# Adaptation

We have already learned about many different types of systems. We started with frequency selective filters and filters designed for their time-domain properties. Then we saw nonfilters that had capabilities that filters lack, such as PLLs that can lock onto desired frequency components. Next we saw how to match a filter to a prespecified signal in order to best detect that signal. We have even glimpsed higher-order signal processing systems that can differentiate between signals with identical power spectra. Yet all these systems are simple in the sense that their design characteristics are known ahead of time. Nothing we have studied so far can treat problems where we are constantly changing our minds as to what the system should do.

In this chapter we briefly discuss adaptive filters, that is, filters that vary in time, adapting their coefficients according to some reference. Of course the term 'adaptive *filter*' is a misnomer since by definition filters must be time-invariant and thus cannot vary at all! However, we allow this shameful usage when the filter coefficients vary much more slowly than the input signal.

You may think that these adaptive filters would be only needed on rare occasions but in practice they are extremely commonplace. In order to understand how and why they turn up we disregard our usual custom and present three applications before tackling the more general theory. These applications, noise cancellation, echo cancellation, and equalization turn out to have a lot in common.

After this motivation we can introduce the more general problem, stressing the connection with the Wiener-Hopf equations. Direct solution of these equations is usually impossible, and so we will learn how to iteratively approximate a solution using the Widrow-Hoff equations and the LMS algorithm. We then briefly present several of the variants to vanilla LMS, and the alternative RLS algorithm.

### 10.1 Adaptive Noise Cancellation

A lecture is to be recorded using a microphone placed at some distance from the lecturer. It is a hot summer day and the lecture hall is packed; a large air-conditioning unit is running noisily, and the fluorescent fixtures are emitting a low buzzing noise. As the lecturer begins to speak the crowd hushes and a tape-recorder starts to record. What exactly is being recorded?

Were we to listen to the recording we would certainly hear the lecturer, but we would soon notice other sounds as well. Fluorescent lamp noise is spectrally localized at harmonics of the AC supply frequency and if truly annoying could be filtered out using techniques we have discussed previously. The air-conditioner sounds and the background talking from the audience are not as easy to remove. They are neither spectrally localized nor stationary in character. Humans are extremely good at 'tuning out' such noises, but our brains use filtering based on content, a difficult feat to duplicate. Is there a practical way to remove these interferences from the recording?

Let's focus on the air-conditioner noise, although the audience's babble could be similarly treated. We propose using a second microphone placed near the air-conditioner so that it picks up mainly its noise and not the speaker's voice. Now since the first microphone is picking up the sum of two signals (the desired speech and the air-conditioner noise) we need to *subtract* the air-conditioner noise signal as picked up by the second microphone from the first signal. If done correctly the speech signal alone will remain.

Simplifying for the sake of presentation, we will assume that the second microphone hears the air-conditioner noise  $q_n$  alone. The lecturer's microphone signal  $y_n$  contains both the desired speech signal  $x_n$  and the air-conditioner noise. However,  $y_n$  will not be simply the sum  $x_n + q_n$  for at least two reasons. First, the amplitude of the air-conditioner noise at the lecturer's microphone will most probably be weaker than that of the microphone directly in front of the unit. Second, the speed of sound is finite, and thus the air-conditioner noise as detected at the lecturer's microphone is delayed as compared to the close microphone. This delay is far from negligible; for example, assume the lecturer's microphone is 15 meters from that of the air-conditioner, take the speed of sound to be 300 meters per second, and let's sample at 48 kilosamples per second. Using these numbers it takes 50 milliseconds for the sound to travel from the air-conditioner microphone to the lecturer's, a delay that corresponds to 2,400 samples! Thus, at least as a rough approximation we believe that

$$y_n = x_n + hq_{n-k} \tag{10.1}$$

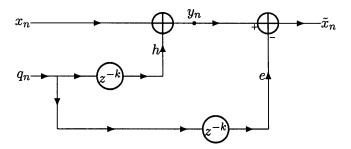


Figure 10.1: Cancelling delayed and attenuated noise by subtracting.

with  $k \approx 2400$  and h < 1. Of course the delay need not be an integer number of samples, and indeed in a closed room we will get multiple noise echoes due to the sound waves bouncing off the walls and other surfaces. Each such echo will arrive at a different time and with a different amplitude, and the total effect is obtained by adding up all these contributions. We will return to the effect of multiple echoes later.

Let's try to regain the desired clean lecturer's voice signal from the noisy received signal  $y_n$  and the reference signal  $q_n$ . Let's assume at first that we know the delay k having measured the distance between the microphones, but have no information regarding the gain h. We can try to subtract out the interference

$$\tilde{x}_n = y_n - eq_{n-k} \tag{10.2}$$

with  $\tilde{x}_n$  representing our attempt at recovering  $x_n$ . This attempt is depicted in Figure 10.1, using a self-explanatory graphical technique to be presented more fully in Chapter 12. We know that this *could* work; were we to know h we could set e = h and

$$\tilde{x}_n = y_n - eq_{n-k} = (x_n + hq_{n-k}) - hq_{n-k} = x_n$$

as required; but since we don't know h we have to find e. When e is improperly chosen we get the desired signal plus a residual interference,

$$\tilde{x}_n = y_n - eq_{n-k} = x_n + (h - e)q_{n-k} = x_n + r_{n-k}$$
(10.3)

with the amplitude of the residual  $r_n$  depending on the value of e.

In order to find e we will make the assumption that the speech signal  $x_n$  and the interference signal  $q_n$  (delayed by any amount) are not correlated. By uncorrelated we mean that the correlation between  $x_n$  and  $q_{n-l}$ , as measured over a certain time interval,

$$C_{xq}(l) = \sum_{n} x_n q_{n-l}$$

is zero for every lag l. This is a reasonable assumption since correlation would imply some connection between the signals that links their values. We believe that the air-conditioner doesn't care what the lecturer is saying, and indeed would be making essentially the same noise were the lecturer not to have started speaking. Now it is true that when the compressor kicks in and the air-conditioner becomes suddenly louder the lecturer might start speaking more loudly, causing some correlation between the speech and the noise, but this is a very slow and weak effect. So we shall assume for now that  $x_n$  and  $q_n$  are uncorrelated.

How does this assumption help us? The lack of correlation is significant because when we sum uncorrelated signals their energies add. Think of taking two flashlights and shining them on the same spot on a wall. It is clear from the conservation of energy that the energy of the spot is the sum of each flashlight's energy. You may recall seeing experiments where two light beams combine and destructively interfere leaving darkness, but for this to happen the beams must be *correlated*. When the light beams are *uncorrelated* their energies add, not their amplitudes, and the same is true for sounds. In large rooms there may be places where echoes constructively or destructively interfere, making localized spots where sounds can be heard from afar or mysteriously disappear; but this is because different echoes of the *same* sound *are* correlated.

Returning to  $\tilde{x}_n = x_n + r_{n-k}$ , since  $r_n$  is  $q_n$  to within a multiplicative constant,  $x_n$  and  $r_n$  are also uncorrelated. Thus the energy of our recovered  $\tilde{x}_n$  signal is the sum of the energy of the original  $x_n$  and that of the residual  $r_n$ . However, the energy of the residual is dependent on our estimate for the coefficient e; the residual has large energy when this estimate is poor, but when we are close to the proper value the residual's energy is close to zero. Of course the energy of  $x_n$  is not affected by our choice of e. Thus we can minimize the energy of the sum signal  $\tilde{x}_n$  by correctly choosing the coefficient e!

To see this mathematically, we write the energy of  $\tilde{x}_n$ 

$$E_{\tilde{x}} = \sum_{n} \tilde{x}_{n}^{2} = \sum_{n} (x_{n} + r_{n-k})^{2} = \sum_{n} x_{n}^{2} + 2 \sum_{n} x_{n} r_{n-k} + \sum_{n} r_{n-k}^{2}$$

but the cross term is precisely lag k of the correlation between  $x_n$  and  $r_n$  that was assumed to be zero.

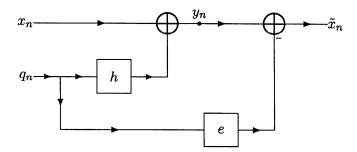


Figure 10.2: Cancellation of filtered noise by subtraction. The filter  $e_k$  is adapted to equal the distorting filter  $h_k$ . When successfully adapted the output of  $e_k$  equals that of  $h_k$  so that the interference is subtracted from the desired signal.

Continuing

$$E_{\tilde{x}} = \sum_{n} x_n^2 + \sum_{n} r_{n-k}^2 = \sum_{n} x_n^2 + (h - e)^2 \sum_{n} q_{n-k}^2 = E_x + E_q (h - e)^2$$

which as a function of e, is a parabola, with its minimum corresponding to  $E_x$ , the energy of the speech signal.

So to find the proper coefficient e all we need to do is to vary it until we find the minimal energy of the reconstructed signal. Since the energy is a parabola there is a single global minimum that is guaranteed to correspond to the original lecturer's voice.

Now, what can we do if the delay k is unknown? And what if the delay is not a integer number of samples? We might as well consider the more general problem of many different paths from the air-conditioner to the lecturer's microphone that all combine with different k and k. In such a case we have

$$y_n = x_n + \sum_{k} h_k q_{n-k} (10.4)$$

which we recognize as corresponding to the adding of a filtered version of the air-conditioner noise  $q_n$  to the desired signal. We try to recover  $x_n$  by looking for the unknown filter

$$\tilde{x}_n = y_n - \sum_k e_k q_{n-k} \tag{10.5}$$

as depicted in Figure 10.2. Once again we are assured that this can be successful, since selecting  $e_k = h_k$  will guarantee  $\tilde{x}_n = x_n$ . Viewed in this light, the problem of noise removal is equivalent to the finding of an unknown filter, with the filter coefficients possibly varying in time.

Following the same path as before we find that due to the assumption of lack of correlation between  $x_n$  and  $q_n$ , the energy of the attempted reconstruction is the sum of two parts.

$$E_{\tilde{x}} = \sum_{n} x_n^2 + \sum_{n} \left( \sum_{k} (h_k - e_k) q_{n-k} \right)^2$$

The first is the energy of the desired signal  $x_n$  and the second is the energy of the residual interference. As a function of the vector of coefficients, the energy  $E(e_1, e_2, \ldots e_N)$  is a hyperparaboloid with a single global minimum to be found. Once again this minimum corresponds to the desired signal.

How does one find this minimum in practice? When there was only a single coefficient e to be found, this was a relatively easy job. For example, we could start with any arbitrary e and then try moving along the e axis by some positive or negative amount. If the energy decreases then we keep moving in the same direction; otherwise we move in the opposite direction. If after several steps that decrease the energy, it starts to rise again, then we have gone too far; so we reduce the step size and 'home in' on the minimum.

The more general case can also be solved by arbitrarily moving around and checking the energy, but such a strategy would take a long time. With one variable there were just two directions in which to move, while with N coefficients there are an infinite number of directions. However, since we know that the energy surface in  $e_k$  space is a hyperparaboloid, we can (with only a little extra work) make a good guess regarding the best direction. The extra work is the calculation of the gradient of the energy in  $e_k$  space,  $\nabla E(e_1, e_2, \dots e_N)$ . Recall that the gradient of a surface is the multidimensional extension of the derivative. The gradient of a function is a vector that points in the direction the function increases most rapidly, and whose length is proportional to the steepness of the function. At a maximum or minimum (like the base of the energy paraboloid) the gradient is the zero vector. Were we to be interested in finding a maximum of the energy, the best strategy would be to move in the direction of the gradient. Any other direction would not be moving to higher energy values as quickly. In order to find the energy's minimum we have to reverse this strategy and move in the direction opposite the gradient. This technique of finding a minimum of a function in N-dimensional space is called steepest descent or gradient descent, and will be more fully explained in Section 10.5.

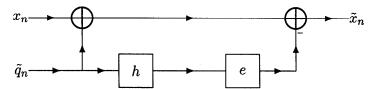


Figure 10.3: Cancelling filtered noise by an inverse filter (equalizer). This time the filter  $e_k$  is adapted to equal the inverse of the distorting filter  $h_k$ . When successfully adapted the output of filter e equals the input of h so that the interference is subtracted from the desired signal.

Before concluding this section we wish to note an alternative solution to the noise cancellation problem. We could have considered the basic noise signal to be that which is added at the lecturer's microphone, and the noise picked up by the reference microphone to be the filtered noise. According to this interpretation the problem is solved when the constructed filter approximates the *inverse filter*, as depicted in Figure 10.3. The desired signal is recovered due to the noise going through a filter and its inverse in series and then being subtracted. Both direct and inverse interpretations are useful, the best one to adopt depending on the application.

#### **EXERCISES**

- 10.1.1 Unlike the air-conditioner, the audience is not located at one well-defined location. Can the audience noise be removed in a manner similar to the air-conditioner noise?
- 10.1.2 Build a random signal and measure its energy. Add to it a sinusoid and measure the resulting energy. Did the energies add? Subtract from the combined signal the same sinusoid with varying amplitudes (but correct phase). Graph the energy as a function of amplitude. What curve did you get? Keep the correct amplitude but vary the phase. Is the behavior the same?
- 10.1.3 Electrocardiographs are required to record weak low-frequency signals and are often plagued by AC line frequency pickup (50 or 60 Hz). Were there are no desired signal components near this frequency a sharp notch filter would suffice, however generally an adaptive technique should be employed. Since we can directly measure the AC line sinusoid, the problem is reduced to finding the optimum gain and phase delay. Explain how to solve this problem. Simulate your solution using a stored waveform as the desired signal and a slowly amplitude- and phase-varying sinusoid as interference.
- 10.1.4 A 'frequency agile notch filter' can remove periodic interference (of unknown frequency) from a nonperiodic desired signal without a separate reference signal. Explain how this can be done.

## 10.2 Adaptive Echo Cancellation

Communications systems can be classified as one-way (simplex) or two-way (full-duplex); radio broadcasts and fax machines are of the former type, while telephones and modems are of the latter. Half-duplex systems, with each side transmitting in turn, lie in between; radio transceivers with push-to-talk microphones are good examples of this mode. True two-way communications systems are often plagued by echo, caused by some of the signal sent in one direction leaking back and being received by the side that transmitted it. This echo signal is always delayed, usually attenuated, and possibly filtered.

For telephones it is useful to differentiate between two types of echo. Acoustic echo is caused by acoustic waves from a loudspeaker being reflected from surfaces such as walls and being picked up by the microphone; this type of echo is particularly annoying for hands-free mobile phones. A device that attempts to mitigate this type of echo is called an acoustic echo canceller. Line echo is caused by reflection of electric signals traveling along the telephone line, and is caused by imperfect impedance matching. The most prevalent source of line echo is the hybrid, the device that connects the subscriber's single two-wire full-duplex telephone line to the four-wire (two simplex) channels used by the telephone company, as depicted in Figure 10.4. We will concentrate on line echo in this section.

Actually, telephones purposely leave some echo to sound natural, i.e., a small amount of the talker's voice as picked up at the handset's microphone is intentionally fed back to the earpiece. This feedback is called 'sidetone' and if not present the line sounds 'dead' and the subscriber may hang up. If there is too little sidetone in his telephone, John will believe that Joan barely hears his voice and compensates by speaking more loudly. When this happens Joan instinctively speaks more softly reinforcing John's impression that he is speaking too softly, resulting in his speaking even more loudly. If there is too much sidetone in Joan's telephone, she will speak more softly causing John to raise his voice, etc.

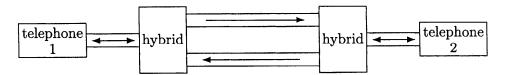


Figure 10.4: The telephone hybrid. At both ends of the telephone connection are two wire channels, but in between the conversation is carried over four-wire circuits.

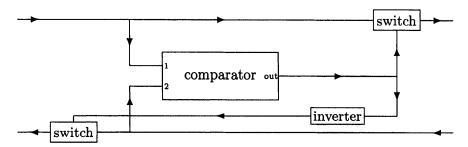


Figure 10.5: The telephone echo suppressor. The top line represents one direction of the four wire telephone channel, and the bottom line the other. When the upper signal is greater than the lower one the comparator gives positive output, thus keeping the upper path open but suppressing the lower signal. When the lower signal is greater the comparator output is negative and the switches open the lower path but shut the upper.

When the delay of line echo is short, it simply combines with the sidetone and is not noticeable. However, when the delay becomes appreciable line echo becomes quite annoying. Most people find it disconcerting to hear their own voice echoing back in their ear if the delay is over 30 milliseconds. An echo suppressor is a simple device that combats line echo by disconnecting one side of the conversation while the other side is talking. The functioning of an echo suppressor is clarified in Figure 10.5. Echo suppressors often cause conversations to be carried out as if the telephone infrastructure were half-duplex rather than full-duplex. Such conversations are unnatural, with each side lecturing the other without interruption, rather than engaging in true dialog. In addition, echo suppressors totally disrupt the operation of data communications devices such as faxes and modems, and must be disabled before these devices can be used. A Line Echo Canceller (LEC) is a more complex device than an echo suppressor; it enables full-duplex conversations by employing adaptive DSP algorithms.

How does an echo canceller work? Like the adaptive noise canceller, the basic idea is that of subtraction; since we know the original signal that has been fed back, we need only subtract it out again. However, we need to know the delay, attenuation, and, more generally, the filter coefficients before such subtraction can be carried out.

Full-duplex modems that fill all of the available bandwidth and use a single pair of wires for both directions always experience echo. Indeed the echo from the nearby modulator may be as strong as the received signal, and demodulation would be completely impossible were it not to be removed effectively. Hence a modem must remove its own transmitted signal from the received signal before attempting demodulation.

Modems typically determine the echo canceller parameters during a short initialization phase before data is transferred. Consider the following common technique to measure the delay. The modem on one side sends an agreed-upon event (e.g., a phase jump of 180° in an otherwise unmodulated sinusoid) while the other side waits for this event to occur. As soon as the event is detected the second modem sends an event of its own (e.g., a phase reversal in its sinusoid), while the first waits. The time the first modem measures between its original event and detecting the other modem's event is precisely the round-trip delay. Similarly, the finding of the filter coefficients can be reduced to a system identification problem, each side transmitting known signals and receiving the filtered echo. While the system identification approach is indeed useful, its results are accurate only at the beginning of the session; in order to remain accurate the echo canceller must continuously adapt to changing line conditions. For this reason modem echo cancellers are initialized using system identification but thereafter become adaptive.

Returning to telephone conversations, it is impractical to require humans to start their conversations with agreed-upon events (although starting with 'hello' may be almost universal), but on the other hand the requirements are not as severe. You will probably not notice hearing an echo of your own voice when the delay is less than 20 milliseconds, and international standards recommend controlling echo when the round-trip delay exceeds 50 milliseconds. This 50 milliseconds corresponds to the round-trip propagation delay of a New York to Los Angeles call, but modern digital networks introduce processing delay as well, and satellite links introduce very annoying half-second round-trip delays. Even when absolutely required voice echo cancellers needn't remove echo as completely as their modem counterparts and are allowed to be even less successful for a short amount of time at the beginning of the conversation.

In the late 1970s the phone companies introduced phone network LECs, an implementation of which is depicted in Figure 10.6. Its philosophy is exactly opposite that of the modem's internal echo canceller discussed above. It filters the signal arriving over the phone network from the far-end (the reference) and subtracts it from the near-end signal to be sent out to the network, aspiring to send only clean echo-free near-end speech. Echo is completely controlled by placing LECs at both ends of the four-wire network.

Figure 10.6 is not hard to understand. After the hybrid in the local telephone company office, the signal to be sent is digitized in order to send it to its destination over the phone system's digital infrastructure. Before the signal is sent out it undergoes two processes, namely subtraction of the echo estimate and NonLinear Processing (NLP).

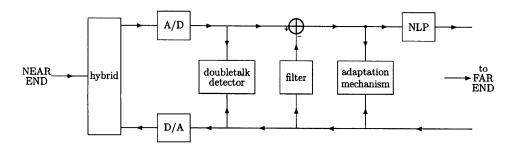


Figure 10.6: A digital telephone network line echo canceller (LEC). In this diagram only signal flow lines are indicated; invisible are the logic indications sent from the double-talk detector to the adaptation mechanism and NLP, and the fact that the adaptation mechanism sets the filter coefficients and the NLP threshold.

The filter processor places digital samples from the far-end into a static buffer (called the 'X register' in LEC terminology), convolves them with the filter (called the H register), and outputs the echo estimate to be subtracted from the near-end samples.

The adaptation mechanism is responsible for adapting the filter coefficients in order to reproduce the echo as accurately as possible. Assume that the far-end subscriber is talking and the near-end silent. In this case the entire signal at the input to the subtracter is unwanted echo generated by the nearby hybrid and the near-end telephone. Consequently, the adaptation mechanism varies the filter coefficients in order to minimize the energy at the output of the subtracter (the place where the energy is measured is marked in the figure). If the far-end is quiet the adaptation algorithm automatically abstains from updating the coefficients.

When the double-talk detector detects that both the near-end and farend subscribers are talking at the same time, it informs the adaptation mechanism to freeze the coefficients. The Geigel algorithm compares the absolute value of the near-end speech plus echo to half the maximum absolute value in the filter's static buffer. Whenever the near-end exceeds the far-end according to this test, we can assume that only the near-end is speaking.

The nonlinear processor (NLP) is a center clipper (see equation (8.7)), that enables the LEC to remove the last tiny bit of perceived echo. For optimal functioning the center clipping threshold should also be adapted.

Although the LEC just described is somewhat complex, the basic filter is essentially the same as that of the adaptive noise canceller. In both cases a filtered reference signal is subtracted from the signal we wish to clean up, and in both cases the criterion for setting the coefficients is energy minimization. These two characteristics are quite general features of adaptive filters.

#### **EXERCISES**

- 10.2.1 Why is an acoustic echo canceller usually more complex than an LEC?
- 10.2.2 Why is the phone network LEC designed to cancel echo from the transmitted signal, rather than from the received signal?
- 10.2.3 Describe the following performance criteria for echo cancellers: convergence speed, ERLE (echo return loss enhancement), and stability (when presented with narrow-band signals). The minimum performance of acoustic echo cancellers is detailed in ITU-T standard G.167, and that of LECs in G.165 and G.168. Research, compare, and contrast these standards.
- 10.2.4 Assume that each tap of the echo cancelling FIR filter takes a single instruction cycle to calculate, that each coefficient update takes a single cycle as well, and that all the other elements are negligible. Estimate the maximum and typical computational complexities (in MIPs) required to echo cancel a standard voice channel (8000 samples per second) assuming a 16-millisecond 'tail' in which echoes can occur.
- 10.2.5 Explain the Geigel algorithm for double-talk detection. Why isn't it sufficient to compare the present near-end to a single far-end value? Why compare to half the maximum far-end? How does it differ from the comparator in the echo suppressor? How can it be improved?

# 10.3 Adaptive Equalization

As a third and final example of adaptive signal processing we will consider adaptive equalization of digital communications signals. We previously defined an *equalizer* as a filter that counteracts the unwanted effects of another filter. For communications signals (to be treated in Chapter 18) this invariably means trying to overcome destructive effects of the communications channel; this channel being universally modeled as a filter followed by addition of noise, as depicted in Figure 10.7.

In general the equalizer cannot overcome noise, and so the optimal equalizer is the inverse filter of the channel. Recall from the previous section how modems calculate their echo cancellers; in similar fashion they use system identification techniques during an initialization phase in order to learn the channel and hence the optimum equalizer. Adaptive equalization is needed thereafter to track changes in the channel characteristics.

Is channel equalization really needed? Let's consider the simplest possible digital communications signal, one that takes on one value for each 0 bit

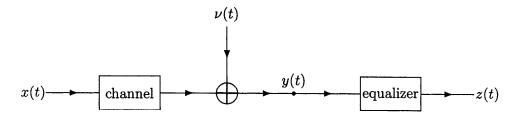


Figure 10.7: An equalizer for digital communications signals. The original signal x(t) transmitted through the communications channel, and subject to additive noise  $\nu(t)$ , is received as signal y(t). The purpose of the equalizer is to construct a signal z(t) that is as close to x(t) as possible.

to be transmitted, and another for each 1 bit. These transmitted values are referred to as 'symbols', and each such symbol is transmitted during a symbol interval. Ideally the signal would be constant at the proper symbol value during each symbol interval, and jump instantaneously from symbol to symbol; in reality it is sufficient for the signal value at the center of the symbol interval to be closer to the correct symbol than to the alternative. When this is the case the receiver, by focusing on times far from transitions, can make correct decisions as to the symbols that were transmitted.

When the modem signal traverses a channel it becomes distorted and the ability of the receiver to properly retrieve the original information is impaired. This effect is conventionally tested using the *eye pattern* (see Figure 10.8). The eye pattern is constructed by collecting multiple traces of the signal at the output of the equalizer. When the 'eye is open' information retrieval is possible, but when the 'eye is closed' it is not. In terms of the eye pattern, the purpose of an equalizer is to open the eye.

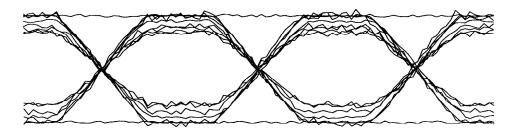


Figure 10.8: The eye pattern display graphically portrays the effect of ISI, noise and possibly other impairments on the receiver's capability to properly decode the symbol. In the present diagram the eye is 'open' and proper decoding is possible.

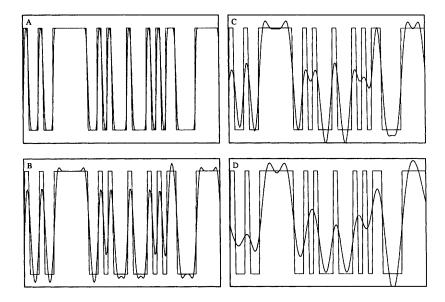


Figure 10.9: The effect of increasing intersymbol interference. The filtered channel output is superposed over the original signal. In (A) (the mildest channel) the received signal is close to the ideal signal. In (B) the bandwidth has been reduced and symbol recovery has become harder. In (C) proper symbol recovery is not always likely. In (D) (the harshest channel) symbol recovery has become impossible.

Why do channels cause the eyes to close? Channels limit the bandwidth of signals that pass through them, and so ideal symbols will never be observed at the channel output. Mild channels merely smooth the symbol-to-symbol jumps, without impairing our ability to observe the proper symbol value far from transitions, but channels with long impulse responses smear each symbol over many symbol intervals, as seen in Figure 10.9. As a result the channel output at any given time is composed not only of the desired symbol, but of contributions of many previous symbols as well, a phenomenon known as InterSymbol Interference (ISI). When the ISI is strong the original information cannot be recovered without equalization.

At first glance the adaptation of an equalizer would seem to be completely different from the applications we discussed in previous sections. In the previous cases there was an interfering signal that contaminated the signal of interest; here the source of contamination is the signal itself! In the previous cases there was a reference signal highly correlated to the contaminating signal; here we observe only a single signal! Notwithstanding

these apparent differences, we can exploit the same underlying principles. The trick is to devise a new signal (based on our knowledge of the original signal) to play the role of the reference signal.

Assuming that the equalizer was initially acquired using system identification techniques, we can presume that the receiver can make proper decisions regarding the symbols that were transmitted, even after some drift in channel characteristics. If proper decisions can be made we can reconstruct a model of the originally transmitted signal and use this artificially reconstructed signal as the reference. This trick is known as Decision Directed Equalization (DDE). Using DDE makes adaptive equalization similar to adaptive noise cancellation and adaptive echo cancellation.

#### **EXERCISES**

- 10.3.1 An alternative to equalization at the receiver as illustrated in Figure 10.7 is 'Tomlinson equalization', where the inverse filter is placed at the transmitter. What are the advantages and disadvantages of this approach? (Hints: What happens if the channel's frequency response has zeros? How can the equalizer be adapted?)
- 10.3.2 DDE is not the only way to adapt an equalizer. Blind equalization uses general characteristics of the signal, without making explicit decisions. Assume the symbol for a 0 bit is -1 and that for a 1 bit is +1. How can the fact that the square of both symbols is unity be used for blind equalization? Describe a blind equalizer for a constant amplitude signal that encodes information in its phase.
- 10.3.3 Signal separation is a generalization of both equalization and echo cancellation. The task is to separate the signal mixtures and recover the original signals. Let  $x_i$  be the original signals we wish to recover, and  $y_i$  the observed combination signals. The most general linear two-signal case is

$$y_1 = h_{11} * x_1 + h_{12} * x_2$$

$$y_2 = h_{21} * x_1 + h_{22} * x_2$$

$$(10.6)$$

where  $h_{ii}$  are the self-filters (which need to be equalized) and the  $h_{i\neq j}$  the cross-filters (which need to be echo-cancelled). Generalize this to N combinations of N signals. What conditions must hold for such problems to be solvable?

## 10.4 Weight Space

After seeing several applications where adaptive filters are commonly used, the time has come to develop the conceptual formalism of adaptive signal processing. In Section 10.1 we saw how to adapt a noise cancellation filter by minimization of energy; in this section we will see that a large family of problems can be solved by finding the minimum of a cost function. A cost function, or loss function, is simply a function that we wish to minimize. If you have to buy a new computer in order to accomplish various tasks, and the computer comes in many configurations and with many different peripherals, you would probably try to purchase the package of minimum cost that satisfies all your needs. Some people, apparently with a more positive mind-set, like to speak of maximizing gain functions rather than minimizing loss functions, but the two approaches are equivalent.

We start by reformulating the difficult FIR system identification problem of Section 6.13. Your opponent has an FIR filter v that produces a desired output signal  $d_m = \sum_{n=1}^N v_n x_{m-n}$ . We can rewrite this using a new notation that stresses the fact that the output is the weighted combination of its inputs.

$$d^{[m]} = \sum_{n=1}^{N} v_n x_n^{[m]} = \underline{v} \cdot \underline{x}^{[m]}$$
 (10.7)

We have introduced this rather unusual vector notation in order to keep our discussion as general as possible. Using the dot product we can consider d to be the output of an FIR filter, in which case  $\underline{x}$  are N consecutive values of a signal; the output of a phased array (see Section 7.9), in which case  $\underline{x}$  are values of N different signals received simultaneously by N sensors; or a two-class linearly separable pattern recognition discrimination function. In this last application there are objects, each of which has N measurable numerical features,  $x_1 \dots x_N$ . Each object belongs to one of two classes, and pattern recognition involves identifying an object's class. Two classes are called linearly separable when there is a linear function  $d(\underline{x}^{[m]}) = \sum_{n=1}^{N} v_n x_n^{[m]}$  that is positive for all objects belonging to one class and negative for all those belonging to the other.

When using this new notation the N coefficients are called 'weights', and  $\underline{v}$  a 'weight vector'. In all three cases, the adaptive filter, the adaptive beamformer, and the two-class discriminator, our task is to find this weight vector given example inputs  $\underline{x}^{[m]}$  and outputs  $d^{[m]}$ . Since this is still the system identification problem, you know that the optimum solution will be given by the Wiener-Hopf equations (6.63). However, we beg your indulgence

as our aim is to rederive these equations in a way that will be more suitable for adaptive filters.

Assume that after seeing m-1 inputs and respective desired outputs we manage to come up with some weight vector  $\underline{w}$ . Then upon observing the  $m^{\text{th}}$  example, we predict the output to be

$$y^{[m]} = \sum_{n=1}^{N} w_n x_n^{[m]} = \underline{w} \cdot \underline{x}^{[m]}$$
 (10.8)

and if the desired output is really  $d^{[m]}$  our output  $y_m$  is in error by  $\delta^{[m]} = d^{[m]} - y^{[m]}$ .

Consider now the abstract N-dimensional vector space of all possible weight vectors  $\underline{w}$ . Before our opponent allows us to observe the system all weight vectors are possible, and all points in weight space are equally plausible. After we have observed a single input-output example only a small subset of weight space remains as plausible weight vectors, since most weight vectors would produce outputs differing significantly from the observed one. We can pick any point in this subset of plausible weight vectors as our guess  $\underline{w}$ . Each successive input-output example we observe reduces the size of the subset of plausible weight vectors; indeed, were there no noise, after seeing N different examples the subset would have been reduced to a single point.

This picture is encouraging, but doesn't provide us a practical heuristic with which to find good weight vectors. To do so we now define the cost (or loss) function  $L(\underline{w})$ . This cost function is defined for every weight vector in weight space, and is simply a measure of how plausible a weight vector  $\underline{w}$  really is. A highly plausible weight vector should have a low cost, while one that noticeably violates the desired examples would be assigned a high cost. An obvious candidate for the cost function is the Mean Squared Error (MSE)

$$L(\underline{w}) = \left\langle (\delta^{[m]})^2 \right\rangle \tag{10.9}$$

the averaging being done over all the observed examples. From its definition the MSE is always nonnegative, and in the absence of noise there is a single weight vector for which the MSE is precisely zero. This weight vector is precisely the weight vector your opponent used, and by finding it you win the game. In the presence of noise there will generally not be any weight vectors with precisely zero MSE, but your best guess will be the weight vector with the Minimum Mean Squared Error (MMSE).

Now you have a strategy with which to proceed. For each example m take the input  $x^{[m]}$ , calculate the corresponding output  $y^{[m]}$  for every weight

vector in weight space according to equation (10.8), and furthermore compute the square of the error  $(\delta^{[m]})^2 = (d^{[m]} - y^{[m]})^2$ . Repeat this procedure for all the examples and compute the average error for each weight vector. In so doing you have assigned a nonnegative number to every point in weight space. You then need only look for the weight vector with the minimum cost, and you're done.

Of course it would be quite time consuming to compute this MSE cost function for *all* points in weight space, so let's use a little mathematical analysis to zoom in on the MMSE. The MSE cost function is

$$L(\underline{w}) \equiv \langle (\delta^{[m]})^2 \rangle = \langle (d^{[m]} - y^{[m]})^2 \rangle$$

$$= \langle (d^{[m]})^2 - 2d^{[m]}y^{[m]} + (y^{[m]})^2 \rangle$$

$$= \langle (d^{[m]})^2 \rangle - 2 \langle d^{[m]}y^{[m]} \rangle + \langle (y^{[m]})^2 \rangle$$
(10.10)

where the expectation  $\langle (d^{[m]})^2 \rangle$  simply means adding up all the errors and dividing by the number of examples. Substituting the basic relation (10.8) we find

$$L(\underline{w}) = \left\langle (d^{[m]})^2 \right\rangle - 2 \left\langle d^{[m]} \sum_{n} w_n x_n \right\rangle + \left\langle \sum_{n} \sum_{l} w_n w_l x_n x_l \right\rangle$$
$$= \left\langle (d^{[m]})^2 \right\rangle - 2 \sum_{n} w_n \left\langle d^{[m]} x_n \right\rangle + \sum_{n} \sum_{l} w_n w_l \left\langle x_n x_l \right\rangle$$

where the sums are all from 1 to N and the expectation on m.

The expressions in the last line have simple interpretations. The first term is the average of the square of the desired outputs; we'll call it  $D^2$ . The second term contains N crosscorrelations between each of the input components  $x_n$  and the desired output  $d^{[m]}$ ,  $C_{dx}(n) \equiv \left\langle d^{[m]}x_n\right\rangle$ . The third term contains all the input autocorrelations  $C_x(n,l) \equiv \langle x_n x_l \rangle$ . Considering the crosscorrelation to be a vector (with index n) and the autocorrelation to be a matrix (with indices n and l), we can write the following matrix equation for the cost function as a function of the weight vectors.

$$L(\underline{w}) = D^2 - 2\sum_{n} w_n (C_{dx})_n + \sum_{n} \sum_{l} w_n w_l (C_x)_{nl}$$

$$= D^2 - 2\underline{w} \cdot \underline{C_{dx}} + \underline{w} \underline{C_x} \underline{w}$$
(10.11)

To find the minimum of the cost function we need to use the gradient operator

$$\nabla \equiv \left(\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \dots, \frac{\partial}{\partial w_N}\right) \tag{10.12}$$

and set the gradient of the cost equal to zero.

$$0 = \nabla L(\underline{w}) = -2(C_{dx})_n + 2\sum_{l} (C_x)_{nl} w_l = -2\underline{C_{dx}} + 2\underline{\underline{C_x}}\underline{w}$$

Solving, we find the following set of N equations

$$(C_{dx})_n = \sum_l (C_x)_{nl} w_l$$
 i.e.  $\underline{C_{dx}} = \underline{\underline{C_x}} \underline{w}$  (10.13)

which we immediately recognize as the Wiener-Hopf equations (6.63). The solution to these equations is immediate.

$$\underline{\underline{w}}^* = \underline{\underline{C}_x}^{-1} \underline{C}_{dx} \tag{10.14}$$

To recap, given M input-output examples, we compute N input-output crosscorrelations  $(C_{dx})_n$  and  $N^2$  input autocorrelations  $(C_x)_{nl}$ . We then write down N coupled algebraic equations that can be solved for  $w_n$ . For realistically large N these equations are difficult to solve explicitly, and it is usually worthwhile to find the MMSE iteratively.

Finding the minimum of a function in high-dimensional space is a hard problem, but one that has been extensively studied. The major problem with numeric methods for finding a global minima is the fact that they tend to get stuck in local minima; in our case, the cost function in weight space defined in equation (10.11) is a quadratic function that can never become negative; as such it is always a hyperparaboloid with a single global minimum.

One family of minima (or maxima) finding methods is *iterative descent*. These methods start with some initial guess and repeatedly update this guess using

$$\underline{w}' = \underline{w} + \underline{\delta w} \tag{10.15}$$

choosing the correction term such that the cost function decreases.

$$L(\underline{w}') < L(\underline{w}) \tag{10.16}$$

If the cost function indeed decreases at each step, we must eventually arrive at a minimum.

The simplest type of iterative step is gradient descent, where the new guess is found by moving in the direction in which the cost function decreases the fastest. To do this we compute the gradient  $\nabla L(\underline{w})$ , which is

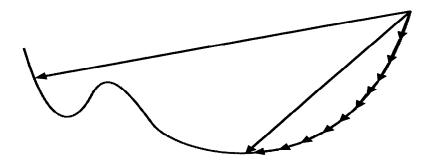


Figure 10.10: The effect of different step sizes on constant step size gradient descent. Small steps waste iterations while large steps may overshoot the minimum.

the direction in which the cost function increases most rapidly, and move in the opposite direction. More sophisticated methods exploit the matrix of second derivatives (the Hessian) as well, but even just calculating and storing the N-by-N matrix can be prohibitive in high dimensions. All of these methods require inverting the Hessian matrix, an operation that is not only computationally costly, but numerically problematic.

In the simplest type of gradient descent we move some arbitrary step size  $\rho$  at every step of the algorithm.

$$\underline{w}' = \underline{w} - \rho \frac{\nabla L(w)}{|\nabla L(w)|} \tag{10.17}$$

In general, this is often not a good idea (see Figure 10.10) since where the gradient is steep this step size may be overly small requiring us to take many small steps where one large one would have sufficed, while where the gradient is shallow we may overshoot the minimum and need to reverse direction at the next iteration. Alternatively, we can save computation by moving some fraction of the value of the gradient

$$\underline{w}' = \underline{w} - \lambda \nabla L(w) \tag{10.18}$$

which is a logical thing to do if the gradient gets larger as we go further from the minimum. There are more complex techniques that search along the line to determine how far to move (requiring much more computation), or vary the step size depending on the absolute value of the gradient or the difference between the present gradient direction and that of the previous iteration.

We have seen that the MMSE weight vector can be found explicitly via the Wiener-Hopf equations, or numerically using minimum finding techniques such as gradient descent. Both of these methods assume that the underlying system is time-invariant. When the system can constantly change we require MMSE finding methods that can dynamically adapt to these changes. The rest of the chapter will be devoted to such methods. It is an interesting coincidence of alliteration that the equations that constitute the simplest adaptive adaptation of the Wiener-Hopf equations is called the Widrow-Hoff equation.

#### **EXERCISES**

- 10.4.1 Assume that there is but a single weight w, so that the Wiener-Hopf equation is simply  $C_{dx} = w^*C_x$ . Show that the cost function as a function of this w is a simple nonnegative parabola with a single minimum. For what weight is the cost precisely zero?
- 10.4.2 Assume that there are two weights  $w_1$  and  $w_2$ . Show that the cost function surface is a paraboloid with a single minimum.
- 10.4.3 What is the computational complexity of the solution in (10.14)?
- 10.4.4 Try directly solving the Wiener-Hopf equations for the case of simple averaging (i.e., the unknown coefficients are all  $\frac{1}{N}$ ). Generate some large number of input-output pairs, compute the correlations, and use the matrix inversion technique of the previous exercise to solve. Have an opponent supply some random w and try to discover it.
- 10.4.5 Show that the MMSE weight vector decorrelates the error from the input vector, (i.e., for  $\underline{w}^*$  the error  $\delta^{[m]}$  and the input  $\underline{x}^{[m]}$  obey  $\left\langle \delta^{[m]} \underline{x}^{[m]} \right\rangle = 0$ ). What is the deeper meaning of this statement, sometimes called the *orthogonality principle*? What can be said about the error-output correlation?

#### 10.5 The LMS Algorithm

In the previous section we saw that by using gradient descent we could approximate the solution to the Wiener-Hopf equations without inverting the autocorrelation matrix. However, we still have to set aside memory and compute the autocorrelation matrix and crosscorrelation vector for some large N. We would really like to avoid these as well. Accordingly we make a further approximation; we assume that we can iteratively update the weight

vector based on each input-output example taken in isolation. In this way each time we observe a new input-output example, we make an independent estimate of the gradient, perform gradient descent, and then discard the example before the next one is presented. Of course in general such a gradient estimate may not be very good, and we will often take 'pseudogradient descent' steps in the wrong direction! Unfortunately, there is no way to avoid this, but if we take small enough steps, and observe enough input-output examples, then the majority tendency toward lower cost will eventually dominate although there will be some small steps in the wrong direction.

Now it really isn't so incredible that the gradient can be approximated by quantities that relate solely to a single input-output example. We originally defined the cost function as the average error; assuming that we are given some finite number of samples M, we could equally well have defined it as the sum of the errors, or half that sum.

$$L(\underline{w}) \equiv \frac{1}{2} \sum_{m=1}^{M} (\delta^{[m]})^2 = \frac{1}{2} \sum_{m=1}^{M} (d^{[m]} - y^{[m]})^2$$
 (10.19)

We can thus expressed the MSE cost function as the sum of M nonnegative single example terms, which can be zero only if all the individual terms are zero. As an inference the gradient of this cost function must also be the sum of single example terms! The problem is that moving  $\underline{w}$  in the direction dictated by one example, although decreasing the present contribution, may increase the contributions from other examples! In particular we may move the weight vector in order to optimize for some input-output example, and then move it right back for the next example; but when we get close enough to the global minimum everything should work out fine.

So let's investigate the single example gradient. Its  $n^{th}$  coördinate is

where we have used the chain rule, equation (10.19) and equation (10.8). Substituting this into equation (10.18) we find

$$\underline{w}^{[m]} = \underline{w}^{[m-1]} + \lambda \delta^{[m]} \underline{x}^{[m]} \tag{10.20}$$

This is the Widrow-Hoff equation. In neural network terminology it is often called the 'delta rule', referring to the  $\delta$  that figures in it so prominently. The iterative algorithm for finding best weight vector based on the Widrow-Hoff equation

```
Initialize: \underline{w}^{[0]} = 0
Loop until converged: get new input \underline{x}^{[m]} and desired output d^{[m]} compute new output y^{[m]} = \underline{w}^{[m]} \cdot \underline{x}^{[m]} calculate error \delta^{[m]} = d^{[m]} - y^{[m]} correct weight vector \underline{w}^{[m+1]} = \underline{w}^{[m]} + \lambda \delta^{[m]} \underline{x}^{[m]}
```

is called the LMS algorithm. LMS stands for Least Mean Squared, referring to our attempt at finding an MMSE solution.

Unlike our attempts at finding the MMSE in the previous section, the LMS algorithm is an *adaptive* algorithm. If the true weights  $\underline{v}$  vary slowly in time, the LMS algorithm will follow these changes, approximating at each instant the best weight vector for that time. Of course if the underlying system varies too rapidly, even an adaptive algorithm may not be able to keep up.

The LMS algorithm is by far the most popular adaptive algorithm, and the Widrow-Hoff equation appears in many contexts, although sometimes it may be hidden. There is an easy way to recognize Widrow-Hoff in disguise; all the correction terms contain the same output error term, while each weight correction term multiplies it by its own input. Remember that the complete correction term is a constant times the input times the output error.

In order to get a 'feel' for the use of the LMS algorithm, let's try a simple example. We'll take a three-dimensional case with the true weight vector  $\underline{w}^0 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ , and start with an initial guess of  $\underline{w} = (0, 0, 0)$ . Now assuming that the first input is  $\underline{x} = (1, 1, 1)$ , we'll be told that the desired output is

$$d = \underline{w}^0 \cdot \underline{x} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}) \cdot (1, 1, 1) = 1$$

while the present system outputs  $\underline{w} \cdot \underline{x} = (0,0,0) \cdot (1,1,1) = 0$ . The output error is thus  $\delta = d - y = 1$ . If we use  $\lambda = \frac{1}{4}$  the corrections will be as follows.

$$\underline{\underline{w}} \leftarrow \underline{\underline{w}} + \lambda \delta \underline{\underline{x}} = (0,0,0) + \frac{1}{4} \cdot 1 \cdot (1,1,1) = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4})$$

Let's take the same input and perform another iteration. The output is

$$\underline{w} \cdot \underline{x} = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}) \cdot (1, 1, 1) = \frac{3}{4}$$

so that the error is  $\delta = d - y = 1 - \frac{3}{4} = \frac{1}{4}$ . The new weight vector is

$$\underline{w} \leftarrow \underline{w} + \lambda \delta \underline{x} = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}) + \frac{1}{4} \cdot \frac{1}{4} \cdot (1, 1, 1) = \left(\frac{5}{16}, \frac{5}{16}, \frac{5}{16}\right)$$

with each component deviating only about 2% from the true value.

In this case two iterations were sufficient to obtain a weight vector quite close to the correct one. Of course all of the components were equal at every iteration, since both the initial guess and inputs had this symmetry. In Figure 10.11 we plot the convergence of the LMS algorithm for a slightly harder problem. The correct weight vector is as before, but we select inputs randomly, and observe the desired output in 10% uniform additive noise. We decided to use a smaller  $\lambda=0.1$  here, in order to better average out the noise and randomness. We see in the figure the three weights are no longer identical, but nevertheless remain close to each other. The convergence takes longer, partially because of the noise but mainly due to the lower  $\lambda$ , but the weights consistently approach their proper values.

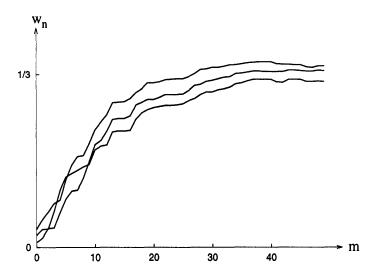


Figure 10.11: The convergence of the Widrow-Hoff algorithm. Here the correct weight vector has three components equal to  $\frac{1}{3}$ , the initial guess is zero, the inputs are random, the desired output is observed with 10% noise, and  $\lambda = 0.1$ . We see that the weights  $w_1$ ,  $w_2$ , and  $w_3$  converge slowly but are close to the proper values after  $m \approx 50$  iterations.

We just used the word 'convergence' without specifying what this means. We may mean that the difference between the derived weight vector and the true one must converge to zero, or that the MSE must converge to zero, but for the LMS algorithm we usually mean that the mean squared error should converge to some small value (nonzero due to the noise). How do we know when the weights have converged? In Figure 10.11 we see that the changes have become much less drastic after about thirty cycles, but can we really be sure that this isn't a temporary phenomenon? Around cycle 15 there was a short stretch where the weights did not change much for a few cycles, but afterward the changes returned. In practice it really can be a tricky decision, and usually the convergence criterion can be concocted only after in-depth study of the particular problem at hand.

Assuming that the LMS algorithm does indeed converge, can we be sure that it will converge to the right answer? Happily we can prove that the expectation of the weight vector approaches the MMSE solution as M increases. To show this we first slightly rewrite the  $n^{\rm th}$  component of the Widrow-Hoff equation.

$$\begin{array}{lll} w_{n}^{[m+1]} & = & w_{n}^{[m]} + \lambda \delta^{[m]} x_{n}^{[m]} \\ & = & w_{n}^{[m]} + \lambda \left( d^{[m]} - y^{[m]} \right) x_{n}^{[m]} \\ & = & w_{n}^{[m]} + \lambda \left( d^{[m]} - \sum_{l} w_{l}^{[m]} x_{l}^{[m]} \right) x_{n}^{[m]} \\ & = & \lambda d^{[m]} x_{n}^{[m]} + \sum_{l} \left( \delta_{nl} - \lambda x_{n}^{[m]} x_{l}^{[m]} \right) w_{l}^{[m]} \end{array}$$

In matrix notation we can write this

$$\underline{\underline{w}}^{[m+1]} = \lambda \underline{d}^{[m]} \underline{\underline{x}}^{[m]} + \left(\underline{\underline{I}} - \lambda \underline{\underline{x}}^{[m]} \underline{\underline{x}}^{[m]}\right) \underline{\underline{w}}^{[m]}$$
(10.21)

where the two input vectors form an outer product. Now we unfold this recursion into an iteration

$$\underline{\underline{w}}^{[m+1]} = \lambda \sum_{\mu=0}^{m} \left( \underline{\underline{I}} - \lambda \underline{\underline{x}}^{[m]} \underline{\underline{x}}^{[m]} \right)^{m-\mu} d^{[\mu]} \underline{\underline{x}}^{[\mu]} + \left( \underline{\underline{I}} - \lambda \underline{\underline{x}}^{[m]} \underline{\underline{x}}^{[m]} \right)^{m} \underline{\underline{w}}^{[0]}$$

and take the expectation of both sides.

$$\left\langle \underline{w}^{[m+1]} \right\rangle = \lambda \sum_{\mu=0}^{m} \left( \underline{\underline{I}} - \lambda \underline{\underline{C}_{x}} \right)^{\mu} \underline{C_{dx}} + \left( \underline{\underline{I}} - \lambda \underline{\underline{C}_{x}} \right)^{m} \underline{w}^{[0]}$$

This last equation, being a matrix equation, must be true in all coördinate systems, in particular in a coördinate system where the input autocorrelation matrix is diagonal. In this system  $\underline{\underline{I}} - \lambda \underline{\underline{C_x}}$  is also diagonalized. The diagonal elements  $1 - \lambda c_n$  will be less than unity in absolute value if  $\lambda$  obeys

$$0 < \lambda < \frac{2}{c} \tag{10.22}$$

where  $c = \max_n c_n$  is the maximum eigenvalue of the input autocorrelation matrix. Assuming that  $\lambda$  obeys this criterion the  $m^{\text{th}}$  power of this matrix approaches the zero matrix as m increases without limit. Hence, with this proviso on  $\lambda$ , the second term above vanishes as  $m \to \infty$ . In the same coördinate system and limit we can sum the geometric series in the first term according to (A.47)

$$\lambda \sum_{\mu=0}^{\infty} (1 - \lambda c_n)^{\mu} = \frac{\lambda}{1 - (1 - \lambda c_n)} = \frac{1}{\lambda c_n}$$

which is the inverse of the correlation matrix in these coördinates. Plugging this back in we find

$$\left\langle \underline{w}^{[m+1]} \right\rangle \to \underline{\underline{C}_x}^{-1} \underline{C_{dx}}$$
 (10.23)

which is precisely the solution to the Wiener-Hopf equations, and so the MMSE solution!

This proof is not only reassuring, it also incidentally provides the maximal step size for convergence. We noted above that we may choose a small step size because of noise, but if  $\lambda$  is chosen too low it will take ages for the weight vector to converge. In adaptive applications we may not even find the weight vector before it changes! So we wish to use as large a  $\lambda$  as possible, but no larger than dictated by equation (10.22) since otherwise the LMS algorithm may diverge. Of course only in unusual cases do we know the value of the largest input autocorrelation eigenvalue, but if it is significantly larger than the rest of the eigenvalues, we may take it to be approximately equal to the trace of the autocorrelation, namely

$$v_{max} \approx \text{Tr } \underline{C_x} = NE_x$$

where  $E_x$  is the energy of the input signal. This leads to a useful approximate range for  $\lambda$ .

$$0 < \lambda < \frac{2}{\langle E_x \rangle} \tag{10.24}$$

Up to now our discussion has been completely general; we end this section by restricting the general discussion of input-output pairs to the system identification case. By using the observed recent inputs and output of an FIR system we can combine FIR convolution and LMS adaptation, thus defining the standard adaptive FIR filter. We can make a new input-output example for *every* new time instant

etc., or we can use only some of these possibilities. By choosing examples at the maximum rate we get the most information for adaptation and track changes in the signal at the highest time resolution. However, this requires the most computational power as well.

#### **EXERCISES**

- 10.5.1 It is easy to extend the derivation of the delta rule to nonlinear combinations, the most important of which is the sigmoidal nonlinearity of equation (8.11). Show that in this case the delta rule reads  $w_n \leftarrow w_n + \lambda \delta y(1-y)x_n$ . (Hint: Use a further chain rule and exercise 8.4.5).
- 10.5.2 Assume the unknown weights of a three-parameter linear combination are  $\underline{w}^0 = (\frac{2}{3}, \frac{1}{3}, 0)$  and that the inputs are  $\underline{x} = (1, 0, 0), (0, 1, 0), (0, 0, 1)$  over and over again. Simulate this system with  $\overline{\lambda} = \frac{1}{2}$  and no noise. Try other values for  $\lambda$  and add noise. How fast can you make the LMS algorithm converge? What happens if  $\lambda$  is too large? What happens if we multiply all the inputs by 100?
- 10.5.3 Although LMS finds the best direction, its choice of constant step size seems overly primitive. A more sophisticated approach would be to search for minimal cost along the gradient direction. Compare LMS and this line-search gradient algorithm on a simulated problem. How many cycles are required for convergence? How many output and error evaluations?
- 10.5.4 Equation (10.20) seems to require two multiplications and one addition for each tap update. Show how this can be reduced. Compare the computational complexity of LMS update to that of running the FIR filter. Why do some DSP processors have an LMS instruction?
- 10.5.5 What is necessary to make the LMS algorithm work for complex-valued signals? What is the complexity compared to a real signal?

10.5.6 In exercise 10.1.3 we discussed the cancellation of power line noise from weak medical signals. Even when a narrow notch filter could be used LMS filters may require less computation. For example, assume that an ECG is a quasiperiodic with period about one Hz. For the purposes of simulation, model the ECG signal as two sinusoidal cycles one after the other, the first with period one-tenth of the whole period, and the second filling the remaining 0.9, but with amplitude one-tenth of the first. Add some white noise and a nominal power line frequency with total energy about the same as the desired signal. Try to remove the power line signal with a static FIR notch filter; how many coefficients are required? Now use an LMS filter; how many taps are required now?

## 10.6 Other Adaptive Algorithms

Although vanilla LMS is the most popular adaptive algorithm, it is certainly not the only one. There are both countless variants on the LMS theme, and also a few completely different algorithms. The LMS variants all start off with standard LMS and try to rectify some potential problem.

What problems does LMS potentially have? The need to guess the best step size, the possibly slow speed of convergence, dependence on initial conditions, and numerical instability are related but distinct problems that many variants try to resolve.

One LMS variant that frequently converges faster and that helps in the step size problem is Normalized LMS (NLMS). In the spirit of equation (10.17) we normalize the input vectors

$$\underline{\underline{w}} \leftarrow \underline{\underline{w}} + \rho \delta \frac{\underline{x}}{E_x} \tag{10.25}$$

where  $E_x$  is the input signal's energy. One way of thinking about NLMS is to cast it in standard LMS form with a normalized  $\lambda$ ;

$$\lambda \equiv \frac{\rho}{E_x}$$

accordingly NLMS is LMS with the step size tuned individually for each input's energy. In many applications NLMS converges faster than vanilla LMS. More good news about NLMS is that it converges when  $0 < \rho < 2$ , so we needn't estimate input energy or autocorrelation eigenvalues. In fact,  $\rho = 1$  is just about always best. One drawback is that NLMS requires the

additional computation of the input signal's energy and a division. Even more worrisome is the fact that for low-energy signals the division is by a small number, causing numeric problems for fixed-point implementations.

Another popular LMS variant, called *block LMS*, strives to speed convergence by smoothing out the weight vector fluctuations while still allowing a relatively large  $\lambda$ . BLMS is less computationally expensive than conventional LMS since it does not perform the actual correction for every input. Instead an averaged estimate of the gradient in weight space is computed

$$\nabla = \sum_{m} \delta^{[m]} \underline{x}^{[m]}$$

by adding up the error times input for all m in the block; then once the entire block has been seen a single correction

$$\underline{w} \leftarrow \underline{w} + \Lambda \nabla \tag{10.26}$$

is performed.

Block LMS is preferable to vanilla LMS when the input signal fluctuates rapidly, but converges more slowly for relatively stationary signals. To convince yourself of this latter fact think of a block of length M in which the signal is constant. Standard LMS will perform M separate iterations while block LMS essentially performs only the first of these.

A compromise between BLMS and vanilla LMS is LMS with momentum. In this variant we smooth the weight changes by a kind of AR filtering

$$\underline{w}^{[m+1]} = \underline{w}^{[m]} + \lambda \delta^{[m]} \underline{x}^{[m]} + \alpha \left(\underline{w}^{[m]} - \underline{w}^{[m-1]}\right)$$
(10.27)

the new term approximating the derivative of the movement in weight space. If  $\alpha = 0$  we have vanilla LMS, while for larger  $\alpha$  the new term tends to cause the weight vector to continue as in the previous iteration (hence the term 'momentum').

After seeing all these LMS variants the time has come to discuss a completely different algorithm. In deriving the LMS algorithm we wrote the MSE as a sum of single example terms. This allowed us to adapt to time-varying systems, but is not the only way to acquire this adaptability. An alternative policy is to take the MSE as the average over the last M examples seen,

$$\frac{1}{M} \sum_{m=1}^{M} |d^{[m]} - \underline{w} \cdot \underline{x}^{[m]}|^2 \tag{10.28}$$

where M is taken small enough that the underlying system does not vary appreciably during the M time instants. This policy automatically provides

a certain amount of averaging, curing one of the aforementioned potential problems. If the underlying system changes rapidly, it is more appropriate to use a recursive filter rather than a plain average. Calling the forgetting factor  $\varphi$ , we have

$$\sum_{m=1}^{M} \varphi^{M-m} |d^{[m]} - \underline{w} \cdot \underline{x}^{[m]}|^2$$

$$(10.29)$$

where we have discarded any normalization factors that will not affect the minimization. For this to be exponentially *decaying* weighting, we require  $0 < \mu < 1$ . We can use the arguments that lead up to equations (10.13) almost without change to show that the MMSE solution here is

$$\underline{w} = \underline{\underline{C}_x}^{-1} \underline{C_{dx}} \tag{10.30}$$

where the correlations appearing here are only slightly different from the usual ones.

$$\underline{\underline{C_x}} = \sum_{m=1}^{M} \varphi^{M-m} \underline{x}^{[m]} \underline{x}^{[m]}$$

$$\underline{C_{dx}} = \sum_{m=1}^{M} \varphi^{M-m} d^{[m]} \underline{x}^{[m]}$$

The useful thing about exponential weighting is that these quantities can be built up recursively.

$$\underline{\underline{C}_{\underline{x}}}^{[m]} = \varphi \underline{\underline{C}_{\underline{x}}}^{[m-1]} + \underline{\underline{x}}^{[m]} \underline{\underline{x}}^{[m]} 
\underline{\underline{C}_{d\underline{x}}}^{[m]} = \varphi \underline{\underline{C}_{d\underline{x}}}^{[m-1]} + \underline{d}^{[m]} \underline{\underline{x}}^{[m]}$$
(10.31)

Now if we only knew how to recursively update the inverse autocorrelation matrix  $\underline{P} \equiv \underline{C_x}^{-1}$  we could substitute these recursions into (10.30) to obtain a recursion for the weight vector. Luckily, there *are* ways to recursively update the inverse of a matrix of this sort. Using the matrix inversion lemma (A.100) with  $\underline{A} = \varphi \underline{C_x}^{[m-1]}$ ,  $\underline{B} = \underline{D} = \underline{x}^{[m]}$  and  $\underline{C} = 1$  we find

$$\left(\varphi \underline{\underline{C}_{\underline{x}}}^{[m-1]} + \underline{x}^{[m]}\underline{x}^{[m]}\right)^{-1} = \varphi^{-1}\underline{\underline{P}}^{[m-1]} - \frac{\varphi^{-2}\underline{P}^{[m-1]}\underline{x}^{[m]}\underline{x}^{[m]}\underline{P}^{[m-1]}}{\varphi^{-1}\underline{x}^{[m]}\underline{P}^{[m-1]}\underline{x}^{[m]} + 1}$$

which looks messy but contains only quantities known at step m of the recursion. In order to clean it up a little we define the  $gain\ vector$ 

$$\underline{k}^{[m]} \equiv \frac{P^{[m-1]}x^{[m]}}{\underline{x}^{[m]}\underline{P}^{[m-1]}\underline{x}^{[m]} + \varphi}$$
(10.32)

in terms of which we can rewrite the recursion for the inverse autocorrelation.

$$\underline{\underline{P}}^{[m]} = \varphi^{-1} \left( \underline{\underline{P}}^{[m-1]} - \underline{\underline{k}}^{[m]} \underline{\underline{x}}^{[m]} \underline{\underline{P}}^{[m-1]} \right)$$
(10.33)

Now the expression for the gain vector looks terrible, but it really has a simple interpretation. Rearranging equation (10.32)

$$\underline{\underline{k}}^{[m]} = \left(\varphi^{-1} \underline{\underline{P}}^{[m-1]} - \varphi^{-1} \underline{\underline{k}}^{[m]} \underline{\underline{x}}^{[m]} \underline{\underline{P}}^{[m-1]}\right) \underline{\underline{x}}^{[m]}$$

and from equation (10.33) we recognize the factor in the parenthesis to be precisely  $\underline{P}^{[m]}$ . As a result the gain vector

$$\underline{\underline{k}^{[m]}} = \underline{\underline{\underline{F}}^{[m]}}\underline{\underline{x}^{[m]}} = \underline{\underline{\underline{C}}\underline{x}^{[m]}}\underline{\underline{x}^{[m]}}$$
(10.34)

is the input partially decorrelated by its own inverse autocorrelation.

Now we can finally substitute the recursions (10.31) back into equation (10.30).

$$\underline{w}^{[m]} = (\underline{\underline{C}_{x}}^{[m]})^{-1}\underline{C}_{dx}^{[m]} 
= \underline{\underline{P}}^{[m]} \left(\varphi\underline{C}_{dx}^{[m-1]} + d^{[m]}\underline{x}^{[m]}\right) 
= \varphi\underline{\underline{P}}^{[m]}\underline{C}_{dx}^{[m-1]} + d^{[m]}\underline{\underline{P}}^{[m]}\underline{x}^{[m]}$$

We now substitute the recursive update of the inverse autocorrelation (10.33) and use equations (10.30) and (10.34).

$$\begin{array}{rcl} \underline{w}^{[m]} & = & \underline{P}^{[m-1]} \underline{C_{dx}}^{[m-1]} - \underline{k}^{[m]} \underline{x}^{[m]} \underline{P}^{[m-1]} \underline{C_{dx}}^{[m-1]} + d^{[m]} \underline{P}^{[m]} \underline{x}^{[m]} \\ & = & \underline{w}^{[m-1]} - \underline{k}^{[m]} \underline{x}^{[m]} \underline{w}^{[m-1]} + d^{[m]} \underline{P}^{[m]} \underline{x}^{[m]} \\ & = & \underline{w}^{[m-1]} - \underline{k}^{[m]} \left( \underline{x}^{[m]} \underline{w}^{[m-1]} - d^{[m]} \right) \end{array}$$

The final step is to recognize the error  $\delta^{[m]}$  in the parentheses and we have found the desired recursion.

$$\underline{w}^{[m]} = \underline{w}^{[m-1]} + \delta^{[m]} \underline{k}^{[m]}$$
 (10.35)

We now understand why we called k the 'gain vector'; it is a directed gain that multiplies the error in the weight update recursion.

We can at last give the Recursive Least Squares (RLS) algorithm.

```
Initialize: \underline{w}^{[0]} = 0
Loop until converged:
  get new input \underline{x}^{[m]} and desired output \delta^{[m]}
  compute new output y^{[m]} = \underline{w}^{[m]} \cdot x^{[m]}
  calculate error \delta^{[m]} = d^{[m]} - y^{[m]}
  compute gain vector \underline{k}^{[m]} using equation (10.32)
  correct weight vector using equation (10.35)
  update inverse autocorrelation \underline{P} using equation (10.33)
```

Comparing this to the LMS algorithm we see some differences but a strong similarity. Recalling equation (10.34) we can write the weight update as

$$\underline{\underline{w}}^{[m]} = \underline{\underline{w}}^{[m-1]} + \underline{\underline{P}}^{[m]} \delta^{[m]} \underline{\underline{x}}^{[m]}$$

Hence RLS can be thought of as LMS with a very intelligent adaptive step size. This step size is a matrix, and hence takes care of cost function surfaces in weight space that are steep in some directions but flat in others. The step size is optimized at every step to ensure rapid convergence.

Each iteration of the RLS algorithm is more complex than an iteration of the LMS algorithm, and indeed RLS is often impractical for real-time applications. However, the RLS algorithm will normally converge faster than the LMS one, at least when the noise is small. When there is strong additive noise a long period of averaging is necessary in order to average out the noise, and so RLS cannot significantly decrease the number of iterations needed.

As with many recursive update formulas, the RLS updates can accumulate numerical error, eventually leading to a noninvertible  $\underline{C_x}$ . This usually isn't a problem when only a few  $\underline{w}$  are needed, but becomes intolerable when we must continuously update weight vectors. One solution to this problem is to iteratively update the Cholesky decomposition of  $\underline{C_x}$  (see equation (A.94)) rather than the matrix itself. Another is the so-called QR-RLS algorithm, which multiplies the equations by an orthogonal matrix in order to keep them triangular; but further discussion of these topics would take us too far astray.

#### **EXERCISES**

- 10.6.1 Compare the LMS, NLMS, and RLS algorithms on benchmarks of your choice. Which is fastest? Which is most robust?
- 10.6.2 One way of ameliorating the numeric difficulties of NLMS is by using the following regularization.

$$\underline{w}' = \underline{w} + \rho \delta \frac{\underline{x}}{\epsilon + E_x}$$

Experiment with NLMS and regularized NLMS for signals with large dynamic range.

## Bibliographical Notes

There are many good textbooks on adaptive signal processing. A classic text is that of Widrow and Stearns [275] and a more recent text is that of Haykin [96].

The invention of adaptive filters was necessitated by the conversion of the phone to digital technologies and the consequent problems of echo cancellation and adaptive differential coding of speech. Adaptive beamforming [272] and equalization were not far behind, and the use of LMS in the related field of pattern recognition helped enrich the field.

In 1960, Bernard Widrow and one of his graduate students, Marcian (Ted) Hoff, presented the Widrow-Hoff approach [274] which is basically the LMS algorithm. Widrow went on to adapt this approach to become one of the fathers of the neural network; Hoff joined Intel and went on to become one of the fathers of the microprocessor.

The applications we presented are covered by review articles of note; adaptive noise cancelling in [273, 59], echo cancellation in [87, 178], and equalizers in [202, 203].

Adaptive equalization for digital communications was originally developed by Lucky at Bell Labs [152, 153] and was the main reason for the increase of speed of telephone modems from 2400 b/s to 9600 b/s.

The technique for equalization at the transmitter discussed in exercise 10.3.1 was developed in the early 1970s by Tomlinson and by Harashima in Japan [260, 92].