1

Introduction to adaptive filters

This chapter introduces the fundamental principles of adaptive filtering and commonly used adaptive filter structures and algorithms. Basic concepts of applying adaptive filters in practical applications are also highlighted. The problems and difficulties encountered in time-domain adaptive filters are addressed. Subband adaptive filtering is introduced to solve these problems. Some adaptive filters are implemented using MATLAB for different applications. These ready-to-run programs can be modified to speed up research and development in adaptive signal processing.

1.1 Adaptive filtering

A filter is designed and used to extract or enhance the desired information contained in a signal. An adaptive filter is a filter with an associated adaptive algorithm for updating filter coefficients so that the filter can be operated in an unknown and changing environment. The adaptive algorithm determines filter characteristics by adjusting filter coefficients (or tap weights) according to the signal conditions and performance criteria (or quality assessment). A typical performance criterion is based on an error signal, which is the difference between the filter output signal and a given reference (or desired) signal.

As shown in Figure 1.1, an adaptive filter is a digital filter with coefficients that are determined and updated by an adaptive algorithm. Therefore, the adaptive algorithm behaves like a human operator that has the ability to adapt in a changing environment. For example, a human operator can avoid a collision by examining the visual information (input signal) based on his/her past experience (desired or reference signal) and by using visual guidance (performance feedback signal) to direct the vehicle to a safe position (output signal).

Adaptive filtering finds practical applications in many diverse fields such as communications, radar, sonar, control, navigation, seismology, biomedical engineering and even in financial engineering [1–7]. In Section 1.6, we will introduce some typical applications using adaptive filtering.

This chapter also briefly introduces subband adaptive filters that are more computationally efficient for applications that need a longer filter length, and are more effective if

Subband Adaptive Filtering: Theory and Implementation Kong-Aik Lee, Woon-Seng Gan, Sen M. Kuo
© 2009 John Wiley & Sons, Ltd

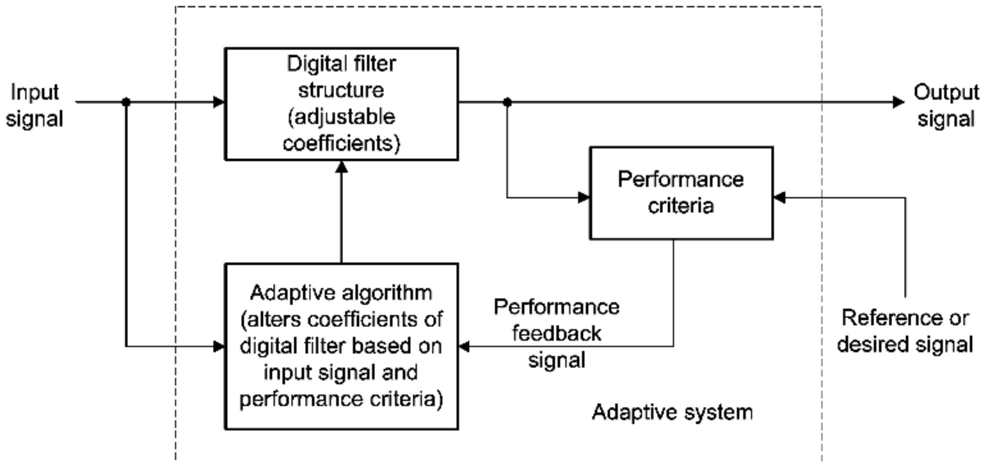
Input signal
Digital filter structure (adjustable coefficients)
Output signal
Performance criteria
Adaptive algorithm (alters coefficients of digital filter based on input signal and performance criteria)
Performance feedback signal
Reference or desired signal
Adaptive system

Figure 1.1 Basic functional blocks of an adaptive filter

the input signal is highly correlated. For example, high-order adaptive filters are required to cancel the acoustic echo in hands-free telephones [2]. The high-order filter together with a highly correlated input signal degrades the performances of most time-domain adaptive filters. Adaptive algorithms that are effective in dealing with ill-conditioning problems are available; however, such algorithms are usually computationally demanding, thereby limiting their use in many real-world applications.

In the following sections we introduce fundamental adaptive filtering concepts with an emphasis on widely used adaptive filter structures and algorithms. Several important properties on convergence rate and characteristics of input signals are also summarized. These problems motivated the development of subband adaptive filtering techniques in the forthcoming chapters. In-depth discussion on general adaptive signal processing can be found in many reference textbooks [1–3, 5–9].

1.2 Adaptive transversal filters

An adaptive filter is a self-designing and time-varying system that uses a recursive algorithm continuously to adjust its tap weights for operation in an unknown environment [6]. Figure 1.2 shows a typical structure of the adaptive filter, which consists of two basic functional blocks: (i) a digital filter to perform the desired filtering and (ii) an adaptive algorithm to adjust the tap weights of the filter. The digital filter computes the output $y(n)$ in response to the input signal $u(n)$, and generates an error signal $e(n)$ by comparing $y(n)$ with the desired response $d(n)$, which is also called the reference signal, as shown in Figure 1.1. The performance feedback signal $e(n)$ (also called the error signal) is used by the adaptive algorithm to adjust the tap weights of the digital filter.

The digital filter shown in Figure 1.2 can be realized using many different structures. The commonly used transversal or finite impulse response (FIR) filter is shown in

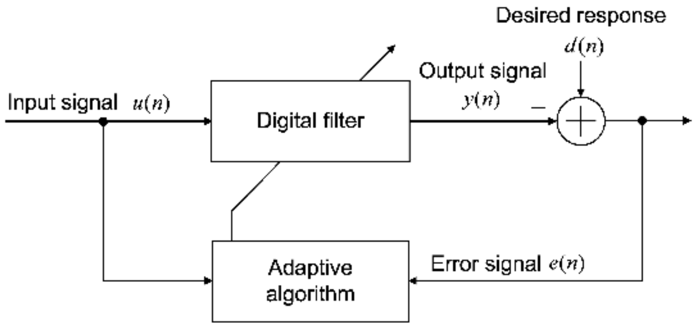

Figure 1.2 Typical structure of the adaptive filter using input and error signals to update its tap weights

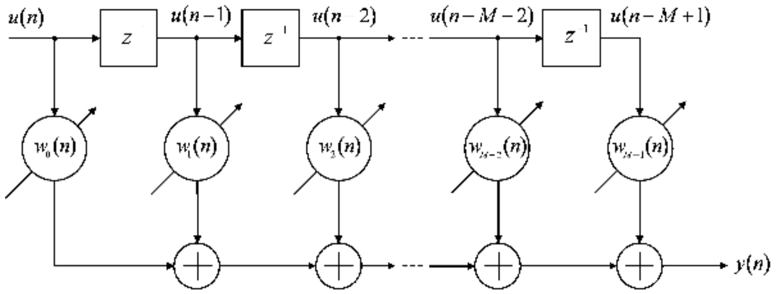

Figure 1.3 An M-tap adaptive transversal filter

Figure 1.3. The adjustable tap weights, $w_m(n)$, $m = 0, 1, \ldots, M-1$, indicated by circles with arrows through them, are the filter tap weights at time instance n and M is the filter length. These time-varying tap weights form an $M \times 1$ weight vector expressed as

$$\mathbf{w}(n) \equiv \left[w_0(n), w_1(n), \ldots, w_{M-1}(n)\right]^T, \tag{1.1}$$

where the superscript T denotes the transpose operation of the matrix. Similarly, the input signal samples, $u(n-m)$, $m = 0, 1, \ldots, M-1$, form an $M \times 1$ input vector

$$\mathbf{u}(n) \equiv [u(n), u(n-1), \ldots, u(n-M+1)]^T. \tag{1.2}$$

With these vectors, the output signal $y(n)$ of the adaptive FIR filter can be computed as the inner product of $\mathbf{w}(n)$ and $\mathbf{u}(n)$, expressed as

$$y(n) = \sum_{m=0}^{M-1} w_m(n)u(n-m) = \mathbf{w}^T(n)\mathbf{u}(n). \tag{1.3}$$

1.3 Performance surfaces

The error signal $e(n)$ shown in Figure 1.2 is the difference between the desired response $d(n)$ and the filter response $y(n)$, expressed as

$$e(n) = d(n) - \mathbf{w}^T(n)\mathbf{u}(n). \tag{1.4}$$

The weight vector $\mathbf{w}(n)$ is updated iteratively such that the error signal $e(n)$ is minimized. A commonly used performance criterion (or cost function) is the minimization of the mean-square error (MSE), which is defined as the expectation of the squared error as

$$J \equiv E\left\{e^2(n)\right\}. \tag{1.5}$$

For a given weight vector $\mathbf{w} = [w_0, w_1, \ldots, w_{M-1}]^T$ with stationary input signal $u(n)$ and desired response $d(n)$, the MSE can be calculated from Equations (1.4) and (1.5) as

$$J = E\left\{d^2(n)\right\} - 2\mathbf{p}^T\mathbf{w} + \mathbf{w}^T\mathbf{R}\mathbf{w}, \tag{1.6}$$

where $\mathbf{R} \equiv E\{\mathbf{u}(n)\mathbf{u}^T(n)\}$ is the input autocorrelation matrix and $\mathbf{p} \equiv E\{d(n)\mathbf{u}(n)\}$ is the cross-correlation vector between the desired response and the input vector. The time index n has been dropped in Equation (1.6) from the vector $\mathbf{w}(n)$ because the MSE is treated as a stationary function.

Equation (1.6) shows that the MSE is a quadratic function of the tap weights $\{w_0, w_1, \ldots, w_{M-1}\}$ since they appear in first and second degrees only. A typical performance (or error) surface for a two-tap transversal filter is shown in Figure 1.4. The corresponding MATLAB script for computing this performance surface is given in Example 1.1. For $M > 2$, the error surface is a hyperboloid. The quadratic performance surface guarantees that it has a single global minimum MSE corresponding to the

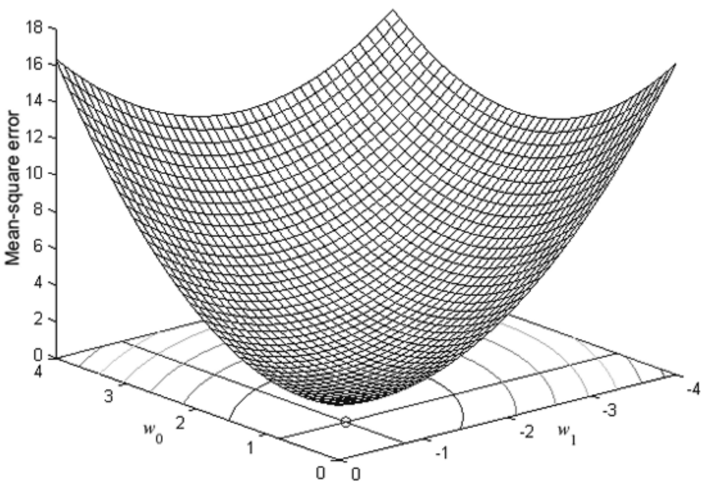

Figure 1.4 A typical performance surface for a two-tap adaptive FIR filter

optimum vector $\mathbf{w}_o$. The optimum solution can be obtained by taking the first derivative of Equation (1.6) with respect to $\mathbf{w}$ and setting the derivative to zero. This results in the Wiener–Hopf equation

$$\mathbf{R}\mathbf{w}_o = \mathbf{p}. \tag{1.7}$$

Assuming that $\mathbf{R}$ has an inverse matrix, the optimum weight vector is

$$\mathbf{w}_o = \mathbf{R}^{-1}\mathbf{p}. \tag{1.8}$$

Substituting Equation (1.8) into (1.6), the minimum MSE corresponding to the optimum weight vector can be obtained as

$$J_{\min} = E\left\{d^2(n)\right\} - \mathbf{p}^T\mathbf{w}_o. \tag{1.9}$$

Example 1.1 (a complete listing is given in the M-file `Example_1P1.m`)
Consider a two-tap case for the FIR filter, as shown in Figure 1.3. Assuming that the input signal and desired response are stationary with the second-order statistics

$$\mathbf{R} = \begin{bmatrix} 1.1 & 0.5 \\ 0.5 & 1.1 \end{bmatrix}, \mathbf{p} = \begin{bmatrix} 0.5272 \\ -0.4458 \end{bmatrix} \text{ and } E\{d^2(n)\} = 3.$$

The performance surface (depicted in Figure 1.4) is defined by Equation (1.6) and can be constructed by computing the MSE at different values of w_0 and w_1, as demonstrated by the following partial listing:

```
R = [1.1 0.5; 0.5 1.1];     % Input autocorrelation matrix
p = [0.5272; -0.4458];      % Cross-correlation vector
dn_var = 3;                 % Variance of desired response d(n)

w0 = 0:0.1:4;               % Range of first tap weight values
w1 = -4:0.1:0;              % Range of second tap weight values
J = zeros(length(w0),length(w1));
                            % Clear MSE values at all (w0, w1)
for m = 1:length(w0)
    for n = 1:length(w1)
        w = [w0(m) w1(n)]';
        J(m,n) = dn_var-2*p'*w + w'*R*w;
                            % Compute MSE values at all (w0, w1)
    end                     %  points, store in J matrix
end

figure; meshc(w0,w1,J');    % Plot combination of mesh and contour
view([-130 22]);            % Set the angle to view 3-D plot
hold on; w_opt = inv(R)*p;
plot(w_opt(1),w_opt(2),'o');
plot(w0,w_opt(2)*ones(length(w0),1));
plot(w_opt(1)*ones(length(w1),1),w1);
xlabel('w_0'); ylabel('w_1');
plot(w0,-4*ones(length(w0),1)); plot(4*ones(length(w1),1),w1);
```

1.4 Adaptive algorithms

An adaptive algorithm is a set of recursive equations used to adjust the weight vector $\mathbf{w}(n)$ automatically to minimize the error signal $e(n)$ such that the weight vector converges iteratively to the optimum solution $\mathbf{w}_o$ that corresponds to the bottom of the performance surface, i.e. the minimum MSE $J_{\min}$. The least-mean-square (LMS) algorithm is the most widely used among various adaptive algorithms because of its simplicity and robustness. The LMS algorithm based on the steepest-descent method using the negative gradient of the instantaneous squared error, i.e. $J \approx e^2(n)$, was devised by Widrow and Stearns [6] to study the pattern-recognition machine. The LMS algorithm updates the weight vector as follows:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\mathbf{u}(n)e(n), \tag{1.10}$$

where μ is the step size (or convergence factor) that determines the stability and the convergence rate of the algorithm. The function M-files of implementing the LMS algorithm are `LMSinit.m` and `LMSadapt.m`, and the script M-file that calls these MATLAB functions for demonstration of the LMS algorithm is `LMSdemo.m`. Partial listing of these files is shown below. The complete function M-files of the adaptive LMS algorithm and other adaptive algorithms discussed in this chapter are available in the companion CD.

The LMS algorithm

There are two MATLAB functions used to implement the adaptive FIR filter with the LMS algorithm. The first function `LMSinit` performs the initialization of the LMS algorithm, where the adaptive weight vector `w0` is normally clear to a zero vector at the start of simulation. The step size `mu` and the leaky factor `leak` can be determined by the user at run time (see the complete `LMSdemo.m` for details). The second function `LMSadapt` performs the actual computation of the LMS algorithm given in Equation (1.10), where `un` and `dn` are the input and desired signal vectors, respectively. `S` is the initial state of the LMS algorithm, which is obtained from the output argument of the first function. The output arguments of the second function consist of the adaptive filter output sequence `yn`, the error sequence `en` and the final filter state `S`.

```
S = LMSinit(zeros(M,1),mu);     % Initialization
S.unknownsys = b;
```

```
[yn,en,S] = LMSadapt(un,dn,S);  % Perform LMS algorithm
...

function S = LMSinit(w0,mu,leak)
% Assign structure fields
S.coeffs        = w0(:);        % Weight (column) vector of filter
S.step          = mu;           % Step size of LMS algorithm
S.leakage       = leak;         % Leaky factor for leaky LMS
S.iter          = 0;            % Iteration count
S.AdaptStart    = length(w0);   % Running effect of adaptive
                                %  filter, minimum M

function [yn,en,S] = LMSadapt(un,dn,S)
M = length(S.coeffs);           % Length of FIR filter
mu = S.step;                    % Step size of LMS algorithm
leak = S.leakage;               % Leaky factor
AdaptStart = S.AdaptStart;
w = S.coeffs;                   % Weight vector of FIR filter
u = zeros(M,1);                 % Input signal vector
%
ITER = length(un);              % Length of input sequence
yn = zeros(1,ITER);             % Initialize output vector to zero
en = zeros(1,ITER);             % Initialize error vector to zero
%
for n = 1:ITER
    u = [un(n); u(1:end-1)];    % Input signal vector contains
                                %  [u(n),u(n-1),...,u(n-M+1)]'
    yn(n) = w'*u;               % Output signal
    en(n) = dn(n) - yn(n);      % Estimation error
    if ComputeEML == 1;
        eml(n) = norm(b-w)/norm_b;
                                % System error norm (normalized)
    end
    if n >= AdaptStart
        w = (1-mu*leak)*w + (mu*en(n))*u; % Tap-weight adaptation
        S.iter = S.iter + 1;
    end
end
```

As shown in Equation (1.10), the LMS algorithm uses an iterative approach to adapt the tap weights to the optimum Wiener–Hopf solution given in Equation (1.8). To guarantee the stability of the algorithm, the step size is chosen in the range

$$0 < \mu < \frac{2}{\lambda_{\max}}, \tag{1.11}$$

where $\lambda_{\max}$ is the largest eigenvalue of the input autocorrelation matrix **R**. However, the eigenvalues of **R** are usually not known in practice so the sum of the eigenvalues (or the trace of $\mathbf{R}$) is used to replace $\lambda_{\max}$. Therefore, the step size is in the range of $0 < \mu < 2/\mathrm{trace}(\mathbf{R})$. Since $\mathrm{trace}(\mathbf{R}) = MP_u$ is related to the average power P_u of the input signal $u(n)$, a commonly used step size bound is obtained as

$$0 < \mu < \frac{2}{MP_u}. \tag{1.12}$$

It has been shown that the stability of the algorithm requires a more stringent condition on the upper bound of μ when convergence of the weight variance is imposed [3]. For Gaussian signals, convergence of the MSE requires $0 < \mu < 2/3MP_u$.

The upper bound on μ provides an important guide in the selection of a suitable step size for the LMS algorithm. As shown in (1.12), a smaller step size μ is used to prevent instability for a larger filter length M. Also, the step size is inversely proportional to the input signal power P_u. Therefore, a stronger signal must use a smaller step size, while a weaker signal can use a larger step size. This relationship can be incorporated into the LMS algorithm by normalizing the step size with respect to the input signal power. This normalization of step size (or input signal) leads to a useful variant of the LMS algorithm known as the normalized LMS (NLMS) algorithm.

The NLMS algorithm [3] includes an additional normalization term $\mathbf{u}^T(n)\mathbf{u}(n)$, as shown in the following equation:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \frac{\mathbf{u}(n)}{\mathbf{u}^T(n)\mathbf{u}(n)} e(n), \tag{1.13}$$

where the step size is now bounded in the range $0 < \mu < 2$. It makes the convergence rate independent of signal power by normalizing the input vector $\mathbf{u}(n)$ with the energy $\mathbf{u}^T(n)\mathbf{u}(n) = \sum_{m=0}^{M-1} u^2(n-m)$ of the input signal in the adaptive filter. There is no significant difference between the convergence performance of the LMS and NLMS algorithms for stationary signals when the step size μ of the LMS algorithm is properly chosen. The advantage of the NLMS algorithm only becomes apparent for nonstationary signals like speech, where significantly faster convergence can be achieved for the same level of MSE in the steady state after the algorithm has converged. The function M-files for the NLMS algorithm are `NLMSadapt.m` and `NLMSadapt.m`. The script M-file that calls these MATLAB functions for demonstration is `NLMSdemo.m`. Partial listing of these files is shown below.

The NLMS algorithm

Most parameters of the NLMS algorithm are the same as the LMS algorithm, except that the step size `mu` is now bounded between 0 and 2. The normalization term, `u'*u + alpha`, makes the convergence rate independent of signal power. An additional constant `alpha` is used to avoid division by zero in normalization operations.

```
...
S = NLMSinit(zeros(M,1),mu);    % Initialization
S.unknownsys = b;
```

```
[yn,en,S] = NLMSadapt(un,dn,S); % Perform NLMS algorithm
...

function [yn,en,S] = NLMSadapt(un,dn,S)
M = length(S.coeffs);           % Length of FIR filter
mu = S.step;                    % Step size (between 0 and 2)
leak = S.leakage;               % Leaky factor
alpha = S.alpha;                % A small constant
AdaptStart = S.AdaptStart;
w = S.coeffs;                   % Weight vector of FIR filter
u = zeros(M,1);                 % Input signal vector
%
ITER = length(un);              % Length of input sequence
yn = zeros(1,ITER);             % Initialize output vector to zero
en = zeros(1,ITER);             % Initialize error vector to zero
...
for n = 1:ITER
    u = [un(n); u(1:end-1)];    % Input signal vector
    yn(n) = w'*u;               % Output signal
    en(n) = dn(n) - yn(n);      % Estimation error
    if ComputeEML == 1;
        eml(n) = norm(b-w)/norm_b;
    end                         % System error norm (normalized)
    if n >= AdaptStart
        w = (1-mu*leak)*w + (mu*en(n)/(u'*u + alpha))*u;
                                % Tap-weight adaptation
        S.iter = S.iter + 1;
    end
end
```

Assuming that all the signals and tap weights are real valued, the FIR filter requires M multiplications to produce the output $y(n)$ and the update equation (1.10) requires $(M+1)$ multiplications. Therefore, a total of $(2M+1)$ multiplications per iteration are required for the adaptive FIR filter with the LMS algorithm. On the other hand, the NLMS algorithm requires additional $(M+1)$ multiplications for the normalization term, giving a total of $(3M+2)$ multiplications per iteration. Since M is generally large for most practical applications, the computational complexity of the LMS and NLMS algorithms is proportional to M, denoted as $O(M)$. Note that the normalization term $\mathbf{u}^T(n)\mathbf{u}(n)$ in Equation (1.13) can be approximated from the average power P_u, which can be recursively estimated at time n by a simple running average as

$$P_u(n) = (1-\beta)P_u(n-1) + \beta u^2(n), \tag{1.14}$$

where $0 < \beta \ll 1$ is the smoothing (or forgetting) parameter.

The affine projection (AP) algorithm [3] is a generalized version of the NLMS algorithm. Instead of minimizing the current error signal given in Equation (1.4), the AP algorithm increases the convergence rate of the NLMS algorithm by using a set of P constraints $d(n-k) = \mathbf{w}^T(n+1)\mathbf{u}(n-k)$ for $k = 0, 1, \ldots, P-1$. Note that P is less than the length M used in the adaptive FIR filter and results in an underdetermined case. Therefore, a pseudo-inversion of the input signal matrix is used to derive the AP algorithm as follows:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \mathbf{A}^T(n)\left[\mathbf{A}(n)\mathbf{A}^T(n)\right]^{-1}\mathbf{e}(n), \tag{1.15}$$

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{A}(n)\mathbf{w}(n), \tag{1.16}$$

where the input signal matrix $\mathbf{A}^T(n) = [\mathbf{u}(n), \mathbf{u}(n-1), \ldots, \mathbf{u}(n-P+1)]$ consists of P columns of input vectors of length M, $\mathbf{e}(n) = [e(n), e(n-1), \ldots, e(n-P+1)]^T$ is the error signal vector and $\mathbf{d}(n) = [d(n), d(n-1), \ldots, d(n-P+1)]^T$ is the desired signal vector. The computation of error signals given in Equation (1.16) is based on P constraints (or order) of the AP algorithm. More constraints (i.e. larger P) results in faster convergence but at the cost of higher complexity. When P reduces to one (i.e. a single column), the updating equation in (1.15) becomes a special case of the NLMS algorithm as shown in Equation (1.13). The computational complexity of the AP algorithm is $O(P^2M)$. The function M-files of the AP algorithm are `APinit.m` and `APadapt.m`, and the script M-file to demonstrate the AP algorithm is `APdemo.m`. The partial listing of these files is shown below.

The AP algorithm

The AP algorithm is a generalized version of the NLMS algorithm with `P` constraints and `M` taps of adaptive filter. In the MATLAB program, `P` $= 10$ and `M` $= 256$ are used in simulation. The normalization term, `A*inv(A'*A + alpha)`, is the pseudo-inversion of the data matrix `A`. The identity matrix `alpha` consists of a small constant of 10^{-4} along its diagonal elements to avoid division by zero in the matrix-inversion operations.

```
...
P = 10; M = 256;
S = APinit(zeros(M,1),mu,P);    % Initialization
S.unknownsys = b;
[yn,en,S] = APadapt(un,dn,S);   % Perform AP algorithm
...

function S = APinit(w0,mu,P)
% Assign structure fields
S.coeffs        = w0(:);        % Weight (column) vector of filter
S.step          = mu;           % Step size
S.iter          = 0;            % Iteration count
S.alpha         = eye(P,P)*1e-4;% A small constant
S.AdaptStart    = length(w0)+P-1;% Running effect of adaptive
                                %   filter and projection matrix
S.order         = P;            % Projection order

function [yn,en,S] = APadapt(un,dn,S)
```

```
M = length(S.coeffs);           % Length of FIR filter
mu = S.step;                    % Step size (between 0 and 2)
P = S.order;                    % Projection order
alpha = S.alpha;                % Small constant
AdaptStart = S.AdaptStart;
w = S.coeffs;                   % Weight vector of FIR filter
u = zeros(M,1);                 % Tapped-input vector
A = zeros(M,P);                 % Projection matrix
d = zeros(P,1);                 % Desired response vector
%
ITER = length(un);              % Length of input sequence
yn = zeros(1,ITER);             % Initialize output vector to zero
en = zeros(1,ITER);             % Initialize error vector to zero
...
for n = 1:ITER
    u = [un(n); u(1:end-1)];    % Input signal vector contains
                                %  [u(n),u(n-1),...,u(n-M+1)]'
    A = [u A(:,1:end-1)];       % Data matrix
    d = [dn(n); d(1:end-1)];    % Desired response vector contains
                                %  [d(n),d(n-1),...,d(n-P+1)]'
    yn(n) = u'*w;               % Output signal
    e = d - A'*w;               % Error estimation
    en(n) = e(1);               % Take the first element of e
    if ComputeEML == 1;
        eml(n) = norm(b-w)/norm_b;
                                % System error norm (normalized)
    end
   if n >= AdaptStart
        w = w + mu*A*inv(A'*A + alpha)*e;
                                % Tap-weight adaptation
    end
end
```

Unlike the NLMS algorithm that is derived from the minimization of the expectation of squared error, the recursive least-square (RLS) [3] algorithm is derived from the minimization of the sum of weighted least-square errors as

$$J_{\mathrm{LS}}(n) = \sum_{i=1}^{n} \lambda^{n-i} e^2(i), \tag{1.17}$$

where λ is the forgetting factor and has a value less than and close to 1. The forgetting factor weights the current error heavier than the past error values to support filter operation in nonstationary environments. Therefore, in the least-square method, the weight vector $\mathbf{w}(n)$ is optimized based on the observation starting from the first iteration ($i = 1$) to the current time ($i = n$). The least-square approach can also be expressed in the form similar to the Wiener–Hopf equation defined in Equation (1.7), where the autocorrelation matrix and cross-correlation vector are expressed as $\mathbf{R} \approx \mathbf{R}(n) = \sum_{i=1}^{n} \lambda^{n-i}\mathbf{u}(i)\mathbf{u}^T(i)$ and $\mathbf{p} \approx \mathbf{p}(n) = \sum_{i=1}^{n} \lambda^{n-i} d(i)\mathbf{u}(i)$, respectively.

Following the derivation given in Reference [3] using the matrix inversion lemma, the RLS algorithm can be written as

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{g}(n)e(n), \tag{1.18}$$

where the updating gain vector is defined as

$$\mathbf{g}(n) = \frac{\mathbf{r}(n)}{1 + \mathbf{u}^T(n)\mathbf{r}(n)} \tag{1.19}$$

and

$$\mathbf{r}(n) = \lambda^{-1}\mathbf{P}(n-1)\mathbf{u}(n). \tag{1.20}$$

The inverse correlation matrix of input data, $\mathbf{P}(n) \equiv \mathbf{R}^{-1}(n)$, can be computed recursively as

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{r}^T(n). \tag{1.21}$$

Note that the AP algorithm approaches the RLS algorithm when P increases to M and $\lambda = 1$. Since both the NLMS and RLS algorithms converge to the same optimum weight vector, there is a strong link between them. Montazeri and Duhamel [10] linked the set of adaptive algorithms including the NLMS, AP and RLS algorithms.

The computational complexity of the RLS algorithm is in the order of $O(M^2)$. There are several efficient versions of the RLS algorithm, such as the fast transversal filter with the reduced complexity to $O(M)$ [3]. Compared to the NLMS and AP algorithms, the RLS algorithm is more expensive to implement. The AP algorithm has a complexity that lies between the NLMS (for $P = 1$) and RLS (for $P = M$) algorithms. The function M-files of the RLS algorithm are `RLSinit.m` and `RLSadapt.m`, and the script M-file to demonstrate the RLS algorithm is `RLSdemo.m`. Partial listing of these files is shown below.

The RLS algorithm

The RLS algorithm whitens the input signal using the inverse of the input autocorrelation matrix. Its convergence speed is faster than the LMS, NLMS and AP algorithms under colored input signals. This MATLAB simulation uses the forgetting factor `lamda = 1` and the initial inverse input correlation matrix `P0 = eye(length(w0))*0.01`. The RLS algorithm updates the filter coefficients according to Equations (1.18) to (1.21).

```
...
S = RLSinit(zeros(M,1),lamda);  % Initialization
```

```
S.unknownsys = b;
[yn,en,S] = RLSadapt(un,dn,S);  % Perform RLS algorithm
...

function S = RLSinit(w0,lamda)
% Assign structure fields
S.coeffs        = w0(:);        % Weight (column) vector of filter
S.lamda         = lamda;        % Exponential weighting factor
S.iter          = 0;            % Iteration count
S.P0            = eye(length(w0))*0.01; % Initialize inverse input
                                %  autocorrelation matrix
S.AdaptStart    = length(w0);   % Running effect

function [yn,en,S] = APadapt(un,dn,S)
M = length(S.coeffs);           % Length of FIR filter
lamda = S.lamda;                % Exponential factor
invlamda = 1/lamda;
AdaptStart = S.AdaptStart;
w = S.coeffs;                   % Weight vector of FIR filter
P = S.P0;                       % Inverse correlation matrix
u = zeros(M,1);                 % Input signal vector
...
for n = 1:ITER
    u = [un(n); u(1:end-1)];    % Input signal vector contains
                                % [u(n),u(n-1),...,u(n-M+1)]';
    yn(n) = w'*u;               % Output of adaptive filter
    en(n)=dn(n)-yn(n);          % Estimated error
    if n >= AdaptStart
        r = invlamda*P*u;       % See equ. (1.20)
        g = r/(1+u'*r);         % See equ. (1.19)
        w = w + g.*en(n);       % Updated tap weights for the
                                %  next cycle, (1.18)
        P = invlamda*(P-g*u'*P);% Inverse correlation matrix
                                %  update, see(1.20) and (1.21)
    end
    if ComputeEML == 1;
        eml(n) = norm(b-w)/norm_b;
                                % System error norm (normalized)
    end
end
```

1.5 Spectral dynamic range and misadjustment

The convergence behavior of the LMS algorithm is related to the eigenvalue spread of the autocorrelation matrix **R** that is determined by the characteristics of the input signal $u(n)$. The eigenvalue spread is measured by the condition number defined as $\kappa(\mathbf{R}) = \lambda_{\max}/\lambda_{\min}$, where $\lambda_{\max}$ and $\lambda_{\min}$ are the maximum and minimum eigenvalues, respectively. Furthermore, the condition number depends on the spectral distribution of the input signal [1, p. 97] and is bounded by the dynamic range of the spectrum $\Gamma_{uu}(e^{j\omega})$ as follows:

$$\kappa(\mathbf{R}) \leq \frac{\max\limits_{\omega} \Gamma_{uu}\left(e^{j\omega}\right)}{\min\limits_{\omega} \Gamma_{uu}\left(e^{j\omega}\right)}, \tag{1.22}$$

where $\max\limits_{\omega} \Gamma_{uu}(e^{j\omega})$ and $\min\limits_{\omega} \Gamma_{uu}(e^{j\omega})$ are the maximum and minimum of the input spectrum, respectively. The ideal condition $\kappa(\mathbf{R}) = 1$ occurs for white input signals. The convergence speed of the LMS algorithm decreases as the ratio of the spectral dynamic range increases. When the input signal is highly correlated with a large spectral dynamic range, the autocorrelation matrix $\mathbf{R}$ is ill-conditioned due to the large eigenvalue spread and the LMS algorithm suffers from slow convergence. The deficiency of LMS and NLMS algorithms has been analyzed in the literature [1, 3, 5, 6, 9]. Subband adaptive filtering will be introduced in Section 1.7.2 to improve the convergence rate when the input signal is highly correlated.

The theoretical convergence curve of the LMS algorithm decays according to the factors $r_m = 1 - \mu\lambda_m$, where λ_m is the mth eigenvalue of $\mathbf{R}$ [6]. Therefore, a time constant that defines the convergence rate of the MSE is obtained as

$$\tau_m = \frac{1}{2\mu\lambda_m}, \quad m = 0, 1, \ldots, M-1. \tag{1.23}$$

Consequently, the slowest convergence is determined by the smallest eigenvalue $\lambda_{\min}$. Even though a large step size can result in faster convergence, it must be limited by the upper bound given in Equation (1.11), which is inversely proportional to the maximum eigenvalue $\lambda_{\max}$.

In practice, the weight vector $\mathbf{w}(n)$ deviates from the optimum weight vector $\mathbf{w}_{\mathrm{o}}$ in the steady state due to the use of the instantaneous squared error by the LMS algorithm. Therefore, the MSE in the steady state after the algorithm has converged is greater than the minimum MSE, $J_{\min}$. The difference between the steady-state MSE $J(\infty)$ and $J_{\min}$ is defined as the excess MSE expressed as $J_{\mathrm{ex}} = J(\infty) - J_{\min}$. In addition, the misadjustment is defined as

$$\mathcal{M} = \frac{J_{\mathrm{ex}}}{J_{\min}} = \frac{\mu}{2} M P_u. \tag{1.24}$$

This equation shows that the misadjustment is proportional to the step size, thus resulting in a tradeoff between the misadjustment and the convergence rate given in Equation (1.23). A small step size results in a slow convergence, but has the advantage of smaller misadjustment, and vice versa. If all the eigenvalues are equal, Equations (1.23) and (1.24) are related by the following simple equation:

$$\mathcal{M} = \frac{M}{4 \times \tau_m} = \frac{M}{\text{settling time}}, \tag{1.25}$$

where the settling time is defined as four times the convergence rate τ_m. Therefore, a long filter requires a long settling time in order to achieve a small value of misadjustment, which is normally expressed as a percentage and has a typical value of 10 % for most applications.

1.6 Applications of adaptive filters

Adaptive filters are used in many applications because of their ability to operate in unknown and changing environments. Adaptive filtering applications can be classified in four categories: adaptive system identification, adaptive inverse modeling, adaptive prediction and adaptive array processing.

In this section, we introduce several applications of adaptive filtering. These applications are implemented in MATLAB code for computer simulation. MATLAB examples include acoustic echo cancellation, adaptive interference cancellation, adaptive line enhancer, active noise control, adaptive channel equalizer, acoustic feedback reduction for hearing aids and adaptive array processing. These applications provide a useful platform to evaluate the performance of different adaptive algorithms. Several plots include an ensemble or time-averaged square error, and the norm of weight error, echo return loss, spectrogram of error signal and magnitude-squared coherence, etc., are used to evaluate the performance of these algorithms. The complete MATLAB programs for these adaptive filtering applications are available in the companion CD.

1.6.1 Adaptive system identification

A structure of adaptive system identification (or modeling) is shown in Figure 1.5, where the adaptive filter is placed in parallel with an unknown system (or plant) to be identified. The adaptive filter provides a linear model that is the best approximation of the unknown system. The excitation signal $u(n)$ serves as the input to both the unknown system and the adaptive filter, while $\eta(n)$ represents the disturbance (or plant noise) occurring within the unknown system. The objective of the adaptive filter is to model the unknown system such that the adaptive filter output $y(n)$ closely matches the unknown system output $d(n)$. This can be achieved by minimizing the error signal $e(n)$, which is the difference between the physical response $d(n)$ and the model response $y(n)$. If the excitation signal

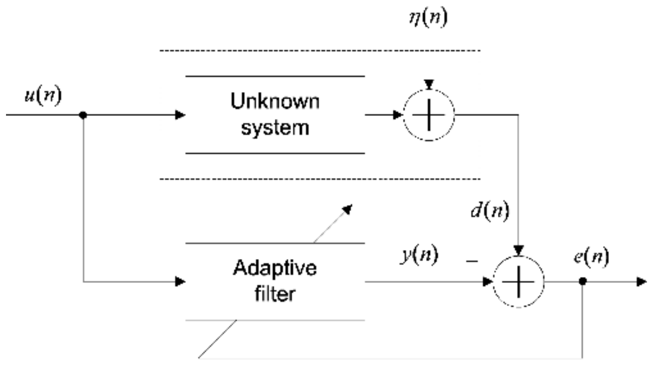

Figure 1.5 Block diagram of adaptive system identification

$u(n)$ is rich in frequency content, such as a white noise, and the plant noise $\eta(n)$ is small and uncorrelated with $u(n)$, the adaptive filter identifies the unknown system from the input–output viewpoint. The script M-file for the adaptive system identification is `SYSID.m` and its partial listing is shown below.

Adaptive system identification

The application of adaptive system identification is illustrated in the MATLAB program, where the plant to be modeled is a 32-tap lowpass FIR filter with a cutoff frequency of 0.3 radians per sample. The adaptive weight vector `w0` containing 32 coefficients and the white Gaussian noise are used as input `un` to both the plant and the adaptive filter. The desired signal `dn` is the plant's output. The following listing uses only the LMS algorithm to update the adaptive filter. Other algorithms can also be used in order to compare their convergence performance with the LMS algorithm.

```
% SYSID.m     System Identification as Shown in Fig.1.5
iter = 1024;
un  = 0.1*randn(1,iter);     % Input signal u(n)
b   = fir1(31,0.3)';         % Unknown FIR system
dn  = filter(b,1,un);        % Desired signal d(n)
w0 = zeros(32,1);            % Initialize filter coefficients to 0

% LMS algorithm

mulms = 0.5;                 % Step size
...
Slms = LMSinit(w0,mulms);    % Initialization
Slms.unknownsys = b;
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
                             % Perform LMS algorithm
```

The adaptive system identification has been widely applied in echo cancellation [2], active noise control [4], digital control [7], geophysics [6] and communications [5]. In fact, many problems can be formulated from the adaptive system identification point of view. A typical example is the acoustic echo cancellation (AEC), which finds many applications in hands-free telephones, audio (or video) conferencing systems, hearing aids, voice-control systems and much more. This book uses the adaptive identification of unknown systems with a long impulse response in later chapters to illustrate the performance of subband adaptive filters.

1.6.1.1 Acoustic echo cancellation

The acoustic echo in hands-free telephones arises when the microphone picks up the sound radiated by the loudspeaker and its reflections in a room [2, 11]. Figure 1.6 shows a general AEC configuration for hands-free telephony systems. Speech originating from the far-end talker is amplified and reproduced by a loudspeaker. The output of the loudspeaker

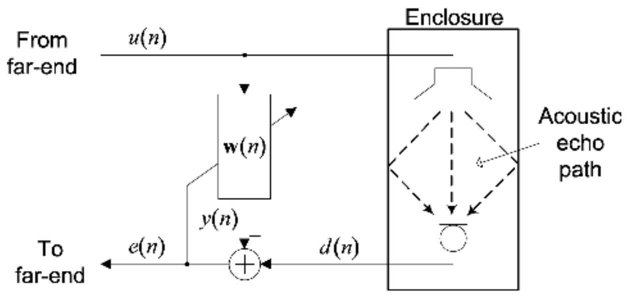

Figure 1.6 Block diagram of an acoustic echo canceller

will be picked up by the microphone via the acoustic echo path in the room. This signal is transmitted back to the far-end user where it is perceived as acoustic echo. The far-end user is annoyed by listening to his/her own speech delayed by the round-trip traveling time of the system.

The acoustic echo can be canceled electronically using an adaptive filter, as illustrated in Figure 1.6. The adaptive filter models the acoustic echo path between the loudspeaker and microphone. The estimated transfer function is used to filter the far-end speech signal $u(n)$ to generate an echo replica $y(n)$, which is then used to cancel the acoustic echo components in the microphone signal $d(n)$, resulting in the echo-free signal $e(n)$. The script M-file of the acoustic echo cancellation application is `AEC.m` and its partial listing is shown below.

Acoustic echo cancellation

Acoustic echo cancellation is an example of the adaptive system identification. The far-end speech signal `un` is used as input to both the room transfer function `ho` and the 1024-tap adaptive filter `w0`. No near-end speech signal is used in this simulation; instead, a room noise `vn` is added to the output of the room transfer function `out` to form the desired signal `dn`. An extension to this simulation adds near-end speech and investigates methods to handle double-talk conditions. In the partial listing, only the LMS algorithm is shown to update the adaptive filter. Other algorithms can also be used to compare their convergence performance and the system error norm with the LMS algorithm. Echo return loss enhancement is also computed in this script M-file.

```
% AEC.m    Application Program to Test Acoustics Echo Cancellation
%          as Shown in Fig.1.6
% Input far-end signal
[far_end,fsf,nbitf] = wavread('far.wav');
                                     % Speech signal from far end
far_end = far_end(1:end);
...
un = far_end';                       % Input signal
```

```
% Input near-end signal (room noise)
vn = 0.01.*randn(1,length(far_end)); % Additive room noise
                                     %  (variance ~ 0.01^2)
near_end = vn';                      % No near-end speech
                                     %  (Single-talk)
...
% Load impulse response of room transfer function (300 ms)
load('h1.dat'); ho = h1(1:1024);     % Truncate impulse response
                                     %  to 1024 samples
out = filter(ho,1,far_end');         % Filtering of far-end signal
                                     %  with room impulse response
dn = out+near_end';

% Perform adaptive filtering using LMS algorithm
mulms = 0.01;                        % Step size mu
...
Slms = LMSinit(w0,mulms);            % Initialization
Slms.unknownsys = ho;
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
                                     % Perform LMS algorithm
```

The following three factors make the AEC a very difficult task:

(i) The acoustic echo path usually has long impulse response. The characteristics of the acoustic path depend on the physical conditions of the room. For a typical office, the duration of the impulse response (i.e. the reverberation time) is about 300 ms. At 8 kHz sampling frequency, an adaptive FIR filter of 2400 taps is required to model the acoustic echo path. As discussed in Section 1.4, a long adaptive FIR filter incurs a heavy computational load and results in slower convergence because of small step sizes, as indicated in (1.12).

(ii) The characteristics of the acoustic echo path are time varying and may change rapidly due to the movement of people, doors opening or closing, temperature changes in the room, etc. Therefore, an adaptive algorithm has to converge quickly to track any changes by continuously updating the adaptive filter.

(iii) The excitation speech signal is highly correlated and nonstationary. For a short speech segment, the spectral density may differ by more than 40 dB at different frequencies. This also results in slow convergence, as shown in Equation (1.22).

Many techniques have been proposed to overcome these problems [2, 3, 8, 11]. In this book, we will focus on using subband adaptive filters to tackle these problems.

1.6.1.2 Active noise control

A single-channel active noise control (ANC) for suppressing acoustic noise in an air duct is shown in Figure 1.7 [4]. The input signal $u(n)$ is acquired by a reference microphone placed near the noise source. The primary noise travels to the error microphone via

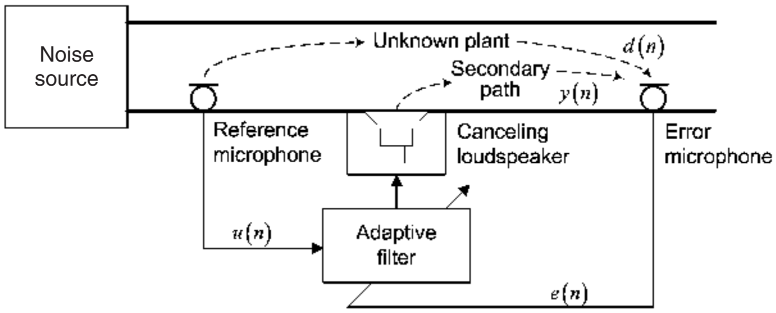

Figure 1.7 An example of single-channel active noise control viewed as an adaptive system identification problem

an unknown plant. At the same time, the canceling loudspeaker generates a secondary canceling noise that propagates to the error microphone via the secondary path. This secondary noise has the same amplitude with opposite phase to cancel the primary noise based on the principle of acoustic superposition. The residual noise picked up by the error microphone is used as the error signal $e(n)$ to update the adaptive filter. The ANC system operates in the electrical domain, while the unknown plant and secondary path are located in the electroacoustic domain. ANC is a challenging application since there are many considerations such as secondary path modeling and compensation, fast changing environment and noise characteristics, interactions between acoustic and electrical domains, placement of microphones and loudspeakers, and undesired acoustic feedback from the canceling loudspeaker to the reference microphone. The script M-file of the ANC application is `ANC.m` and its partial listing is shown below.

Active noise control

The application of active noise control is illustrated in MATLAB, where the engine wave file `engine_2.wav` is used as the source noise. The primary path is modeled as a rational transfer function with denominator `pri_den` and numerator `pri_num`. Similarly, the secondary path is modeled by denominator `sec_den` and numerator `sec_num`. A primary difference between the active noise control and other adaptive noise cancellation techniques is that the active noise control system does not generate an error signal electrically; instead, the error signal `en` is measured by the error microphone. Due to the presence of the secondary path, the filtered-x LMS (FXLMS) algorithm is used to adapt the coefficients of the adaptive filter. This FXLMS algorithm is implemented by function M-files `FXLMSinit.m` and `FXLMSadapt.m`.

```
% ANC.m           Application Program to Test Active Noise Control
%                 as Shown in Fig.1.7
[noise,fs] = wavread('engine_2.wav');% Engine noise
```

```
pri_num = load('p_z.asc');           % Primary path, numerator
pri_den = load('p_p.asc');           % Primary path, denominator
sec_num = load('s_z.asc');           % Secondary path, numerator
sec_den = load('s_p.asc');           % Secondary path, denominator
ir_sec = impz(sec_num,sec_den);      % Actual secondary path
est_sec = ir_sec(1:150)';            % Estimated secondary path
dn = filter(pri_num,pri_den,noise)'; % Desired signal
un = noise';                         % Input signal
M = 32;                              % Filter length
w0 = zeros(M,1);                     % Initialize coefs to 0

% Perform adaptive filtering using FXLMS algorithm

mulms = 0.05;                        % Step size
...
Sfxlms = FXLMSinit(w0,mulms,est_sec,sec_num,sec_den);
                                     % Initialization
[ylms,enlms,Sfxlms] = FXLMSadapt(un,dn,Sfxlms);
                                     % Perform FXLMS algorithm
```

1.6.1.3 Adaptive noise cancellation

Adaptive noise (or interference) cancellation, illustrated in Figure 1.8, can be classified as an adaptive system identification problem [6]. The system response $d(n)$ consists of the desired signal $s(n)$ corrupted by noise $v'(n)$, i.e. $d(n) = s(n) + v'(n)$. Since there is spectra overlap between the desired signal and noise, conventional fixed filters may not be an effective solution in attenuating this noise. As shown in Figure 1.8, the input signal $u(n)$ consists of a noise $v(n)$ that is correlated to $v'(n)$, but must be uncorrelated with $s(n)$. The adaptive filter produces an output signal $y(n)$ that estimates $v'(n)$. Thus the error signal approximates the signal $s(n)$. If the signal $s(n)$ is considered as the disturbance $\eta(n)$ shown in Figure 1.5 and the noise path is treated as the unknown system, adaptive noise cancellation can be considered as a system identification problem.

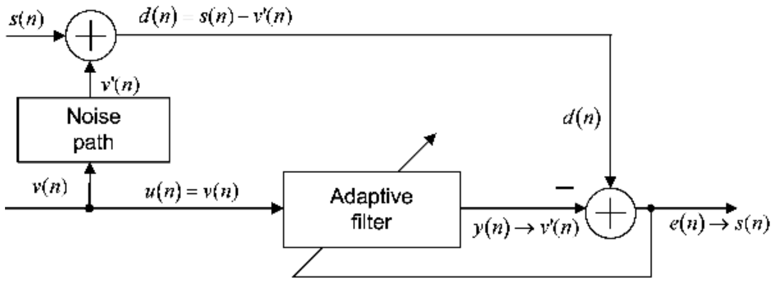

Figure 1.8 Block diagram of the adaptive noise canceller

When the signal $s(n)$ is absent during silent periods, the error signal approaches zero and an accurate system identification of the noise path can be obtained. The script M-file of the adaptive interference cancellation system is `AIC.m` and its partial listing is shown below.

Adaptive interference canceller

In MATLAB implementation of the adaptive interference canceller, the speech signal `timit.wav` is corrupted with white Gaussian noise. The input signal `un` is the source noise and the signal `dn` is the corrupted speech signal. The error signal `enlms` consists of a cleaner version of the speech signal after the output `ynlms` of the adaptive LMS filter has converged to the broadband noise.

```
% AIC.m          Application Program to Test Adaptive Noise
%               (Interference) Canceller as Shown in Fig.1.8
[y,fs] = wavread('timit.wav');      % Extract normalized wave file
noise=0.1*randn(size(y));           % Noisy signal
b = fir1(16,0.25);                  % Filter to simulate the noise
                                    %  path
filt_noise = filter(b,1,noise);     % Filtered version of noise
                                    %  signal
dn = (y+filt_noise)';               % Speech corrupted with
                                    %  filtered noise
un = noise';                        % Input signal to adaptive
                                    %  filter
M = 32;                             % Filter length
w0 = zeros(M,1);                    % Initialize coefs to 0

% Perform adaptive filtering using LMS algorithm

mulms = 0.05;                       % Step size
...
Slms = LMSinit(w0,mulms);           % Initialization
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
                                    % Perform LMS algorithm
```

1.6.1.4 Acoustic feedback cancellation in hearing aids

A hearing aid modifies and amplifies incoming sound to make it audible for people with hearing loss [2]. Hearing aids are normally small in size to fit into the ear canal or worn behind the ear. Due to the close proximity between the sound emitting transducer (or receiver) and the microphone, acoustic feedback can occur in hearing aids or other hearing devices such as Bluetooth headsets. Figure 1.9 shows the model of hearing aids and the application of the adaptive filter to remove the acoustic feedback (leakage) from the receiver to the microphone. The forward path is the processing unit of the hearing aids and its output $u(n)$ is fed into the adaptive filter and the receiver. The reference signal $d(n)$ picked up by the microphone is the summation of incoming sound and the

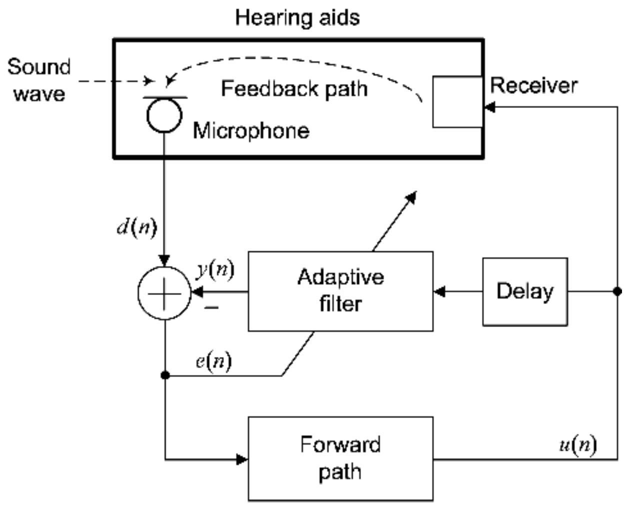

Figure 1.9 Block diagram of the hearing aids with acoustic feedback cancellation

feedback signal. The error signal $e(n)$ is derived by subtracting the adaptive filter output from the reference signal.

Due to the high correlation between the reference signal $d(n)$ and the input signal $u(n)$, the adaptive filter cannot model the feedback path accurately. A delay is used in the cancellation path to decorrelate the reference signal and the input signal. This acoustic feedback cancellation is also classified as the adaptive system identification problem that uses the adaptive filter to model the feedback path. The script M-file of the hearing aids application is `HA.m` and its partial listing is shown below.

Acoustic feedback reduction in hearing aids

In MATLAB implementation of acoustic feedback reduction, a speech signal `timit.wav` at the receiver is corrupted with white Gaussian noise `noise` to form the desired signal `dn`. The feedback path loaded from the data file `fb.dat` is used to simulate the feedback of the speech signal from the receiver to the microphone. A delay unit is used as the feedforward path `ff` for the simulation. The input signal `un` consists of the received speech signal when no acoustic feedback exists.

```
% HA.m          Application Program to Test Acoustic Feedback
%               Reduction for Hearing Aids as Shown in Fig.1.9
load fb.dat;                        % Load the feedback path
norm_fb = norm(fb);                 % Norm-2 of feedback path
delay = 10;                         % Delay in feedforward path
ff= [zeros(1,delay+1) 4];           % Simple feedforward path
[un,fs] = wavread('timit.wav');     % Extract normalized wave file
noise=0.01*randn(size(un));         % Noisy signal
```

```
dn = (un+noise)';                   % Speech corrupted with noise
num = conv(ff,fb);                  % Numerator
den = [1; -num(2:end)];             % Denominator
y = filter(num,den,dn);
M = 64;                             % Filter length
w0 = zeros(M,1);                    % Initialize filter coefs to 0

% Perform adaptive filtering using LMS algorithm

mulms = 0.005;                      % Step size
...
Slms = HALMSinit(w0,mulms,fb,ff,delay);
                                    % Initialization
[ylms,enlms,fblms,Slms] = HALMSadapt(un,Slms);
                                    % Perform LMS algorithm
```

1.6.2 Adaptive prediction

Adaptive prediction is illustrated in Figure 1.10, where the desired signal $d(n)$ is assumed to have predictable signal components [3]. This signal is delayed by Δ samples to form the input signal $u(n) = d(n - \Delta)$ for the adaptive filter to minimize the error signal $e(n)$. The adaptive filter predicts the current value of the desired signal based on the past samples. A major application of adaptive prediction is the waveform coding of speech, such as the adaptive differential pulse code modulation [1]. In this application, the adaptive filter exploits the correlation between adjacent samples of speech such that the prediction error is much smaller than the original signal on average. This prediction error is then quantized using fewer bits for transmission, resulting in a lower bit rate. Other applications of adaptive prediction include spectrum estimation and signal whitening.

Adaptive prediction is also used to separate the signal from noise based on differences between the signal and noise correlation. Depending on whether $y(n)$ or $e(n)$ is the desired output, the adaptive predictor enhances the narrowband signal corrupted by broadband noise or removes the narrowband interference from the corrupted broadband signal. With reference to Figure 1.10, the input of the adaptive line enhancement consists

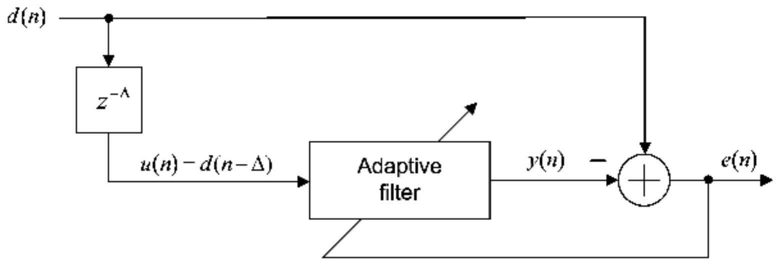

Figure 1.10 Block diagram of the adaptive predictor

of desired narrowband (or tonal) components $t(n)$ corrupted by broadband noise $v(n)$, i.e. $d(n) = t(n) + v(n)$. The delay Δ must be sufficiently large to decorrelate the broadband noise components, but the narrowband components still correlate in $d(n)$ and $u(n)$. The adaptive filter compensates for the delay (or phase shift) in order to cancel the narrowband components in $d(n)$. Therefore, the filter output $y(n)$ estimates the narrowband components $t(n)$ only, while the error signal $e(n)$ approximates the wideband noise $v(n)$. This structure enhances the narrowband signal at the adaptive filter output $y(n)$ and is called the adaptive line enhancer. Usually, $\Delta = 1$ is adequate for white noise and $\Delta > 1$ is used for color noise. The script M-file of the adaptive line enhancer is `ALE.m` and allows the user to select different modes of operations.

Adaptive line enhancer

The adaptive line enhancer can be used either to enhance the tonal signal or enhance the broadband (such as speech) signal corrupted by tonal noise. As shown in Figure 1.10, the input signal `un` to the adaptive filter is the delayed version of the mixed input of desired signal and noise. The delay unit is used to decorrelate the broadband noise component so that the adaptive filter output `ylms` converges to the narrowband component of the mixed signal. The desired signal `dn` is the delayless version of the mixed signal and is subtracted from the output signal to obtain the error signal `enlms`, which consists of the enhanced broadband signal such as speech. Therefore, the adaptive line enhancer can be programmed to extract the narrowband or broadband component depending on the given application.

```
% ALE.m         Application Program to Test Adaptive Line Enhancer
%               as Shown in Fig.1.10
select = input('Select (1) separate noise from tone, (2) separate
speech from tone:','s’);
 if select == '1'
    f = 400; fs = 4000;         % Tonal signal at 400 Hz and
                                %  sampling at 4000 Hz
    iter = 8000; n = 0:1:iter-1;% Length of signal and time index
    sn = sin(2*pi*f*n/fs);      % Generate sinewave
    noise=randn(size(sn));      % Generate random noise
    dn = sn+0.2*noise;          % Mixing sinewave with white noise
    delay = 1;                  % Delay by 1
    un = [zeros(1,delay) dn(1,1:length(dn)-delay)];
                                % Generate delayed version of d(n)
else
    [y,fs] = wavread('timit.wav');% Extract normalized wave file
    f = 1000;                   % Tone frequency 1000 Hz
    iter = length(y);           % Length of signal
    n = 0:1:iter-1;             % Time index
    sn = sin(2*pi*f*n/fs);      % Generate sinewave
    dn = sn+y';                 % Mixing sinewave with wave file
    delay = 5;                  % Delay by 5
    un = [zeros(1,delay) dn(1,1:length(dn)-delay)];
```

```
                                % Generate delayed version of d(n)
end
M = 64;                         % Filter length
w0 = zeros(M,1);                % Initialize filter coefs to 0

% Perform adaptive filtering using LMS algorithm

mulms = 0.01;                   % Step size
...
Slms = LMSinit(w0,mulms);       % Initialization
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
                                % Perform LMS algorithm
```

One practical application is to enhance the tonal signal in the dual-tone multi-frequency [12] signals found in touchtone phones. Alternatively, this structure can be used as an adaptive notch filter where the error signal $e(n)$ is the system output. One possible application of the adaptive notch filter is to remove any tonal noise (e.g. power hum) from the wideband signal, such as a biomedical measurement signal.

1.6.3 Adaptive inverse modeling

Adaptive filters can be applied for adaptive inverse modeling (or channel equalization) applications [7], where an unknown system is cascaded with an adaptive filter, as illustrated in Figure 1.11. The desired signal $d(n)$ is derived by delaying L samples of the input signal $s(n)$ using a delay unit z^{-L} to compensate for the overall propagation delay through the unknown system and the adaptive filter. The proper delay allows the adaptive filter to converge to a causal filter that is the inverse of the unknown system. Thus the adaptive filter equalizes the unknown system to recover the delayed version of signal $s(n - L)$ at the output $y(n)$. The combined transfer function of the unknown system and the adaptive filter approaches z^{-L}. In some applications, such as the adaptive channel equalizer in data modems, the desired signal is derived from the predefined data sequence during the training stage.

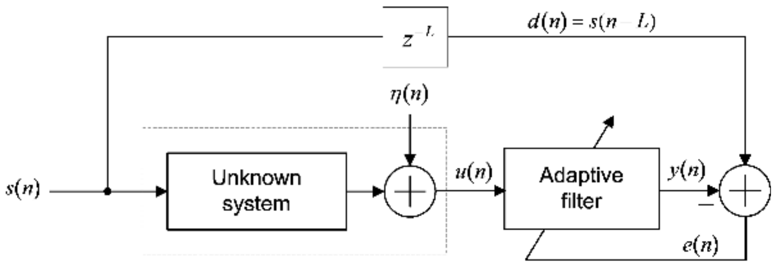

Figure 1.11 Inverse channel modeling using an adaptive filter

Similar to the system modeling shown in Figure 1.5, if $s(n)$ has a flat spectrum and the plant noise is small and uncorrelated with $s(n)$, the adaptive filter can adapt to an accurate inverse model of the unknown system. It is interesting to note that the adaptive line enhancer (or adaptive predictor) described in Figure 1.10 can be considered as inverse modeling by treating the delay $z^{-\Delta}$ as the unknown channel, and the adaptive filter converges to $z^{-L+\Delta}$. If $L = 0$ (as in the case of an adaptive predictor), the adaptive filter converges to a noncausal system of z^{Δ}. Therefore, the adaptive filter predicts the input at Δ samples ahead. The script M-file of the adaptive inverse modeling is `INVSYSID.m` and its partial listing is shown below.

Inverse system identification

The application of adaptive inverse system identification is illustrated in the MATLAB program `INVSYSID`. Here, a binary random signal is generated and transmitted to a channel modeled by a three-tap FIR filter `h_1`. The parameter `W` controls the amount of amplitude distortion produced by the channel. The output of the channel `out` becomes the input `un` to the adaptive filter. The length of the adaptive filter `eq_length` is 11 and the coefficients of adaptive filter are all initialized to zero. The desired signal `dn` is the delayed version of the binary signal. By selecting the delay to match the midpoint of the combined length of channel and adaptive filter impulse responses, the adaptive LMS filter is able to approximate the inverse of the channel response.

```
% INVSYSID.m   Inverse System Identification as Shown in Fig. 1.11

...
s  = 2*(rand(1,iter) > 0.5) - 1;     % Generate binary input signal

% Impulse response of channel 1
W = [2.9 3.1 3.3 3.5];               % Parameters affect the
                                     %  eigenvalue spread (evs)
h_1   = 0.5*(1+cos((2*pi/W(1))*((1:3) - 2)));
                      % Channel 1 impulse response (evs = 6.0782)
% Select channel 1

h = h_1;
ch_delay = fix(length(h)/2);         % Channel delay = 2
eq_length = 11;                      % Length of adaptive equalizer
eq_delay = fix(eq_length/2);         % Delay of channel equalizer
total_delay = ch_delay + eq_delay;   % Total delay
out  = filter(h,1,s);                % Channel output signal
un = out + sqrt(0.001).*randn(1,iter);% Input to adaptive filter
dn = [zeros(1,total_delay) s(1,1:length(un)-total_delay)];
                                     % Desired signal
w0 = zeros(eq_length,1);             % Initialize filter coefs to 0

% LMS algorithm
```

```
mulms = 0.075;                       % Step size
...
Slms = LMSinit(w0,mulms);            % Initialization
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
                                     % Perform LMS algorithm
```

The above description of an adaptive channel equalizer assumes that the additive noise $\eta(n)$ shown in Figure 1.11 can be ignored. When significant noise is present, the adaptive filter cannot converge to the exact inverse of the channel as effort is also needed to minimize the additive noise. In addition, due to the channel inversion function, the adaptive filter may amplify the noise at frequency bands where the unknown system has small magnitude responses. Therefore, special considerations are needed to prevent excessive equalization at the expense of noise enhancement.

Inverse modeling is widely used in adaptive control, deconvolution and channel equalization. For example, the adaptive equalizer is used to compensate the distortion caused by transmission over telephone and radio channels [5]. In adaptive channel equalization, $s(n)$ is the original data at the transmitter that is transmitted to the receiver through the unknown communication channel. At the receiver, the received data $u(n)$ is distorted because each data symbol transmitted over a time-dispersive channel extends beyond the time interval used to represent that symbol, thus causing an overlay of received symbols. Since most channels are time varying and unknown in advance, the adaptive filter is required to converge to the inverse of the unknown and time-varying channel so that $y(n)$ approximates the delayed version of $s(n)$.

After a short training period using pseudo-random numbers known to both the transmitter and receiver in modems, the transmitter begins to transmit the modulated data signal $s(n)$. To track the possible changes in the channel during data transmission, the adaptive equalizer must be adapted continuously while receiving the data sequence. However, the desired signal is no longer available to the adaptive equalizer in the receiver located at the other end of the channel. This problem can be solved by treating the output of the decision device at the receiver as correct and using it as the desired signal $d(n)$ to compute the error signal, as shown in Figure 1.12. This technique is called the decision-feedback equalizer, which works well when decision errors occur infrequently. The script M-file of the adaptive channel equalizer is `CH_EQ.m` and its partial listing is shown below.

Adaptive channel equalization

The channel equalization is commonly used to illustrate the principle behind inverse system identification. As illustrated in Figure 1.12, there are two modes of channel equalization: training and decision-directed modes. The main difference between these modes lies in the generation of the desired signal. In the training mode, the desired signal is obtained from the delayed version of the binary random signal, as described in the previous inverse system identification example. In the decision-directed mode, the desired signal is derived from the decision device at the receiver end. Therefore, a different MATLAB function `LMSadapt_dec` is used for the decision-directed mode.

```
% CH_EQ.m         Channel Equalizer as Shown in Fig.1.12.

...
% Select channel 1

h = h_1;
ch_delay = fix(length(h)/2);        % Channel delay, 2
eq_length = 11;                     % Length of adaptive equalizer
eq_delay = fix(eq_length/2);        % Delay of channel equalizer
total_delay = ch_delay + eq_delay;
out  = filter(h,1,s);               % Channel output signal
un = out(1:iter_train) + sqrt(0.001).*randn(1,iter_train);
                                    % Input to adaptive filter
un_dec = out(iter_train+1:iter_total) +
sqrt(0.001).*randn(1,iter_dec);     % Input to adaptive filter
                                    %  in decision directed mode
dn = [zeros(1,total_delay) s(1,1:iter_train-total_delay)];
                                    % Desired signal in
                                    %  training mode
w0 = zeros(eq_length,1);            % Initialize filter coefs to 0

% Adaptive filtering using LMS algorithm

mulms = 0.075;                      % Step size mu
...
Slms = LMSinit(w0,mulms);           % Training mode
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
                                    % Initialization
Slms_dec = LMSinit(Slms.coeffs,mulms/10); % Decision-directed mode
[ylms_dec,enlms_dec,Slms_dec] = LMSadapt_dec(un_dec,Slms_dec);
                                    % Perform LMS algorithm
```

1.6.4 Adaptive array processing

So far, we have introduced adaptive filtering applications that combine weighted input signal samples at different time instants to produce an output signal that is related to the desired signal. This type of filtering is classified as temporal filtering. In this section, we discuss an alternate adaptive filtering application that acquires input signals at different positions to form a spatial filtering approach. The term adaptive array processing is commonly used to describe multiple adaptive filters that perform spatial filtering. An example of adaptive array processing [5] is shown in Figure 1.13, where the array of multiple sensors (spaced l units apart) with steering delays are used to pick up signals and interference arriving from different directions. The main objective of adaptive array

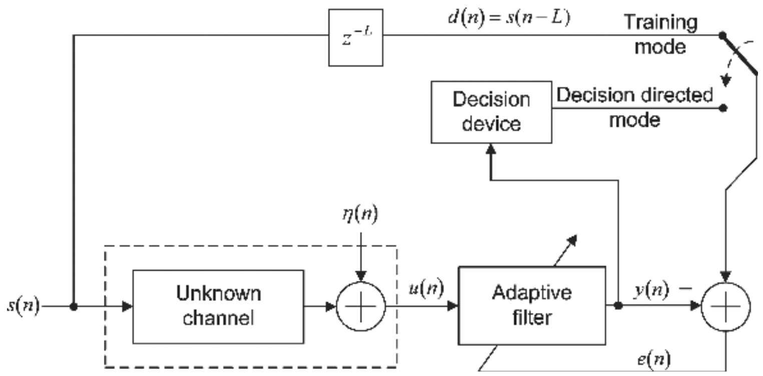

Figure 1.12 Two modes (training and decision directed) of operation in adaptive channel equalization

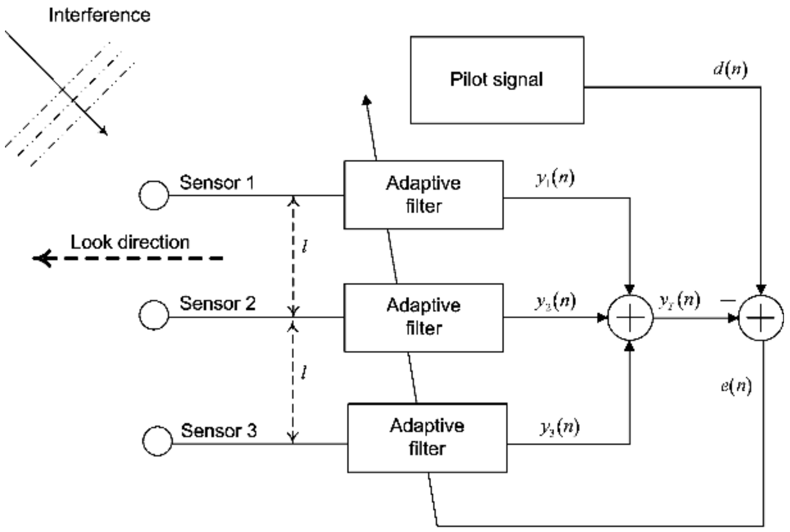

Figure 1.13 An example of adaptive array processing with three sensors

processing is to form a beam that produces a peak array gain in the desired look direction and adaptively to cancel interference outside the desired beam.

Each array element is connected to an adaptive filter that produces an output that is subsequently combined with the outputs of other adaptive filters to form an array pattern in the desired look direction. In order to pick up the desired signal in the look direction, a pilot signal that consists of the desired spectral and directional characteristics is used as the desired signal $d(n)$. The error signal $e(n)$ is used to adjust the adaptive filters to form a beam pattern in the look direction, and at the same time eliminate interference coming from other directions.

The sensor array in the adaptive array processing system is analogous to the tapped-delay line of the adaptive FIR filter shown in Figure 1.3. At any particular instant of

Table 1.1 Summary of adaptive filtering applications

Application types	**Desired (or reference) signal, *d(n)***	**Input signal, *u(n)***	**Error signal, *e(n)***	**Applications**
System identification	Output from the unknown plant/system, may be corrupted by noise.	Excitation signal or processed version of excitation signal.	Difference between the unknown system output and the adaptive filter output. The adaptive filter converges to the approximation of the unknown system.	• Acoustic/network echo cancellation. • Active noise control. • Adaptive noise cancellation. • Feedback cancellation.
Inverse system identification	A delayed or processed version of the reference signal.	Output from the unknown plant/system, may be corrupted by noise.	Difference between the desired signal and the adaptive filter output. The adaptive filter converges to an inverse of the unknown system.	• Channel equalization for communication systems and disc drive.
Prediction	Excitation signal consists of both wideband signal (noise) and narrowband noise (signal).	Delayed version of the excitation signal.	Error signal consists of the wideband signal (noise), while the output of the adaptive filter consists of narrowband noise (signal). The adaptive filter converges to a narrowband bandpass filter.	• Adaptive line enhancer. • Predictive coding. • Spectral analysis.
Array processing	Pilot signal or some reference waveform in the look direction.	An array of input signals that detect signal from different directions.	Error signal is the difference between the combined adaptive filters' outputs and the pilot signal. Adaptive filter converges to a beam pattern in the look direction.	• Adaptive array processing for hearing aids. • Antenna adaptive array.

time, spatial signals captured in the sensor array are individually filtered by its respective adaptive filter and combined to capture the target signal with the maximum signal-to-noise ratio. Similarly, the adaptive filter adaptively combines the input samples to form an output signal that approximates the desired (target) signal.

1.6.5 Summary of adaptive filtering applications

A summary of adaptive filtering applications is listed in Table 1.1 to highlight the main differences among different applications. It is noted that the desired (or reference) signal is not always available for a given application. The reference signal $d(n)$ may be derived from the input signal $u(n)$. For example, in the adaptive channel equalization operating in the decision-directed mode (Figure 1.12), the desired signal is derived from a nonlinear decision device. Alternatively, the error signal $e(n)$ may be derived without the reference signal. For example, a constant modulus adaptive algorithm [5] is applied to a constant-envelope signal in communications. Based on a priori information that the amplitude of signal is constant, we can compare the adaptive filter output to this constant value and derive an error signal without the desired signal. Another example is the active noise control shown in Figure 1.7, where the noise cancellation (acoustic superposition) is performed in the acoustic domain. Therefore, a microphone is used to sense the residual noise as the error signal $e(n)$ for filter adaptation.

1.7 Transform-domain and subband adaptive filters

As explained in Sections 1.2 and 1.3, the most widely used adaptive structure is the transversal (or FIR) filter shown in Figure 1.3. This structure operates in the time domain and can be implemented in either the sample or block processing mode [13]. However, the time-domain LMS-type adaptive algorithms associated with the FIR filter suffer from high computational cost if applications (such as acoustic echo cancellation) demand a high-order filter and slow convergence if the input signal is highly correlated.

This section describes the transform-domain and subband adaptive filters [1, 3] that have advantages of low computational cost and fast convergence. The differences between subband and transform-domain adaptive filters are highlighted. Some insights into the relation between these structures are briefly introduced. This section also provides a brief introduction to subband adaptive filtering, which is suitable for adaptive filtering applications that need a long filter length, such as acoustics echo cancellation.

1.7.1 Transform-domain adaptive filters

This subsection highlights two classes of transform-domain adaptive filters including frequency-domain adaptive filters and self-orthogonalizing adaptive filters.

1.7.1.1 Frequency-domain adaptive filters

The frequency-domain adaptive filter transforms the desired signal $d(n)$ and the input signal $u(n)$ using the FFT and performs frequency-domain filtering and adaptation based on these transformed signals. As discussed in Sections 1.2 and 1.3, the time-domain adaptive filter performs linear convolution (filtering) and linear

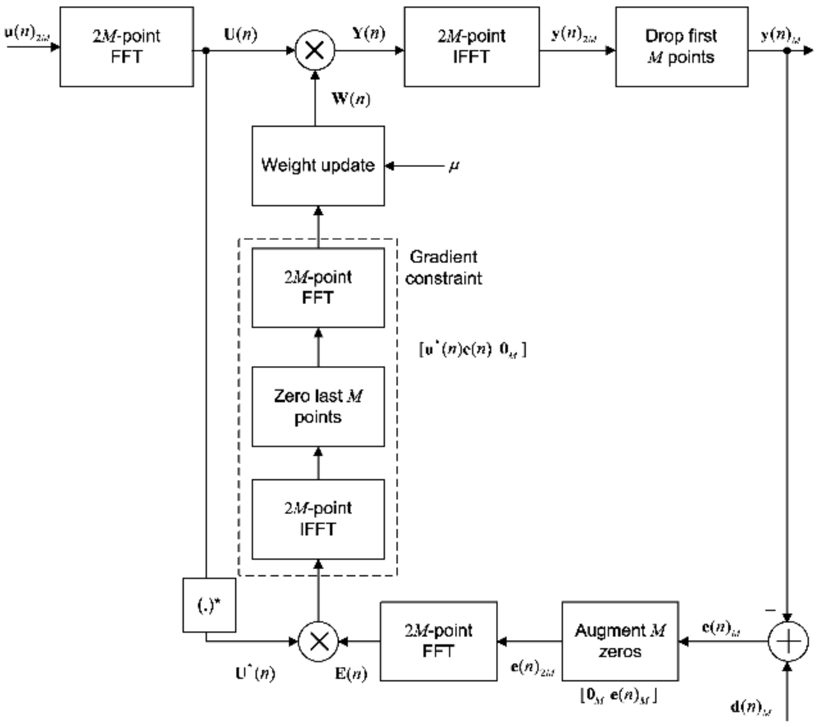

Figure 1.14 Block diagram of the frequency-domain fast LMS algorithms

correlation (weight updating) iteratively based on the arrival of new data samples. The frequency-domain adaptive filter shown in Figure 1.14 performs frequency transforms and processing over a block of input and desired signals, thus achieving substantial computational savings by using an FFT, especially for applications that use high-order filters. Unfortunately, the frequency-domain operations result in circular convolution and circular correlation. Thus a more complicated overlap-add or overlap-save method [13] is required to overcome the error introduced in these circular operations.

Figure 1.14 shows the block diagram of the frequency-domain adaptive filter using the fast LMS algorithm [3, 4]. This algorithm employs the overlap-save method with 50 % overlap and requires five $2M$-point FFT/IFFT operations.

In order to achieve the correct linear convolution and correlation results in frequency-domain filtering, extended data vectors are required. This type of processing is known as block processing, with the following definitions:
Input vector:

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

FFT of input vector:

$$\mathbf{U}(n) = \text{FFT}\,[\mathbf{u}(n)] = \left[U_{2M-1}(n), \ldots, U_M(n), U_{M-1}(n), \ldots, U_1(n), U_0(n)\right]^T$$

Error vector:

$$\mathbf{e}(n)_M = [e(n), e(n-1), \ldots, e(n-M+1)]^T$$

FFT of error vector:

$$\mathbf{E}(n) = \text{FFT}\,[0_M \mathbf{e}(n)_M] = \left[E_0(n), E_1(n), \ldots, E_{2M-1}(n)\right]^T$$

Note that the subscripts $2M$ and M denotes the length of the vectors $\mathbf{u}(n)_{2M}$ and $\mathbf{e}(n)_M$, respectively. The input vector $\mathbf{u}(n)_{2M}$ concatenates the previous M data samples with the present M data samples, and a $2M$-point FFT operates on this input vector to form the frequency-domain vector $\mathbf{U}(n)$. Note that $\mathbf{U}(n)$ is a $2M$-point complex vector.

The output signal vector from the adaptive filter can be computed by taking the element-by-element multiplication between the input vector and the $2M$-point weight vector as

$$\mathbf{Y}(n) = \mathbf{W}(n).\mathbf{U}(n), \tag{1.26}$$

where the operator '.' is the corresponding element-by-element multiplication for the frequency index $k = 0, 1, \ldots, 2M-1$ and $\mathbf{W}(n) = [W_0(n), W_1(n), \ldots, W_{2M-1}(n)]^T$ is the complex-valued weight vector at time n. In order to obtain the correct result using circular convolution, the frequency-domain output vector, $\mathbf{Y}(n) = [Y_0(n), Y_1(n), \ldots, Y_{2M-1}(n)]^T$, is transformed back to the time domain using the inverse FFT (IFFT) and the first M points of the $2M$-point IFFT outputs are discarded to obtain the output vector $\mathbf{y}(n)_M = [y(n), y(n-1), \ldots, y(n-M+1)]^T$. The output vector is subtracted from the desired vector $\mathbf{d}(n) = [d(n), d(n-1), \ldots, d(n-M+1)]^T$ to produce the error signal vector $\mathbf{e}(n)_M$. The error vector $\mathbf{e}(n)$ is then augmented with M zeros and transformed to the frequency-domain vector $\mathbf{E}$(n) using the FFT, as shown in Figure 1.14. The adaptation of the filter coefficients can be expressed as

$$W_k(n+M) = W_k(n) + \mu U_k^*(n) E_k(n), \;\; k = 0, 1, \ldots, 2M-1, \tag{1.27}$$

where $U_k^*(n)$ is the complex conjugate of the frequency-domain signal $U_k(n)$. This weight-updating equation is known as the block complex LMS algorithm [3]. Note that the weight vector is not updated sample by sample as the time-domain LMS algorithm; it is updated once for a block of M samples. Also, the weight variables are complex valued due to the FFT.

As shown in the gradient constraint box in Figure 1.14, the weight-updating terms $[\mu \mathbf{U}^*(n).\mathbf{E}(n)]$ are inverse transformed, and the last M points are set to zeros before taking the $2M$-point FFT for weight updating. An alternate algorithm, called the unconstrained frequency-domain adaptive filter [14], removes the gradient constraint block, thus producing a simpler implementation that involves only three transform operations. Unfortunately, this simplified algorithm no longer produces a linear correlation between the transformed error and input vectors, thus resulting in poorer performance compared to the frequency-domain adaptive filters with gradient constraints. Nevertheless, frequency-domain adaptive filters with fast convolution and correlation schemes are able to achieve lower computational cost as compared to high-order time-domain adaptive filters. The function M-files of the frequency-domain adaptive filter algorithm are

`FDAFinit.m` and `FDAFadapt.m`, and the script M-file that calls these MATLAB functions is `FDAFdemo.m`. Partial listing of these files is shown below.

The frequency-domain adaptive filter (FDAF) algorithm

The function `FDAFinit` performs the initialization of the FDAF algorithm, where the weight vector `w0` is normally set to a zero vector at the start of the simulation. The function `FDAFadapt` performs the actual computation of the FDAF algorithm given in Equation (1.27). Note that both constrained and unconstrained versions of the FDAF algorithm are implemented in the program, and users have the option to select one. The step sizes `mu` and `mu_unconst` are used in the FDAF algorithm with gradient constraint and gradient unconstraint, respectively. Note that the function `FDAFadapt` calls another MATLAB function `fdaf` to perform a frequency-domain adaptive filter algorithm, which is modified from John Forte, BWV Technologies Inc. Version v1.0.

```
...
M = 256;
mu = 0.5;                    % Step size for constrained FDAF
mu_unconst = 0.001;          % Step size for unconstrained FDAF
...
S = FDAFinit(zeros(M,1),mu,mu_unconst,iter);
                             % Initialization
S.unknownsys = b;
[en,S] = FDAFadapt(un,dn,S); % Perform FDAF algorithm
...

function S = FDAFinit(w0,mu,mu_unconst,iter)
...
S.coeffs       = w0;         % Coefficients of FIR filter
S.length       = M;          % Length of adaptive filter
S.step         = mu;         % Step size of constrained FDAF
S.step_unconst = mu_unconst; % Step size of unconstrained FDAF
S.iter         = 0;          % Iteration count
S.AdaptStart   = length(w0); % Running effect of adaptive filter
S.weight       = WEIGHT;     % FFT of zero-padded weight vector
S.palpha       = 0.5;        % Constant to update power in
                             %   each frequency bin
S.ref          = un_blocks;  % 2M-sample block of input signal

function [en,S] = FDAFadapt(un,dn,S)
...
for n = 1:ITER
    u_new = [u_new(2:end); un(n)];
                             % Start input signal blocks
    dn_block = [dn_block(2:end); dn(n)];
                             % Start desired signal block
    if mod(n,M)==0           % If iteration == block length
```

```
        un_blocks = [u_old; u_new];
        u_old = u_new;
        b = S.unknownsys;
        if n >= AdaptStart   % Frequency-domain adaptive filtering
            [error,WEIGHT,power,misalign] =
fdaf(M,mu,mu_unconst,power_alpha,WEIGHT,dn_block,un_blocks,power,b
,select);
            en = [en; error];% Update error block
            eml = [eml; misalign];
                             % Update misalignment at each block
        end
    end
end
```

1.7.1.2 Self-orthogonalizing adaptive filters

As explained in Section 1.5, the eigenvalue spread plays an important role in determining the convergence speed of the time-domain adaptive filters. When the input signal is highly correlated, the convergence of the LMS algorithm is very slow. Many methods were developed to solve this problem. One method is to use the RLS algorithm (see Section 1.4), which extracts past information to decorrelate the present input signal [3]. This approach suffers from high computational cost, poor robustness, low numerical stability and slow tracking ability in nonstationary environments. The other method is to decorrelate the input signal using a transformation such as the discrete Fourier transform (DFT) or discrete cosine transform (DCT) [1]. This method is known as the self-orthogonalizing adaptive filter. Compared with the RLS approach, the self-orthogonalizing adaptive filter saves computational cost and is independent of the characteristics of the input signal.

Figure 1.15 shows the self-orthogonalizing adaptive filter. It has a similar structure to the M-tap adaptive transversal filter shown in Figure 1.3, but with preprocessing (transform and normalization) on the input signal using the DFT or DCT. The transform is carried out for every new input signal sample. The transformed signals $U_k(n)$ are normalized by the square root of its respective power in the transform domain as

$$U'_k(n) = \frac{U_k(n)}{\sqrt{P_k(n) + \delta}}, \ \ k = 0, 1, \ldots, M-1, \tag{1.28}$$

where δ is a small constant to prevent division by zero and $P_k(n)$ is the power that can be estimated recursively as

$$P_k(n) = (1-\beta)P_k(n-1) + \beta U_k^2(n). \tag{1.29}$$

Note that the power $P_k(n)$ is updated iteratively for every new sample arriving at the tapped-delay line and β is a forgetting factor that is usually chosen as $1/M$. The resulting transform-domain signals with similar power are processed by the adaptive linear combiner whose tap weights are updated by the LMS algorithm given in Equation (1.10).

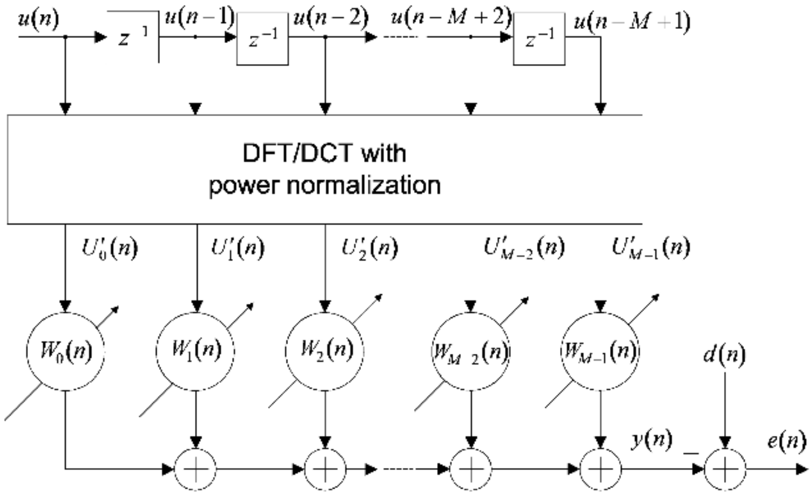

Figure 1.15 Block diagram of the self-orthogonalizing algorithm

Therefore, the extra computational cost over the conventional LMS algorithm is the additional transform power estimator given in Equation (1.29) and the normalization described in Equation (1.28).

The normalization defined in Equation (1.28) can be absorbed into the LMS algorithm to become the normalized LMS algorithm given in Equation (1.13). It is also noted that when DFT is used to transform the input signal, the DFT outputs are complex variables and the complex LMS algorithm must be used to update the weight vector as

$$W_k(n+1) = W_k(n) + \mu_k U_k^*(n)e(n), \quad k = 0, 1, \ldots, M-1, \tag{1.30}$$

where $\mu_k = \mu/\sqrt{P_k(n)+\delta}$ is the normalized step size. In this case, the output of the algorithm is still real valued if the desired signal is real valued because the complex error signal $e(n)$ has a negligible imaginary part. The function M-files of the self-orthogonalizing adaptive filter algorithm are `SOAFinit.m` and `SOAFadapt.m`, and the script M-file `SOAFdemo.m` calls these MATLAB functions. Partial listing of these M-files is shown below.

The self-orthogonalizing adaptive filter algorithm

The self-orthogonalizing adaptive filter requires a transformed input vector. In this MATLAB example, we use a DCT-transform matrix `S.T` to transform the input data vector `u`. The transformed input `U` is applied to a set of parallel coefficients `w` and updates similar to the normalized LMS algorithm. The normalized step size at each channel is based on the individual signal power `power_vec` at that frequency bin.

```
...
S = SOAFinit(zeros(M,1),mu,iter); % Initialization
S.unknownsys = b;
[yn,en,S] = SOAFadapt(un,dn,S);   % Perform algorithm

function S = SOAFinit(w0,mu,leak)
                                  % Assign structure fields
S.coeffs        = w0(:);          % Coefficients of FIR filter
S.step          = mu;             % Step size
S.leakage       = leak;           % Leaky factor
S.iter          = 0;              % Iteration count
S.AdaptStart    = length(w0);     % Running effects
M               = length(w0);     % Length of filter
S.beta          = 1/M;            % Forgetting factor
%
for j=1:M,                        % DCT-transform matrix
   S.T(1,j)=1/sqrt(M);
   for i=2:M,
     S.T(i,j)=sqrt(2/M)*cos(pi*(i-1)*(2*j-1)/2/M);
   end
end

function [yn,en,S] = SOAFadapt(un,dn,S)
...
for n = 1:ITER
    u = [un(n); u(1:end-1)];      % Input signal vector contains
                                  %  [u(n),u(n-1),...,u(n-M+1)]'
    U = T*u;                      % Transformed input vector
    yn(n) = w'*U;                 % Output signal
    power_vec= (1-S.beta)*power_vec+S.beta*(U.*U);
                                  % Estimated power
    inv_sqrt_power = 1./(sqrt(power_vec+(0.00001.*ones(M,1))));
    en(n) = dn(n) - yn(n);        % Estimation error
    if ComputeEML == 1;
        eml(n) = norm(T*b-w)/norm_Tb;
                                  % System error norm (normalized)
    end
    if n >= AdaptStart
        w = w + (mu*en(n)*inv_sqrt_power).*U;
                                  % Tap-weight adaptation
        S.iter = S.iter + 1;
    end
end
```

The following section briefly introduces a different structure known as the subband adaptive filter, which closely resembles the self-orthogonalizing filter.

1.7.2 Subband adaptive filters

As stated in Section 1.6.1, the acoustic echo canceller in a typical room requires a high-order adaptive FIR filter to cover long echo tails. This is a disadvantage of using FIR filters for adaptive filtering because of the high computational load and slower convergence. Subband filtering structures have been proposed to overcome these problems [1, 3].

Figure 1.16 shows the block diagram of a general subband adaptive filter (SAF) that partitions the fullband input and desired signals into N subbands and decimates the subband signals from the original sampling rate f_s to the lower rate of f_s/N. The long M-tap filter (shown in Figure 1.3) is now replaced by N shorter M_S-tap FIR filters (where $M_S < M$) operating in parallel at a lower rate. The analysis filter bank consists of N parallel bandpass filters that are designed to partition the fullband signal into N overlapped frequency bands. The details of designing the analysis filter bank will be discussed in Chapters 2 and 5.

Also shown in Figure 1.16, a synthesis filter bank is used to combine the N subband output signals, $y_{i,D}(n), i = 0, 1, \ldots, N - 1$, into a fullband signal $y(n)$. The synthesis filter bank consists of a bank of interpolators that upsample the subband signals before filtering and adds these subband signals. Again, the details of designing synthesis filter banks will be discussed in Chapters 2 and 5. A synthesis filter bank is only required when fullband signals, such as the output signal $y(n)$ or error signal $e(n)$, are required. A detailed description of subband adaptive filters and their important properties will be given in Chapters 4, 5, 6 and 7.

A problem with subband adaptive filters is the introduction of substantial delay in the signal path caused by analysis and synthesis filter banks. Several modifications to the subband structures, known as delayless subband adaptive filters, are introduced in Chapter 4. A new multiband-structured SAF that has several useful properties as compared to conventional subband adaptive filters will be discussed in Chapter 6. Chapter 7 deals with the stability and performance analysis of SAF. Chapter 8 reports new research directions and recent works in the area of subband adaptive filtering.

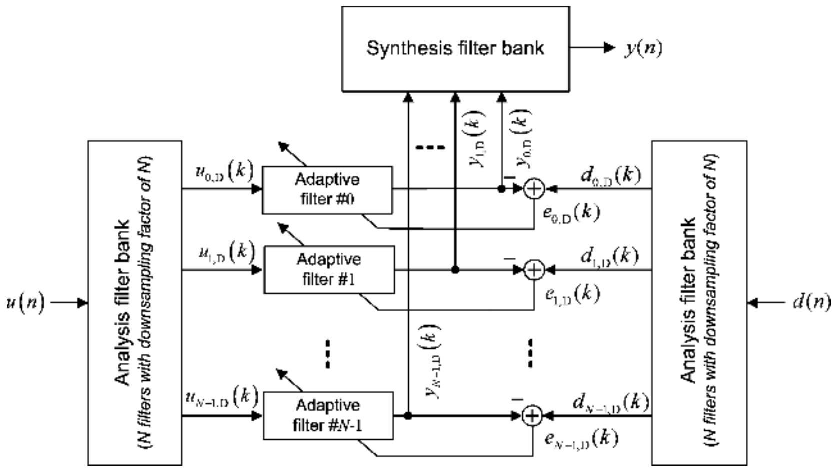

Figure 1.16 Block diagram of the subband adaptive filter

There are some similarities and differences between the SAF and the self-orthogonalizing adaptive filter shown in Figure 1.15. Both structures are designed to improve convergence over the FIR filter with the LMS algorithm by partitioning the fullband input signal $u(n)$ into a bank of parallel bandpass filters. The differences include: (i) the SAF involves the decimation operation to reduce the sampling rate for adaptive filtering, while there is no decimation in the self-orthogonalizing adaptive filter, (ii) N subband error signals are produced to update respective subband adaptive filters, but only a single error signal is used to update all the tap weights in the self-orthogonalizing adaptive filter, (iii) if the DFT is used in the self-orthogonalizing adaptive filter, the tap weights are complex numbers, while the SAF has real-valued tap weights, which leads to a simpler computational architecture, and (iv) the subband adaptive filter needs the synthesis filter bank to reconstruct the fullband output and error signals, which is not required in the self-orthogonalizing adaptive filter. These attributes of the subband adaptive filters and the comparison with other adaptive filters will be further explored in subsequent chapters.

1.8 Summary

This chapter introduced fundamental concepts of adaptive filtering. The basic block diagrams of filter structures, adaptive algorithms and applications were briefly discussed. Adaptive algorithms discussed were limited to the class that is based on minimizing the mean-square error, e.g. the LMS-type algorithms. A brief mention of another class of algorithms that is based on the deterministic framework of finding the least-square error over a period of past error samples was also given. In addition, important features of the transform-domain and subband adaptive filters were presented to illustrate how they differ from each other and the time-domain approaches. A more detailed description and analysis of these time-domain and transform-domain adaptive algorithms can be found in References [1] and [3].

References

[1] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications*, New York: John Wiley & Sons, Inc., 1998.

[2] E. Hänsler and G. Schmidt, *Acoustic Echo and Noise Control: A Practical Approach*, New York: John Wiley & Sons, Inc., 2004.

[3] S. Haykin, *Adaptive Filter Theory*, 4th edition, Upper Saddle River, New Jersey: Prentice Hall, 2002.

[4] S. M. Kuo and D. R. Morgan, *Active Noise Control Systems: Algorithms and DSP Implementations*, New York: John Wiley & Sons, Inc., 1996.

[5] J. R. Treichler, C. R. Johnson and M. G. Larimore, *Theory and Design of Adaptive Filters*, Upper Saddle River, New Jersey: Prentice Hall, 2001.

[6] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Upper Saddle River, New Jersey: Prentice Hall, 1985.

[7] B. Widrow and E. Walach, *Adaptive Inverse Control*, Upper Saddle River, New Jersey: Prentice Hall, 1996.

[8] S. Haykin and B. Widrow, *Least-Mean-Square Adaptive Filters*, New York: John Wiley & Sons, Inc., 2003.

[9] A. H. Sayed, *Fundamentals of Adaptive Filtering*, New York: John Wiley & Sons, Inc., 2003.

[10] M. Montazeri and P. Duhamel, 'A set of algorithms linking NLMS and block RLS algorithms', *IEEE Trans. Signal Processing*, **43**(2), February 1995, 444–453.

[11] S. M. Kuo, B. H. Lee and W. Tian, *Real-Time Digital Signal Processing: Implementations and Application*, 2nd edition, Chichester, West Sussex: John Wiley & Sons, Ltd, 2006.

[12] W. S. Gan and S. M. Kuo, *Embedded Signal Processing with the Micro Signal Architecture*, Hoboken, New Jersey: John Wiley & Sons, Inc., 2007.

[13] S. M. Kuo and W. S. Gan, *Digital Signal Processors: Algorithms, Implementations, and Applications*, Upper Saddle River, New Jersey: Prentice Hall, 2005.

[14] D. Mansour and A. H. Gray, 'Unconstrained frequency-domain adaptive filters', *IEEE Trans. Acoust. Speech Signal Processing*, **30**(3), October 1982, 726–734.