\cdots\big(1-z_M(n)z^{-1}\big)\end{aligned}\tag{16.12.8}$$

where again the factor $a_0(n)$ ensures the unit-norm constraint. In the zero-tracking algorithm, the weight update equation (16.12.5) is replaced by a zero-update equation of the form:

$$z_i(n+1)=z_i(n)+\Delta z_i(n)\,,\quad i=1,2,\dots,M\tag{16.12.9}$$

where the zero updates $\Delta z_i(n)$ must be such that to ensure the convergence of the zeros to their asymptotic values z_i. One way to do this is to make the algorithm equivalent to the LMS algorithm. The functional dependence of $z_i(n)$ on $a_m(n)$ defined by Eq. (16.12.8) implies that if the weights $a_m(n)$ are changed by a small amount $\Delta a_m(n)$ given by Eq. (16.12.6), then a small change $\Delta z_i(n)$ will be induced on the corresponding zeros. This is given as follows:

$$\Delta z_i(n)=\sum_{m=0}^{M}\frac{\partial z_i(n)}{\partial a_m}\Delta a_m(n)\tag{16.12.10}$$

where the partial derivatives are given by [12]

$$\frac{\partial z_i(n)}{\partial a_m} = -\frac{1}{a_0(n)}\,\frac{z_i(n)^{M-m}}{\prod_{j\neq i}\big(z_i(n)-z_j(n)\big)}\,,\quad 0\le m\le M \tag{16.12.11}$$

Equation (16.12.10) is strictly valid for infinitesimal changes, but for small μ, it can be taken to be an adequate approximation for the purpose of computing $\Delta z_i(n)$. The advantage of this expression is that only the current zeros $z_i(n)$ are needed to compute $\Delta z_i(n)$. The complete algorithm is summarized as follows:

1. At time n, the zeros $z_i(n)$, $i = 1,2,\dots,M$ are available.
2. Using convolution, compute the corresponding filter weights and normalize them to unit norm, that is, first convolve the factors of Eq. (16.12.8) to obtain the vector

   $$\begin{aligned}\mathbf{b}(n)^T &= [1, b_1(n), b_2(n), \dots, b_M(n)]\\ &= [1, -z_1(n)] * [1, -z_2(n)] * \cdots * [1, -z_M(n)]\end{aligned}$$

   and then normalize $\mathbf{b}(n)$ to unit norm:

   $$\mathbf{a}(n) = \frac{\mathbf{b}(n)}{\|\mathbf{b}(n)\|}$$

3. Compute the filter output $e_n = \mathbf{a}(n)^T\mathbf{y}(n)$.
4. Compute the LMS coefficient updates $\Delta a_m(n)$ using Eq. (16.12.6). Compute the zero updates $\Delta z_i(n)$ using Eqs. (16.12.10) and (16.12.11), and update the zeros using Eq. (16.12.9).

The algorithm may be initialized by a random selection of the initial zeros inside the unit circle in the z-plane. Next, we present a simulation example consisting of a fourth order filter and four sinusoids

$$y_n = v_n + e^{j\omega_1 n} + e^{j\omega_2 n} + e^{j\omega_3 n} + e^{j\omega_4 n}$$

with frequencies

$$\omega_1 = 0.25\pi\,,\quad \omega_2 = -0.25\pi\,,\quad \omega_3 = 0.75\pi\,,\quad \omega_4 = -0.75\pi$$

and a zero-mean, unit-variance, white noise sequence v_n (this corresponds to all sinusoids having 0 dB signal to noise ratio). The value of μ was 0.001. Figure 7.14 shows the adaptive trajectories of the four filter zeros as they converge onto the unit circle at the above frequency values. After convergence, the adaptive zeros remain within small neighborhoods about the asymptotic zeros. The diameter of these neighborhoods decreases with smaller μ, but so does the speed of convergence [1409].

The transient behavior of the zeros can be seen by plotting $z_i(n)$ versus iteration number n. Fig. 16.12.3 shows the real and imaginary parts of the adaptive trajectory of the zero $z_2(n)$ converging to the real and imaginary parts of the asymptotic zero $z_2 = e^{j\omega_2} = e^{-j0.25\pi} = (1 - j)/\sqrt{2}$.

When the number L of sinusoids is less than the order M of the filter, only L of the M zeros $z_i(n)$ of the filter will be driven to the unit circle at the right frequency angles. The remaining $(M - L)$ zeros correspond to spurious degrees of freedom (the degeneracy of the minimum eigenvalue σ_v^2), and are affected by the adaptation process only insofar as the M zero trajectories are not entirely independent of each other but are mutually coupled through Eq. (16.12.11). Where these spurious zeros converge to depends on the particular initialization. For some special initial conditions it is possible for the spurious zeros to move close to the unit circle, thus causing a confusion as to which are the true sinusoid zeros. To safeguard against such a possibility, the algorithm may be run again with a different choice of initial zeros. Fig. 16.12.4 shows the adaptive

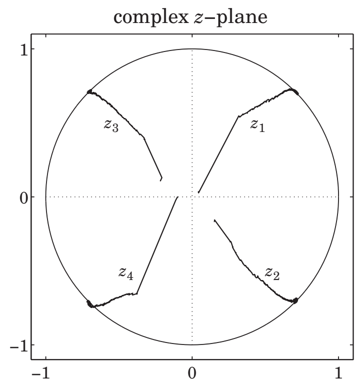


Fig. 16.12.2 z-Plane trajectories of the four adaptive zeros $z_i(n)$, $i = 1,2,3,4$.

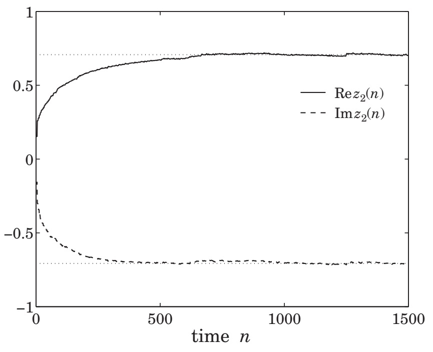


Fig. 16.12.3 Real and imaginary parts of $z_2(n)$ versus n.

trajectory of a single sinusoid, $L = 1$, using a third order filter, $M = 3$. The sinusoid's frequency was $\omega_1 = 0.25\pi$, its SNR was 0 dB, and μ was 0.001. One of the three filter zeros is driven to the unit circle at the desired angle ω_1, while the two spurious zeros traverse fairly short paths which depend on their initial positions.

16.13 Gradient Adaptive Lattice Filters

In this section we discuss the "gradient adaptive lattice" implementations of linear prediction and lattice Wiener filters [1411–1416]. They are based on a gradient-descent, LMS-like approach applied to the weights of the lattice representations rather than to the weights of the direct-form realization. Taking advantage of the decoupling of the successive stages of the lattice, and properly choosing the adaptation constants μ, all lattice weights can be made to converge fast and, in contrast to the LMS weights, with a convergence rate that is essentially *independent* of the eigenvalue spread of the input covariance matrix. The gradient lattice algorithms are very similar but not identical to the recursive least-squares lattice algorithms (RLSL) [1436–1444], and

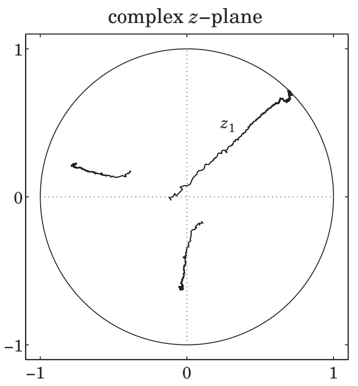


Fig. 16.12.4 Single sinusoid with order-3 adaptive filter.

they share the same properties of fast convergence and computational efficiency with the latter. Typically, the gradient lattice converges somewhat more slowly than RLSL. Some comparisons between the two types of algorithms are given in [1416,1443].

We start by casting the ordinary lattice filter of linear prediction in a gradient-adaptive form, and then discuss the gradient-adaptive form of the lattice Wiener filter, the stationary version of which was presented in Sec. 12.11.

The lattice recursion for an Mth order prediction-error filter of a stationary signal y_n was found in Sec. 12.7 to be

$$\begin{aligned} e^+_{p+1}(n) &= e^+_p(n) - \gamma_{p+1} e^-_p(n-1) \\ e^-_{p+1}(n) &= e^-_p(n-1) - \gamma_{p+1} e^+_p(n) \end{aligned} \tag{16.13.1}$$

for $p = 0, 1, \dots, M-1$, and where $e^\pm_0(n) = y_n$. The optimal value of the reflection coefficient γ_{p+1} can be obtained by minimizing the performance index

$$\mathcal{E}_{p+1} = E[e^+_{p+1}(n)^2 + e^-_{p+1}(n)^2] \tag{16.13.2}$$

Differentiating with respect to γ_{p+1}, we find

$$\frac{\partial \mathcal{E}_{p+1}}{\partial \gamma_{p+1}} = E\left[e^+_{p+1}(n)\frac{\partial e^+_{p+1}(n)}{\partial \gamma_{p+1}} + e^-_{p+1}(n)\frac{\partial e^-_{p+1}(n)}{\partial \gamma_{p+1}}\right]$$

and using Eq. (16.13.1)

$$\frac{\partial \mathcal{E}_{p+1}}{\partial \gamma_{p+1}} = -2E\big[e^+_{p+1}(n)e^-_p(n-1) + e^-_{p+1}(n)e^+_p(n)\big] \tag{16.13.3}$$

Inserting Eq. (16.13.1) into (16.13.3), we rewrite the latter as

$$\frac{\partial \mathcal{E}_{p+1}}{\partial \gamma_{p+1}} = -2(C_{p+1} - \gamma_{p+1}D_{p+1}) \tag{16.13.4}$$

where

$$C_{p+1} = 2E\big[e^+_p(n)e^-_p(n-1)\big] \tag{16.13.5}$$

$$D_{p+1} = E\big[e^+_p(n)^2 + e^-_p(n-1)^2\big] \tag{16.13.6}$$

Setting the gradient (16.13.4) to zero, we find the optimal value of γ_{p+1}

$$\gamma_{p+1} = \frac{C_{p+1}}{D_{p+1}} = \frac{2E\big[e^+_p(n)e^-_p(n-1)\big]}{E\big[e^+_p(n)^2 + e^-_p(n-1)^2\big]} \tag{16.13.7}$$

which, due to the assumed stationarity, agrees with Eq. (12.7.3). Replacing the numerator and denominator of Eq. (16.13.7) by time averages leads to Burg's method.

The gradient adaptive lattice is obtained by solving $\partial\mathcal{E}_{p+1}/\partial\gamma_{p+1} = 0$ iteratively by the gradient-descent method

$$\gamma_{p+1}(n+1) = \gamma_{p+1}(n) - \mu_{p+1}\frac{\partial \mathcal{E}_{p+1}}{\partial \gamma_{p+1}(n)} \tag{16.13.8}$$

where μ_{p+1} is a small positive adaptation constant. Before we drop the expectation instructions in Eq. (16.13.3), we use the result of Eq. (16.13.4) to discuss qualitatively the convergence rate of the algorithm. Inserting Eq. (16.13.4) into (16.13.8), we find

$$\gamma_{p+1}(n+1) = \gamma_{p+1}(n) + 2\mu_{p+1}(C_{p+1} - \gamma_{p+1}(n)D_{p+1})$$

or,

$$\gamma_{p+1}(n+1) = (1 - 2\mu_{p+1}D_{p+1})\gamma_{p+1}(n) + 2\mu_{p+1}C_{p+1} \tag{16.13.9}$$

Actually, if we replace γ_{p+1} by $\gamma_{p+1}(n)$ in Eq. (16.13.1), the stationarity of the lattice is lost, and it is not correct to assume that C_{p+1} and D_{p+1} are independent of n. The implicit dependence of C_{p+1} and D_{p+1} on the (time-varying) reflection coefficients of the previous lattice stages makes Eq. (16.13.9) a nonlinear difference equation in the reflection coefficients. In the analogous discussion of the LMS case in Sec. 16.6, the corresponding difference equation for the weights was linear with constant coefficients. Because of the tapped delay-line structure, the stationarity of the input signal $\mathbf{y}(n)$ was not affected by the time-varying weights. Nevertheless, we will use Eq. (16.13.9) in a qualitative manner, replacing C_{p+1} and D_{p+1} by their constant asymptotic values, but only for the purpose of motivating the final choice of the adaptation parameter μ_{p+1}. The solution of Eq. (16.13.9), then, is

$$\gamma_{p+1}(n) = \gamma_{p+1} + (1 - 2\mu_{p+1}D_{p+1})^n(\gamma_{p+1}(0) - \gamma_{p+1}) \tag{16.13.10}$$

where γ_{p+1} is the asymptotic value of the weight given in Eq. (16.13.7). The stability of Eqs. (16.13.9) and (16.13.10) requires that

$$|1 - 2\mu_{p+1}D_{p+1}| < 1 \tag{16.13.11}$$

If we choose μ_{p+1} as

$$2\mu_{p+1} = \frac{\alpha}{D_{p+1}} \tag{16.13.12}$$

then $1 - 2\mu_{p+1}D_{p+1} = 1 - \alpha$ will satisfy Eq. (16.13.11). Note that α was chosen to be independent of the order p. This implies that all reflection coefficients $\gamma_{p+1}(n)$ will essentially converge at the same rate. Using Eqs. (16.13.3) and (16.13.12), we write Eq. (16.13.8) as follows:

$$\gamma_{p+1}(n+1) = \gamma_{p+1}(n) + \frac{\alpha}{D_{p+1}}E\big[e^+_{p+1}(n)e^-_p(n-1) + e^-_{p+1}(n)e^+_p(n)\big] \tag{16.13.13}$$

The practical implementation of this method consists of ignoring the expectation instruction, and using a least-squares approximation for D_{p+1} of the form [1411–1413]

$$D_{p+1}(n) = (1-\lambda)\sum_{k=0}^{n}\lambda^{n-k}\big[e^+_p(k)^2 + e^-_p(k-1)^2\big] \tag{16.13.14}$$

where $0 < \lambda < 1$. It may also be computed recursively by

$$D_{p+1}(n) = \lambda D_{p+1}(n-1) + (1-\lambda)\big[e_p^+(n)^2 + e_p^-(n-1)^2\big] \qquad (16.13.15)$$

This quantity is a measure of D_{p+1} of Eq. (16.13.6); indeed, taking expectations of both sides and assuming stationarity, we find

$$\begin{aligned} E\big[D_{p+1}(n)\big] &= (1-\lambda)\sum_{k=0}^{n}\lambda^{n-k}E\big[e_p^+(k)^2 + e_p^-(k-1)^2\big] \\ &= (1-\lambda)\sum_{k=0}^{n}\lambda^{n-k}D_{p+1} = (1-\lambda^{n+1})D_{p+1} \end{aligned}$$

which converges to D_{p+1} for large n. With the above changes, we obtain the adaptive version of Eq. (16.13.13),

$$\gamma_{p+1}(n+1) = \gamma_{p+1}(n) + \frac{\alpha}{D_{p+1}(n)}\big[e_{p+1}^+(n)e_p^-(n-1) + e_{p+1}^-(n)e_p^+(n)\big] \qquad (16.13.16)$$

It can be written in a slightly different form by defining the quantity

$$\begin{aligned} d_{p+1}(n) &= \sum_{k=0}^{n}\lambda^{n-k}\big[e_p^+(k)^2 + e_p^-(k-1)^2\big] \\ &= \lambda d_{p+1}(n-1) + \big[e_p^+(n)^2 + e_p^-(n-1)^2\big] \end{aligned} \qquad (16.13.17)$$

and noting that $D_{p+1}(n) = (1-\lambda)d_{p+1}(n)$. Defining the new parameter $\beta = \alpha/(1-\lambda)$, we rewrite Eq. (16.13.16) in the form

$$\gamma_{p+1}(n+1) = \gamma_{p+1}(n) + \frac{\beta}{d_{p+1}(n)}\big[e_{p+1}^+(n)e_p^-(n-1) + e_{p+1}^-(n)e_p^+(n)\big] \qquad (16.13.18)$$

This is usually operated with $\beta = 1$ or, equivalently, $\alpha = 1-\lambda$. This choice makes Eq. (16.13.18) equivalent to a recursive reformulation of Burg's method [1411–1413]. This may be seen as follows. Set $\beta = 1$ and define the quantity $c_{p+1}(n)$ by

$$c_{p+1}(n) = \sum_{k=0}^{n}\lambda^{n-k}\big[2e_p^+(k)e_p^-(k-1)\big]$$

Then, inserting Eq. (16.13.1), with γ_{p+1} replaced by $\gamma_{p+1}(n)$, into Eq. (16.13.18), we find after some algebra

$$\gamma_{p+1}(n+1) = \frac{c_{p+1}(n)}{d_{p+1}(n)}$$

or, written explicitly

$$\gamma_{p+1}(n+1) = \frac{2\displaystyle\sum_{k=0}^{n}\lambda^{n-k}\big[e_p^+(k)e_p^-(k-1)\big]}{\displaystyle\sum_{k=0}^{n}\lambda^{n-k}\big[e_p^+(k)^2 + e_p^-(k-1)^2\big]} \qquad (16.13.19)$$

which corresponds to Burg's method, and also guarantees that $|\gamma_{p+1}(n+1)|$ will remain less than one at each iteration. The adaptive lattice is depicted in Fig. 16.13.1. At each time instant n, the order recursions (16.13.1) are

$$\begin{aligned} e_{p+1}^+(n) &= e_p^+(n) - \gamma_{p+1}(n)e_p^-(n-1) \\ e_{p+1}^-(n) &= e_p^-(n-1) - \gamma_{p+1}(n)e_p^+(n) \end{aligned} \qquad (16.13.20)$$

for $p = 0, 1, \dots, M-1$, with $\gamma_{p+1}(n)$ updated in time using Eq. (16.13.18) or Eq. (16.13.19). Initialize (16.13.20) by $e_0^\pm(n) = y_n$. We summarize the computational steps as follows:

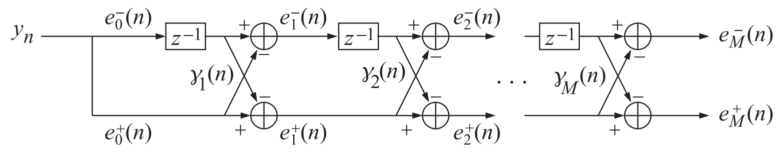


Fig. 16.13.1 Adaptive lattice predictor.

1. At time n, the coefficients $\gamma_{p+1}(n)$ and $d_{p+1}(n-1)$ are available.
2. Iterate Eq. (16.13.20) for $p = 0, 1, \dots, M-1$.
3. Using Eq. (16.13.17), compute $d_{p+1}(n)$ for $p = 0, 1, \dots, M-1$.
4. Using Eq. (16.13.18), compute $\gamma_{p+1}(n+1)$ for $p = 0, 1, \dots, M-1$.
5. Go to $n \to n+1$.

Next, we discuss the adaptive lattice realization of the FIR Wiener filter of Sec. 12.11. We use the same notation as in that section. The time-invariant lattice weights g_p are chosen optimally to minimize the mean-square estimation error

$$\mathcal{E} = E[e_n^2] = \min \qquad (16.13.21)$$

where $e_n = x_n - \hat{x}_n$, and

$$\hat{x}_n = \sum_{p=0}^{M} g_p e_p^-(n) = [g_0, g_1, \dots, g_M]\begin{bmatrix} e_0^-(n) \\ e_1^-(n) \\ \vdots \\ e_M^-(n) \end{bmatrix} = \mathbf{g}^T\mathbf{e}^-(n) \qquad (16.13.22)$$

The gradient with respect to $\mathbf{g}$ is

$$\frac{\partial\mathcal{E}}{\partial\mathbf{g}} = -2E\big[e_n\mathbf{e}^-(n)\big] \qquad (16.13.23)$$

Inserting Eq. (16.13.22) into (16.13.23), we rewrite the latter as

$$\frac{\partial\mathcal{E}}{\partial\mathbf{g}} = -2(\mathbf{r} - R\mathbf{g}) \qquad (16.13.24)$$

where $\mathbf{r}$ and R are defined in terms of the *backward* lattice signals $e_p^-(n)$ as

$$\mathbf{r} = E\big[x_n\mathbf{e}^-(n)\big], \quad R = E\big[\mathbf{e}^-(n)\mathbf{e}^-(n)^T\big] \qquad (16.13.25)$$

The gradient-descent method applied to the weights $\mathbf{g}$ is

$$\mathbf{g}(n+1) = \mathbf{g}(n) - \mathcal{M}\frac{\partial\mathcal{E}}{\partial\mathbf{g}(n)} \qquad (16.13.26)$$

where, following the discussion of Sec. 16.6, we have used a positive definite symmetric adaptation matrix $\mathcal{M}$, to be chosen below. Then, Eq. (16.13.26) becomes

$$\mathbf{g}(n+1) = (I - 2\mathcal{M}R)\mathbf{g}(n) + 2\mathcal{M}\mathbf{r} \qquad (16.13.27)$$

The orthogonality of the backward prediction errors $\mathbf{e}^-(n)$ causes their covariance matrix R to be diagonal

$$R = \mathrm{diag}\{E_0, E_1, \dots, E_M\} \qquad (16.13.28)$$

where E_p is the variance of $e_p^-(n)$

$$E_p = E[e_p^-(n)^2]\,,\quad p = 0,1,\dots,M \tag{16.13.29}$$

If we choose $\mathcal{M}$ to be diagonal, say, $\mathcal{M} = \text{diag}\{\mu_0,\mu_1,\dots,\mu_M\}$, then the state matrix $(I - 2\mathcal{M}R)$ of Eq. (16.13.27) will also be diagonal and, therefore, Eq. (16.13.27) will decouple into its individual components

$$g_p(n+1) = (1 - 2\mu_p E_p)g_p(n) + 2\mu_p r_p\,,\quad p = 0,1,\dots,M \tag{16.13.30}$$

where $r_p = E[x_n e_p^-(n)]$. Its solution is

$$g_p(n) = g_p + (1 - 2\mu_p E_p)^n(g_p(0) - g_p) \tag{16.13.31}$$

where $g_p = r_p/E_p$ are the *optimal weights*. The convergence rate depends on the quantity $(1 - 2\mu_p E_p)$. Choosing μ_p such that

$$2\mu_p = \frac{\alpha}{E_p}\,,\quad 0 < \alpha < 1 \tag{16.13.32}$$

implies that all lattice weights $g_p(n)$ will have the same rate of convergence. Using Eqs. (16.13.32) and (16.13.23) we can rewrite Eq. (16.13.26) component-wise as follows

$$g_p(n+1) = g_p(n) + \frac{\alpha}{E_p}E[e_n e_p^-(n)]$$

Ignoring the expectation instruction, and replacing E_p by its time average,

$$E_p(n) = (1-\lambda)\sum_{k=0}^{n}\lambda^{n-k}e_p^-(k)^2 = \lambda E_p(n-1) + (1-\lambda)e_p^-(n)^2 \tag{16.13.33}$$

we obtain the adaptation equation for the pth weight

$$g_p(n+1) = g_p(n) + \frac{\alpha}{E_p(n)}\,e_n e_p^-(n)\,,\quad p = 0,1,\dots,M \tag{16.13.34}$$

Defining

$$d_p^-(n) = \sum_{k=0}^{n}\lambda^{n-k}e_p^-(k)^2 = \lambda d_p^-(n-1) + e_p^-(n)^2 \tag{16.13.35}$$

and noting that $E_p(n) = (1-\lambda)d_p^-(n)$, we rewrite Eq. (16.13.34) as

$$g_p(n+1) = g_p(n) + \frac{\beta}{d_p^-(n)}\,e_n e_p^-(n)\,,\quad p = 0,1,\dots,M \tag{16.13.36}$$

where $\beta = \alpha/(1-\lambda)$. Typically, Eq. (16.13.36) is operated with $\beta = 1$, or $\alpha = 1-\lambda$, [1411–1413]. The realization of the adaptive lattice Wiener filter is shown in Fig. 16.13.2.

A slightly different version of the algorithm is obtained by replacing e_n in Eq. (16.13.36) by $e_p(n)$, that is, the estimation error based on a pth order Wiener filter:

$$e_p(n) = x_n - \hat{x}_p(n)\,,\quad \hat{x}_p(n) = \sum_{i=0}^{p} g_i e_i^-(n)$$

It satisfies the recursions (12.11.10) through (12.11.11). This version arises by minimizing the order-p performance index $\mathcal{E}_p = E[e_p(n)^2]$ rather than the order-M performance index (16.13.21). This version is justified by the property that all lower order portions of $\mathbf{g}$ are already optimal. If $\{g_0, g_1, \dots, g_{p-1}\}$ are already optimal, then to go to the next order p it is only necessary to determine the optimal value of the new weight g_p, which is obtained by minimizing $\mathcal{E}_p$ with respect to g_p. The overall algorithm is summarized below:

1. At time n, the quantities $\gamma_p(n)$, $d_p(n-1)$, for $p = 1,2,\dots,M$ and $g_p(n)$, $d_p^-(n-1)$, for $p = 0,1,\dots,M$, are available, as well as the current input samples x_n, y_n.
2. Initialize in order by

$$e_0^\pm(n) = y_n\,,\quad \hat{x}_0(n) = g_0(n)e_0^-(n)\,,\quad e_0(n) = x_n - \hat{x}_0(n)$$
$$d_0^-(n) = \lambda d_0^-(n-1) + e_0^-(n)^2$$
$$g_0(n+1) = g_0(n) + \frac{\beta}{d_0^-(n)}\,e_0(n)e_0^-(n)$$

3. For $p = 1,2,\dots,M$, compute:

$$e_p^+(n) = e_{p-1}^+(n) - \gamma_p(n)e_{p-1}^-(n-1)$$
$$e_p^-(n) = e_{p-1}^-(n-1) - \gamma_p(n)e_{p-1}^+(n)$$
$$d_p(n) = \lambda d_p(n-1) + e_{p-1}^+(n)^2 + e_{p-1}^-(n-1)^2$$
$$\gamma_p(n+1) = \gamma_p(n) + \frac{\beta}{d_p(n)}\left[e_p^+(n)e_{p-1}^-(n-1) + e_p^-(n)e_{p-1}^+(n)\right]$$
$$\hat{x}_p(n) = \hat{x}_{p-1}(n) + g_p(n)e_p^-(n)$$
$$e_p(n) = e_{p-1}(n) - g_p(n)e_p^-(n)$$
$$d_p^-(n) = \lambda d_p^-(n-1) + e_p^-(n)^2$$
$$g_p(n+1) = g_p(n) + \frac{\beta}{d_p^-(n)}\,e_p(n)e_p^-(n)$$

4. Go to the next time instant, $n \to n+1$.

The adaptation of the reflection coefficients $\gamma_p(n)$ provides a gradual orthogonalization of the backward error signals $e_p^-(n)$, which in turn drive the adaptation equations for the lattice weights $g_p(n)$.

The algorithm is initialized in time by setting $\gamma_p(0) = 0$, $d_p(-1) = 0$, $g_p(0) = 0$, $d_p^-(-1) = 0$. Because initially all the γs and the delay registers of the lattice are zero, it follows that the backward output of the pth lattice section, $e_p^-(n)$, will be zero for $n < p$. The corresponding

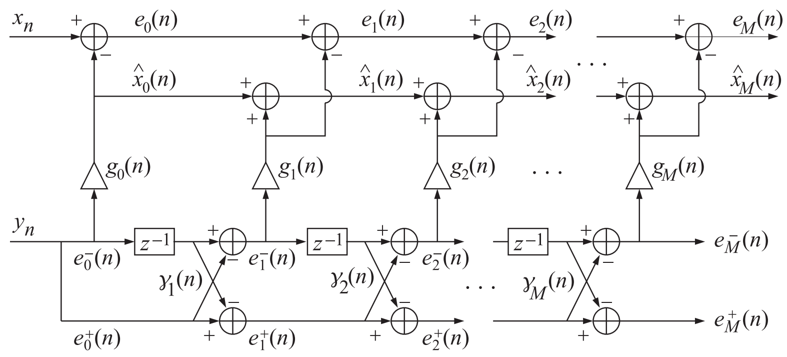


Fig. 16.13.2 Adaptive lattice Wiener filter.

$d_p^-(n)$ will also be zero and thus cannot be used in the updating of $g_p(n)$. During this startup period, we keep $g_p(n)= 0$, $n < p$. A similar problem does not arise for the γs because $d_p(n)$ contains contributions from the forward lattice outputs, which are not zero.

The function **glwf** is an implementation of the gradient lattice Wiener filter. It is the same as **lwf** with the weight adaptation parts added to it. Next, we present a simulation example. The signals x_n and y_n were generated by

$$x_n = y_n + 1.5y_{n-1} - 2y_{n-2} + u_n\,,\quad y_n = 0.75y_{n-1} - 0.5y_{n-2} + \epsilon_n$$

where u_n and ϵ_n were mutually independent, zero-mean, unit-variance, white noises. It follows from our general discussion in Sec. 16.5 that we must use a Wiener filter of order at least $M = 2$ to cancel completely the y-dependent part of x_n. Solving the order-two linear prediction problem for y_n using **bkwlev**, we find the theoretical L matrix and reflection coefficients

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0.5 & -0.75 & 1 \end{bmatrix},\quad \gamma_1 = 0.5\,,\quad \gamma_2 = -0.5 \tag{16.13.37}$$

The direct-form coefficients of the Wiener filter are precisely the coefficients of the y-dependent part of x_n. Thus, we have

$$\mathbf{h} = \begin{bmatrix} 1 \\ 1.5 \\ -2 \end{bmatrix},\quad \mathbf{g} = L^{-T}\mathbf{h} = \begin{bmatrix} 2 \\ 0 \\ -2 \end{bmatrix} \tag{16.13.38}$$

In the simulation we generated 100 samples of x_n and y_n (after letting the transients of the difference equation of y_n die out). The function **glwf** was run on these samples with $\lambda = 1$ and $\beta = 1$. Fig. 16.13.3 shows the adaptive reflection coefficients $\gamma_1(n)$ and $\gamma_2(n)$ versus iteration number n. The figure shows on the right the three coefficients $g_p(n)$, $p = 0,1,2$, versus n, converging to their theoretical values g_p above. For comparison purposes, we have also included the direct-form weight $h_2(n)$ adapted according to the standard LMS algorithm with $\mu = 0.01$. It should be compared to $g_2(n)$ because by construction the last elements of **g** and **h** are the same; here, $g_2 = h_2$. The LMS algorithm can be accelerated somewhat by using a larger μ, but at the expense of increasing the noisiness of the weights.

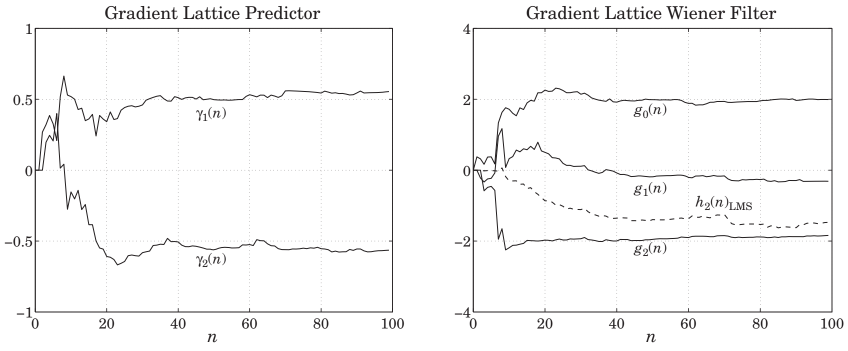


Fig. 16.13.3 Adaptive coefficients $\gamma_p(n)$ and $g_p(n)$.

16.14 Adaptive Gram-Schmidt Preprocessors

In this section we derive the *spatial* analogs of the gradient adaptive lattice algorithms. The main function of the adaptive lattice filter is to decorrelate the tapped delay-line data vector $\mathbf{y}(n)= [y_n, y_{n-1},\dots,y_{n-M}]^T$. In effect, it carries out the Gram-Schmidt orthogonalization of the components of $\mathbf{y}(n)$ at each time instant n. In array processing problems, because the data vector $\mathbf{y}(n)= [y_0(n), y_1(n),\dots,y_M(n)]^T$ does not have the tapped-delay line property, the Gram-Schmidt orthogonalization cannot be done by a simple a lattice filter. It requires a more complicated structure that basically amounts to carrying out the lower triangular linear transformation $\mathbf{y} = B\boldsymbol{\epsilon}$, which decorrelates the covariance matrix of $\mathbf{y}$.

The Gram-Schmidt construction of an arbitrary random vector $\mathbf{y}$ was discussed in Sec. 1.6. Here, we recast these results in a way that can be used directly in gradient-adaptive implementations. The Gram-Schmidt construction proceeds recursively starting at one end, say, $\epsilon_0 = y_0$. At the mth step of the recursion, we have available the mutually decorrelated components $\{\epsilon_0,\epsilon_1,\dots,\epsilon_{m-1}\}$. The next component ϵ_m is defined by

$$\epsilon_m = y_m - \sum_{i=0}^{m-1} b_{mi}\epsilon_i\,,\quad b_{mi} = \frac{1}{E_i}E[y_m\epsilon_i] \tag{16.14.1}$$

where $E_i = E[\epsilon_i^2]$. By construction, ϵ_m is decorrelated from all the previous ϵ_is, that is, $E[\epsilon_m\epsilon_i]= 0$, $i = 0,1\dots,m-1$. The summation term in Eq. (16.14.1) represents the optimum estimate of y_m based on the previous ϵ_is and ϵ_m represents the estimation error. Therefore, the coefficients b_{mi} can also be derived by the mean-square criterion

$$\mathcal{E}_m = E[\epsilon_m^2]= \min \tag{16.14.2}$$

The gradient with respect to b_{mi} is

$$\frac{\partial\mathcal{E}_m}{\partial b_{mi}} = -2E[\epsilon_m\epsilon_i]= -2\big(E[y_m\epsilon_i]-b_{mi}E_i\big) \tag{16.14.3}$$

where we used the fact that the previous ϵ_is are already decorrelated, so that $E[\epsilon_i\epsilon_j]= \delta_{ij}E_i$, for $i,j = 0,1,\dots,m-1$. Setting the gradient to zero gives the optimum solution (16.14.1) for b_{mi}. In a gradient-adaptive approach, the coefficients b_{mi} will be time-dependent, $b_{mi}(n)$, and updated by

$$b_{mi}(n+1)= b_{mi}(n)-\mu_{mi}\frac{\partial\mathcal{E}_m}{\partial b_{mi}(n)} = b_{mi}(n)+2\mu_{mi}E[\epsilon_m\epsilon_i] \tag{16.14.4}$$

Using the above expression for the gradient, we find the difference equation

$$b_{mi}(n+1)= (1-2\mu_{mi}E_i)b_{mi}(n)+2\mu_{mi}E[y_m\epsilon_i]$$

with solution, for $n \ge 0$

$$b_{mi}(n)= b_{mi} + (1-2\mu_{mi}E_i)^n\big(b_{mi}(0)-b_{mi}\big)$$

where b_{mi} is the optimum solution (16.14.1). As in Sec. 16.13, because of the diagonal nature of the covariance matrix of the previous ϵ_is, the system of difference equations for the b_{mi}s decouples into separate scalar equations. Choosing μ_{mi} by

$$2\mu_{mi} = \frac{\alpha}{E_i}\,,\quad 0 < \alpha < 1$$

implies that all coefficients $b_{mi}(n)$ will converge at the same rate. With this choice, Eq. (16.14.4) becomes

$$b_{mi}(n+1)= b_{mi}(n)+\frac{\alpha}{E_i}E[\epsilon_m\epsilon_i]$$

As before, we may replace E_i by its weighted time average $E_i(n) = (1-\lambda)d_i(n)$, where

$$d_i(n) = \sum_{k=0}^{n} \lambda^{n-k}\epsilon_i(k)^2 = \lambda d_i(n-1) + \epsilon_i(n)^2$$

Setting $\beta = \alpha/(1-\lambda)$ and dropping the expectation values, we obtain the adaptive Gram-Schmidt algorithm:

1. At time n, $b_{mi}(n)$ and $d_i(n-1)$ are available, and also the current data vector $\mathbf{y}(n) = [y_0(n), y_1(n), \dots, y_M(n)]^T$. (The algorithm is initialized in time by $b_{mi}(0) = 0$ and $d_i(-1) = 0$.)
2. Set $\epsilon_0(n) = y_0(n)$.
3. For $m = 1, 2, \dots, M$, compute:

$$\epsilon_m(n) = y_m(n) - \sum_{i=0}^{m-1} b_{mi}(n)\epsilon_i(n)$$

$$d_{m-1}(n) = \lambda d_{m-1}(n) + \epsilon_{m-1}(n)^2$$

for $i = 0, 1 \dots, m-1$, compute:

$$b_{mi}(n+1) = b_{mi}(n) + \frac{\beta}{d_i(n)}\,\epsilon_m(n)\epsilon_i(n)$$

4. Go to the next time instant, $n \to n+1$.

The conventional Gram-Schmidt construction builds up the matrix B row-wise; for example in the case $M = 3$

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ b_{10} & 1 & 0 & 0 \\ b_{20} & b_{21} & 1 & 0 \\ b_{30} & b_{31} & b_{32} & 1 \end{bmatrix}$$

According to Eq. (16.14.1), ϵ_m is constructed from the entries of the mth row of B. This gives rise to the block-diagram realization of the Gram-Schmidt construction shown in Fig. 16.14.1. We will see shortly that each circular block represents an elementary correlation canceling operation of the type [1355,1417–1421]

$$e = u - bv \qquad \begin{tikzcd} v \arrow[dr] & & \\ u \arrow[r] & b \arrow[r] & e \end{tikzcd}$$

with

$$E[ev] = 0 \quad \Rightarrow \quad b = \frac{E[uv]}{E[v^2]}$$

Therefore, each block can be replaced by an ordinary adaptive CCL or by an accelerated CCL, as discussed below. This point of view leads to an alternative way of organizing the Gram-Schmidt construction with better numerical properties, known as the *modified Gram-Schmidt* procedure [1166], which builds up the matrix B column-wise. Let $\mathbf{b}_i$ be the ith column of B, so that

$$\mathbf{y} = B\boldsymbol{\epsilon} = [\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_M]\begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_M \end{bmatrix} = \sum_{j=0}^{M} \mathbf{b}_j\epsilon_j$$

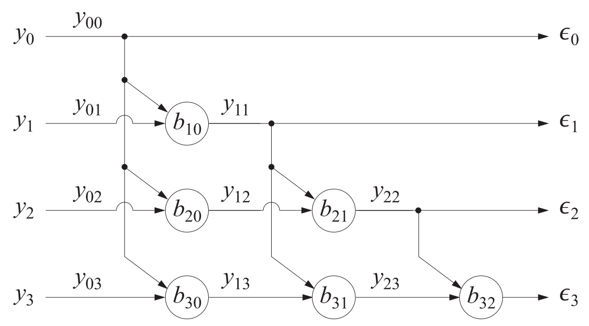


Fig. 16.14.1 Gram-Schmidt array preprocessor.

Removing the contribution of the first i columns, we define for $i = 1, 2, \dots, M$

$$\mathbf{y}_i = \mathbf{y} - \sum_{j=0}^{i-1} \mathbf{b}_j\epsilon_j = \sum_{j=i}^{M} \mathbf{b}_j\epsilon_j \tag{16.14.5}$$

Component-wise, we write

$$y_{im} = \sum_{j=i}^{M} b_{mj}\epsilon_j\,, \quad m = 0, 1, \dots, M$$

It follows from the lower-triangular nature of B that $y_{im} = 0$ for $m < i$. Moreover, because B has unit diagonal, we have at $m = i$ that $y_{ii} = b_{ii}\epsilon_i = \epsilon_i$. Thus,

$$\epsilon_i = y_{ii} \tag{16.14.6}$$

Equation (16.14.5) can be written recursively as follows

$$\mathbf{y}_i = \mathbf{b}_i\epsilon_i + \sum_{j=i+1}^{M} \mathbf{b}_j\epsilon_j = \mathbf{b}_i\epsilon_i + \mathbf{y}_{i+1}$$

or,

$$\mathbf{y}_{i+1} = \mathbf{y}_i - \mathbf{b}_i\epsilon_i \qquad \begin{tikzcd} \epsilon_i \arrow[dr] & & \\ \mathbf{y}_i \arrow[r] & \mathbf{b}_i \arrow[r] & \mathbf{y}_{i+1} \end{tikzcd}$$

and component-wise, $y_{i+1,m} = y_{im} - b_{mi}\epsilon_i$. The recursion is initialized by $\mathbf{y}_0 = \mathbf{y}$. It is evident by inspecting Fig. 16.14.1 that $\mathbf{y}_i$ represents the output column vector after each column operation. It follows also that each circular block is an elementary correlation canceler. This follows by noting that $\mathbf{y}_{i+1}$ is built out of ϵ_j with $j \geq i+1$, each being uncorrelated with ϵ_i. Thus,

$$E[\epsilon_i\mathbf{y}_{i+1}] = E[\epsilon_i\mathbf{y}_i] - \mathbf{b}_iE_i = 0 \quad \Rightarrow \quad \mathbf{b}_i = \frac{1}{E_i}E[\epsilon_i\mathbf{y}_i]$$

or, component-wise

$$b_{mi} = \frac{1}{E_i}E[\epsilon_i y_{im}]\,, \quad m = i+1, i+2, \dots, M \tag{16.14.7}$$

An adaptive implementation can be obtained easily by writing

$$\mathbf{b}_i(n+1) = \mathbf{b}_i(n) + 2\mu_i E[\epsilon_i\mathbf{y}_{i+1}] = (1 - 2\mu_i E_i)\mathbf{b}_i(n) + 2\mu_i E[\epsilon_i\mathbf{y}_i]$$

As usual, we set $2\mu_i = \alpha/E_i$, replace E_i by $E_i(n) = (1-\lambda)d_i(n)$, and drop the expectation values to obtain the following algorithm, which adapts the matrix elements of B column-wise:

1. At time n, $b_{mi}(n)$ and $d_i(n-1)$ are available, and also the current data vector $\mathbf{y}(n)=[y_0(n),y_1(n),\dots,y_M(n)]^T$.
2. Define $y_{0m}(n)=y_m(n)$, for $m=0,1,\dots,M$.
3. For $i=0,1,\dots,M$, compute:

```
$\epsilon_i(n)=y_{ii}(n)$
$d_i(n)=\lambda d_i(n-1)+\epsilon_i(n)^2$
For $i+1\le m\le M$, compute:
    $y_{i+1,m}(n)=y_{im}(n)-b_{mi}(n)\epsilon_i(n)$
    $b_{mi}(n+1)=b_{mi}(n)+\dfrac{\beta}{d_i(n)}\epsilon_i(n)y_{i+1,m}(n)$
```

4. Go to the next time instant, $n\to n+1$.

The algorithm may be appended to provide an overall Gram-Schmidt implementation of the adaptive linear combiner of Sec. 16.4. In the decorrelated basis, the estimate of x_n and estimation error may be written order recursively as

$$\hat{x}_i(n)=\hat{x}_{i-1}(n)+g_i(n)\epsilon_i(n)\,,\quad e_i(n)=e_{i-1}(n)-g_i(n)\epsilon_i(n) \tag{16.14.8}$$

with the weights $g_i(n)$ adapted by

$$g_i(n+1)=g_i(n)+\frac{\beta}{d_i(n)}e_i(n)\epsilon_i(n)\,,\quad i=0,1,\dots,M \tag{16.14.9}$$

The function **mgs** is an implementation of the adaptive modified Gram-Schmidt procedure. At each call, the function reads the snapshot vector $\mathbf{y}$, computes the decorrelated vector $\boldsymbol{\epsilon}$, and updates the matrix elements of B in preparation for the next call. An LMS-like version can be obtained by replacing the accelerated CCLs by ordinary CCLs [1355]

$$b_{mi}(n+1)=b_{mi}(n)+2\mu\epsilon_i(n)y_{i+1,m}(n) \tag{16.14.10}$$

An exact recursive least squares version of the modified Gram-Schmidt algorithm can also be derived [1421]. It bears the same relationship to the above gradient-based version that the exact RLS lattice filter bears to the gradient lattice filter. The computational complexity of the algorithm is high because there are $M(M+1)/2$ coefficients to be adapted at each time instant, namely, the matrix elements in the strictly lower triangular part of B. By contrast, in the lattice structure there are only M reflection coefficients to be adapted. Despite its computational complexity, the algorithm is quite modular, built out of elementary CCLs.

Next, we present a simulation example of order $M=2$. The vectors $\mathbf{y}$ were constructed by

$$\mathbf{y}=\begin{bmatrix}1&0&0\\-2&1&0\\1&2&1\end{bmatrix}\begin{bmatrix}\epsilon_0\\\epsilon_1\\\epsilon_2\end{bmatrix}=B\boldsymbol{\epsilon}$$

with the components of $\boldsymbol{\epsilon}$ having variances $E_0=1$, $E_1=4$, and $E_2=9$. We generated 100 independent snapshots $\boldsymbol{\epsilon}$ and computed the corresponding $\mathbf{y}=B\boldsymbol{\epsilon}$. Fig. 16.14.2 shows the two matrix elements $b_{10}(n)$ and $b_{21}(n)$ adapted by running **mgs** on the 100 snapshots with $\lambda=1$ and $\beta=1$. They are converging to the theoretical values $b_{10}=-2$ and $b_{21}=2$. On the right, the figure shows the same two matrix elements adapted by the LMS algorithm (16.14.10) with $\mu=0.01$.

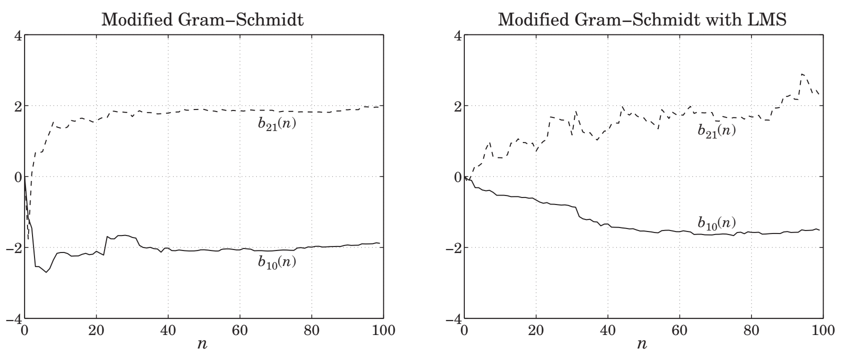


Fig. 16.14.2 Modified Gram-Schmidt algorithm and its LMS version.

16.15 Rank-One Modification of Covariance Matrices

All recursive least-squares (RLS) algorithms, conventional, lattice, and fast direct-form structures, can be derived from the rank-one updating properties of covariance matrices. In this section we discuss these properties and derive all the necessary algebraic steps and computational reductions that make the fast RLS versions possible. In the succeeding sections, we couple these results with the so-called *shift-invariance* property to close the loop, as it were, and complete the derivation of the fast RLS algorithms.

The rank-one modification of a covariance matrix R_0 is obtained by adding the rank-one term

$$R_1=R_0+\mathbf{y}\mathbf{y}^T \tag{16.15.1}$$

where $\mathbf{y}$ is a vector of the same dimension as R_0. Similarly, the modification of a cross-correlation vector $\mathbf{r}_0$ will be defined as follows, where x is a scalar

$$\mathbf{r}_1=\mathbf{r}_0+x\mathbf{y} \tag{16.15.2}$$

We define the *Wiener solutions* based on the pairs $R_0,\mathbf{r}_0$ and $R_1,\mathbf{r}_1$ by

$$\mathbf{h}_0=R_0^{-1}\mathbf{r}_0\,,\quad \mathbf{h}_1=R_1^{-1}\mathbf{r}_1 \tag{16.15.3}$$

and the corresponding *estimates* of x and estimation errors

$$\hat{x}_0=\mathbf{h}_0^T\mathbf{y}\,,\quad e_0=x-\hat{x}_0\quad\text{and}\quad \hat{x}_1=\mathbf{h}_1^T\mathbf{y}\,,\quad e_1=x-\hat{x}_1 \tag{16.15.4}$$

Similarly, using the notation of Sec. 1.8, we will consider the solution of the forward and backward prediction problems

$$R_0\mathbf{a}_0=E_{0a}\mathbf{u}\,,\quad R_1\mathbf{a}_1=E_{1a}\mathbf{u} \tag{16.15.5}$$

and

$$R_0\mathbf{b}_0=E_{0b}\mathbf{v}\,,\quad R_1\mathbf{b}_1=E_{1b}\mathbf{v} \tag{16.15.6}$$

and the corresponding forward and backward prediction errors

$$e_{0a}=\mathbf{a}_0^T\mathbf{y}\,,\quad e_{1a}=\mathbf{a}_1^T\mathbf{y}\quad\text{and}\quad e_{0b}=\mathbf{b}_0^T\mathbf{y}\,,\quad e_{1b}=\mathbf{b}_1^T\mathbf{y} \tag{16.15.7}$$

The basic question that we pose is how to construct the solution of the filtering and prediction problems 1 from the solution of the corresponding problems 0; that is, to construct $\mathbf{h}_1$ from $\mathbf{h}_0$, $\mathbf{a}_1$ from $\mathbf{a}_0$, and $\mathbf{b}_1$ from $\mathbf{b}_0$. We will generally refer to the various quantities of problem-0 as *a priori* and to the corresponding quantities of problem-1 as *a posteriori*. The constructions are carried out with the help of the so-called a priori and a posteriori *Kalman gain* vectors defined by

$$\mathbf{k}_0 = R_0^{-1}\mathbf{y}\,, \quad \mathbf{k}_1 = R_1^{-1}\mathbf{y} \tag{16.15.8}$$

We also define the so-called *likelihood variables*

$$\nu = \mathbf{y}^T R_0^{-1}\mathbf{y}\,, \quad \mu = \frac{1}{1+\nu} = \frac{1}{1+\mathbf{y}^T R_0^{-1}\mathbf{y}} \tag{16.15.9}$$

Note that the positivity condition $\nu > 0$ is equivalent to $0 < \mu < 1$. Multiplying Eq. (16.15.1) from the left by R_1^{-1} and from the right by R_0^{-1}, we obtain

$$R_0^{-1} = R_1^{-1} + R_1^{-1}\mathbf{y}\mathbf{y}^T R_0^{-1} = R_1^{-1} + \mathbf{k}_1\mathbf{k}_0^T \tag{16.15.10}$$

Acting on $\mathbf{y}$ and using the definitions (16.15.8) and (16.15.9), we find

$$R_0^{-1}\mathbf{y} = R_1^{-1}\mathbf{y} + \mathbf{k}_1\mathbf{k}_0^T\mathbf{y} \quad \Rightarrow \quad \mathbf{k}_0 = \mathbf{k}_1 + \mathbf{k}_1\nu = (1+\nu)\mathbf{k}_1 = \frac{1}{\mu}\mathbf{k}_1$$

or,

$$\mathbf{k}_1 = \mu\mathbf{k}_0 \tag{16.15.11}$$

It follows that

$$\mathbf{y}^T R_1^{-1}\mathbf{y} = \mathbf{k}_1^T\mathbf{y} = \mu\,\mathbf{k}_0^T\mathbf{y} = \mu\nu = \frac{\nu}{1+\nu} = 1 - \frac{1}{1+\nu} = 1-\mu$$

Thus, solving for μ

$$\mu = 1 - \mathbf{y}^T R_1^{-1}\mathbf{y} = \frac{1}{1+\mathbf{y}^T R_0^{-1}\mathbf{y}} \tag{16.15.12}$$

Solving Eq. (16.15.10) for R_1^{-1}, we obtain

$$R_1^{-1} = R_0^{-1} - \mathbf{k}_1\mathbf{k}_0^T = R_0^{-1} - \mu\,\mathbf{k}_0\mathbf{k}_0^T = R_0^{-1} - \frac{1}{1+\mathbf{y}^T R_0^{-1}\mathbf{y}} R_0^{-1}\mathbf{y}\mathbf{y}^T R_0^{-1} \tag{16.15.13}$$

which is recognized as the application of the matrix inversion lemma to Eq. (16.15.1). It provides the rank-one update of the inverse matrices. Denoting $P_0 = R_0^{-1}$ and $P_1 = R_1^{-1}$, we may rewrite Eq. (16.15.13) in the form

$$P_1 = P_0 - \mu\,\mathbf{k}_0\mathbf{k}_0^T\,, \quad \mathbf{k}_0 = P_0\mathbf{y}\,, \quad \mu = \frac{1}{1+\mathbf{y}^T P_0\mathbf{y}} \tag{16.15.14}$$

Before we derive the relationship between the Wiener solutions Eq. (16.15.3), we may obtain the relationship between the a priori and a posteriori estimation errors. Noting that the estimates can be written as,

$$\hat{x}_0 = \mathbf{h}_0^T\mathbf{y} = \mathbf{r}_0^T R_0^{-1}\mathbf{y} = \mathbf{r}_0^T\mathbf{k}_0$$

$$\hat{x}_1 = \mathbf{h}_1^T\mathbf{y} = \mathbf{r}_1^T R_1^{-1}\mathbf{y} = \mathbf{r}_1^T\mathbf{k}_1$$

and using Eq. (16.15.2), we obtain

$$\hat{x}_1 = \mathbf{k}_1^T\mathbf{r}_1 = (\mu\mathbf{k}_0)^T(\mathbf{r}_0 + x\mathbf{y}) = \mu\hat{x}_0 + \mu\nu x = \mu\hat{x}_0 + (1-\mu)x = x - \mu e_0$$

from which it follows that

$$e_1 = \mu e_0 \tag{16.15.15}$$

The simplest method of determining the relationship between the $\mathbf{h}_1$ and $\mathbf{h}_0$ is to act on $\mathbf{h}_0$ by the covariance matrix R_1 of problem-1, and then use the recursions (16.15.1) and (16.15.2), that is,

$$R_1\mathbf{h}_0 = (R_0 + \mathbf{y}\mathbf{y}^T)\mathbf{h}_0 = \mathbf{r}_0 + \hat{x}_0\mathbf{y} = (\mathbf{r}_1 - x\mathbf{y}) + \hat{x}_0\mathbf{y} = \mathbf{r}_1 - e_0\mathbf{y}$$

Multiplying by R_1^{-1}, we find

$$\mathbf{h}_0 = R_1^{-1}\mathbf{r}_1 - e_0 R_1^{-1}\mathbf{y} = \mathbf{h}_1 - e_0\mathbf{k}_1$$

or, solving for $\mathbf{h}_1$ and using Eqs. (16.15.11) and (16.15.15)

$$\mathbf{h}_1 = \mathbf{h}_0 + e_0\mathbf{k}_1 = \mathbf{h}_0 + \mu e_0\mathbf{k}_0 = \mathbf{h}_0 + e_1\mathbf{k}_0 \tag{16.15.16}$$

Note that the update term can be expressed either in terms of the a priori estimation error e_0 and a posteriori Kalman gain $\mathbf{k}_1$, or the a posteriori error e_1 and a priori Kalman gain $\mathbf{k}_0$. Next, we summarize what may be called the *conventional RLS* computational sequence:

1. $\mathbf{k}_0 = P_0\mathbf{y}$
2. $\nu = \mathbf{k}_0^T\mathbf{y}\,, \quad \mu = \dfrac{1}{1+\nu}$
3. $\mathbf{k}_1 = \mu\mathbf{k}_0$
4. $P_1 = P_0 - \mathbf{k}_1\mathbf{k}_0^T$
5. $\hat{x}_0 = \mathbf{h}_0^T\mathbf{y}\,, \quad e_0 = x - \hat{x}_0\,, \quad e_1 = \mu e_0\,, \quad \hat{x}_1 = x - e_1$
6. $\mathbf{h}_1 = \mathbf{h}_0 + e_0\mathbf{k}_1$

Because in step 4 an entire matrix is updated, the computational complexity of the algorithm grows quadratically with the matrix order; that is, $O(M^2)$ operations.

Next, we consider the forward and backward prediction solutions. Equations (1.8.28) and (1.8.35) applied to R_0 become

$$R_0^{-1} = \begin{bmatrix} 0 & \mathbf{0}^T \\ \mathbf{0} & \tilde{R}_0^{-1} \end{bmatrix} + \frac{1}{E_{0a}}\mathbf{a}_0\mathbf{a}_0^T = \begin{bmatrix} \bar{R}_0^{-1} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} + \frac{1}{E_{0b}}\mathbf{b}_0\mathbf{b}_0^T$$

Acting on $\mathbf{y}$ and using Eq. (16.15.7), we find

$$\mathbf{k}_0 = \begin{bmatrix} 0 \\ \tilde{\mathbf{k}}_0 \end{bmatrix} + \frac{e_{0a}}{E_{0a}}\mathbf{a}_0 = \begin{bmatrix} \bar{\mathbf{k}}_0 \\ 0 \end{bmatrix} + \frac{e_{0b}}{E_{0b}}\mathbf{b}_0 \tag{16.15.17}$$

where $\tilde{\mathbf{k}}_0 = \tilde{R}_0^{-1}\tilde{\mathbf{y}}$ and $\bar{\mathbf{k}}_0 = \bar{R}_0^{-1}\bar{\mathbf{y}}$, where we recall the decompositions (1.8.2) and (1.8.3)

$$\mathbf{y} = \begin{bmatrix} y_a \\ \tilde{\mathbf{y}} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{y}} \\ y_b \end{bmatrix}$$

Similarly, we obtain for the a posteriori gains

$$\mathbf{k}_1 = \begin{bmatrix} 0 \\ \tilde{\mathbf{k}}_1 \end{bmatrix} + \frac{e_{1a}}{E_{1a}}\mathbf{a}_1 = \begin{bmatrix} \bar{\mathbf{k}}_1 \\ 0 \end{bmatrix} + \frac{e_{1b}}{E_{1b}}\mathbf{b}_1 \tag{16.15.18}$$

Because $\mathbf{b}_0$ and $\mathbf{b}_1$ have last coefficients of unity, it follows that the *last* coefficients of the Kalman gains will be

$$k_{0b} = \frac{e_{0b}}{E_{0b}}\,, \quad k_{1b} = \frac{e_{1b}}{E_{1b}} \tag{16.15.19}$$

Similarly, the *first* coefficients will be

$$k_{0a} = \frac{e_{0a}}{E_{0a}}\,, \quad k_{1a} = \frac{e_{1a}}{E_{1a}} \tag{16.15.20}$$

Taking the dot product of Eq. (16.15.17) with $\mathbf{y}$ and using the definition (16.15.9) and (16.15.7), we obtain

$$\nu = \tilde{\nu} + \frac{e_{0a}^2}{E_{0a}} = \bar{\nu} + \frac{e_{0b}^2}{E_{0b}}$$

or,

$$\nu = \tilde{\nu} + e_{0a}k_{0a} = \bar{\nu} + e_{0b}k_{0b} \tag{16.15.21}$$

where $\tilde{\nu} = \tilde{\mathbf{k}}_0^T\tilde{\mathbf{y}}$ and $\bar{\nu} = \bar{\mathbf{k}}_0^T\bar{\mathbf{y}}$. Similarly, using $\mathbf{k}_1^T\mathbf{y} = 1 - \mu$ and taking the dot product of Eq. (16.15.18) with $\mathbf{y}$, we find

$$1 - \mu = 1 - \tilde{\mu} + \frac{e_{1a}^2}{E_{1a}} = 1 - \bar{\mu} + \frac{e_{1b}^2}{E_{1b}}$$

or,

$$\mu = \tilde{\mu} - \frac{e_{1a}^2}{E_{1a}} = \bar{\mu} - \frac{e_{1b}^2}{E_{1b}} \tag{16.15.22}$$

This is equivalent to Eq. (16.15.21). To relate $\mathbf{a}_1$ and $\mathbf{a}_0$, we apply the usual method of acting on the a priori solution $\mathbf{a}_0$ by the a posteriori covariance matrix R_1:

$$R_1\mathbf{a}_0 = (R_0 + \mathbf{y}\mathbf{y}^T)\mathbf{a}_0 = R_0\mathbf{a}_0 + \mathbf{y}(\mathbf{y}^T\mathbf{a}_0) = E_{0a}\mathbf{u} + e_{0a}\mathbf{y}$$

Multiplying by R_1^{-1} and using $R_1^{-1}\mathbf{u} = \mathbf{a}_1/E_{1a}$, we obtain

$$\mathbf{a}_0 = \frac{E_{0a}}{E_{1a}}\mathbf{a}_1 + e_{0a}\mathbf{k}_1 \tag{16.15.23}$$

This has five useful consequences. First, equating first coefficients and using Eq. (16.15.20), we obtain

$$1 = \frac{E_{0a}}{E_{1a}} + e_{0a}k_{1a} = \frac{E_{0a}}{E_{1a}} + \frac{e_{0a}e_{1a}}{E_{1a}} \tag{16.15.24}$$

or,

$$E_{1a} = E_{0a} + e_{0a}e_{1a} \tag{16.15.25}$$

Second, writing Eq. (16.15.24) in the form $E_{0a}/E_{1a} = 1 - e_{0a}k_{1a}$, we rewrite Eq. (16.15.23) as

$$\mathbf{a}_0 = (1 - e_{0a}k_{1a})\mathbf{a}_1 + e_{0a}\mathbf{k}_1 = \mathbf{a}_1 + e_{0a}(\mathbf{k}_1 - k_{1a}\mathbf{a}_1) = \mathbf{a}_1 + e_{0a}\begin{bmatrix} 0 \\ \tilde{\mathbf{k}}_1 \end{bmatrix}$$

where we used Eq. (16.15.18). Thus,

$$\mathbf{a}_1 = \mathbf{a}_0 - e_{0a}\begin{bmatrix} 0 \\ \tilde{\mathbf{k}}_1 \end{bmatrix} \tag{16.15.26}$$

Third, taking the dot product with $\mathbf{y}$ and using $\tilde{\mathbf{k}}_1^T\tilde{\mathbf{y}} = 1 - \tilde{\mu}$, we find

$$e_{1a} = \mathbf{a}_1^T\mathbf{y} = \mathbf{a}_0^T\mathbf{y} - e_{0a}(\tilde{\mathbf{k}}_1^T\tilde{\mathbf{y}}) = e_{0a} - (1 - \tilde{\mu})e_{0a} = \tilde{\mu}e_{0a} \qquad \text{or,}$$

$$e_{1a} = \tilde{\mu}e_{0a} \tag{16.15.27}$$

This is analogous to Eq. (16.15.15). Fourth, writing $e_{0a} = e_{1a}/\tilde{\mu} = (1+\tilde{\nu})e_{1a}$, it follows by adding one to Eq. (16.15.21) that

$$(1 + \nu) = (1 + \tilde{\nu}) + (1 + \tilde{\nu})e_{1a}\frac{e_{0a}}{E_{0a}} = (1 + \tilde{\nu})\frac{E_{0a} + e_{0a}e_{1a}}{E_{0a}} = (1 + \tilde{\nu})\frac{E_{1a}}{E_{0a}}$$

and inverting,

$$\mu = \tilde{\mu}\frac{E_{0a}}{E_{1a}} \tag{16.15.28}$$

This, in turn, is equivalent to Eq. (16.15.22) as can be seen by

$$\mu = \tilde{\mu}\frac{E_{1a} - e_{0a}e_{1a}}{E_{1a}} = \tilde{\mu} - (\tilde{\mu}e_{0a})\frac{e_{1a}}{E_{1a}} = \tilde{\mu} - \frac{e_{1a}^2}{E_{1a}}$$

Fifth, using Eq. (16.15.27) and the result $\tilde{\mathbf{k}}_1 = \tilde{\mu}\tilde{\mathbf{k}}_0$, we may rewrite Eq. (16.15.26) in terms of the a posteriori error e_{1a} and the a priori gain $\tilde{\mathbf{k}}_0$ as follows

$$\mathbf{a}_1 = \mathbf{a}_0 - e_{1a}\begin{bmatrix} 0 \\ \tilde{\mathbf{k}}_0 \end{bmatrix} \tag{16.15.29}$$

Defining the inverse matrices $\tilde{P}_0 = \tilde{R}_0^{-1}$ and $\tilde{P}_1 = \tilde{R}_1^{-1}$, we summarize the conventional RLS computational sequence for the *forward predictor:*

1. $\tilde{\mathbf{k}}_0 = \tilde{P}_0\tilde{\mathbf{y}}$
2. $\tilde{\nu} = \tilde{\mathbf{k}}_0^T\tilde{\mathbf{y}}, \quad \tilde{\mu} = \dfrac{1}{1+\tilde{\nu}}$
3. $\tilde{\mathbf{k}}_1 = \tilde{\mu}\,\tilde{\mathbf{k}}_0$
4. $\tilde{P}_1 = \tilde{P}_0 - \tilde{\mathbf{k}}_1\tilde{\mathbf{k}}_0^T$
5. $e_{0a} = \mathbf{a}_0^T\mathbf{y}, \quad e_{1a} = \tilde{\mu}e_{0a}$
6. $\mathbf{a}_1 = \mathbf{a}_0 - e_{0a}\begin{bmatrix} 0 \\ \tilde{\mathbf{k}}_1 \end{bmatrix}$

The fast RLS algorithms make use also of the backward predictors. Starting with $R_1\mathbf{b}_0 = (R_0 + \mathbf{y}\mathbf{y}^T)\mathbf{b}_0 = E_{0b}\mathbf{v} + e_{0b}\mathbf{y}$, and following similar steps as for the forward case, we obtain parallel results for the backward predictor, that is,

$$\mathbf{b}_0 = \frac{E_{0b}}{E_{1b}}\mathbf{b}_1 + e_{0b}\mathbf{k}_1 \tag{16.15.30}$$

from which it follows that

$$1 = \frac{E_{0b}}{E_{1b}} + e_{0b}k_{1b} = \frac{E_{0b}}{E_{1b}} + \frac{e_{0b}e_{1b}}{E_{1b}} \tag{16.15.31}$$

or,

$$E_{1b} = E_{0b} + e_{0b}e_{1b} \tag{16.15.32}$$

Similarly, we have $\bar{\mathbf{k}}_1 = \bar{\mu}\,\bar{\mathbf{k}}_0$, and

$$e_{1b} = \bar{\mu}e_{0b} \tag{16.15.33}$$

and the equivalencies

$$\nu = \bar{\nu} + \frac{e_{0b}^2}{E_{0b}} \quad\Leftrightarrow\quad \mu = \bar{\mu} - \frac{e_{1b}^2}{E_{1b}} \quad\Leftrightarrow\quad \mu = \bar{\mu}\frac{E_{0b}}{E_{1b}} \tag{16.15.34}$$

Finally, the update equations of $\mathbf{b}_1$ are

$$\mathbf{b}_1 = \mathbf{b}_0 - e_{0b}\begin{bmatrix} \bar{\mathbf{k}}_1 \\ 0 \end{bmatrix} = \mathbf{b}_0 - e_{1b}\begin{bmatrix} \bar{\mathbf{k}}_0 \\ 0 \end{bmatrix} \tag{16.15.35}$$

Writing Eq. (16.15.31) in the form $E_{1b}/E_{0b} = 1/(1 - e_{0b}k_{1b})$, and solving Eq. (16.15.30) for $\mathbf{b}_1$, we have the alternative expression

$$\mathbf{b}_1 = \frac{E_{1b}}{E_{0b}}(\mathbf{b}_0 - e_{0b}\mathbf{k}_1) = \frac{\mathbf{b}_0 - e_{0b}\mathbf{k}_1}{1 - e_{0b}k_{1b}} \tag{16.15.36}$$

This is used in the so-called *fast Kalman* (FK) [1422,1423] computational sequence, which we summarize below

1. $e_{0a} = \mathbf{a}_0^T\mathbf{y}$
2. $\mathbf{a}_1 = \mathbf{a}_0 - e_{0a}\begin{bmatrix}0\\ \tilde{\mathbf{k}}_1\end{bmatrix}$
3. $e_{1a} = \mathbf{a}_1^T\mathbf{y}$
4. $E_{1a} = E_{0a} + e_{0a}e_{1a}$
5. Compute the first element of $\mathbf{k}_1$, $k_{1a} = \dfrac{e_{1a}}{E_{1a}}$
6. $\mathbf{k}_1 = \begin{bmatrix}0\\ \tilde{\mathbf{k}}_1\end{bmatrix} + k_{1a}\mathbf{a}_1$, and extract the last element of $\mathbf{k}_1$, k_{1b}
7. $e_{0b} = \mathbf{b}_0^T\mathbf{y}$
8. $\mathbf{b}_1 = \dfrac{\mathbf{b}_0 - e_{0b}\mathbf{k}_1}{1 - e_{0b}k_{1b}}$
9. $\begin{bmatrix}\bar{\mathbf{k}}_1\\ 0\end{bmatrix} = \mathbf{k}_1 - k_{1b}\mathbf{b}_1$
10. $\hat{x}_0 = \mathbf{h}_0^T\mathbf{y}\,,\quad e_0 = x - \hat{x}_0\,,\quad \mathbf{h}_1 = \mathbf{h}_0 + e_0\mathbf{k}_1\,,\quad \hat{x}_1 = \mathbf{h}_1^T\mathbf{y}\,,\quad e_1 = x - \hat{x}_1$

Step 9 is obtained from Eq. (16.15.18). Steps 1–9 perform the calculation and update of the Kalman gain vector $\mathbf{k}_1$, which is used in step 10 for the Wiener filtering part. This algorithm avoids the updating of the inverse autocorrelation matrices P_0 and P_1. The computationally intensive parts of the algorithm are the computation of the inner products and the vector updates. Steps 1, 2, 3, 6, 7, and 9 require M operations each, and step 8 requires $2M$ operations. Thus, the gain calculation in steps 1–9 requires a total of $8M$ operations. The Wiener filtering and updating part in step 10 require an additional $3M$ operations. Thus, the overall complexity grows like $8M + 3M = 11M$ operations; that is, linearly in the order M.

Several of the above operations can be avoided. In particular, the computation of the error e_{1a} in step 3 can be done by Eq. (16.15.27), thus, avoiding the inner product. Similarly, the inner product in step 7 can be avoided by solving Eq. (16.15.19) for e_{0b}, that is, $e_{0b} = k_{0b}E_{0b}$. Also, the division by the overall scalar factor $1/(1-e_{0b}k_{1b})$ in step 8 can be avoided by using Eq. (16.15.35) instead. This saves $3M$ out of the $8M$ computations—a 40% reduction. Similarly, the operation $\hat{x}_1 = \mathbf{h}_1^T\mathbf{y}$ in the Wiener filtering part can be avoided by $e_1 = \mu e_0$ and $\hat{x}_1 = x - e_1$. The resulting computational sequence is the so-called *fast a posteriori error sequential technique* (FAEST) [1424]. It uses the a posteriori errors and the a priori Kalman gains, and is summarized below

1. $e_{0a} = \mathbf{a}_0^T\mathbf{y}$
2. $e_{1a} = \tilde{\mu}e_{0a} = e_{0a}/(1+\tilde{\nu})$
3. Compute the first element of $\mathbf{k}_0$, $k_{0a} = \dfrac{e_{0a}}{E_{0a}}$
4. $E_{1a} = E_{0a} + e_{0a}e_{1a}$
5. $\mathbf{k}_0 = \begin{bmatrix}0\\ \tilde{\mathbf{k}}_0\end{bmatrix} + k_{0a}\mathbf{a}_0$, and extract the last element of $\mathbf{k}_0$, k_{0b}
6. $e_{0b} = k_{0b}E_{0b}$
7. $\begin{bmatrix}\bar{\mathbf{k}}_0\\ 0\end{bmatrix} = \mathbf{k}_0 - k_{0b}\mathbf{b}_0$
8. $\nu = \tilde{\nu} + e_{0a}k_{0a}\,,\quad \bar{\nu} = \nu - e_{0b}k_{0b}$
9. $e_{1b} = \bar{\mu}e_{0b} = e_{0b}/(1+\bar{\nu})$
10. $E_{1b} = E_{0b} + e_{0b}e_{1b}$

11. $\mathbf{a}_1 = \mathbf{a}_0 - e_{1a}\begin{bmatrix}0\\ \tilde{\mathbf{k}}_0\end{bmatrix}$
12. $\mathbf{b}_1 = \mathbf{b}_0 - e_{1b}\begin{bmatrix}\bar{\mathbf{k}}_0\\ 0\end{bmatrix}$
13. $\hat{x}_0 = \mathbf{h}_0^T\mathbf{y}\,,\quad e_0 = x - \hat{x}_0\,,\quad e_1 = \mu e_0 = e_0/(1+\nu)\,,\quad \hat{x}_1 = x - e_1$
14. $\mathbf{h}_1 = \mathbf{h}_0 + e_1\mathbf{k}_0$

Step 8 was obtained from Eq. (16.15.21). Steps l, 5, 7, 11, and 12 require M operations each. Therefore, the gain calculation can be done with $5M$ operations. The last two Wiener filtering steps require an additional $2M$ operations. Thus, the total operation count grows like $5M + 2M = 7M$. The so-called *fast transversal filter* (FTF) [1425] computational sequence is essentially identical to FAEST, but works directly with the variables μ instead of ν. The only change is to replace step 8 by the following:

8. $\mu = \tilde{\mu}\dfrac{E_{0a}}{E_{1a}}\,,\quad \bar{\mu} = \dfrac{\mu}{1 - e_{0b}k_{0b}\mu}$ (FTF)

The second equation is obtained from (16.15.34), (16.15.31), and the proportionality $\mathbf{k}_1 = \mu\,\mathbf{k}_0$, which implies the same for the last elements of these vectors, $k_{1b} = \mu k_{0b}$. We have

$$\bar{\mu} = \mu\,\frac{E_{1b}}{E_{0b}} = \frac{\mu}{1 - e_{0b}k_{1b}} = \frac{\mu}{1 - e_{0b}k_{0b}\mu}$$

The above computational sequences are organized to start with the tilde quantities, such as $\tilde{\nu}$ and $\tilde{\mathbf{k}}_0$, and end up with the bar quantities such as $\bar{\nu}$ and $\bar{\mathbf{k}}_0$. The reason has to do with the shift-invariance property, which implies that all bar quantities computed at the present iteration become the corresponding tilde quantities of the *next* iteration; for example,

$$\tilde{\nu}(n+1) = \bar{\nu}(n)\,,\quad \tilde{\mathbf{k}}_0(n+1) = \bar{\mathbf{k}}_0(n)$$

This property allows the repetition of the computational cycle from one time instant to the next. As we have seen, the computational savings of FAEST over FK, and FK over conventional RLS, have nothing to do with shift invariance but rather are consequences of the rank-one updating properties.

The FAEST, FTF, and FK algorithms are the fastest known RLS algorithms. Unfortunately, they can exhibit numerically unstable behavior and require the use of rescue devices and re-initializations for continuous operation [1426–1435]. Next, we consider the lattice formulations. Equations (1.8.50) can be applied to the *a priori lattice*

$$\begin{aligned} e_{0a} &= \bar{e}_{0a} - \gamma_{0b}\tilde{e}_{0b} \\ e_{0b} &= \tilde{e}_{0b} - \gamma_{0a}\bar{e}_{0a} \end{aligned} \tag{16.15.37}$$

and *a posteriori lattice*

$$\begin{aligned} e_{1a} &= \bar{e}_{1a} - \gamma_{1b}\tilde{e}_{1b} \\ e_{1b} &= \tilde{e}_{1b} - \gamma_{1a}\bar{e}_{1a} \end{aligned} \tag{16.15.38}$$

with the reflection coefficients computed by

$$\gamma_{0a} = \frac{\Delta_0}{\bar{E}_{0a}}\,,\quad \gamma_{0b} = \frac{\Delta_0}{\tilde{E}_{0b}}\quad\text{and}\quad \gamma_{1a} = \frac{\Delta_1}{\bar{E}_{1a}}\,,\quad \gamma_{1b} = \frac{\Delta_1}{\tilde{E}_{1b}} \tag{16.15.39}$$

To find the relationship between Δ_1 and Δ_0, we use Eq. (1.8.44) applied to R_1

$$R_1\begin{bmatrix}0\\ \tilde{\mathbf{b}}_1\end{bmatrix} = \Delta_1\mathbf{u} + \tilde{E}_{1b}\mathbf{v}\,,\quad R_1\begin{bmatrix}\bar{\mathbf{a}}_1\\ 0\end{bmatrix} = \Delta_1\mathbf{v} + \bar{E}_{1a}\mathbf{u} \tag{16.15.40}$$

Applying Eq. (1.8.44) also to R_0, we obtain

$$R_1\begin{bmatrix}\bar{\mathbf{a}}_0\\0\end{bmatrix} = \big(R_0+\mathbf{y}\mathbf{y}^T\big)\begin{bmatrix}\bar{\mathbf{a}}_0\\0\end{bmatrix} = \Delta_0\mathbf{v}+\bar{E}_{0a}\mathbf{u}+\bar{e}_{0a}\mathbf{y} \tag{16.15.41}$$

and

$$R_1\begin{bmatrix}0\\\tilde{\mathbf{b}}_0\end{bmatrix} = \big(R_0+\mathbf{y}\mathbf{y}^T\big)\begin{bmatrix}0\\\tilde{\mathbf{b}}_0\end{bmatrix} = \Delta_0\mathbf{u}+\tilde{E}_{0b}\mathbf{v}+\tilde{e}_{0b}\mathbf{y} \tag{16.15.42}$$

Forming the dot products,

$$[0,\,\tilde{\mathbf{b}}_1^T]R_1\begin{bmatrix}\bar{\mathbf{a}}_0\\0\end{bmatrix} \quad\text{and}\quad [0,\,\tilde{\mathbf{b}}_0^T]R_1\begin{bmatrix}\bar{\mathbf{a}}_1\\0\end{bmatrix}$$

we obtain the two alternative expressions

$$\Delta_1 = \Delta_0+\bar{e}_{0a}\tilde{e}_{1b}\,,\quad \Delta_1 = \Delta_0+\bar{e}_{1a}\tilde{e}_{0b} \tag{16.15.43}$$

They represent the least-squares modifications of the partial correlation (1.8.53). The two expressions are equivalent. Applying Eq. (16.15.33) to $\tilde{e}_{1b}$, we have $\tilde{e}_{1b} = \bar{\tilde{\mu}}\tilde{e}_{0b}$. Applying Eq. (16.15.27) to $\bar{e}_{1a}$, we have $\bar{e}_{1a} = \tilde{\bar{\mu}}\bar{e}_{0a}$. But, $\bar{\tilde{\nu}} = \tilde{\bar{\nu}}$ because, as is evident from Eq. (1.8.51), the tilde part of $\bar{\mathbf{y}}$ is the same as the bar part of $\tilde{\mathbf{y}}$, namely, $\mathbf{y}_c$. Thus, $\bar{\tilde{\nu}} = \tilde{\bar{\nu}} = \mathbf{y}_c^T R_{0c}^{-1}\mathbf{y}_c$, which implies $\bar{\tilde{\mu}} = \tilde{\bar{\mu}}$. Applying Eq. (16.15.34), we have the updating equation $\tilde{\mu} = \bar{\tilde{\mu}} - \tilde{e}_{1b}^2/\tilde{E}_{1b}$.

As for the Wiener filtering part, we can apply the order-updating equations (1.8.24) through (1.8.27) to the a priori and a posteriori problems to get

$$\begin{aligned}\hat{x}_0 &= \bar{x}_0+g_{0b}e_{0b}\,,\quad e_0 = \bar{e}_0-g_{0b}e_{0b}\\ \hat{x}_1 &= \bar{x}_1+g_{1b}e_{1b}\,,\quad e_1 = \bar{e}_1-g_{1b}e_{1b}\end{aligned} \tag{16.15.44}$$

where g_{0b} and g_{1b} are the *last* components of the lattice weight vectors $\mathbf{g}_0$ and $\mathbf{g}_1$. Because of the relationship $\mathbf{h} = L^T\mathbf{g}$, it follows that the last component of $\mathbf{h}$ is equal to the last component of $\mathbf{g}$. Thus, extracting the last components of the relationship $\mathbf{h}_1 = \mathbf{h}_0+e_0\mathbf{k}_1$, we find

$$g_{1b} = g_{0b}+e_0k_{1b} = g_{0b}+e_0\,\frac{e_{1b}}{E_{1b}} \tag{16.15.45}$$

This provides a *direct* way to update the gs. The more conventional updating method is indirect; it is obtained by writing

$$g_{0b} = \frac{\rho_{0b}}{E_{0b}}\,,\quad g_{1b} = \frac{\rho_{1b}}{E_{1b}} \tag{16.15.46}$$

Using Eq. (16.15.44), we can find a recursion for the ρs as follows

$$\rho_{1b} = E_{1b}g_{1b} = E_{1b}g_{0b}+(\bar{e}_0-g_{0b}e_{0b})e_{1b} = (E_{1b}-e_{0b}e_{1b})g_{0b}+\bar{e}_0e_{1b}$$

or, using $E_{1b}-e_{0b}e_{1b} = E_{0b}$ and $\rho_{0b} = E_{0b}g_{0b}$, we obtain

$$\rho_{1b} = \rho_{0b}+\bar{e}_0e_{1b} = \rho_{0b}+\frac{1}{\bar{\mu}}\,\bar{e}_1e_{1b} \tag{16.15.47}$$

The *conventional RLS lattice* (RLSL) computational sequence is summarized below [1436–1444]:

1. $\Delta_1 = \Delta_0+\tilde{e}_{1b}\bar{e}_{0a} = \Delta_0+\tilde{e}_{1b}\bar{e}_{1a}/\tilde{\bar{\mu}}$
2. $\gamma_{1a} = \dfrac{\Delta_1}{\bar{E}_{1a}}\,,\quad \gamma_{1b} = \dfrac{\Delta_1}{\tilde{E}_{1b}}$

3. $e_{1a} = \bar{e}_{1a}-\gamma_{1b}\tilde{e}_{1b}\,,\quad e_{1b} = \tilde{e}_{1b}-\gamma_{1a}\bar{e}_{1a}$
4. $E_{1a} = \bar{E}_{1a}-\gamma_{1b}\Delta_1\,,\quad E_{1b} = \tilde{E}_{1b}-\gamma_{1a}\Delta_1$
5. $\tilde{\mu} = \bar{\tilde{\mu}}-\dfrac{\tilde{e}_{1b}^2}{\tilde{E}_{1b}}$
6. $\rho_{1b} = \rho_{0b}+\bar{e}_1e_{1b}/\bar{\mu}$
7. $g_{1b} = \dfrac{\rho_{1b}}{E_{1b}}$
8. $e_1 = \bar{e}_1-g_{1b}e_{1b}\,,\quad \hat{x}_1 = x-e_1$

This is referred to as the *a posteriori* RLS lattice because it uses the a posteriori lattice equations (16.15.38). There are 14 multiplication/division operations in this sequence. We will see later that the use of the so-called forgetting factor λ requires 2 more multiplications. Thus, the total number of operations is 16. Because this sequence must be performed once per order, it follows that, for an order-M problem, the computational complexity of the RLS lattice will be $16M$ operations per time update. This is to be compared with $7M$ for the FAEST direct-form version. However, as we have already mentioned, the direct-form versions can exhibit numerical instabilities. By contrast, the lattice algorithms are numerically stable [1431,1445].

Many other variations of the RLS lattice are possible. For example, there is a version based on Eq. (16.15.37), called the *a priori* RLS lattice algorithm [1358,1444], or a version called the *double* (a priori/a posteriori) RLS algorithm [1441,1444] that uses Eqs. (16.15.37) and (16.15.38) simultaneously. This version avoids the computation of the likelihood parameter μ. Like Eq. (16.15.45), we can also obtain direct updating formulas for the reflection coefficients, thereby avoiding the recursion (16.15.43) for the partial correlations Δ. Using the second term of Eqs. (16.15.43) and (16.15.25) applied to $\bar{E}_{1a}$, that is, $\bar{E}_{1a}+\bar{E}_{0a}+\bar{e}_{0a}\bar{e}_{1a}$. we find

$$\begin{aligned}\gamma_{1a} &= \frac{\Delta_1}{\bar{E}_{1a}} = \frac{\Delta_0+\bar{e}_{1a}\tilde{e}_{0b}}{\bar{E}_{1a}} = \frac{\gamma_{0a}\bar{E}_{0a}+\bar{e}_{1a}\tilde{e}_{0b}}{\bar{E}_{1a}}\\ &= \frac{\gamma_{0a}(\bar{E}_{1a}-\bar{e}_{0a}\bar{e}_{1a})+\bar{e}_{1a}\tilde{e}_{0b}}{\bar{E}_{1a}} = \gamma_{0a}+\frac{\bar{e}_{1a}}{\bar{E}_{1a}}(\tilde{e}_{0b}-\gamma_{0a}\bar{e}_{0a})\end{aligned}$$

and using Eq. (16.15.37), we obtain

$$\gamma_{1a} = \gamma_{0a}+e_{0b}\,\frac{\bar{e}_{1a}}{\bar{E}_{1a}} \tag{16.15.48}$$

Similarly, working with the first term of Eq. (16.15.43), we find

$$\gamma_{1b} = \gamma_{0b}+e_{0a}\,\frac{\tilde{e}_{1b}}{\tilde{E}_{1b}} \tag{16.15.49}$$

Replacing $\bar{e}_{1a} = \tilde{\bar{\mu}}\bar{e}_{0a}$ and $\tilde{e}_{1b} = \bar{\tilde{\mu}}\tilde{e}_{0b}$ in the above equations gives rise to the so-called *a priori direct-updating* RLS lattice [1445], also called the *a priori error-feedback* lattice because the outputs e_{0a} and e_{0b} of the a priori lattice equations (16.15.37) are used to update the reflection coefficients.

An *a posteriori* direct or error-feedback algorithm [1445] can also be obtained by working with the a posteriori lattice Eq. (16.15.38). In this case, we must express e_{0a} and e_{0b} in terms of the a posteriori quantities as follows:

$$e_{0a} = \bar{e}_{0a}-\gamma_{0b}\tilde{e}_{0b} = (\bar{e}_{1a}-\gamma_{0b}\tilde{e}_{1b})/\tilde{\bar{\mu}} \quad\text{and}\quad e_{0b} = (\tilde{e}_{1b}-\gamma_{0a}\bar{e}_{1a})/\tilde{\bar{\mu}}$$

The a priori and a posteriori error-feedback lattice algorithms are computationally somewhat more expensive—requiring $O(20M)$ operations—than the conventional RLS lattice. But, they have much better *numerical accuracy* under quantization [1445] and, of course, their long-term behavior is numerically stable.

Below we list the computational sequence of what may be called the *double/direct* RLS lattice algorithm that, on the one hand, uses direct-updating for increased numerical accuracy, and on the other, has the same computational complexity as the conventional a posteriori RLS lattice, namely, $16M$ operations [1487]:

1. $e_{0a} = \bar{e}_{0a} - \gamma_{0b}\bar{e}_{0b}\,, \quad e_{0b} = \bar{e}_{0b} - \gamma_{0a}\bar{e}_{0a}$
2. $\gamma_{1a} = \gamma_{0a} + e_{0b}\,\dfrac{\bar{e}_{1a}}{\bar{E}_{1a}}\,, \quad \gamma_{1b} = \gamma_{0b} + e_{0a}\,\dfrac{\bar{e}_{1b}}{\bar{E}_{1b}}$
3. $e_{1a} = \bar{e}_{1a} - \gamma_{1b}\bar{e}_{1b}\,, \quad e_{1b} = \bar{e}_{1b} - \gamma_{1a}\bar{e}_{1a}$
4. $E_{1a} = E_{0a} + e_{1a}e_{0a}\,, \quad E_{1b} = E_{0b} + e_{1b}e_{0b}$
5. $e_0 = \bar{e}_0 - g_{0b}e_{eb}$
6. $g_{1b} = g_{0b} + e_0\,\dfrac{e_{1b}}{E_{1b}}$
7. $e_1 = \bar{e}_1 - g_{1b}e_{1b}\,, \quad \hat{x}_1 = x - e_1$

It uses simultaneously the a priori and a posteriori lattice equations (16.15.37) and (16.15.38). There are 14 operations (plus 2 for the forgetting factor) per order per time update, that is, a total of $16M$ per time update.

Finally, we discuss the sense in which the a priori and a posteriori backward errors e_{0b} and e_{1b} provide a decorrelation of the covariance matrices R_0 and R_1. Following Eqs. (1.8.13) and (1.8.17), we write the LU factorizations of the a priori and a posteriori problems

$$L_0R_0L_0^T = D_{0b}\,, \quad L_1R_1L_1^T = D_{1b} \tag{16.15.50}$$

where L_0 and L_1 have as rows the backward predictors $\mathbf{b}_0^T = [\boldsymbol{\beta}_0^T, 1]$ and $\mathbf{b}_1^T = [\boldsymbol{\beta}_1^T, 1]$.

$$L_0 = \begin{bmatrix} \bar{L}_0 & \mathbf{0} \\ \boldsymbol{\beta}_0^T & 1 \end{bmatrix}, \quad L_1 = \begin{bmatrix} \bar{L}_1 & \mathbf{0} \\ \boldsymbol{\beta}_1^T & 1 \end{bmatrix} \tag{16.15.51}$$

The corresponding backward basis vectors are constructed by

$$\mathbf{e}_{0b} = L_0\mathbf{y} = \begin{bmatrix} \bar{L}_0 & \mathbf{0} \\ \boldsymbol{\beta}_0^T & 1 \end{bmatrix}\begin{bmatrix} \bar{\mathbf{y}} \\ y_b \end{bmatrix} = \begin{bmatrix} \bar{L}_0\bar{\mathbf{y}} \\ \mathbf{b}_0^T\mathbf{y} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{e}}_{0b} \\ e_{0b} \end{bmatrix} \tag{16.15.52}$$

and

$$\mathbf{e}_{1b} = L_1\mathbf{y} = \begin{bmatrix} \bar{L}_1 & \mathbf{0} \\ \boldsymbol{\beta}_1^T & 1 \end{bmatrix}\begin{bmatrix} \bar{\mathbf{y}} \\ y_b \end{bmatrix} = \begin{bmatrix} \bar{L}_1\bar{\mathbf{y}} \\ \mathbf{b}_1^T\mathbf{y} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{e}}_{1b} \\ e_{1b} \end{bmatrix} \tag{16.15.53}$$

The rank-one updating property (16.15.1) for the Rs can be translated into an updating equation for the LU factorizations[112–114], in the following form:

$$L_1 = LL_0 \tag{16.15.54}$$

It turns out that the unit lower triangular matrix L can be built entirely out of the a priori backward errors $\mathbf{e}_{0b}$, as we show below. The determining equation for L may be found by

$$D_{1b} = L_1R_1L_1^T = LL_0(R_0 + \mathbf{y}\mathbf{y}^T)L_0^TL^T = L(D_{0b} + \mathbf{e}_{0b}\mathbf{e}_{0b}^T)L^T \tag{16.15.55}$$

Thus, L performs the LU factorization of the rank-one update of a diagonal matrix, namely, $D_{0b} + \mathbf{e}_{0b}\mathbf{e}_{0b}^T$. The solution is easily found by introducing the block decompositions

$$L = \begin{bmatrix} \bar{L} & \mathbf{0} \\ \boldsymbol{\beta}^T & 1 \end{bmatrix}, \quad D_{1b} = \begin{bmatrix} \bar{D}_{1b} & \mathbf{0} \\ \mathbf{0}^T & E_{1b} \end{bmatrix}, \quad D_{0b} + \mathbf{e}_{0b}\mathbf{e}_{0b}^T = \begin{bmatrix} \bar{D}_{0b} + \bar{\mathbf{e}}_{0b}\bar{\mathbf{e}}_{0b}^T & e_{0b}\bar{\mathbf{e}}_{0b} \\ e_{0b}\bar{\mathbf{e}}_{0b}^T & E_{0b} + e_{0b}^2 \end{bmatrix}$$

Using the methods of Sec. 1.8, e.g., Eqs. (1.8.7) and (1.8.11) applied to this problem, we find the solution

$$\boldsymbol{\beta} = -\bar{\mu}e_{0b}\bar{D}_{0b}^{-1}\bar{\mathbf{e}}_{0b}\,, \quad \bar{\mu} = \frac{1}{1 + \bar{\mathbf{e}}_{0b}^T\bar{D}_{0b}^{-1}\bar{\mathbf{e}}_{0b}} \tag{16.15.56}$$

Using $\bar{R}_0^{-1} = \bar{L}_0^T\bar{D}_{0b}^{-1}\bar{L}_0$, we recognize

$$\bar{\mathbf{e}}_{0b}^T\bar{D}_{0b}^{-1}\bar{\mathbf{e}}_{0b} = \bar{\mathbf{y}}^T\bar{L}_0^T\bar{D}_{0b}^{-1}\bar{L}_0\bar{\mathbf{y}} = \bar{\mathbf{y}}^T\bar{R}_0^{-1}\bar{\mathbf{y}} = \bar{\nu}$$

Therefore, the quantity $\bar{\mu}$ defined above is the usual one. Similarly, we find

$$E_{1b} = (E_{0b} + e_{0b}^2) + e_{0b}\bar{\mathbf{e}}_{0b}^T\boldsymbol{\beta} = E_{0b} + e_{0b}^2 - \bar{\mu}e_{0b}^2\bar{\nu}$$

Noting that $1 - \bar{\mu}\bar{\nu} = \bar{\mu}$, this reduces to Eq. (16.15.32). Writing $\bar{D}_{0b}^{-1}\bar{\mathbf{e}}_{0b} = \bar{L}_0^{-T}\bar{R}_0^{-1}\bar{\mathbf{y}} = \bar{L}_0^{-T}\bar{\mathbf{k}}_0$, we may express $\boldsymbol{\beta}$ in terms of the Kalman gain vector:

$$\boldsymbol{\beta} = -\bar{\mu}e_{0b}\bar{L}_0^{-T}\bar{\mathbf{k}}_0 \tag{16.15.57}$$

It easy to verify that the block-decomposed form of Eq. (16.15.54) is equivalent to

$$\bar{L}_1 = \bar{L}\bar{L}_0\,, \quad \boldsymbol{\beta}_1 = \boldsymbol{\beta}_0 + \bar{L}_0^T\boldsymbol{\beta} \tag{16.15.58}$$

Because of Eq. (16.15.57), the updating equation for the $\boldsymbol{\beta}$s is equivalent to Eq. (16.15.35). Using this formalism, we may show the proportionality between the a posteriori and a priori backward errors. We have $\mathbf{e}_{1b} = L_1\mathbf{y} = LL_0\mathbf{y} = L\mathbf{e}_{0b}$, and in block form

$$\mathbf{e}_{1b} = \begin{bmatrix} \bar{L} & \mathbf{0} \\ \boldsymbol{\beta}^T & 1 \end{bmatrix}\begin{bmatrix} \bar{\mathbf{e}}_{0b} \\ e_{0b} \end{bmatrix} = \begin{bmatrix} \bar{L}\mathbf{e}_{0b} \\ e_{0b} + \boldsymbol{\beta}^T\bar{\mathbf{e}}_{0b} \end{bmatrix}$$

Therefore, $e_{1b} = e_{0b} + \boldsymbol{\beta}^T\bar{\mathbf{e}}_{0b} = e_{0b} - \bar{\mu}e_{0b}\bar{\nu} = \bar{\mu}e_{0b}$. It follows that L acting on $\mathbf{e}_{0b}$ can be replaced by the diagonal matrix of $\bar{\mu}$s acting on $\mathbf{e}_{0b}$. The double/direct lattice algorithm effectively provides the error signals required to build L. For example, Eq. (16.15.56) can be written in a form that avoids the computation of the μs

$$\boldsymbol{\beta} = -\bar{\mu}e_{0b}\bar{D}_{0b}^{-1}\bar{\mathbf{e}}_{0b} = -e_{1b}\bar{D}_{0b}^{-1}\bar{\mathbf{e}}_{0b} \tag{16.15.59}$$

The a priori and a posteriori estimates $\hat{x}_0$ and $\hat{x}_1$ may also be expressed in the backward bases. Defining $\mathbf{g}_0 = L_0^{-T}\mathbf{h}_0$, we find $\hat{x}_0 = \mathbf{h}_0^T\mathbf{y} = \mathbf{g}_0^TL_0\mathbf{y} = \mathbf{g}_0^T\mathbf{e}_{0b}$, and similarly, defining $\mathbf{g}_1 = L_1^{-T}\mathbf{h}_1$, we find $\hat{x}_1 = \mathbf{g}_1^T\mathbf{e}_{1b}$. Thus,

$$\mathbf{g}_1 = L_1^{-T}\mathbf{h}_1\,, \quad \mathbf{g}_0 = L_0^{-T}\mathbf{h}_0 \tag{16.15.60}$$

and

$$\hat{x}_1 = \mathbf{g}_1^T\mathbf{e}_{1b}\,, \quad \hat{x}_0 = \mathbf{g}_0^T\mathbf{e}_{0b} \tag{16.15.61}$$

Finally, the updating equation (16.15.16) for the direct-form weights translates into an updating equation for the lattice weights:

$$\mathbf{g}_1 = L_1^{-T}\mathbf{h}_1 = L_1^{-T}(\mathbf{h}_0 + e_0\mathbf{k}_1) = L^{-T}L_0^{-T}\mathbf{h}_0 + e_0L_1^{-T}\mathbf{k}_1$$

where we used the factorization (16.15.54) for the first term. Using $R_1^{-1} = L_1^TD_{1b}^{-1}L_1$, we find for the second term $L_1^{-T}\mathbf{k}_1 = L_1^{-T}R_1^{-1}\mathbf{y} = D_{1b}^{-1}L_1\mathbf{y} = D_{1b}^{-1}\mathbf{e}_{1b}$. Therefore,

$$\mathbf{g}_1 = L^{-T}\mathbf{g}_0 + e_0D_{1b}^{-1}\mathbf{e}_{1b} \tag{16.15.62}$$

Extracting the last elements we obtain Eq. (16.15.45).

16.16 RLS Adaptive Filters

The LMS and gradient lattice adaptation algorithms, based on the steepest descent method, provide a gradual, iterative, minimization of the performance index. The adaptive weights are not optimal at each time instant, but only after convergence. In this section, we discuss *recursive least-squares* (RLS) adaptation algorithms that are based on the *exact minimization* of least-squares criteria. The filter weights are optimal at each time instant n.

Adaptive RLS algorithms are the time-recursive analogs of the block processing methods of linear prediction and FIR Wiener filtering that we discussed in Sections 12.12 and 12.14. They may be used, in place of LMS, in any adaptive filtering application. Because of their fast convergence they have been proposed for use in fast start-up channel equalizers [1448–1451]. They are also routinely used in real-time system identification applications. Their main disadvantage is that they require a fair amount of computation, $O(M^2)$ operations per time update. In biomedical applications, they can be easily implemented on minicomputers. In other applications, such as the equalization of rapidly varying channels or adaptive arrays [1355,1453–1455], they may be too costly for implementation.

The fast reformulations of RLS algorithms, such as the RLSL, FK, FAEST, and FTF, have $O(M)$ computational complexity. The fast RLS algorithms combine the best of the LMS and RLS, namely, the computational efficiency of the former and the fast convergence of the latter. Among the fast RLS algorithms, the RLS lattice has better numerical stability properties than the direct-form versions.

We start with the RLS formulation of the FIR Wiener filtering problem. The estimation criterion, $\mathcal{E} = E[e(n)^2] = \min$, is replaced with a least-squares weighted time-average that includes all estimation errors from the initial time instant to the current time n, that is, $e(k)$, $k = 0, 1, \dots, n$:

$$\mathcal{E}_n = \sum_{k=0}^{n} e^2(k) = \min \tag{16.16.1}$$

where

$$e(k) = x(k) = \hat{x}(k)$$

and $\hat{x}(k)$ is the estimate of $x(k)$ produced by the order-M Wiener filter

$$\hat{x}(k) = \sum_{m=0}^{M} h_m y_{k-m} = [h_0, h_1, \dots, h_M]\begin{bmatrix} y_k \\ y_{k-1} \\ \vdots \\ y_{k-M}\end{bmatrix} = \mathbf{h}^T\mathbf{y}(k)$$

Note that in adaptive array problems, $\mathbf{y}(k)$ represents the vector of measurements at the array elements, namely, $\mathbf{y}(k) = [y_0(k), y_1(k), \dots, y_M(k)]$. To better track possible non-stationarities in the signals, the performance index may be modified by introducing exponential weighting

$$\mathcal{E}_n = \sum_{k=0}^{n} \lambda^{n-k} e^2(k) = e^2(n) + \lambda e^2(n-1) + \lambda^2 e^2(n-2) + \cdots + \lambda^n e^2(0) \tag{16.16.2}$$

where the *forgetting factor* λ is positive and less than one. This performance index emphasizes the most recent observations and exponentially ignores the older ones. We will base our discussion on this criterion. Setting the derivative with respect to $\mathbf{h}$ to zero, we find the least-square analogs of the *orthogonality equations*

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{h}} = -2\sum_{k=0}^{n} \lambda^{n-k} e(k)\mathbf{y}(k) = 0$$

which may be cast in a *normal equation* form

$$\sum_{k=0}^{n} \lambda^{n-k}\big[x(k) - \mathbf{h}^T\mathbf{y}(k)\big]\mathbf{y}(k) = 0\,, \quad \text{or,}$$

$$\left[\sum_{k=0}^{n} \lambda^{n-k}\mathbf{y}(k)\mathbf{y}(k)^T\right]\mathbf{h} = \sum_{k=0}^{n} \lambda^{n-k} x(k)\mathbf{y}(k)$$

Defining the quantities

$$\begin{aligned} R(n) &= \sum_{k=0}^{n} \lambda^{n-k}\mathbf{y}(k)\mathbf{y}(k)^T \\ \mathbf{r}(n) &= \sum_{k=0}^{n} \lambda^{n-k} x(k)\mathbf{y}(k) \end{aligned} \tag{16.16.3}$$

we write the normal equations as $R(n)\mathbf{h} = \mathbf{r}(n)$, with solution $\mathbf{h} = R(n)^{-1}\mathbf{r}(n)$. Note that the n-dependence of $R(n)$ and $\mathbf{r}(n)$ makes $\mathbf{h}$ depend on n; we shall write, therefore,

$$\mathbf{h}(n) = R(n)^{-1}\mathbf{r}(n) \tag{16.16.4}$$

These are the least-squares analogs of the ordinary Wiener solution, with $R(n)$ and $\mathbf{r}(n)$ playing the role of the covariance matrix $R = E[\mathbf{y}(n)\mathbf{y}^T(n)]$ and cross-correlation vector $\mathbf{r} = E[x(n)\mathbf{y}(n)]$. These quantities satisfy the rank-one updating properties

$$R(n) = \lambda R(n-1) + \mathbf{y}(n)\mathbf{y}(n)^T \tag{16.16.5}$$

$$\mathbf{r}(n) = \lambda\mathbf{r}(n-1) + x(n)\mathbf{y}(n) \tag{16.16.6}$$

Thus, the general results of the previous section can be applied. We have the correspondences:

$\mathbf{y}$	$\to$	$\mathbf{y}(n)$	x	$\to$	$x(n)$
R_1	$\to$	$R(n)$	R_0	$\to$	$\lambda R(n-1)$
P_1	$\to$	$P(n) = R(n)^{-1}$	P_0	$\to$	$\lambda^{-1}P(n-1) = \lambda^{-1}R(n-1)^{-1}$
$\mathbf{r}_1$	$\to$	$\mathbf{r}(n)$	$\mathbf{r}_0$	$\to$	$\lambda\mathbf{r}(n-1)$
$\mathbf{h}_1$	$\to$	$\mathbf{h}(n) = R(n)^{-1}\mathbf{r}(n)$	$\mathbf{h}_0$	$\to$	$\mathbf{h}(n-1) = R(n-1)^{-1}\mathbf{r}(n)$
$\hat{x}_1$	$\to$	$\hat{x}(n) = \mathbf{h}(n)^T\mathbf{y}(n)$	$\hat{x}_0$	$\to$	$\hat{x}(n/n-1) = \mathbf{h}(n-1)^T\mathbf{y}(n)$
e_1	$\to$	$e(n) = x(n) - \hat{x}(n)$	e_0	$\to$	$e(n/n-1) = x(n) - \hat{x}(n/n-1)$
$\mathbf{k}_1$	$\to$	$\mathbf{k}(n) = R(n)^{-1}\mathbf{y}(n)$	$\mathbf{k}_0$	$\to$	$\mathbf{k}(n/n-1) = \lambda^{-1}R(n-1)^{-1}\mathbf{y}(n)$
ν	$\to$	$\nu(n) = \mathbf{k}(n/n-1)^T\mathbf{y}(n)$	μ	$\to$	$\mu(n) = 1/\big(1+\nu(n)\big)$

We used the notation $\hat{x}(n/n-1)$, $e(n/n-1)$, and $\mathbf{k}(n/n-1)$ to denote the *a priori* estimate, estimation error, and Kalman gain. Note that $R_0, \mathbf{r}_0$ are the quantities $R(n-1)$, $\mathbf{r}(n-1)$ scaled by the forgetting factor λ. In the a priori solution $\mathbf{h}_0 = R_0^{-1}\mathbf{r}_0$, the factors λ cancel to give $\big[\lambda R(n-1)\big]^{-1}\big[\lambda\mathbf{r}(n-1)\big] = R(n-1)^{-1}\mathbf{r}(n-1) = \mathbf{h}(n-1)$. Thus, the a priori Wiener solution is the solution at the *previous* time instant $n-1$. With the above correspondences, the conventional RLS algorithm listed in the previous section becomes

1. $\mathbf{k}(n/n-1) = \lambda^{-1}P(n-1)\mathbf{y}(n)$

2. $\nu(n) = \mathbf{k}(n/n-1)^T\mathbf{y}(n)\,, \quad \mu(n) = \dfrac{1}{1+\nu(n)}$

3. $\mathbf{k}(n)=\mu(n)\mathbf{k}(n/n-1)$
4. $P(n)=\lambda^{-1}P(n-1)-\mathbf{k}(n)\mathbf{k}(n/n-1)^T$
5. $\hat{x}(n/n-1)=\mathbf{h}(n-1)^T\mathbf{y}(n)\,,\quad e(n/n-1)=x(n)-\hat{x}(n/n-1)$
6. $e(n)=\mu(n)e(n/n-1)\,,\quad \hat{x}(n)=x(n)-e(n)$
7. $\mathbf{h}(n)=\mathbf{h}(n-1)+e(n/n-1)\mathbf{k}(n)$

The algorithm may be initialized in time by taking $R(-1)=0$, which would imply $P(-1)=\infty$. Instead, we may use $P(-1)=\delta^{-1}I$, where δ is a very small number, and I the identity matrix. The algorithm is quite insensitive to the choice of δ. Typical values are $\delta=0.1$, or $\delta=0.01$.

The function **rls** is an implementation of the algorithm. Because the algorithm can also be used in array problems, we have designed the function so that its inputs are the old weights $\mathbf{h}(n-1)$, the current sample $x(n)$, and the entire data vector $\mathbf{y}(n)$ (in time series problems only the current time sample y_n is needed, the past samples y_{n-i}, $i=1,2,\dots,M$ being stored in the tapped delay line). The outputs of the function are $\mathbf{h}(n)$, $\hat{x}(n)$, and $e(n)$. A simulation example will be presented in the next section.

The term *Kalman gain* arises by interpreting $\mathbf{h}(n)=\mathbf{h}(n-1)+e(n/n-1)\mathbf{k}(n)$ as a Kalman predictor/corrector algorithm, where the first term $\mathbf{h}(n-1)$ is a prediction of the weight $\mathbf{h}(n)$ based on the past, $e(n/n-1)=x(n)-\mathbf{h}(n-1)^T\mathbf{y}(n)$ is the tentative estimation error made on the basis of the prediction $\mathbf{h}(n-1)$, and the second term $e(n/n-1)\mathbf{k}(n)$ is the correction of the prediction. The fast convergence properties of the algorithm can be understood by making the replacement $\mathbf{k}(n)=R(n)^{-1}\mathbf{y}(n)$ in the update equation

$$\mathbf{h}(n)=\mathbf{h}(n-1)+R(n)^{-1}\mathbf{y}(n)e(n/n-1) \tag{16.16.7}$$

It differs from the LMS algorithm by the presence of $R(n)^{-1}$ in the weight update term. Because $R(n)$ is an estimate of the covariance matrix $R=E[\mathbf{y}(n)\mathbf{y}(n)^T]$, the presence of $R(n)^{-1}$ makes the RLS algorithm behave like Newton's method, hence its fast convergence properties [1456,1457]. Another important conceptual difference with the LMS algorithm is that in the RLS algorithm the filters $\mathbf{h}(n)$ and $\mathbf{h}(n-1)$ are the *exact* Wiener solutions of two different minimization criteria; namely, $\mathcal{E}_n=\min$ and $\mathcal{E}_{n-1}=\min$, whereas in the LMS algorithm they are successive gradient-descent approximations to the optimum solution.

The role of the forgetting factor λ may be understood qualitatively, by considering the quantity

$$n_\lambda=\frac{\displaystyle\sum_{n=0}^{\infty}n\lambda^n}{\displaystyle\sum_{n=0}^{\infty}\lambda^n}=\frac{\lambda}{1-\lambda}$$

to be a *measure* of the effective memory of the performance index $\mathcal{E}_n$. Smaller λs correspond to shorter memory n_λ, and can track better the non-stationary changes of the underlying signals. The memory n_λ of the performance index should be as short as the effective duration of the non-stationary segments, but not shorter because the performance index will not be taking full advantage of all the available samples (which could extend over the entire non-stationary segment); as a result, the computed weights $\mathbf{h}(n)$ will exhibit more noisy behavior. In particular, if the signals are stationary, the best value of λ is unity.

In Sec. 16.12, we considered the adaptive implementation of eigenvector methods based on an LMS gradient-projection method. Adaptive eigenvector methods can also be formulated based on the rank-one updating property (16.16.5). For example, one may use standard numerical methods for the rank-one updating of the entire eigenproblem of $R(n)$ [1166,1458,1459].

If one is interested only in a few largest or smallest eigenvalues and corresponding eigenvectors, one can use the more efficient power method or inverse power method and their generalizations, such as the simultaneous and subspace iterations, or Lanczos methods, which are essentially the subspace iteration improved by Rayleigh-Ritz methods [1246,1460].

The basic procedure for making these numerical methods adaptive is as follows [1461–1467]. The power method generates the maximum eigenvector by the iteration $\mathbf{e}(n)=R\mathbf{e}(n-1)$, followed by normalization of $\mathbf{e}(n)$ to unit norm. Similarly, the minimum eigenvector may be generated by the inverse power iteration $\mathbf{e}(n)=R^{-1}\mathbf{e}(n-1)$. Because R and R^{-1} are not known, they may be replaced by their estimates $R(n)$ and $P(n)=R(n)^{-1}$, which are being updated from one time instant to the next by Eq. (16.16.5) or by step 4 of the RLS algorithm, so that one has $\mathbf{e}(n)=R(n)\mathbf{e}(n-1)$ for the power iteration, or $\mathbf{e}(n)=P(n)\mathbf{e}(n-1)$ for the inverse power case.

This can be generalized to the simultaneous iteration case. For example, to generate adaptively the K minimum eigenvectors spanning the noise subspace one starts at each iteration n with K mutually orthonormalized vectors $\mathbf{e}_i(n-1)$, $i=0,1,\dots,K-1$. Each is subjected to the inverse power iteration $\mathbf{e}_i(n)=P(n)\mathbf{e}_i(n-1)$ and finally, the K updated vectors $\mathbf{e}_i(n)$ are mutually orthonormalized using the Gram-Schmidt or modified Gram-Schmidt procedure for vectors. Similar simultaneous iteration methods can also be applied to the gradient-projection method of Sec. 16.12. The main limitation of applying the simultaneous iteration methods is that one must know in advance the dimension K of the noise subspace.

16.17 Fast RLS Filters

In this section, we present fast RLS algorithms based on a direct-form realization [1422–1424,1436–1445,1468–1477]. Fast RLS lattice filters are discussed in the next section. The fast direct-form RLS algorithms make use of the forward and backward predictors. The subblock decompositions of the $(M+1)$-dimensional data vector $\mathbf{y}(n)$ are

$$\mathbf{y}(n)=\begin{bmatrix}y_n\\y_{n-1}\\\vdots\\y_{n-M}\end{bmatrix}=\begin{bmatrix}\bar{\mathbf{y}}(n)\\y_{n-M}\end{bmatrix}=\begin{bmatrix}y_n\\\tilde{\mathbf{y}}(n)\end{bmatrix} \tag{16.17.1}$$

Therefore, the two M-dimensional parts of $\mathbf{y}(n)$ are

$$\bar{\mathbf{y}}(n)=\begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-M+1}\end{bmatrix}, \quad \tilde{\mathbf{y}}(n)=\begin{bmatrix} y_{n-1} \\ y_{n-2} \\ \vdots \\ y_{n-M}\end{bmatrix} \tag{16.17.2}$$

The covariance matrices of these subvectors will be

$$\bar{R}(n)=\sum_{k=0}^{n}\lambda^{n-k}\bar{\mathbf{y}}(k)\bar{\mathbf{y}}(k)^T, \quad \tilde{R}(n)=\sum_{k=0}^{n}\lambda^{n-k}\tilde{\mathbf{y}}(k)\tilde{\mathbf{y}}(k)^T \tag{16.17.3}$$

The definitions (16.17.2) imply the *shift-invariance* property

$$\tilde{\mathbf{y}}(n+1)=\bar{\mathbf{y}}(n) \tag{16.17.4}$$

Using this property, we find

$$\begin{aligned}\tilde{R}(n+1) &= \sum_{k=0}^{n+1}\lambda^{n+1-k}\tilde{\mathbf{y}}(k)\tilde{\mathbf{y}}(k)^T=\sum_{k=-1}^{n}\lambda^{n-k}\tilde{\mathbf{y}}(k+1)\tilde{\mathbf{y}}(k+1)^T \\ &= \sum_{k=-1}^{n}\lambda^{n-k}\bar{\mathbf{y}}(k)\bar{\mathbf{y}}(k)^T=\bar{R}(n)+\lambda^{n+1}\bar{\mathbf{y}}(-1)\bar{\mathbf{y}}(-1)^T\end{aligned}$$

If we make the *prewindowing* assumption that $\bar{\mathbf{y}}(-1)=0$, we obtain the shift-invariance property for the covariance matrices

$$\tilde{R}(n+1)=\bar{R}(n) \tag{16.17.5}$$

Before we use the shift-invariance properties, we make some additional correspondences from the previous section:

$\bar{\mathbf{y}}$	$\to$	$\bar{\mathbf{y}}(n)$
$\tilde{\mathbf{y}}$	$\to$	$\tilde{\mathbf{y}}(n)$
$R_1\mathbf{a}_1 = E_{1a}\mathbf{u}$	$\to$	$R(n)\mathbf{a}(n)=E^+(n)\mathbf{u}$
$R_1\mathbf{b}_1 = E_{1b}\mathbf{v}$	$\to$	$R(n)\mathbf{b}(n)=E^-(n)\mathbf{v}$
$R_0\mathbf{a}_0 = E_{0a}\mathbf{u}$	$\to$	$\lambda R(n-1)\mathbf{a}(n-1)=\lambda E^+(n-1)\mathbf{u}$
$R_0\mathbf{b}_0 = E_{0b}\mathbf{v}$	$\to$	$\lambda R(n-1)\mathbf{b}(n-1)=\lambda E^-(n-1)\mathbf{v}$
$e_{1a} = \mathbf{a}_1^T\mathbf{y}$	$\to$	$e^+(n)=\mathbf{a}(n)^T\mathbf{y}(n)$
$e_{1b} = \mathbf{b}_1^T\mathbf{y}$	$\to$	$e^-(n)=\mathbf{b}(n)^T\mathbf{y}(n)$
$e_{0a} = \mathbf{a}_0^T\mathbf{y}$	$\to$	$e^+(n/n-1)=\mathbf{a}(n-1)^T\mathbf{y}(n)$
$e_{0b} = \mathbf{b}_0^T\mathbf{y}$	$\to$	$e^-(n/n-1)=\mathbf{b}(n-1)^T\mathbf{y}(n)$
$E_{1a} = E_{0a}+e_{1a}e_{0a}$	$\to$	$E^+(n)=\lambda E^+(n-1)+e^+(n)e^+(n/n-1)$
$E_{1b} = E_{0b}+e_{1b}e_{0b}$	$\to$	$E^-(n)=\lambda E^-(n-1)+e^-(n)e^-(n/n-1)$
$\tilde{\mathbf{k}}_1 = \tilde{R}_1^{-1}\tilde{\mathbf{y}}$	$\to$	$\tilde{\mathbf{k}}(n)=\tilde{R}(n)^{-1}\tilde{\mathbf{y}}(n)$
$\bar{\mathbf{k}}_1 = \bar{R}_1^{-1}\bar{\mathbf{y}}$	$\to$	$\bar{\mathbf{k}}(n)=\bar{R}(n)^{-1}\bar{\mathbf{y}}(n)$
$\tilde{\mathbf{k}}_0 = \tilde{R}_0^{-1}\tilde{\mathbf{y}}$	$\to$	$\tilde{\mathbf{k}}(n/n-1)=\lambda^{-1}\tilde{R}(n-1)^{-1}\tilde{\mathbf{y}}(n)$
$\bar{\mathbf{k}}_0 = \bar{R}_0^{-1}\bar{\mathbf{y}}$	$\to$	$\bar{\mathbf{k}}(n/n-1)=\lambda^{-1}\bar{R}(n-1)^{-1}\bar{\mathbf{y}}(n)$
$\tilde{\nu} = \tilde{\mathbf{k}}_0^T\tilde{\mathbf{y}}$	$\to$	$\tilde{\nu}(n)=\tilde{\mathbf{k}}(n/n-1)^T\tilde{\mathbf{y}}(n)$
$\bar{\nu} = \bar{\mathbf{k}}_0^T\bar{\mathbf{y}}$	$\to$	$\bar{\nu}(n)=\bar{\mathbf{k}}(n/n-1)^T\bar{\mathbf{y}}(n)$
$\tilde{\mu} = 1/(1+\tilde{\nu})$	$\to$	$\tilde{\mu}(n)=1/\big(1+\tilde{\nu}(n)\big)$
$\bar{\mu} = 1/(1+\bar{\nu})$	$\to$	$\bar{\mu}(n)=1/\big(1+\bar{\nu}(n)\big)$

We have used the superscripts $\pm$ to indicate the forward and backward quantities. Again, note the cancellation of the factors λ from the a priori normal equations, which implies that the a priori predictors are the predictors of the previous time instant; that is, $\mathbf{a}_0 \to \mathbf{a}(n-1)$ and $\mathbf{b}_0 \to \mathbf{b}(n-1)$.

Using the shift-invariance properties (16.17.4) and (16.17.5), we find that all the tilde quantities at the next time instant $n+1$ are equal to the bar quantities at the present instant n; for example,

$$\tilde{\mathbf{k}}(n+1)=\tilde{R}(n+1)^{-1}\tilde{\mathbf{y}}(n+1)=\bar{R}(n)^{-1}\bar{\mathbf{y}}(n)=\bar{\mathbf{k}}(n)$$

Similarly,

$$\tilde{\mathbf{k}}(n+1/n)=\lambda^{-1}\tilde{R}(n)^{-1}\tilde{\mathbf{y}}(n+1)=\lambda^{-1}\bar{R}(n-1)^{-1}\bar{\mathbf{y}}(n)=\bar{\mathbf{k}}(n/n-1)$$

and for the likelihood variables

$$\tilde{\nu}(n+1)=\tilde{\mathbf{k}}(n+1/n)^T\tilde{\mathbf{y}}(n+1)=\bar{\mathbf{k}}(n/n-1)^T\bar{\mathbf{y}}(n)=\bar{\nu}(n)$$

and similarly for the μs. We summarize:

$$\begin{aligned}\tilde{\mathbf{k}}(n+1) &= \bar{\mathbf{k}}(n), \quad \tilde{\mathbf{k}}(n+1/n)=\bar{\mathbf{k}}(n/n-1) \\ \tilde{\nu}(n+1) &= \bar{\nu}(n), \quad \tilde{\mu}(n+1)=\bar{\mu}(n)\end{aligned} \tag{16.17.6}$$

These equations can be added at the ends of the computational sequences of the previous section to complete the computational cycle at each time instant. In the present notation, the complete *fast Kalman algorithm* [1422,1423] is:

0. At time n, we have available the quantities $\mathbf{h}(n-1)$, $\mathbf{a}(n-1)$, $\mathbf{b}(n-1)$, $\tilde{\mathbf{k}}(n)$, $E^+(n-1)$, $x(n)$, and $\mathbf{y}(n)$

1. $e^+(n/n-1)=\mathbf{a}(n-1)^T\mathbf{y}(n)$

2. $\mathbf{a}(n)=\mathbf{a}(n-1)-e^+(n/n-1)\begin{bmatrix}0\\ \tilde{\mathbf{k}}(n)\end{bmatrix}$

3. $e^+(n)=\mathbf{a}(n)^T\mathbf{y}(n)$

4. $E^+(n)=\lambda E^+(n-1)+e^+(n)e^+(n/n-1)$

5. Compute the first element of $\mathbf{k}(n)$, $k_0(n)=\dfrac{e^+(n)}{E^+(n)}$

6. $\mathbf{k}(n)=\begin{bmatrix}0\\ \tilde{\mathbf{k}}(n)\end{bmatrix}+k_0(n)\mathbf{a}(n)$, extract the last element of $\mathbf{k}(n)$, $k_M(n)$

7. $e^-(n/n-1)=\mathbf{b}(n-1)^T\mathbf{y}(n)$

8. $\mathbf{b}(n)=\dfrac{\mathbf{b}(n-1)-e^-(n/n-1)\mathbf{k}(n)}{1-e^-(n/n-1)k_M(n)}$

9. $\begin{bmatrix}\bar{\mathbf{k}}(n)\\ 0\end{bmatrix}=\mathbf{k}(n)-k_M(n)\mathbf{b}(n)$

10. $\hat{x}(n/n-1) = \mathbf{h}(n-1)^T\mathbf{y}(n)\,,\quad e(n/n-1) = x(n) - \hat{x}(n/n-1)$

11. $\mathbf{h}(n) = \mathbf{h}(n-1) + e(n/n-1)\mathbf{k}(n)$

12. $\hat{x}(n) = \mathbf{h}(n)^T\mathbf{y}(n)\,,\quad e(n) = x(n) - \hat{x}(n)$

13. $\tilde{\mathbf{k}}(n+1) = \bar{\mathbf{k}}(n)$

14. Go to the next time instant, $n \to n+1$

The first and last entries of the a posteriori Kalman gain vector $\mathbf{k}(n)$ were denoted by $k_0(n)$ and $k_M(n)$, that is, $\mathbf{k}(n) = [k_0(n), k_1(n), \dots, k_M(n)]^T$. Similarly, we obtain the complete *FAEST algorithm* [1424]:

0. At time n, we have available the quantities $\mathbf{h}(n-1)$, $\mathbf{a}(n-1)$, $\mathbf{b}(n-1)$, $\tilde{\mathbf{k}}(n/n-1)$, $\tilde{\nu}(n)$, $E^{\pm}(n-1)$, $x(n)$, and $\mathbf{y}(n)$

1 . $e^+(n/n-1) = \mathbf{a}(n-1)^T\mathbf{y}(n)$

2. $e^+(n) = e^+(n/n-1)/\big(1+\tilde{\nu}(n)\big) = \tilde{\mu}(n)e^+(n/n-1)$

3. Compute the first element of $\mathbf{k}(n/n-1)$, $k_0(n/n-1) = \dfrac{e^+(n/n-1)}{\lambda E^+(n-1)}$

4. $E^+(n) = \lambda E^+(n-1) + e^+(n)e^+(n/n-1)$

5. $\mathbf{k}(n/n-1) = \begin{bmatrix} 0 \\ \tilde{\mathbf{k}}(n/n-1) \end{bmatrix} + k_0(n/n-1)\mathbf{a}(n-1)$

6. Extract the last element of $\mathbf{k}(n/n-1)$, $k_M(n/n-1)$

7. $e^-(n/n-1) = k_M(n/n-1)\big[\lambda E^-(n-1)\big]$

8. $\begin{bmatrix} \bar{\mathbf{k}}(n/n-1) \\ 0 \end{bmatrix} = \mathbf{k}(n/n-1) - k_M(n/n-1)\mathbf{b}(n-1)$

9. $\nu(n) = \tilde{\nu}(n) + e^+(n/n-1)k_0(n/n-1)\,,\quad \bar{\nu}(n) = \nu(n) - e^-(n/n-1)k_M(n/n-1)$

10. $e^-(n) = e^-(n/n-1)/\big(1+\bar{\nu}(n)\big) = \bar{\mu}(n)e^-(n/n-1)$

11. $E^-(n) = \lambda E^-(n-1) + e^-(n)e^-(n/n-1)$

12. $\mathbf{a}(n) = \mathbf{a}(n-1) - e^+(n)\begin{bmatrix} 0 \\ \tilde{\mathbf{k}}(n/n-1) \end{bmatrix}$

13. $\mathbf{b}(n) = \mathbf{b}(n-1) - e^-(n)\begin{bmatrix} \bar{\mathbf{k}}(n/n-1) \\ 0 \end{bmatrix}$

14. $\hat{x}(n/n-1) = \mathbf{h}(n-1)^T\mathbf{y}(n)\,,\quad e(n/n-1) = x(n) - \hat{x}(n/n-1)$

15. $e(n) = e(n/n-1)/\big(1+\nu(n)\big) = \mu(n)e(n/n-1)\,,\quad \hat{x}(n) = x(n) - e(n)$

16. $\mathbf{h}(n) = \mathbf{h}(n-1) + e(n)\mathbf{k}(n/n-1)$

17. $\tilde{\mathbf{k}}(n+1/n) = \bar{\mathbf{k}}(n)\,,\quad \tilde{\nu}(n+1) = \bar{\nu}(n)$

19. Go to the next time instant, $n \to n+1$

The algorithm is initialized in time by clearing the tapped delay line of the filter and setting $\mathbf{h}(-1) = 0$, $\mathbf{a}(-1) = \mathbf{u} = [1, \mathbf{0}^T]^T$, $\mathbf{b}(-1) = \mathbf{v} = [\mathbf{0}^T, 1]^T$, $\tilde{\mathbf{k}}(0/-1) = 0$, $\tilde{\nu}(0) = 0$, and $E^{\pm}(-1) = \delta$, where δ is a small constant. Exact initialization procedures have been discussed in [1426]. The *FTF algorithm* [1426] is obtained by replacing step 9 by the following:

$$\mu(n) = \tilde{\mu}(n)\,\frac{\lambda E^+(n-1)}{E^+(n)}\,,\quad \bar{\mu}(n) = \frac{\mu(n)}{1 - e^-(n/n-1)k_M(n/n-1)\mu(n)} \qquad \text{(FTF)}$$

The function **faest** is an implementation of the FAEST algorithm. The function transforms an input pair of samples $\{x, y\}$ into an output pair $\{\hat{x}, e\}$, updates the tapped delay line of the filter, and updates the filter $\mathbf{h}(n)$.

Next, we present a simulation example comparing the FAEST and LMS algorithms. The example is the same as that discussed in Sec. 16.13 and defined theoretically by Eqs. (16.13.37) and (16.13.38). Fig. 16.17.1 shows two of the adaptive weights, $h_1(n)$ and $h_2(n)$, adapted by FAEST and LMS. The weights are converging to their theoretical values of $h_1 = 1.5$ and $h_2 = -2$. The RLS parameters were $\lambda = 1$ and $\delta = 0.01$; the LMS parameter was $\mu = 0.01$.

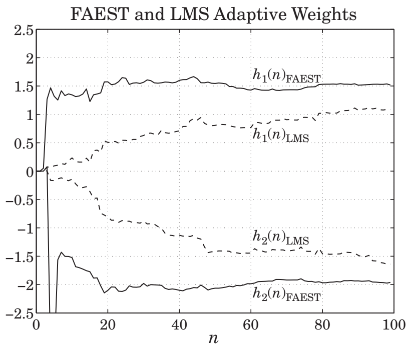


Fig. 16.17.1 Comparison of FAEST and LMS adaptive weights.

16.18 RLS Lattice Filters

The fast direct-form RLS filters were fixed-order filters. By contrast, the RLS lattice algorithms [1436–1445], for each time instant n, do a recursion in the order, $p = 0, 1, \dots, M$. Therefore, it is necessary to indicate the order p by using an extra index in all the quantities of the past two sections. For example, the order-p data vector and its bar and tilde parts will be denoted by

$$\mathbf{y}_p(n)=\begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-p}\end{bmatrix}, \quad \bar{\mathbf{y}}_p(n)=\begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-p+1}\end{bmatrix}, \quad \tilde{\mathbf{y}}_p(n)=\begin{bmatrix} y_{n-1} \\ y_{n-2} \\ \vdots \\ y_{n-p}\end{bmatrix} \qquad (16.18.1)$$

Therefore, we have

$$\bar{\mathbf{y}}_p(n)=\mathbf{y}_{p-1}(n)\,, \quad \tilde{\mathbf{y}}_p(n)=\bar{\mathbf{y}}_p(n-1)=\mathbf{y}_{p-1}(n-1) \qquad (16.18.2)$$

Similarly, the covariance matrices will be

$$\bar{R}_p(n)=R_{p-1}(n)\,, \quad \tilde{R}_p(n)=R_{p-1}(n-1) \qquad (16.18.3)$$

The order-p predictors will be denoted by $\mathbf{a}_p(n)$ and $\mathbf{b}_p(n)$, with error signals $e_p^+(n)=\mathbf{a}_p(n)^T\mathbf{y}_p(n)$ and $e_p^-(n)=\mathbf{b}_p(n)^T\mathbf{y}_p(n)$ The corresponding mean-square errors will be denoted by $E_p^\pm(n)$. Similarly, the a priori estimation errors are denoted by $e_p^+(n/n-1)=\mathbf{a}_p(n-1)^T\mathbf{y}_p(n)$ and $e_p^-(n/n-1)=\mathbf{b}_p(n-1)^T\mathbf{y}_p(n)$. Using Eq. (16.18.3), we find the following correspondences between the order-$(p-1)$ and order-p problems:

$$\begin{array}{lll lll lll}
\bar{R}_1 & \to & R_{p-1}(n), & \bar{\mathbf{a}}_1 & \to & \mathbf{a}_{p-1}(n), & \bar{E}_{1a} & \to & E^+_{p-1}(n) \\
\bar{R}_0 & \to & \lambda R_{p-1}(n-1), & \bar{\mathbf{a}}_0 & \to & \mathbf{a}_{p-1}(n-1), & \bar{E}_{0a} & \to & \lambda E^+_{p-1}(n-1) \\
\tilde{R}_1 & \to & R_{p-1}(n-1), & \tilde{\mathbf{b}}_1 & \to & \mathbf{b}_{p-1}(n-1), & \tilde{E}_{1b} & \to & E^-_{p-1}(n-1) \\
\tilde{R}_0 & \to & \lambda R_{p-1}(n-2), & \tilde{\mathbf{b}}_0 & \to & \mathbf{b}_{p-1}(n-2), & \tilde{E}_{0b} & \to & \lambda E^-_{p-1}(n-1)
\end{array}$$

$$\begin{array}{rcl}
\bar{e}_{1a}=\bar{\mathbf{a}}_1^T\bar{\mathbf{y}} & \to & e^+_{p-1}(n)=\mathbf{a}_{p-1}(n)^T\mathbf{y}_{p-1}(n) \\
\tilde{e}_{1b}=\tilde{\mathbf{b}}_1^T\tilde{\mathbf{y}} & \to & e^-_{p-1}(n-1)=\mathbf{b}_{p-1}(n-1)^T\mathbf{y}_{p-1}(n-1) \\
\bar{e}_{0a}=\bar{\mathbf{a}}_0^T\bar{\mathbf{y}} & \to & e^+_{p-1}(n/n-1)=\mathbf{a}_{p-1}(n-1)^T\mathbf{y}_{p-1}(n) \\
\tilde{e}_{0b}=\tilde{\mathbf{b}}_0^T\tilde{\mathbf{y}} & \to & e^-_{p-1}(n-1/n-2)=\mathbf{b}_{p-1}(n-2)^T\mathbf{y}_{p-1}(n-1)
\end{array}$$

$$\begin{array}{rcl}
\gamma_{1a} & \to & \gamma_p^+(n) \\
\gamma_{0a} & \to & \gamma_p^+(n-1) \\
\gamma_{1b} & \to & \gamma_p^-(n) \\
\gamma_{0b} & \to & \gamma_p^-(n-1)
\end{array}$$

$$\begin{array}{rcl}
e_{1a}=\bar{e}_{1a}-\gamma_{1b}\tilde{e}_{1b} & \to & e_p^+(n)=e^+_{p-1}(n)-\gamma_p^-(n)e^-_{p-1}(n-1) \\
e_{1b}=\tilde{e}_{1b}-\gamma_{1a}\bar{e}_{1a} & \to & e_p^-(n)=e^-_{p-1}(n-1)-\gamma_p^+(n)e^+_{p-1}(n) \\
e_{0a}=\bar{e}_{0a}-\gamma_{0b}\tilde{e}_{0b} & \to & e_p^+(n/n-1)=e^+_{p-1}(n/n-1)-\gamma_p^-(n-1)e^-_{p-1}(n-1/n-2) \\
e_{0b}=\tilde{e}_{0b}-\gamma_{0a}\bar{e}_{0a} & \to & e_p^-(n/n-1)=e^-_{p-1}(n-1/n-2)-\gamma_p^+(n-1)e^+_{p-1}(n/n-1)
\end{array}$$

$$\mathbf{a}_1=\begin{bmatrix}\bar{\mathbf{a}}_1\\0\end{bmatrix}-\gamma_{1b}\begin{bmatrix}0\\\tilde{\mathbf{b}}_1\end{bmatrix} \;\to\; \mathbf{a}_p(n)=\begin{bmatrix}\mathbf{a}_{p-1}(n)\\0\end{bmatrix}-\gamma_p^-(n)\begin{bmatrix}0\\\mathbf{b}_{p-1}(n-1)\end{bmatrix}$$

$$\mathbf{b}_1=\begin{bmatrix}0\\\tilde{\mathbf{b}}_1\end{bmatrix}-\gamma_{1a}\begin{bmatrix}\bar{\mathbf{a}}_1\\0\end{bmatrix} \;\to\; \mathbf{b}_p(n)=\begin{bmatrix}0\\\mathbf{b}_{p-1}(n-1)\end{bmatrix}-\gamma_p^+(n)\begin{bmatrix}\mathbf{a}_{p-1}(n)\\0\end{bmatrix}$$

$$\mathbf{a}_0=\begin{bmatrix}\bar{\mathbf{a}}_0\\0\end{bmatrix}-\gamma_{0b}\begin{bmatrix}0\\\tilde{\mathbf{b}}_0\end{bmatrix} \;\to\; \mathbf{a}_p(n-1)=\begin{bmatrix}\mathbf{a}_{p-1}(n-1)\\0\end{bmatrix}-\gamma_p^-(n-1)\begin{bmatrix}0\\\mathbf{b}_{p-1}(n-2)\end{bmatrix}$$

$$\mathbf{b}_0=\begin{bmatrix}0\\\tilde{\mathbf{b}}_0\end{bmatrix}-\gamma_{0a}\begin{bmatrix}\bar{\mathbf{a}}_0\\0\end{bmatrix} \;\to\; \mathbf{b}_p(n-1)=\begin{bmatrix}0\\\mathbf{b}_{p-1}(n-2)\end{bmatrix}-\gamma_p^+(n-1)\begin{bmatrix}\mathbf{a}_{p-1}(n-1)\\0\end{bmatrix}$$

$$\begin{array}{rcl}
\gamma_{1a}=\gamma_{0a}+e_{0b}\dfrac{\bar{e}_{1a}}{\bar{E}_{1a}} & \to & \gamma_p^+(n)=\gamma_p^+(n-1)+e_p^-(n/n-1)\dfrac{e^+_{p-1}(n)}{E^+_{p-1}(n)} \\
\gamma_{1b}=\gamma_{0b}+e_{0a}\dfrac{\tilde{e}_{1b}}{\tilde{E}_{1b}} & \to & \gamma_p^-(n)=\gamma_p^-(n-1)+e_p^+(n/n-1)\dfrac{e^-_{p-1}(n-1)}{E^-_{p-1}(n-1)} \\
e_0=\bar{e}_0-g_{0b}e_{0b} & \to & e_p(n/n-1)=e_{p-1}(n/n-1)-g_p(n-1)e_p^-(n/n-1) \\
g_{1b}=g_{0b}+e_0\dfrac{e_{1b}}{E_{1b}} & \to & g_p(n)=g_p(n-1)+e_p(n/n-1)\dfrac{e_p^-(n)}{E_p^-(n)} \\
e_1=\bar{e}_1-g_{1b}e_{1b} & \to & e_p(n)=e_{p-1}(n)-g_p(n)e_p^-(n)
\end{array}$$

We have denoted the forward/backward reflection coefficients by $\gamma_p^\pm(n)$, and the lattice Wiener weights by $g_p(n)$. The order-p a priori and a posteriori estimation errors are $e_p(n/n-1)=x(n)-\hat{x}_p(n/n-1)$ and $e_p(n)=x(n)-\hat{x}_p(n)$. The likelihood variable $\mu=1-\mathbf{y}^TR_1^{-1}\mathbf{y}$ is

$$\mu_p(n)=1-\mathbf{y}_p(n)^TR_p(n)^{-1}\mathbf{y}_p(n) \qquad (16.18.4)$$

and can also be written as

$$\mu_p(n)=\frac{1}{1+\nu_p(n)}=\frac{1}{1+\lambda^{-1}\mathbf{y}_p(n)^TR_p(n-1)^{-1}\mathbf{y}_p(n)}$$

Similarly, we have

$$\begin{aligned}
\tilde{\mu}_p(n) &= 1-\tilde{\mathbf{y}}_p(n)^T\tilde{R}_p(n)^{-1}\tilde{\mathbf{y}}_p(n) \\
&= 1-\mathbf{y}_{p-1}(n-1)^TR_{p-1}(n-1)^{-1}\mathbf{y}_{p-1}(n-1) \\
&= \mu_{p-1}(n-1)
\end{aligned}$$

and

$$\begin{aligned}
\bar{\mu}_p(n) &= 1-\bar{\mathbf{y}}_p(n)^T\bar{R}_p(n)^{-1}\bar{\mathbf{y}}_p(n) \\
&= 1-\mathbf{y}_{p-1}(n)^TR_{p-1}(n)^{-1}\mathbf{y}_{p-1}(n) \\
&= \mu_{p-1}(n)
\end{aligned}$$

Therefore,

$$\tilde{\mu}_p(n)=\mu_{p-1}(n-1)\,, \quad \bar{\mu}_p(n)=\mu_{p-1}(n) \qquad (16.18.5)$$

Thus, the proportionality between a posteriori and a priori errors will be

$$e_p^+(n)=\tilde{\mu}_p(n)e_p^+(n/n-1)\,, \quad e_p^-(n)=\bar{\mu}_p(n)e_p^-(n/n-1) \qquad (16.18.6)$$

Using either of Eq. (16.18.5), we find for the quantity $\bar{\tilde{\mu}}=\tilde{\bar{\mu}}$

$$\bar{\tilde{\mu}}_p(n)=\bar{\mu}_{p-1}(n-1)=\tilde{\mu}_{p-1}(n)=\mu_{p-2}(n-1) \qquad (16.18.7)$$

Based on the above correspondences, we can obtain all versions of RLS lattice algorithms, such as the conventional a posteriori, a priori, double, and a priori and a posteriori error-feedback. In particular, we summarize the complete *double/direct RLS lattice algorithm* [156]:

0. At time n, we have available the quantities $\gamma_p^\pm(n-1)$, $g_p(n-1)$, $E_p^\pm(n-1)$, and $x(n), y(n)$.

1. Initialize in order by

$$e_0^\pm(n/n-1) = e_0^\pm(n) = y(n)$$
$$E_0^\pm(n) = \lambda E_0^\pm(n-1) + e_0^\pm(n)e_0^\pm(n/n-1)$$

$$e_0(n/n-1) = x(n) - g_0(n-1)e_0^-(n/n-1)$$
$$g_0(n) = g_0(n-1) + e_0(n/n-1)\frac{e_0^-(n)}{E_0^-(n)}$$
$$e_0(n) = x(n) - g_0(n)e_0^-(n)$$

2. For $p = 1, 2, \dots, M$, compute

$$e_p^+(n/n-1) = e_{p-1}^+(n/n-1) - \gamma_p^-(n-1)e_{p-1}^-(n-1/n-2)$$
$$e_p^-(n/n-1) = e_{p-1}^-(n-1/n-2) - \gamma_p^+(n-1)e_{p-1}^+(n/n-1)$$
$$\gamma_p^+(n) = \gamma_p^+(n-1) + e_p^-(n/n-1)\frac{e_{p-1}^+(n)}{E_{p-1}^+(n)}$$
$$\gamma_p^-(n) = \gamma_p^-(n-1) + e_p^+(n/n-1)\frac{e_{p-1}^-(n-1)}{E_{p-1}^-(n-1)}$$
$$e_p^+(n) = e_{p-1}^+(n) - \gamma_p^-(n)e_{p-1}^-(n-1)$$
$$e_p^-(n) = e_{p-1}^-(n-1) - \gamma_p^+(n)e_{p-1}^+(n)$$
$$E_p^\pm(n) = \lambda E_p^\pm(n-1) + e_p^\pm(n)e_p^\pm(n/n-1)$$
$$e_p(n/n-1) = e_{p-1}(n/n-1) - g_p(n-1)e_p^-(n/n-1)$$
$$g_p(n) = g_p(n-1) + e_p(n/n-1)\frac{e_p^-(n)}{E_p^-(n)}$$
$$e_p(n) = e_{p-1}(n) - g_p(n)e_p^-(n)$$

3. $\hat{x}_M(n) = x(n) - e_M(n)$, and go to the next time instant, $n \to n+1$.

The algorithm is initialized in time by clearing the delay registers of both lattices and setting $\gamma_p^\pm(-1) = 0$, $E_p^\pm(-1) = 0$, and $g_p(-1) = 0$. As in the case of the gradient lattice, it follows that the backward outputs from the pth lattice section, $e_p^-(n/n-1)$, will be zero for $n < p$; therefore, we must keep $\gamma_p^-(n) = g_p(n) = 0$ for $n < p$ because these quantities require divisions by $E_p^-(n)$. There are 16 multiplications/divisions in step 2; therefore, the complexity of the algorithm grows like $16M$ per time update.

The **rlsl** is an implementation of the above algorithm. It is essentially the same as **lwf** used twice for the a priori and a posteriori lattices and with the weight adaptation parts added to it.

Fig. 16.18.1 shows the reflection coefficients $\gamma_1^\pm(n)$ and $\gamma_2^\pm(n)$ adapted by the RLS lattice algorithm, for the same example presented in Sec. 16.13, which was also used in

the FAEST simulation. Note that, after some initial transients, the forward and backward reflection coefficients become more or less the same as they converge to their theoretical values. Compare also with the reflection coefficients of Fig. 16.13.3 adapted by the gradient lattice. The version of the gradient lattice that we presented uses one set of reflection coefficients, which may be thought of as some sort of average combination of the forward/backward ones. Indeed, the curves for the gradient lattice reflection coefficients fall mostly between the curves of the forward and backward ones. Similarly, the lattice Wiener weights $g_p(n)$ have almost the same behavior as those of Fig. 16.13.3. We finish this section by discussing LU factorizations. Equations (16.15.20) become

$$L_p(n)R_p(n)L_p(n)^T = D_p^-(n)\,, \quad \lambda L_p(n-1)R_p(n-1)L_p(n-1)^T = \lambda D_p^-(n-1) \tag{16.18.8}$$

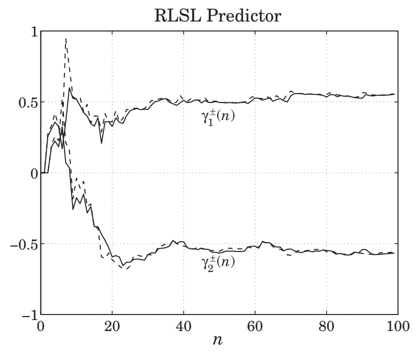


Fig. 16.18.1 Reflection coefficients adapted by the double/direct RLSL algorithm.

where

$$D_p^-(n) = \text{diag}\{E_0^-(n), E_1^-(n), \dots, E_p^-(n)\}$$

The vectors of a posteriori and a priori backward error signals are constructed by

$$\mathbf{e}_p^-(n) = \begin{bmatrix} e_0^-(n) \\ e_1^-(n) \\ \vdots \\ e_p^-(n) \end{bmatrix} = L_p(n)\mathbf{y}_p(n)\,,$$

$$\mathbf{e}_p^-(n/n-1) = \begin{bmatrix} e_0^-(n/n-1) \\ e_1^-(n/n-1) \\ \vdots \\ e_p^-(n/n-1) \end{bmatrix} = L_p(n-1)\mathbf{y}_p(n)$$

This follows from the fact that the rows of the matrices $L_p(n)$ are the backward predictors of successive orders. The $L_p(n)$ matrices are related by Eq. (16.15.54), which reads

$$L_p(n) = L_p(n/n-1)L_p(n-1) \tag{16.18.9}$$

The rows of the unit lower triangular updating matrix $L_p(n/n-1)$ are constructed by (16.15.59), that is,

$$\boldsymbol{\beta}_p = -e_p^-(n)\big[\lambda D_{p-1}^-(n-1)\big]^{-1}\mathbf{e}_{p-1}^-(n/n-1) \qquad (16.18.10)$$

or, component-wise

$$\beta_{pi} = -e_p^-(n)\frac{e_i^-(n/n-1)}{\lambda E_i^-(n-1)} = -\bar{\mu}_p(n)e_p^-(n/n-1)\frac{e_i^-(n/n-1)}{\lambda E_i^-(n-1)}\,, \quad i = 0,1,\dots,p-1$$

The direct and lattice Wiener weights are related by Eq. (16.15.60), i.e., $\mathbf{g}_p(n) = L_p(n)^{-T}\mathbf{h}_p(n)$, and the a posteriori and a priori estimation errors are given by (16.15.61)

$$\hat{x}_p(n) = \mathbf{g}_p(n)^T\mathbf{e}_p(n)\,, \quad \hat{x}_p(n/n-1) = \mathbf{g}_p(n-1)^T\mathbf{e}_p^-(n/n-1) \qquad (16.18.11)$$

and satisfy the recursions in order

$$\hat{x}_p(n) = \hat{x}_{p-1}(n) + g_p(n)e_p^-(n)\,, \quad \hat{x}_p(n/n-1) = \hat{x}_{p-1}(n/n-1) + g_p(n-1)e_p^-(n/n-1)$$

This implies the following recursions for the estimation errors

$$e_p(n) = e_{p-1}(n) - g_p(n)e_p^-(n)\,, \quad e_p(n/n-1) = e_{p-1}(n/n-1) - g_p(n-1)e_p^-(n/n-1)$$

Finally, the time updating equation (16.15.62) for the lattice weights takes the form

$$\mathbf{g}_p(n) = L_p(n/n-1)^{-T}\mathbf{g}_p(n-1) + e_p(n/n-1)D_p^-(n)^{-1}\mathbf{e}_p^-(n)$$

and extracting the last component, we obtain

$$g_p(n) = g_p(n-1) + e_p(n/n-1)\frac{e_p^-(n)}{E_p^-(n)}$$

RLS lattice and gradient adaptive lattice filters may be used in any Wiener filtering application. Their attractive features are: (a) computational *efficiency*; (b) very *fast* rate of convergence, which is essentially independent of the eigenvalue spread of the input covariance matrix; (c) *modularity* of structure admitting parallel VLSI implementations; and (d) numerical *stability* and accuracy under quantization.

16.19 Computer Project – Adaptive Wiener Filters

It is desired to design an adaptive Wiener filter to enhance a sinusoidal signal buried in noise. The noisy sinusoidal signal is given by

$$x_n = s_n + v_n\,, \qquad \text{where} \quad s_n = \sin(\omega_0 n)$$

with $\omega_0 = 0.075\pi$. The noise v_n is related to the secondary signal y_n by

$$v_n = y_n + y_{n-1} + y_{n-2} + y_{n-3} + y_{n-4} + y_{n-5} + y_{n-6}$$

The signal y_n is assumed to be an order-4 AR process with reflection coefficients:

$$\{\gamma_1, \gamma_2, \gamma_3, \gamma_4\} = \{0.5, -0.5, 0.5, -0.5\}$$

The variance σ_ϵ^2 of the driving white noise of the model must be chosen in such a way as to make the variance σ_v^2 of the noise component v_n approximately unity.

a. For a Wiener filter of order $M = 6$, determine the theoretical direct-form Wiener filter coefficient vector:

$$\mathbf{h} = [h_0, h_1, \dots, h_6]$$

for estimating x_n (or, rather v_n) from y_n. Determine also the theoretical lattice/ladder realization coefficients:

$$\boldsymbol{\gamma} = [\gamma_1, \gamma_2, \dots, \gamma_6]\,, \qquad \mathbf{g} = [g_0, g_1, \dots, g_6]$$

b. Generate input pairs $\{x_n, y_n\}$ (making sure that the transients introduced by the filter have died out), and filter them through the LMS algorithm to generate the filter output pairs $\{\hat{x}_n, e_n\}$. On the same graph, plot e_n together with the desired signal s_n.

Plot also a few of the adaptive filter coefficients such as $h_4(n)$, $h_5(n)$, and $h_6(n)$. Observe their convergence to the theoretical Wiener solution.

You must generate enough input pairs in order to achieve convergence of the LMS algorithm and observe the steady-state converged output of the filter.

Experiment with the choice of the adaptation parameter μ. Start by determining $\lambda_{\max}$, $\lambda_{\min}$, the eigenvalue spread $\lambda_{\max}/\lambda_{\min}$ of R and the corresponding time constant.

c. Repeat (b), using the gradient lattice adaptive filter. Plot all of the adaptive reflection coefficients $\gamma_p(n)$ versus n, and a few of the ladder coefficients, such as $g_4(n)$, $g_5(n)$, and definitely $g_6(n)$.

(Because theoretically $g_6 = h_6$ (why?), plotting $h_6(n)$ and $g_6(n)$ will let you compare the convergence speeds of the LMS and lattice adaptive filters.)

You must experiment with a couple of values of λ (use $\beta = 1$). You must work of course with exactly the same set of input pairs as in part (b).

d. Next, we change this experiment into a non-stationary one. Suppose the total number of input pairs that you used in parts (b) and (c) is N. And suppose that at time $n = N$, the input statistics changes suddenly so that the primary signal is given now by the model:

$$x_n = s_n + v_n\,, \qquad \text{where} \quad v_n = y_n + y_{n-1} + y_{n-2} + y_{n-3}$$

and y_n changes from a fourth-order AR model to a second-order model with reflection coefficients (use the same σ_ϵ^2

$$\{\gamma_1, \gamma_2\} = \{0.5, -0.5\}$$

Repeat parts (a,b,c), keeping the filter order the same, $M = 6$. Use $2N$ input pairs, such that the first N follow the original statistics and the second N follow the changed statistics. Compare the capability of the LMS and lattice adaptive filters in tracking such changes.

Here, the values of μ for the LMS case and λ for the lattice case, will make more of a difference in balancing the requirements of learning speed and quality of estimates.

e. Finally, feel free to "tweak" the statements of all of the above parts as well as the definition of the models in order to show more clearly and more dramatically the issues involved, namely, LMS versus lattice, learning speed versus quality, and the effect of the adaptation parameters, eigenvalue spread, and time constants. One other thing to notice in this experiment is that, while the adaptive weights tend to fluctuate a lot as they converge, the actual filter outputs $\hat{x}_n, e_n$ behave better and are closer to what one might expect.

16.20 Problems

16.1 *Computer Experiment.* (a) Reproduce the results of Fig. 16.3.2.

(b) On the same graph of part (a), plot the theoretical convergence curve of the weight $h(n)$ obtained by using Eq. (16.2.8).

(c) Using 10 different realizations of x_n and y_n, compute 10 different realizations of the adaptive weight of Eq. (16.3.2). Compute the average weight over the 10 realizations and plot it versus n, together with the theoretical weight of Eq. (16.2.8). Use $\mu = 0.03$.

(d) Reproduce the results of Fig. 16.5.2.

16.2 In steered adaptive arrays [1093] and other applications, one has to solve a constrained Wiener filtering problem. Suppose the $(M+1)$-dimensional weight vector $\mathbf{h} = [h_0, h_1, \dots, h_M]^T$ satisfies the L linear constraints $\mathbf{c}_i^T\mathbf{h} = f_i$, $i = 1, 2, \dots, L$, where $L \le M$ and the $\mathbf{c}_i$ are given $(M+1)$-dimensional vectors, and f_i are given scalars. The set of constraints may be written compactly as $C^T\mathbf{h} = \mathbf{f}$, where $C = [\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_L]$ and $\mathbf{f} = [f_1, f_2, \dots, f_L]^T$.

(a) Show that the solution of the minimization problem $\mathcal{E} = E[e_n^2] = \min$, subject to the constraint $C^T\mathbf{h} = \mathbf{f}$, is given by

$$\mathbf{h} = \mathbf{h}_u + R^{-1}C(C^TR^{-1}C)^{-1}(\mathbf{f} - C^T\mathbf{h}_u)$$

where $\mathbf{h}_u = R^{-1}\mathbf{r}$ is the unconstrained Wiener solution and $R = E[\mathbf{y}(n)\mathbf{y}(n)^T]$, $\mathbf{r} = E[x_n\mathbf{y}(n)]$.

(b) In an adaptive implementation, $\mathbf{h}(n+1) = \mathbf{h}(n) + \Delta\mathbf{h}(n)$, the constraint must be satisfied at each iteration. The time update term, therefore, must satisfy $C^T\Delta\mathbf{h}(n) = 0$. Show that the following (gradient projection) choice satisfies this condition

$$\Delta\mathbf{h}(n) = -\mu\mathcal{P}\frac{\partial\mathcal{E}}{\partial\mathbf{h}(n)}, \quad \mathcal{P} = I - C(C^TC)^{-1}C^T$$

Moreover, show that this choice moves the performance index closer to its minimum at each iteration.

(c) Show that the resulting difference equation can be written as

$$\mathbf{h}(n+1) = \mathcal{P}\big[\mathbf{h}(n) - 2\mu R\mathbf{h}(n) + 2\mu\mathbf{r}\big] + \mathbf{h}_{LS}$$

where $\mathbf{h}_{LS} = C(C^TC)^{-1}\mathbf{f}$ is recognized as the least-squares solution of the linear equation $C^T\mathbf{h} = \mathbf{f}$. And, show that $C^T\mathbf{h}(n+1) = \mathbf{f}$.

(d) Show that the LMS adaptive algorithm resulting by dropping the expectation values is, with $e_n = x_n - \hat{x}_n = x_n - \mathbf{h}(n)^T\mathbf{y}(n)$

$$\mathbf{h}(n+1) = \mathcal{P}\big[\mathbf{h}(n) + 2\mu e_n\mathbf{y}(n)\big] + \mathbf{h}_{LS}$$

16.3 Rederive the results in parts (c) and (d) of Problem 16.2 using the following approach. Introduce a Lagrange multiplier vector $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \dots, \lambda_L]^T$ into the performance index enforcing the constraint equations; that is, $\mathcal{E} = E[e_n^2] + \boldsymbol{\lambda}^T(\mathbf{f} - C^T\mathbf{h})$. Show that the ordinary unconstrained gradient descent method $\mathbf{h}(n+1) = \mathbf{h}(n) - \mu\partial\mathcal{E}/\partial\mathbf{h}(n)$ gives rise to the difference equation

$$\mathbf{h}(n+1) = (I - 2\mu R)\mathbf{h}(n) + 2\mu\mathbf{r} - \mu C\boldsymbol{\lambda}(n)$$

Impose the constraint $C^T\mathbf{h}(n+1) = \mathbf{f}$, eliminate $\boldsymbol{\lambda}(n)$, and show that this equation is equivalent to that in part (c) of the previous problem.

16.4 Verify that Eq. (16.6.5) is the solution of Eq. (16.6.4).

16.5 Consider an adaptive filter with two taps:

$$\hat{x}_n = h_0(n)y_n + h_1(n)y_{n-1} = [h_0(n), h_1(n)]\begin{bmatrix} y_n \\ y_{n-1}\end{bmatrix} = \mathbf{h}(n)^T\mathbf{y}(n)$$

The optimal filter weights are found adaptively by the gradient descent algorithm

$$\mathbf{h}(n+1) = \mathbf{h}(n) - \mu\frac{\partial\mathcal{E}}{\partial\mathbf{h}(n)}$$

where $\mathcal{E} = E[e_n^2]$ and e_n is the estimation error.

(a) Show that the above difference equation may be written as

$$\mathbf{h}(n+1) = \mathbf{h}(n) + 2\mu\big(\mathbf{r} - R\mathbf{h}(n)\big)$$

where

$$\mathbf{r} = \begin{bmatrix} R_{xy}(0) \\ R_{xy}(1)\end{bmatrix}, \quad R = \begin{bmatrix} R_{yy}(0) & R_{yy}(1) \\ R_{yy}(1) & R_{yy}(0)\end{bmatrix}$$

(b) Suppose $R_{xy}(0) = 10$, $R_{xy}(1) = 5$, $R_{yy}(0) = 3$, $R_{yy}(1) = 2$. Find the optimal weights $\mathbf{h} = \lim \mathbf{h}(n)$ as $n \to \infty$.

(c) Select $\mu = 1/6$. Explain why such a value is sufficiently small to guarantee convergence of the difference equation of part (a). What other values of μ also guarantee convergence?

(d) With $\mu = 1/6$, solve the difference equation of part (a) in closed form for $n \ge 0$. Discuss the rate of convergence.

16.6 Consider a single CCL as shown in Fig. 16.3.1.

(a) Suppose the reference signal is set equal to a unit step signal; that is, $y(n) = u(n)$. Show that the CCL will behave as a time-invariant linear filter with input x_n and output e_n. Determine the transfer function $H(z)$ from x_n to e_n.

(b) Find and interpret the poles and zeros of $H(z)$.

(c) Determine the range of μ-values for which $H(z)$ is stable.

16.7 Repeat Problem 16.6 when the reference signal is the alternating unit step; that is, $y(n) = (-1)^n u(n)$.

16.8 Let $\mathbf{h}_R$ and $\mathbf{h}_I$ be the real and imaginary parts of the complex weight vector $\mathbf{h} = \mathbf{h}_R + j\mathbf{h}_I$. Show that

$$\frac{\partial\mathcal{E}}{\partial\mathbf{h}^*} = \frac{1}{2}\left[\frac{\partial\mathcal{E}}{\partial\mathbf{h}_R} + j\frac{\partial\mathcal{E}}{\partial\mathbf{h}_I}\right]$$

Consider the simultaneous gradient descent with respect to $\mathbf{h}_R$ and $\mathbf{h}_I$, that is, $\mathbf{h}_R \to \mathbf{h}_R + \Delta\mathbf{h}_R$ and $\mathbf{h}_I \to \mathbf{h}_I + \Delta\mathbf{h}_I$, with

$$\Delta\mathbf{h}_R = -\mu\frac{\partial\mathcal{E}}{\partial\mathbf{h}_R}, \quad \Delta\mathbf{h}_I = -\mu\frac{\partial\mathcal{E}}{\partial\mathbf{h}_I}$$

Show that it is equivalent to the gradient descent $\mathbf{h} \to \mathbf{h} + \Delta\mathbf{h}$, where

$$\Delta\mathbf{h} = -2\mu\frac{\partial\mathcal{E}}{\partial\mathbf{h}^*}$$

Note the conjugation and the factor of two.

16.9 Using the transfer function of Eq. (16.9.1), derive an approximate expression for the 3-dB width of the notch. You may work to lowest order in μ.

16.10 *Computer Experiment.* Consider the noise canceling example discussed in Sec. 12.11 and in Problems 12.25–12.27 and defined by the following choice of parameters:

$$\omega_0 = 0.075\pi \ \text{[rads/sample]}, \quad \phi = 0, \quad a_1 = -0.5, \quad a_2 = 0.8, \quad M = 4$$

(a) Generate a realization of the signals $x(n)$ and $y(n)$ and process them through the adaptive noise canceler of Sec. 16.9, using an Mth order adaptive filter and adaptation parameter μ. By trial and error select a value for μ that makes the LMS algorithm convergent, but not too small as to make the convergence too slow. Plot one of the filter weights $h_m(n)$ versus iteration number n, and compare the asymptotic value with the theoretical value obtained in Problem 12.26.

(b) After the weights have converged, plot 100 output samples of the error signal $e(n)$, and observe the noise cancellation property.

(c) Repeat (a) and (b) using an adaptive filter of order $M = 6$.

16.11 *Computer Experiment.* (a) Plot the magnitude of the frequency response of the adaptive noise canceler notch filter of Eq. (16.9.1) versus frequency ω $(z = e^{j\omega})$. Generate several such plots for various values of μ and observe the effect of μ on the width of the notch.

(b) Let $x(n) = e^{j\omega_0 n}$ and $y(n) = Ae^{j\omega_0 n}$, and select the parameters as

$$\omega_0 = 0.075\pi, \quad M = 2, \quad A = 0.01, \quad \mu = 0.1$$

Process $x(n)$ and $y(n)$ through the adaptive noise canceler of Sec. 16.9, and plot the output $e(n)$ versus n and observe the cancellation of the signal $x(n)$ due to the notch filter created by the presence of the weak sinusoidal reference signal $y(n)$.

16.12 *Computer Experiment.* Let $x(n) = x_1(n) + x_2(n)$, where $x_1(n)$ is a narrowband component defined by

$$x_1(n) = \sin(\omega_0 n + \phi), \quad \omega_0 = 0.075\pi \ \text{[rads/sample]}$$

where ϕ is a random phase uniformly distributed over $[0, 2\pi]$, and $x_2(n)$ is a fairly broadband component generated by sending zero-mean, unit-variance, white noise $\epsilon(n)$ through the filter

$$x_2(n) = \epsilon(n) + 2\epsilon(n-1) + \epsilon(n-2)$$

(a) Compute the autocorrelation functions of $x_1(n)$ and $x_2(n)$ and sketch them versus lag k. Based on this computation, select a value for the delay Δ to be used in the adaptive line enhancer discussed in Sec. 16.10.

(b) Generate a realization of $x(n)$ and process it through the ALE with an appropriately chosen adaptation parameter μ. Plot the output signals $\hat{x}(n)$ and $e(n)$, and compare them with the components $x_1(n)$ and $x_2(n)$, respectively.

16.13 The response of the ALE to an input sinusoid in noise can be studied as follows: Let the input be

$$x_n = A_1 e^{j\omega_1 n + j\phi} + v_n$$

where ϕ is a random phase independent of the zero-mean white noise v_n. The optimum Wiener filter weights of the ALE are given by

$$\mathbf{h} = R^{-1}\mathbf{r}$$

where $R_{ij} = R_{xx}(i-j)$ and $r_i = R_x(i+\Delta)$, as discussed in Sec. 16.10.

(a) Using the methods of Sec. 14.2, show that the optimum filter $\mathbf{h}$ is given by

$$\mathbf{h} = \frac{e^{j\omega_1\Delta}}{\dfrac{\sigma_v^2}{P_1} + M + 1}\mathbf{s}_{\omega_1}$$

where the phasing vector $\mathbf{s}_{\omega_1}$ was defined in Sec. 14.2, and $P_1 = |A_1|^2$ is the power of the sinusoid.

(b) Show that the mean output power of the ALE is given by

$$E\big[|\hat{x}_n|^2\big] = \mathbf{h}^\dagger R\mathbf{h} = \sigma_v^2\mathbf{h}^\dagger\mathbf{h} + P_1|\mathbf{h}^\dagger\mathbf{s}_{\omega_1}|^2$$

(c) Show that the SNR at the output is enhanced by a factor $M+1$ over the SNR at the input; that is, show that

$$(SNR)_{\text{out}} = \frac{P_1|\mathbf{h}^\dagger\mathbf{s}_{\omega_1}|^2}{\sigma_v^2\mathbf{h}^\dagger\mathbf{h}} = \frac{P_1}{\sigma_v^2}(M+1) = (M+1)(SNR)_{\text{in}}$$

(d) Derive an expression for the eigenvalue spread $\lambda_{\max}/\lambda_{\min}$ in terms of the parameters σ_v^2, P_1, and M.

(e) Show that if the delay Δ is removed; that is, $\Delta = 0$, then the optimal weight vector becomes equal to the unit vector

$$\mathbf{h} = [1, 0, 0, \dots, 0]^T$$

and that this choice corresponds to complete cancellation of the input signal $x(n)$ from the output $e(n)$.

16.14 *Computer Experiment.* Consider the autoregressive process y_n generated by the difference equation

$$y_n = -a_1 y_{n-1} - a_2 y_{n-2} + \epsilon_n$$

where $a_1 = -1.6$, $a_2 = 0.8$, and ϵ_n is zero-mean, unit-variance, white noise. Generate a realization of y_n and process it through the LMS adaptive predictor of order 2, as discussed in Sec. 16.11. Use a value for the adaptation parameter μ of your own choice. Plot the adaptive prediction coefficients $a_1(n)$ and $a_2(n)$ versus n, and compare their converged values with the theoretical values given above.

16.15 The adaptive predictor may be considered as the linearly constrained minimization problem $\mathcal{E} = E[e_n^2] = \min$, subject to the constraint that the first element of $\mathbf{a} = [1, a_1, \dots, a_M]^T$ be unity. This constraint may be written compactly as $\mathbf{u}^T\mathbf{a} = 1$, where $\mathbf{u} = [1, 0, \dots, 0]^T$. Rederive the adaptation equations of Sec. 16.11 using the formalism and results of Problem 16.2.

16.16 *Computer Experiment.* A complex-valued version of the LMS adaptive predictor of Sec. 16.11 is defined by

$$e_n = y_n + a_1(n)y_{n-1} + a_2(n)y_{n-2} + \cdots + a_M(n)y_{n-M}$$

$$a_m(n+1) = a_m(n) - 2\mu e_n y_{n-m}^*\,, \quad m = 1, 2, \dots, M$$

Let y_n consist of two complex sinusoids in zero-mean white noise

$$y_n = A_1 e^{j\omega_1 n} + A_2 e^{j\omega_2 n} + v_n$$

where the frequencies and the SNRs are

$$\omega_1 = 0.3\pi\,, \quad \omega_2 = 0.7\pi \;\text{[radians/sample]}$$

$$10\log_{10}\left[|A_1|^2/\sigma_v^2\right] = 10\log_{10}\left[|A_2|^2/\sigma_v^2\right] = 20 \text{ dB}$$

(a) Generate a realization of y_n (using a complex-valued v_n) and process it through an Mth order LMS adaptive predictor using an adaptation constant μ. Experiment with several choices of M and μ. In each case, stop the algorithm after convergence has taken place and plot the AR spectrum $S(\omega) = 1/|A(\omega)|^2$ versus frequency ω. Discuss your results.

(b) Using the same realization of y_n, iterate the adaptive Pisarenko algorithm defined by Eqs. (16.12.5) and (16.12.6). After convergence of the Pisarenko weights, plot the Pisarenko spectrum estimate $S(\omega) = 1/|A(\omega)|^2$ versus frequency ω.

(c) Repeat (a) and (b) when the SNR of the sinewaves is lowered to 0 dB. Compare the adaptive AR and Pisarenko methods.

16.17 *Computer Experiment.* Reproduce the results of Figs. 7.19 and 7.20.

16.18 Derive Eqs. (16.14.8) and (16.14.9) that describe the operation of the adaptive linear combiner in the decorrelated basis provided by the Gram-Schmidt preprocessor.

16.19 *Computer Experiment.* Reproduce the results of Fig. 16.14.2.

16.20 What is the exact operational count of the conventional RLS algorithm listed in Sec. 16.15? Note that the inverse matrices P_0 and P_1 are symmetric and thus only their lower-triangular parts need be updated.

16.21 Verify the solution (16.15.56) for the rank-one updating of the LU factors L_0 and L_1. Also verify that Eq. (16.15.58) is equivalent to (16.15.54).

16.22 *Computer Experiment.* Reproduce the results of Fig. 16.17.1. Carry out the same experiment (with the same input data) using the conventional RLS algorithm and compare with FAEST. Carry out both experiments with various values of λ and comment on the results.

16.23 *Computer Experiment.* Reproduce the results of Fig. 16.18.1.

17 Appendices

A Matrix Inversion Lemma

The matrix inversion lemma, also known as Woodbury's identity, is useful in Kalman filtering and recursive least-squares problems. Consider the matrix relationship,

$$\boxed{R = A + UBV} \tag{A.1}$$

where

$$A \in \mathbb{C}^{N\times N}\,, \quad U \in \mathbb{C}^{N\times M}\,, \quad B \in \mathbb{C}^{M\times M}\,, \quad V \in \mathbb{C}^{M\times N}$$

and assume that A, B are both *invertible* and that $M \le N$. Then, the term UBV has rank M, while R, A have rank N. The matrix inversion lemma states that the inverse of R can be obtained from the inverses of A, B via the formula,

$$\boxed{R^{-1} = (A + UBV)^{-1} = A^{-1} - A^{-1}U\left[B^{-1} + VA^{-1}U\right]^{-1}VA^{-1}} \tag{A.2}$$

Proof: Multiply both sides of (A.1) by R^{-1} from the right, and then by A^{-1} from the left to obtain,

$$A^{-1} = R^{-1} + A^{-1}UBVR^{-1} \tag{A.3}$$

then, multiply both sides from the left by V,

$$VA^{-1} = VR^{-1} + VA^{-1}UBVR^{-1} \quad \Rightarrow \quad VA^{-1} = \left[I_M + VA^{-1}UB\right]VR^{-1}$$

where I_M is the $M \times M$ identity matrix, and solve for BVR^{-1},

$$VA^{-1} = \left[B^{-1} + VA^{-1}U\right]BVR^{-1} \quad \Rightarrow \quad BVR^{-1} = \left[B^{-1} + VA^{-1}U\right]^{-1}VA^{-1}$$

and substitute back into (A.3), after solving for R^{-1},

$$R^{-1} = A^{-1} - A^{-1}UBVR^{-1} = A^{-1} - A^{-1}U\left[B^{-1} + VA^{-1}U\right]^{-1}VA^{-1}$$

Thus given A^{-1} and B^{-1}, the inverse of the $N \times N$ matrix R requires only the inverse of the smaller $M \times M$ matrix, $B^{-1} + VA^{-1}U$.