Chapter 7

Adaptive Filtering

7.1 Introduction

The principle property of an *adaptive filter* is its time-varying, self-adjusting characteristics. An adaptive filter usually takes on the form of an FIR filter structure, with an adaptive algorithm that continually updates the filter coefficients, such that an *error* signal is minimised according to some criterion. The *error* signal is derived in some way from the signal flow diagram of the application, so that it is a measure of how close the filter is to the optimum. Most adaptive algorithms can be regarded as approximations to the *Wiener filter*, which is therefore central to the understanding of adaptive filters.

7.2 Examples of Adaptive Filtering

To illustrate how adaptive filtering can be used, three examples are given below, together with the finite difference equations associated with the so called LMS algorithm. Having described how this method can be used, we will then go on to explain how this filter was derived, why it works and what the ideas are behind the method.

7.2.1 Telephone Echo Canceller

A telephone echo canceller, as depicted in Figure 7.1(a) below, is a typical application of an adaptive filter. The "hybrid" is an electronic device that should ensure that all signals from the far telephone are sent to the local telephone without any of the far telephone signals getting through to be sent back to the far telephone as if it were an echo. In practice, the hybrid cannot be perfect because it relies on perfect impedance matching, and the impedance of the phone and line will vary with time. Such echoes are often noticeable on international phone lines when you hear an echo of yourself substantially delayed.

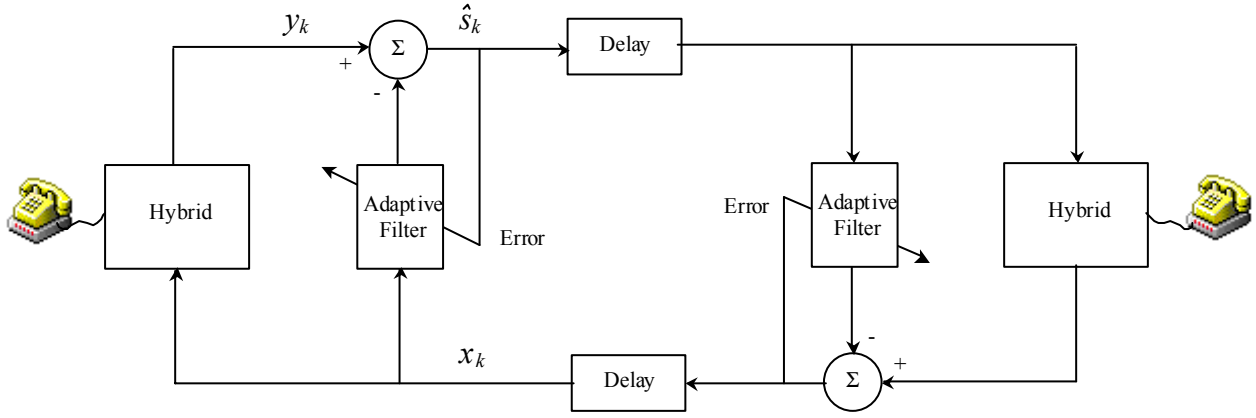


Figure 7.1(a): Telephone echo cancellation.

The echo canceller depicted in the diagram works by removing the echoes from the mismatch in the hybrid, by making the adaptive filter exactly match the reflection path, and track the changes to this echo path. The adaptive filter is an FIR filter of length N with coefficients $w(i)$. Concentrating on the left hand adaptive filter, the following finite difference equations would eventually achieve an optimal reduction of the echo:

(a) Calculate the k^{th} signal sample: $$\hat{s}_k = y_k - \sum_{i=0}^{N-1} w_k(i) \cdot x_{k-i}$$

(b) Update the filter weights for the next sample: $$w_{k+1}(i) = w_k(i) + 2\mu \hat{s}_k x_{k-i} \text{ , for } i = 0 \text{ to } N-1$$

where $\hat{s}_k$ is an "estimate" of the signal from the local telephone, and μ is a small positive constant. The filter weights $w_k(i)$ can start with any arbitrary values (i.e. for k=0) – for example all set to zero. The derivation of these apparently very simple equations is given in later sections in this Chapter.

7.2.2 Aircraft Cockpit Noise Canceller

The second example concerns the cancellation of noise interference when using a microphone. Figure 7.2(b) depicts an aircraft cockpit – the problem is that the loud background noise makes it difficult for ground staff to understand what the pilot is saying. The solution is also shown in the figure, and again uses an adaptive filter. A second microphone (B) located some distance from the pilot picks up a second version of the engine noise that is correlated with that picked by (A). The adaptive filter adjusts this to be as close a replica as possible to that contained in the speech.

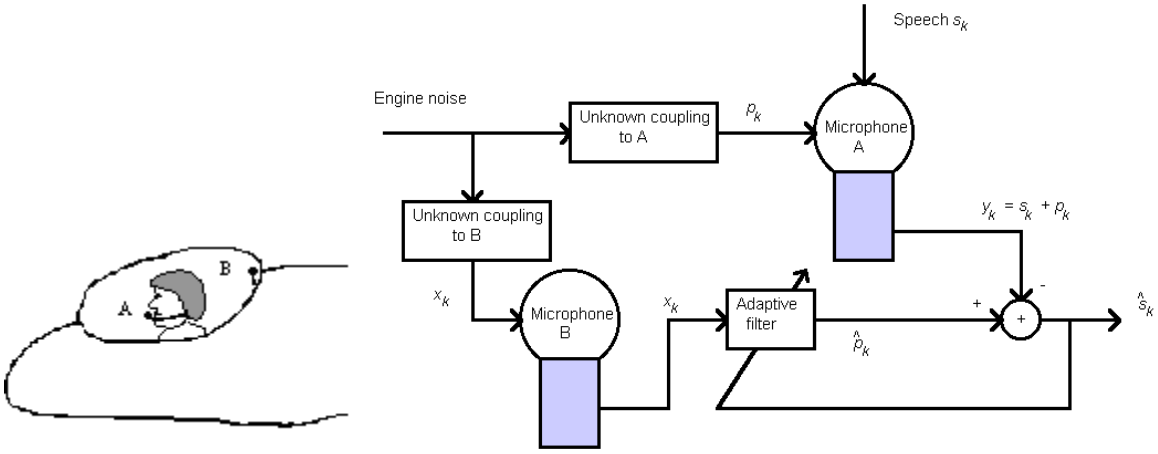


Figure 7.1(b)

The equations required are the same as for the previous example:

(a) Calculate the k^{th} signal sample: $\hat{s}_k = y_k - \sum_{i=0}^{N-1} w_k(i) \cdot x_{k-i}$

(b) Update the filter weights for the next sample: $w_{k+1}(i) = w_k(i) + 2\mu\hat{s}_k x_{k-i}$, for $i = 0$ to $N-1$

7.2.3 Adaptive Equalisation

The third example is concerned with digital receivers, and is illustrated in Figure 7.1(c). A typical mobile phone channel distorts the binary signal by introducing dispersion and multipath, so that the binary signals may no longer be detected reliably. The purpose of the adaptive filter is to provide the inverse response of the channel so that the output of the filter is an estimate $\hat{s}_k$ of the transmitted signal. Since the channel characteristic changes with time, it must be adaptive in order to track these changes. An "error" signal is derived by finding the difference between the filter output and a "squared up" version of it. The finite difference equations required are given below:

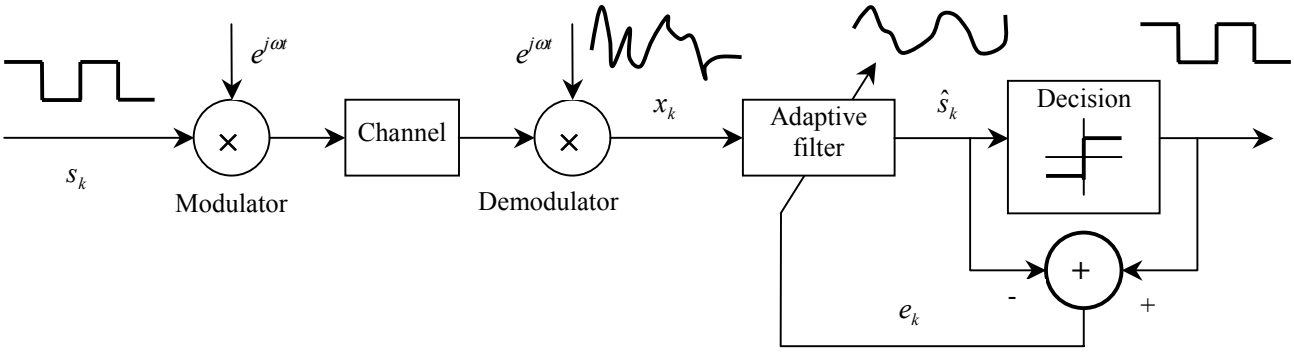


Figure 7.1(c)

(a) Calculate the k^{th} signal sample: $\hat{s}_k = \sum_{i=0}^{N-1} w_k(i) \cdot x_{k-i}$

(b) Update the filter weights for the next sample: $w_{k+1}(i) = w_k(i) + 2\mu e_k x_{k-i}$, for $i = 0$ to $N-1$

7.3 Wiener Filter Theory

The starting point for deriving the equations for the adaptive filter, is to define very clearly what we mean by an *optimum* filter. The Wiener filter is probably the most common definition in use, and it relates to the configuration depicted in Figure 7.2. The k^{th} sample of signal y, y_k, consists of two components: the principal signal s_k, and a noise component n_k which is correlated with x_k. The Wiener filter provides an optimal estimate of n_k, known as $\hat{n}_k$.

Now let us suppose that the Wiener filter is an FIR filter with N coefficients, the estimated error signal e_k is found by subtracting the Wiener filter noise estimate $\hat{n}_k$ from the input signal y_k, this is mathematically defined by Equation 7.1 below.

$$e_k = y_k - \hat{n}_k = y_k - \sum_{i=0}^{N-1} w(i) \cdot x_{k-i} \tag{7.1}$$

where $w(i)$ is the i^{th} coefficient of the Wiener filter. Since we are dealing with discrete values, the input signal and Wiener filter coefficients can be represented in matrix notation. Such that:

$$\mathbf{X}_k = \begin{bmatrix} x_k \\ x_{k-1} \\ \vdots \\ x_{k-(N-1)} \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} w(0) \\ w(1) \\ \vdots \\ w(N-1) \end{bmatrix} \tag{7.2}$$

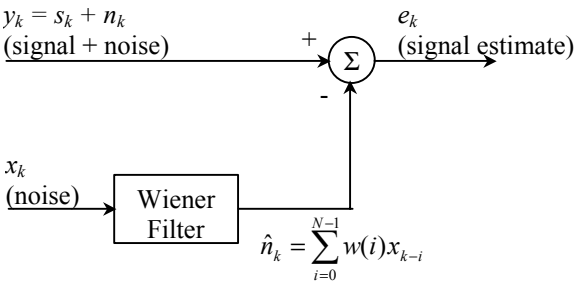


Figure 7.2: The Wiener filter configuration.

By substituting for the matrix notation into Equation 7.1, it is possible to represent the estimated error signal by Equation 7.3 below.

$$e_k = y_k - \mathbf{W}^T \mathbf{X}_k = y_k - \mathbf{X}_k^T \mathbf{W} \tag{7.3}$$

The instantaneous *squared error* of the signal can be found by squaring Equation 7.3 such that it can be represented as the following equation:

$$e_k^2 = y_k^2 - 2\mathbf{W}^T (y_k \mathbf{X}_k) + \mathbf{W}^T \mathbf{X}_k \mathbf{X}_k^T \mathbf{W} \tag{7.4}$$

The *mean square error* (MSE), ξ, is defined by the "expectation" of the squared error, from Equation 7.4. Hence the MSE can be represented by Equation 7.5.

$$\xi = E[e_k^2] = E[y_k^2] - 2\mathbf{W}^T E[y_k \mathbf{X}_k] + \mathbf{W}^T E[\mathbf{X}_k \mathbf{X}_k^T] \mathbf{W} \tag{7.5}$$

The mean square error function can be more conveniently expressed by replacing the term $E\left[\mathbf{X}_k \mathbf{X}_k^T\right]$ in Equation 7.5 with the *Autocorrelation Matrix* $\mathbf{R}_{\mathbf{XX}}$. In this example, we have illustrated the format of the matrix for when $N = 4$ coefficients:

$$\mathbf{R}_{\mathbf{XX}} = E\left[\mathbf{X}_k \mathbf{X}_k^T\right] = \begin{bmatrix} x_k \\ x_{k-1} \\ x_{k-2} \\ x_{k-3} \end{bmatrix} \begin{bmatrix} x_k & x_{k-1} & x_{k-2} & x_{k-3} \end{bmatrix}$$

$$= E\begin{bmatrix} x_k x_k & x_k x_{k-1} & x_k x_{k-2} & x_k x_{k-3} \\ x_{k-1} x_k & x_{k-1} x_{k-1} & x_{k-1} x_{k-2} & x_{k-1} x_{k-3} \\ x_{k-2} x_k & x_{k-2} x_{k-1} & x_{k-2} x_{k-2} & x_{k-2} x_{k-3} \\ x_{k-3} x_k & x_{k-3} x_{k-1} & x_{k-3} x_{k-2} & x_{k-3} x_{k-3} \end{bmatrix}$$

$$= \begin{bmatrix} r_{xx}[0] & r_{xx}[1] & r_{xx}[2] & r_{xx}[3] \\ r_{xx}[1] & r_{xx}[0] & r_{xx}[1] & r_{xx}[2] \\ r_{xx}[2] & r_{xx}[1] & r_{xx}[0] & r_{xx}[1] \\ r_{xx}[3] & r_{xx}[2] & r_{xx}[1] & r_{xx}[0] \end{bmatrix} \tag{7.6}$$

where $r_{xx}[m]$ is the m^{th} autocorrelation coefficient (denoted as $\phi_{xx}[m]$ in Chpater 6). In addition, the $E\{y_k \mathbf{X}_k\}$ term in Equation 7.5 can also be replaced with the *Cross-correlation Matrix* $\mathbf{R}_{y\mathbf{X}}$, as illustrated by Equation 7.7 below. Again, we have illustrated the format of the matrix for when $N = 4$ coefficients.

$$\mathbf{R}_{y\mathbf{X}} = E\left[y_k \mathbf{X}_k\right] = E\begin{bmatrix} y_k x_k \\ y_k x_{k-1} \\ y_k x_{k-2} \\ y_k x_{k-3} \end{bmatrix} = \begin{bmatrix} r_{yx}[0] \\ r_{yx}[1] \\ r_{yx}[2] \\ r_{yx}[3] \end{bmatrix} \tag{7.7}$$

It is now possible to express the MSE of Equation 7.5 in a simpler manner by substituting in Equation 7.6 and Equation 7.7, so that:

$$\xi = E\{e_k^2\} = E\{y_k^2\} - 2\mathbf{W}^T \mathbf{R}_{y\mathbf{X}} + \mathbf{W}^T \mathbf{R}_{\mathbf{XX}} \mathbf{W} \tag{7.8}$$

It is clear from this expression that the mean square error ξ is a *quadratic function* of the weight vector **W** (filter coefficients). That is, when Equation 7.8 is expanded, the elements of **W** will appear in the first and second order only. This is valid when the input components and desired response inputs are wide-sense stationary stochastic (random) variables.

7.4 Performance Surface

A portion of a typical two-dimensional mean square error function is illustrated in Figure 7.3 below. The vertical axis represents the mean square error and the two horizontal axes represent the values of two filter coefficients. The quadratic error function, or *performance surface*, can be used to determine the optimum weight vector $\mathbf{W}_{\text{opt}}$ (or Wiener filter coefficients). With a quadratic performance function there is only one global optimum; no local minima exist. The shape of the function would be hyper-parabolic if there were more than two weights.

In this example, the filter coefficient $w(0)$ varies between [1,…,3] while $w(1)$ varies between the range [-1,…,1]. The optimum weight vector is given by $\mathbf{W}_{opt} = [2, -0.1]$, corresponding to the values for which the mean square error takes the minimum value, ξ_{min}.

Many adaptive processes that cause the weight vector to search for the minimum of the performance surface do so by the *gradient method*. The gradient of the mean square error of the performance surface, designated ∇, can be obtained by differentiating Equation 7.8 with respect to each component of the weight vector.

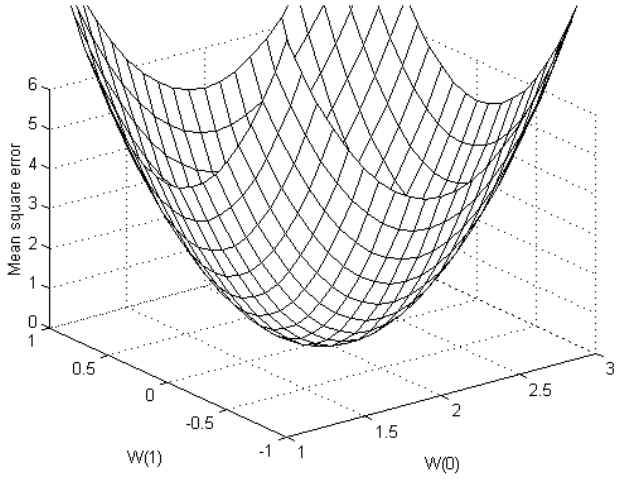


Figure 7.3: A two-dimensional quadratic performance surface.

$$\nabla = \frac{\partial \xi}{\partial \mathbf{W}} = \begin{bmatrix} \frac{\partial \xi}{\partial w(0)} \\ \frac{\partial \xi}{\partial w(1)} \\ \vdots \\ \frac{\partial \xi}{\partial w(N-1)} \end{bmatrix} \tag{7.9}$$

Remember that the MSE was derived from the expectation of the squared error function, from Equation 7.5. So as an alternative method, the gradient can also be found by differentiating the expected squared error function with respect to the weight vector.

$$\nabla = E\left\{\frac{\partial e_k^2}{\partial \mathbf{W}}\right\} = E\left\{2e_k \frac{\partial e_k}{\partial \mathbf{W}}\right\} = 2E\left\{\left(y_k - \mathbf{X}_k^T\mathbf{W}\right)\frac{\partial}{\partial \mathbf{W}}\left(y_k - \mathbf{X}_k^T\mathbf{W}\right)\right\}$$

$$\nabla = -2E\left\{\left(y_k - \mathbf{X}_k^T\mathbf{W}\right)\mathbf{X}_k\right\}$$

$$\nabla = -2E\{\mathbf{X}_k y_k\} + 2E\{\mathbf{X}_k\mathbf{X}_k^T\}\mathbf{W}$$

$$\nabla = -2\mathbf{R}_{y\mathbf{X}} + 2\mathbf{R}_{\mathbf{XX}}\mathbf{W} \tag{7.10}$$

When the weight vector (filter coefficient) is at the optimum $\mathbf{W}_{\text{opt}}$, the mean square error will be at its minimum. Hence, the gradient ∇ will be zero, i.e. $\nabla = 0$. Therefore, setting Equation 7.10 to zero we get:

$$0 = -2\mathbf{R}_{y\mathbf{X}} + 2\mathbf{R}_{\mathbf{XX}}\mathbf{W}_{opt}$$

$$\mathbf{W}_{opt} = \mathbf{R}_{\mathbf{XX}}^{-1}\mathbf{R}_{y\mathbf{X}} \tag{7.11}$$

This equation is known as the *Wiener-Hopf* equation in matrix form, and the filter given by $\mathbf{W}_{opt}$ in Equation (7.11) is the Wiener filter. However, in practice it is not usual to evaluate In addition, $\mathbf{W}_{opt}$ has to be calculated repeatedly for non-stationary signals and this can be computationally intensive. An iterative solution to the Wiener-hopf equation is the steepest decent algorithm.

7.5 The Steepest Descent Algorithm

In practice it is not usual to calculate the optimum filter $\mathbf{W}_{opt}$ using Equation (7.11) directly. The problem is that the evaluation of $\mathbf{R}_{\mathbf{XX}}^{-1}$ involves the inversion of a matrix of dimension N by N which is computationally very complex.

Furthermore, if the signal statistics are non-stationary, which is quite often the case, then the calculation has to be undertaken periodically in order to track the changing conditions.

An alternative method of calculation is therefore the *steepest descent* algorithm. In this method the weights are adjusted iteratively in the direction of the gradient.

$$\mathbf{W}_{p+1} = \mathbf{W}_p - \mu\nabla_p \tag{7.12}$$

where $\mathbf{W}_p$ is the weight vector after the p^{th} iteration, and ∇_p is the gradient vector after the p^{th} iteration evaluated by substituting $\mathbf{W}_p$ into Equation (7.10). The parameter μ is a constant that regulates the step size and therefore controls stability, and the rate of convergence.

7.6 The LMS Algorithm

The least mean square (LMS) algorithm is extremely popular because of its simplicity and ease of computation. The LMS algorithm is based on the steepest descent method, but simplifies it further by undertaking just one iteration per sample, and by calculating only an *estimate* of the gradient vector – but a new one $\hat{\nabla}_k$ at each sample k.

The estimate of the gradient vector at the k^{th} sample, $\hat{\nabla}_k$, can be derived from the definition of the error, from Equation 7.3:

$$e_k = y_k - \mathbf{X}_k^T\mathbf{W}$$

$$\hat{\nabla}_k = \begin{bmatrix} \dfrac{\partial e_k^2}{\partial w(0)} \\ \dfrac{\partial e_k^2}{\partial w(1)} \\ \vdots \\ \dfrac{\partial e_k^2}{\partial w(N-1)} \end{bmatrix} = 2e_k \begin{bmatrix} \dfrac{\partial e_k}{\partial w(0)} \\ \dfrac{\partial e_k}{\partial w(1)} \\ \vdots \\ \dfrac{\partial e_k}{\partial w(N-1)} \end{bmatrix} = -2e_k\mathbf{X}_k \tag{7.13}$$

This estimate of the gradient can be substituted into Equation 7.12 to obtain:

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu e_k\mathbf{X}_k \tag{7.14}$$

which can also be expressed as:

$$w_{k+1}(i) = w_k(i) + 2\mu e_k x_{k-i}, \text{ for } \quad i = 0 \text{ to } N-1 \tag{7.14a}$$

This expression is more commonly known as the LMS algorithm. As before, the parameter μ is a constant that regulates the stability and the rate of convergence. With respect to Equation 7.14, the LMS algorithm is extremely easy to implement in hardware because it does not involve squaring, averaging or differentiation.

7.6.1 Implementation of the LMS Algorithm

The computational steps for the LMS algorithm are as follows:

1. Initialise the filter coefficients $w_k(i)$ to zero.

2. At each sampling period:

   (a) Compute the filter output: $\hat{n}_k = \sum_{i=0}^{N-1} w_k(i) \cdot x_{k-i}$

   (b) Calculate the error estimate: $e_k = y_k - \hat{n}_k$

(c) Update the new filter weights: $w_{k+1}(i) = w_k(i) + 2\mu e_k x_{k-i}$, for $i = 0$ to $N-1$

In matrix form this can be expressed as: $\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu e_k \mathbf{X}_k$

7.6.2 Convergence Properties

The performance of the LMS algorithm is illustrated in Figure 7.4 below. Contours of ξ are shown in the figure and the optimum filter coefficients for this particular example are $w(0) = 3.784$ and $w(1) = -4.178$. Illustrated in Figure 7.4 are two weight value tracks for the LMS algorithm, which have the characteristics, as depicted in Table 7.1 below:

	Start weight values: $w(0)$, $w(1)$	Value of μ	N^{os} of iterations k
Upper track	0, 0	0.10	250
Lower track	4, -10	0.05	500

Table 7.1: LMS algorithm characteristics.

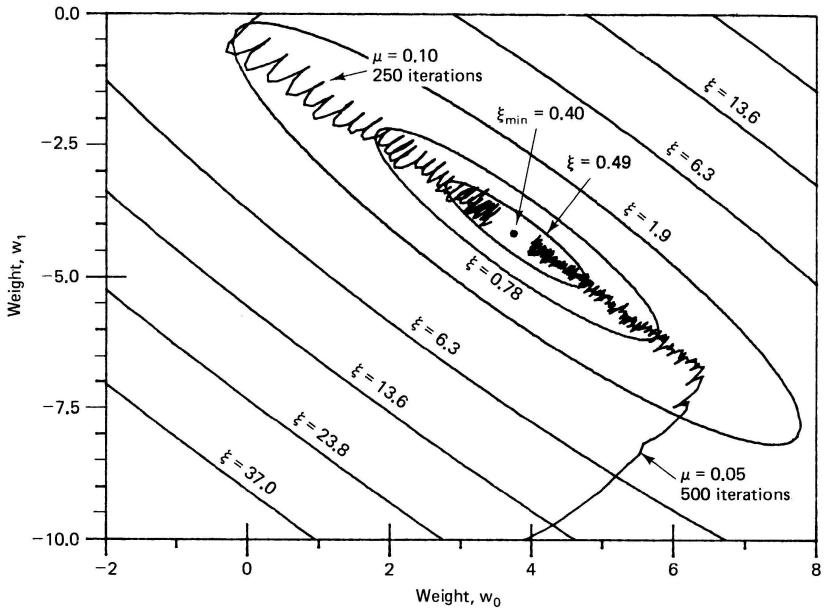


Figure 7.4: Performance surface contours and weight value tracks for the LMS [Widrow 85].

The track that has the largest value of μ appears to be more erratic because the weight of the adjustment is greater at each iteration step, but it arrives at the same distance from ξ_{min} for half the number of iterations taken for the track with the smaller value of μ.

The choice of μ is very important as it controls the rate of convergence. If the value is too small, it may take too long to converge towards ξ_{min}. On the other hand, the value of μ must not be too large as it will not be able to reach ξ_{min} because it will cause the weights to jump around their optimum value; a common problem called misadjustment. In general, the LMS algorithm will converge to the Wiener filter providing it satisfies the following boundary conditions; where λ_{max} is the maximum eigenvalue of the input data covariance matrix.

$$0 < \mu < \frac{1}{\lambda_{max}} \tag{7.15}$$

7.7 Adaptive Filter Configurations

The adaptive filter can be used in a wide variety of applications. Most of these can be classified into one of three types.

7.7.1 Linear Prediction

The prediction configuration in Figure 7.5(a) is perhaps the simplest. The desired signal is the input signal, *s*, and a delayed version is sent to the adaptive filter, which must try to *predict* the current input signal in order to drive the error *e*, towards zero. The prediction configuration is often used in signal encoding and noise reduction techniques.

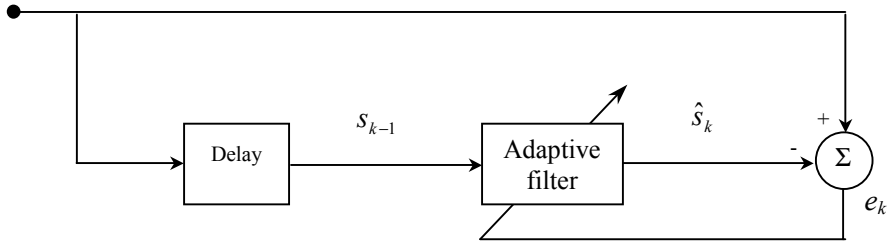


Figure 7.5(a) Prediction

7.7.2 Direct System Modelling

In the direct system modelling application in Figure 7.5(b), the input signal x_k is the input to both the adaptive filter as well as to an *unknown system*. To reduce the error e_k the adaptive processor tries to emulate the system's transfer characteristic. After adaptation the system is modelled in the sense that its transfer function is the same as the adaptive processor. Adaptive system modelling can be used to evaluate a model for a system whose input and output signals are available. Typical applications also include echo cancellation and noise cancellation, as illustrated in sections 7.2.1 and 7.2.2.

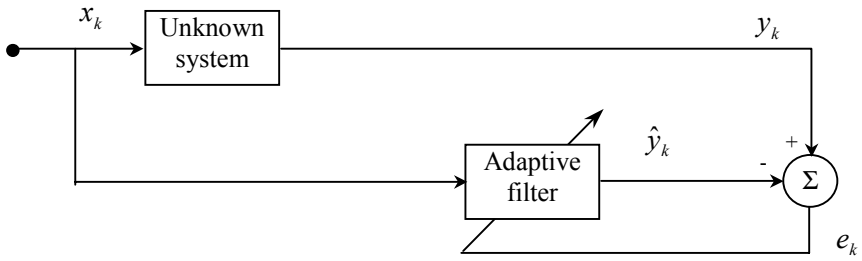


Figure 7.5(b) Direct system modelling

7.7.3 Inverse System Modelling

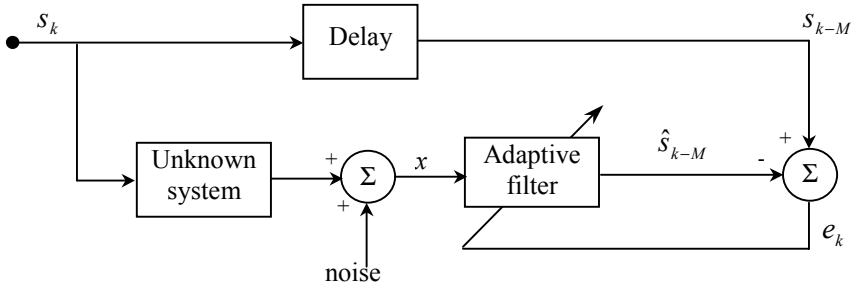


Figure 7.5(c) Inverse system modelling

The inverse system modelling configuration in Figure 7 (c) works by the adaptive processor recovering a delayed version of the signal, s_k, which is assumed to have been altered by the unknown system and to contain additive noise. The delay in the figure is to allow for the propagation through the unknown system and the adaptive processor itself. Inverse system modelling can be used to undo the effects of a communication channel on a signal, or it can be used to

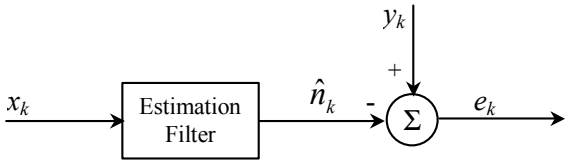
produce an inverse model of an unknown system. This type of configuration is what is used in the *adaptive equalisation* example of section 7.2.3.

7.8 Numerical Examples of Wiener Filter

Example 1: A signal estimation problem is illustrated in the diagram below, where the observed input sequence is x_k and the desired (ideal) signal is y_k, such that:

$$x_k = \sin(2\pi k/4) + \eta_k$$

$$y_k = 2\cos(2\pi k/4)$$



η_k is a noise sequence of power = 0.1, and the estimation filter is of order 2 (i.e. it has two coefficients).

Calculate: 1). The 2 x 2 autocorrelation matrix $\mathbf{R_{xx}}$

2). The 1 x 2 cross-correlation matrix $\mathbf{R}_{y\mathbf{x}}$

3). The optimum Wiener filter coefficients for this case.

1).

$$\mathbf{R_{xx}} = E\left[\mathbf{X}_k \mathbf{X}_k^T\right] = E\begin{bmatrix} x_k^2 & x_k x_{k-1} \\ x_k x_{k-1} & x_{k-1}^2 \end{bmatrix} \qquad \text{k = 0, 1}$$

$$E\left[x_k^2\right] = E\left[x_{k-1}^2\right] = E[\sin\frac{2\pi k}{4} + \eta_k]^2 = [\frac{1}{2} + 0.1] = 0.6$$

$$E\left[x_k x_{k-1}\right] = E\left[\left(\sin\frac{2\pi k}{4} + \eta_k\right)\cdot\left(\sin\frac{2\pi(k-1)}{4} + \eta_{(k-1)}\right)\right] = 0$$

Hence by inserting these values into the autocorrelation matrix, we have:

$$\mathbf{R_{xx}} = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$$

2).

$$\mathbf{R}_{y\mathbf{x}} = E\left[y_k \mathbf{X}_k\right] = E\begin{bmatrix} y_k x_k \\ y_k x_{k-1} \end{bmatrix} \qquad \text{k = 0, 1}$$

$$E\left[y_k x_k\right] = E\left[2\cos\frac{2\pi k}{4}\cdot\sin\frac{2\pi k}{4}\right] = 0$$

$$E\left[y_k x_{k-1}\right] = E\left[2\cos\frac{2\pi k}{4}\cdot\sin\frac{2\pi(k-1)}{4}\right] = -1$$

N.B. The expectation of a random noise sequence is zero, so it can be removed from the calculation. Hence, by inserting these values into the cross-correlation matrix, we have:

$$\mathbf{R}_{y\mathbf{X}} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

3).

$$\mathbf{W}_{opt} = \mathbf{R}_{\mathbf{XX}}^{-1}\mathbf{R}_{y\mathbf{X}}$$

$$\mathbf{R}_{\mathbf{XX}}^{-1} = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}^{-1} = \frac{1}{0.6^2}\begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix} = \begin{bmatrix} 1.67 & 0 \\ 0 & 1.67 \end{bmatrix} =$$

$$\mathbf{W}_{opt} = \begin{bmatrix} 1.67 & 0 \\ 0 & 1.67 \end{bmatrix}\begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1.67 \end{bmatrix}$$

Hence, the optimum coefficients relating to the Wiener filter in this example are $w(0) = 0$ and $w(1) = -1.67$.

Example 2: A signal is characterised by the following properties, where $\mathbf{r}_{x\mathbf{x}}$ is an autocorrelation vector and $\mathbf{R}_{\mathbf{XX}}$ and $\mathbf{R}_{\mathbf{nn}}$ are autocorrelation matrices.

$$\mathbf{r}_{x\mathbf{y}} = \mathbf{r}_{x\mathbf{x}}$$

$$\mathbf{R}_{\mathbf{YY}} = \mathbf{R}_{\mathbf{XX}} + \mathbf{R}_{\mathbf{nn}}$$

If the first three values of $\mathbf{r}_{x\mathbf{x}}$ and $\mathbf{r}_{n\mathbf{n}}$ are (0.7, 0.5, 0.3) and (0.4, 0.2, 0.0) respectively, evaluate $\mathbf{r}_{xy}$ and $\mathbf{R}_{\mathbf{YY}}$.

1).

$$\mathbf{r}_{x\mathbf{y}} = \mathbf{r}_{x\mathbf{x}} = \begin{bmatrix} 0.7 \\ 0.5 \\ 0.3 \end{bmatrix}$$

2).

$$\mathbf{R}_{\mathbf{XX}} = E\left[\mathbf{X}_k\mathbf{X}_k^T\right] = \begin{bmatrix} r_{xx}[0] & r_{xx}[1] & r_{xx}[2] \\ r_{xx}[1] & r_{xx}[0] & r_{xx}[1] \\ r_{xx}[2] & r_{xx}[1] & r_{xx}[0] \end{bmatrix} = \begin{bmatrix} 0.7 & 0.5 & 0.3 \\ 0.5 & 0.7 & 0.5 \\ 0.3 & 0.5 & 0.7 \end{bmatrix}$$

$$\mathbf{R}_{\mathbf{nn}} = E\left[\mathbf{N}_k\mathbf{N}_k^T\right] = \begin{bmatrix} r_{nn}[0] & r_{nn}[1] & r_{nn}[2] \\ r_{nn}[1] & r_{nn}[0] & r_{nn}[1] \\ r_{nn}[2] & r_{nn}[1] & r_{nn}[0] \end{bmatrix} = \begin{bmatrix} 0.4 & 0.2 & 0 \\ 0.2 & 0.4 & 0.2 \\ 0 & 0.2 & 0.4 \end{bmatrix}$$

$$\mathbf{R}_{\mathbf{YY}} = \mathbf{R}_{\mathbf{XX}} + \mathbf{R}_{\mathbf{nn}}$$

$$\mathbf{R}_{\mathbf{XY}} = \begin{bmatrix} 1.1 & 0.7 & 0.3 \\ 0.7 & 1.1 & 0.7 \\ 0.3 & 0.7 & 1.1 \end{bmatrix}$$