Chapter 13

Analysis and Modeling of Random Signals*

The random signals encountered in this book until now were chiefly white noise sequences. White noise sequences appeared in Section 6.5, in the context of frequency measurement, and in Section 11.7, in the context of quantization noise. We devote this chapter to a more general treatment of random signals, not necessarily white noise sequences. Randomness is inherent in many physical phenomena. Communication, radar, biomedicine, acoustics, imaging—are just a few examples in which random signals are prevalent. It is not exaggeration to say that knowledge of DSP cannot be regarded as satisfactory if it does not include methods of analysis of random signals. The methods presented in this chapter are elementary. We begin by discussing simple methods for estimating the power spectral density of a WSS random signal: the periodogram and some of its variations. Most of the chapter, however, is devoted to parametric modeling of random signals. We present general rational models for WSS random signals, then specialize to all-pole, or autoregressive models, and finally discuss joint signal modeling by FIR filters.

13.1 Spectral Analysis of Random Signals

In Chapter 6 we discussed the measurement of sinusoidal signal parameters, both without and with additive noise. Sinusoidal signals have a well-defined shape. If we know the amplitude, the frequency, and the initial phase of a sinusoidal signal, we can exactly compute its value at any desired time. Let us look, for example, at a 10-second recording of power-line voltage. Depending on where you live, it will have a frequency of 50 or 60 Hz, and its amplitude will be between $\sqrt{2} \cdot 110$ and $\sqrt{2} \cdot 240$ volts, so we will see either 500 or 600 periods. If we look at another 10-second recording, taken later, we will again see the same number of periods, with the same amplitude. Only the initial phase may be different, depending on the starting instant of the recording.

Not all signals encountered in real life are sinusoidal. In particular, we often must deal with signals that are random to a certain extent. Even the power-line signal, if examined carefully, will be seen to exhibit fluctuations of amplitude and frequency. As another example, suppose that we are interested in measuring the time variation of the height of ocean waves. We pick a spot and observe the height of the water at that spot as a function of time. Figure 13.1 shows a possible waveform of such a measurement and the magnitude of its Fourier transform (in dB). Like the power-line voltage, this waveform is oscillatory. However, its frequency and amplitude are not constant, and the overall shape bears little resemblance to a sinusoid. Moreover, if we repeat the experiment at a later time, we will record a waveform whose overall shape may resemble the one shown in Figure 13.1, but the details will most likely be different. Ocean waves are an example of a random signal.

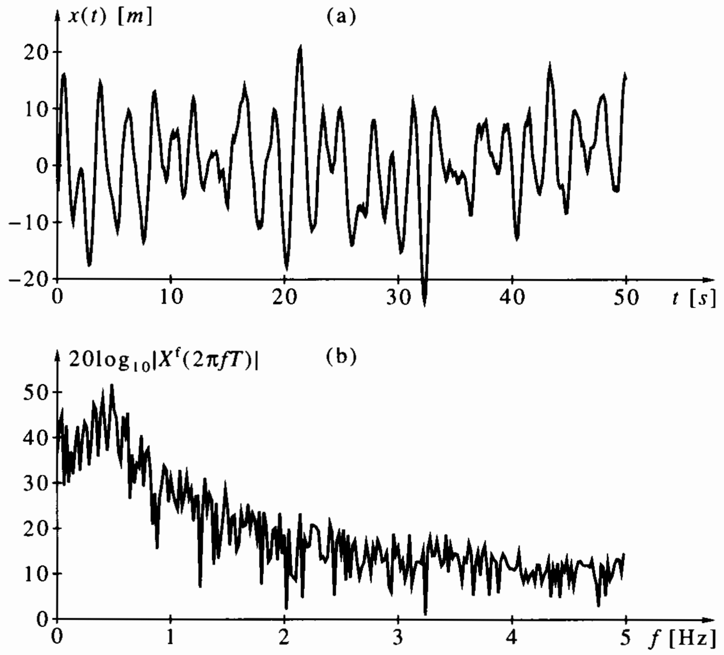

Figure 13.1 Ocean wave heights: (a) as a function of time; (b) as a function of frequency.

The method of spectral analysis we described in Chapter 6, consisting of windowing followed by examining the magnitude of the windowed DFT, is not satisfactory when applied to random signals. When the signal is random, so is its Fourier transform. Thus, the shape of the DFT will vary from experiment to experiment, limiting the information we can extract from it. Furthermore, it can be shown mathematically that increasing the length of the sampled sequence does not help; the longer DFTs show more details of the frequency response, but most of those details are random and vary from experiment to experiment. The randomness of the DFT is evident in Figure 13.1(b).

A possible solution, in case of a stationary random signal, is to arrange the samples in (relatively short) segments of equal length, compute the DFT of each segment, and average the resulting DFTs. Averaging reduces the randomness and provides a relatively smooth spectrum. The average spectrum displays the macroscopic characteristics of the signal and suppresses the random details. The more segments we use for averaging, the better the smoothing and the stronger the randomness suppression. However, averaging is effective only if we know beforehand that our signal is stationary during the entire time interval spanning the union of all segments. For example, it makes no sense to average ocean wave height measurements if some are taken when the sea is calm and some when it is stormy.

Once we understand how averaging works in principle, the question arises: Exactly what entities should we use for averaging? Averaging the DFTs themselves is not the right answer. The DFT is a complex sequence, and the relative phases of the DFTs of different segments are likely to be random. When we average complex numbers having random phases, we are most likely to end up with a number close to zero, which is not a meaningful result. To preserve the spectral information, we average the *square magnitudes* of the DFTs. The term *periodogram* is a synonym for the square magnitude of the DFT.[1] Suppose we have a total of NL data points of the random signal $x[n]$. We divide these points into L consecutive segments of length N each. Then, the *averaged periodogram* is defined as

$$\hat{K}_x^{\mathrm{f}}(\theta) = \frac{1}{L}\sum_{l=0}^{L-1}\left\{\frac{1}{N}\left|\sum_{n=0}^{N-1} x[n+lN]e^{-j\theta n}\right|^2\right\}. \tag{13.1}$$

The justification for the averaging procedure in (13.1) and for the notation $\hat{K}_x^{\mathrm{f}}(\theta)$ comes from the Wiener-Khintchine theorem, which we studied in Section 2.9. As we recall from (2.131), the expectation of each of the square magnitudes of the transforms

$$\frac{1}{N}\left|\sum_{n=0}^{N-1} x[n+lN]e^{-j\theta n}\right|^2$$

converges to the power spectral density $K_x^{\mathrm{f}}(\theta)$ as N tends to infinity. The right side of (13.1) is the empirical mean of the individual square magnitudes of the transforms; therefore we may hope that if both L and N are allowed to approach infinity, the empirical mean will converge to the theoretical mean and we will get[2]

$$\lim_{\substack{N\to\infty\\ L\to\infty}} \hat{K}_x^{\mathrm{f}}(\theta) = K_x^{\mathrm{f}}(\theta) \tag{13.2}$$

In practice, the simple averaged periodogram defined in (13.1) is rarely used. Instead, we use a *windowed averaged periodogram*, defined by

$$\hat{K}_x^{\mathrm{f}}(\theta) = \frac{1}{L}\sum_{l=0}^{L-1}\left\{\frac{1}{N}\left|\sum_{n=0}^{N-1} w[n]x[n+lN]e^{-j\theta n}\right|^2\right\}, \tag{13.3}$$

where $w[n]$ is a window chosen by the user. It can be shown that, if the limit (13.2) holds for the simple averaged periodogram, it also holds for the windowed averaged periodogram. In reality, neither N nor L is infinite. Practice shows that, for finite N and L, windowing helps smoothing the randomness of $\hat{K}_x^{\mathrm{f}}(\theta)$ and gives better results. The shape of the periodogram depends not only on the window, but also on the choice of N and L. These parameters must be chosen to make their product equal to the total number of given data points. Thus, increasing L necessarily decreases N and vice versa. In general, increasing N provides more details in $\hat{K}_x^{\mathrm{f}}(\theta)$, but also increases its randomness. Increasing L makes $\hat{K}_x^{\mathrm{f}}(\theta)$ smoother, with fewer details and less randomness.

Figure 13.2 depicts the periodogram obtained by averaging 100 DFTs of simulated ocean wave measurements, of the kind shown in Figure 13.1. Each segment is 50 seconds long and contains 500 samples. The Hann window was used for all segments. The frequency scale was converted to a physical frequency in hertz. The information in the periodogram should be interpreted as follows: Its value at each frequency indicates the energy density at that frequency. Thus, in Figure 13.1, low-frequency waves are relatively strong, the strongest ones having a frequency of about 0.5 Hz. The energy density decays gradually as the frequency increases. We also notice the slight randomness of the graph, because averaging is not perfect when the number of data points is finite.

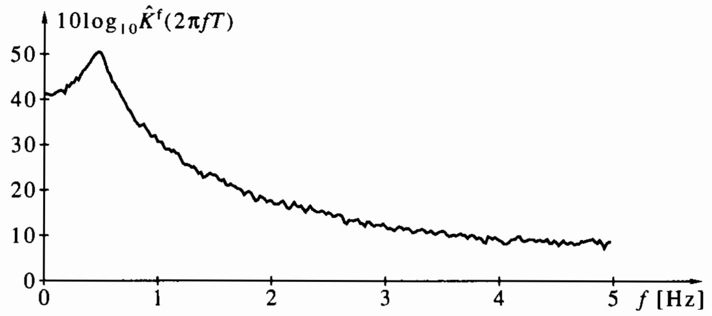

Figure 13.2 Averaged windowed periodogram of ocean wave heights.

When dividing the total number of data into segments, it is common to let the segments overlap, as shown in Figure 13.3; overlap of 50 percent is typical.

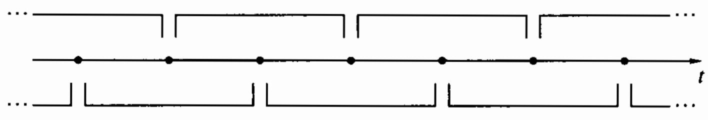

Figure 13.3 Segment overlapping for periodogram averaging.

The rationale for segment overlap is as follows: Since we window each segment, the relative weight given to the points near the ends of the segment is less than the weight given to the points near the center. When 50 percent overlap is used, points near the end of one segment will be near the center of a neighboring segment, as is evident from Figure 13.3. Therefore, all data points will be equally represented on the average. The method of averaging periodograms with windowing and overlapping was developed by P. D. Welch [1970] and is known as the *Welch periodogram*. The Welch periodogram is a common and useful method for spectral analysis of stationary random signals.

The mathematical expression of the Welch periodogram is as follows. Assume that the segment length N is even and that the total number of segments to be averaged is L. Then, with 50 percent overlap, we have

$$\hat{K}_x^{\mathrm{f}}(\theta) = \frac{1}{L}\sum_{l=0}^{L-1}\left\{\frac{1}{N}\left|\sum_{n=0}^{N-1} w[n]x[n+0.5lN]e^{-j\theta n}\right|^2\right\}. \tag{13.4}$$

The frequencies θ can be taken as integer multiples of $2\pi/N$ (usual DFT), integer multiples of $2\pi/M$ for $M > N$ (zero-padded DFT), or dense points in a small frequency interval (chirp Fourier transform).

Figure 13.4 summarizes the stages of practical spectral analysis of random signals. The continuous-time signal from the physical source is filtered to avoid aliasing and then sampled at or above the Nyquist rate. It is then divided into segments, with optional overlap. Each segment is windowed, then undergoes FFT. Variations on simple FFT include zero-padded FFT to increase display resolution and chirp Fourier transform to display only part of the frequency range. Absolute value and squaring is then performed on the output of the FFT, followed by optional averaging. As we have explained, averaging should be restricted to signals that are known to be random and stationary. In any case, the number of segments to be averaged should be chosen such that the total time interval for all segments does not exceed the stationarity time of the signal. Finally, the result is converted to dB and sent to a graphical display or other uses.

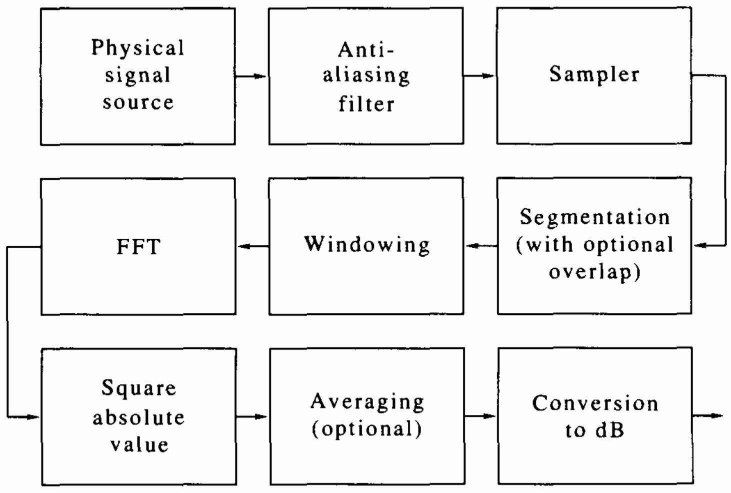

Figure 13.4 A complete short-time spectral analysis scheme.

The procedure stsa in Program 13.1 implements spectral analysis in MATLAB. It provides for optional overlapping and averaging, as well as optional zero padding or chirp Fourier transform. The window has to be supplied externally. The rows of the output matrix are the DFTs of the segments (or averages thereof, depending on the input parameters). The program does not convert the results to dB, so this must be done externally if desired.

Example 13.1 In Example 3.13 we introduced the digital communication signaling method BPSK. In this example we introduce the signaling method *offset quadrature phase-shift keying*, or OQPSK, and examine its frequency domain characteristics.

Suppose we are given a sequence of bits $b[n]$, appearing every T seconds. We split the sequences into two subsequences, one consisting of the even-index bits and the other of the odd-index bits, that is,

$$b_e[n] = b[2n], \quad b_o[n] = b[2n+1].$$

Each of the subsequences has a rate $1/2T$ and they are offset by T seconds (hence the word *offset* in the name). We generate an NRZ signal for each sequence, as in Example 3.13. We use the sequence $b_e[n]$ to modulate the phase of a cosine waveform and the sequence $b_o[n]$ to modulate the phase of a sine waveform. Finally, we subtract the two waveforms. This sequence of operations is described by the formula

$$x(t) = p_r(t)\cos(\omega_0 t) - p_i(t)\sin(\omega_0 t), \tag{13.5}$$

where

$$p_r(t) = \begin{cases} 1, & b_e[n] = 0, \\ -1, & b_e[n] = 1, \end{cases} \quad 2nT \le t < 2(n+1)T, \tag{13.6a}$$

$$p_i(t) = \begin{cases} 1, & b_o[n] = 0, \\ -1, & b_o[n] = 1, \end{cases} \quad (2n+1)T \le t < (2n+3)T. \tag{13.6b}$$

The signals $p_r(t)$ and $p_i(t)$ are depicted in Figure 13.5(a, b).

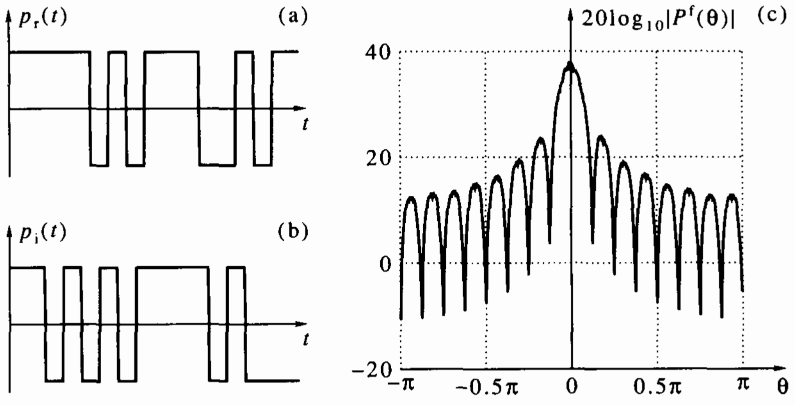

Figure 13.5 An offset quadrature phase-shift keying signal: (a) real part; (b) imaginary part; (c) spectrum.

An alternative way to express (13.5) is as

$$x(t) = \Re\{p(t)e^{j\omega_0 t}\}, \quad \text{where} \quad p(t) = p_r(t) + jp_i(t). \tag{13.7}$$

The signal $p(t)$ is called the *complex envelope* of the modulated signal. The spectrum of the complex envelope provides all the information about the frequency-domain characteristics of the modulated signal, since multiplication by $e^{j\omega_0 t}$ shifts the spectrum by ω_0, and the operation $\Re$ generates the conjugate image at $-\omega_0$.

Figure 13.5(c) illustrates the spectrum of the complex envelope of an OQPSK signal (*spectrum* herewith refers to the estimated PSD, computed by averaged periodogram). It was generated from 4096 random bits using the program stsa discussed earlier. The sampling rate is 8 times the bit duration. The segment length is $N = 512$, corresponding to 64 bits. There is no overlap between adjacent segments, and all segments (64 in total) are averaged to provide one spectral plot. As we see, OQPSK has relatively high side lobes. This feature results from the discontinuities of the modulating waveform and is typical of many digital communication signaling methods. High side lobes are undesirable, since they potentially interfere with neighboring communication channels. Communication regulations in most countries (FCC regulations in the United States) forbid high side lobes such as the ones shown in Figure 13.5(c).

A common way to reduce the spectral side lobes is to filter the signal $p(t)$ before feeding it to the RF modulator, that is, to form the signal

$$q(t) = \{p * h\}(t). \tag{13.8}$$

The filtered complex envelope $q(t)$ is then used for modulating the carrier as in (13.7). The low-pass filter $h(t)$ used for this purpose typically has bandwidth equal to that of the main lobe of $P^F(\omega)$ or slightly higher. Figure 13.6(a, b) illustrates the filtered signal and Figure 13.6(c), generated in the same way as Figure 13.5(c), shows its spectrum. The bandwidth of the filtered signal is approximately $1/2T$.

The complex envelope $p(t)$ has the property that $|p(t)| = \sqrt{2}$, so unfiltered OQPSK signal is a *constant envelope* signal. Constant envelope signals are highly desirable, since they are convenient to amplify by analog circuitry. The filtering operation has an

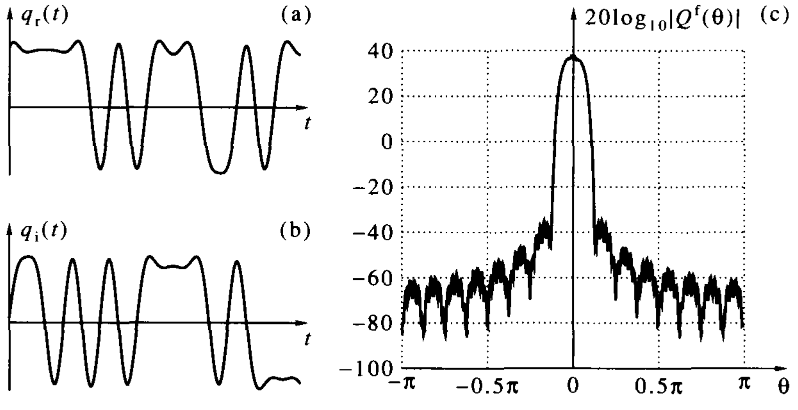

Figure 13.6 A filtered offset quadrature phase-shift keying signal: (a) real part; (b) imaginary part; (c) spectrum.

undesirable side effect, however. The constant envelope property is lost in the filtering process; the absolute value of $q(t)$ varies by a factor of about 2 between minimum and maximum.

Many communication systems, especially ones designed to be inexpensive and power efficient, hard limit the envelope signal in the process of generating the final transmitted signal. Hard limiting is represented mathematically (at least approximately) by the operation

$$r(t) = Ae^{j\measuredangle q(t)}, \tag{13.9}$$

where A is the minimum value of $|q(t)|$ (or slightly smaller), and $\measuredangle q(t)$ is the phase of $q(t)$. The signal $r(t)$ contains the same phase information as $q(t)$ (it is the phase that carries the information about the bits!), but has a constant envelope and is therefore easier to handle by the electronic circuitry. However, hard limiting introduces discontinuities to the derivative of the signal, so it inevitably increases the side-lobe level again. This phenomenon is called *spectrum regrowth* and is an undesirable side effect of hard limiting.

Figure 13.7(a, b) illustrates the hard-limited signal, and Figure 13.7(c) shows its spectrum. The spectral side lobes are now only slightly higher than those before hard limiting and in any case much lower than those of the unfiltered OQPSK signal. We conclude that an OQPSK signal has modest spectral regrowth as a result of hard limiting. This is one of the most attractive features of this signaling method. By comparison, filtering a BPSK signal followed by envelope hard limiting causes severe spectrum regrowth, which almost nullifies the filtering effect. □

13.2 Spectral Analysis by a Smoothed Periodogram

The Welch periodogram, presented in the preceding section, has limited use if the data length is relatively short, because it may be difficult to perform effective segmentation in such a case. We now introduce another spectrum estimation method for random signals—the *smoothed periodogram* method of Blackman and Tukey [1958]. The smoothed periodogram method performs smoothing of the square magnitude of the DFT in the frequency domain *without* segmentation and averaging. It is therefore useful in cases where the data sequence is short.

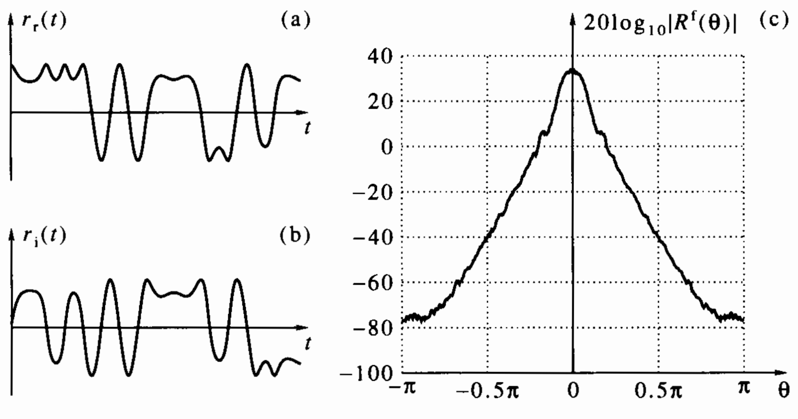

Figure 13.7 A filtered and hard-limited offset quadrature phase-shift keying signal: (a) real part; (b) imaginary part; (c) spectrum.

The smoothed periodogram of a zero-mean sequence $x[n]$ of length N is defined as

$$\hat{K}_x^f(\theta) = \frac{1}{2\pi}\int_{-\pi}^{\pi} N^{-1}|X^f(\theta-\lambda)|^2 W^f(\lambda)d\lambda, \tag{13.10}$$

where $W^f(\lambda)$ is the kernel function of an odd-length symmetric window $\{w[m], -M \le m \le M\}$. Equation (13.10) is a frequency-domain convolution between the periodogram of the sequence and the kernel function; it therefore acts to smooth the randomness of $|X^f(\theta)|^2$. Spectral lines, resulting from periodicities in $x[n]$, will be smeared as well. Therefore, we face the usual window trade-off: side-lobe level versus main-lobe width.

The convolution operation (13.10) is not convenient to perform in the frequency domain. The corresponding time-domain expression is a multiplication of the window $w[m]$ by the inverse Fourier transform of $N^{-1}|X^f(\theta)|^2$. This, in turn, can be found as follows:

$$\begin{aligned} N^{-1}|X^f(\theta)|^2 &= N^{-1}\left|\sum_{n=0}^{N-1} x[n]e^{-j\theta n}\right|^2 = N^{-1}\left(\sum_{n=0}^{N-1} x[n]e^{-j\theta n}\right)\overline{\left(\sum_{l=0}^{N-1} x[l]e^{-j\theta l}\right)} \\ &= N^{-1}\sum_{n=0}^{N-1}\sum_{l=0}^{N-1} x[n]x[l]e^{-j\theta(n-l)} = \sum_{m=-(N-1)}^{N-1} \hat{\kappa}_x[m]e^{-j\theta m}, \end{aligned} \tag{13.11}$$

where

$$\hat{\kappa}_x[m] \triangleq N^{-1}\sum_{i=0}^{N-1-|m|} x[i]x[i+|m|], \quad -(N-1) \le m \le N-1. \tag{13.12}$$

In passing to the rightmost form of (13.11), we made the substitutions

$$m = n - l, \quad i = l \text{ if } m \ge 0, \quad i = n \text{ if } m < 0.$$

Therefore, the inverse Fourier transform of $N^{-1}|X^f(\theta)|^2$ is the finite-length sequence $\{\hat{\kappa}_x[m], -(N-1) \le m \le N-1\}$. It follows that (13.10) can be expressed as

$$\hat{K}_x^f(\theta) = \sum_{m=-M}^{M} \hat{\kappa}_x[m]w[m]e^{-j\theta m}, \tag{13.13}$$

assuming that $M \le N - 1$.

The quantity $\hat{\kappa}_x[m]$ can be interpreted as an estimate of the covariance $\kappa_x[m]$, defined in (2.119). Comparing (13.13) with (2.122), we see that the smoothed periodogram $\hat{K}_x^{\mathrm{f}}(\theta)$ can be interpreted as an estimate of $K_x^{\mathrm{f}}(\theta)$, obtained by (1) replacing the true covariance sequence $\kappa_x[m]$ by the estimated covariance sequence $\hat{\kappa}_x[m]$ and (2) truncation and windowing of the estimated covariance sequence.

If the mean of the random signal $x[n]$ is not zero, we define its *estimated mean* as

$$\hat{\mu}_x = N^{-1} \sum_{n=0}^{N-1} x[n] \tag{13.14}$$

and the *estimated covariance* as

$$\hat{\kappa}_x[m] \triangleq N^{-1} \sum_{i=0}^{N-1-|m|} (x[i] - \hat{\mu}_x)(x[i+|m|] - \hat{\mu}_x). \tag{13.15}$$

Again, the smoothed periodogram is given by (13.13).

Under certain conditions on the random signal $x[n]$, $\hat{K}_x^{\mathrm{f}}(\theta)$ can be made to converge to $K_x^{\mathrm{f}}(\theta)$ as N tends to infinity, by a proper choice of the window $w[m]$ and its length M as a function of N. However, in the present application, N is relatively small, so there is no guarantee that $\hat{\kappa}_x[m]$ will be a good approximation of $\kappa_x[m]$ or that $\hat{K}_x^{\mathrm{f}}(\theta)$ will be a good approximation of $K_x^{\mathrm{f}}(\theta)$.

We reiterate the computational procedure for the smoothed periodogram of a sequence $x[n]$. First we compute $\hat{\kappa}_x[m]$, according to (13.12) if $x[n]$ is zero mean, or to (13.15) if it is not. We then multiply $\hat{\kappa}_x[m]$ by the window $w[m]$, and finally compute the DFT of the result. Basic, zero-padded, or chirp DFT can be used. The procedure `smooper` in Program 13.2 implements this computation. The program first checks that the window length is odd. It then subtracts from $x[n]$ its mean and generates the sequence $\hat{\kappa}_x[m]$ by convolving the input sequence with its reversed copy. The product of $\hat{\kappa}_x[m]$ and $w[m]$ is formed next, taking care to center the latter with respect to the former. If the length $2M+1$ of $w[m]$ is shorter than $2N-1$, the product $\hat{\kappa}_x[m]w[m]$ will contain $N-M-1$ zeros at each end. The FFT of the product is then computed. The length of the FFT is $2N-1$, out of which the first N elements are retained, corresponding to the positive frequencies. The FFT introduces linear phase, because it treats the sequence as causal, although this sequence is symmetric with respect to $m=0$. By taking the absolute value of the FFT we remove the linear phase. This program does not provide zero-padded or chirp DFT.

In practical use of the smoothed periodogram, the window length is always smaller than $2N-1$, but not much smaller, to prevent excessive smearing of periodic components that may be present. Typical values of the ratio M/N are between 0.2 and 0.5. Weak windows, such as Hann or Hamming, are more commonly used than strong windows, such as Blackman or Kaiser.

Example 13.2 The data we examine in this example—annual average sunspot numbers—are not directly related to electrical engineering, but they are important to the history of random signal analysis, since they have been used on numerous occasions to test methods of random signal analysis. Sunspot statistics are important in astronomy and radio communications.[3] Sunspot numbers vary irregularly, but approximately follow a periodic pattern, with a period of about 11 years. Figure 13.8(a) shows the annual average sunspot numbers in the years 1749–1924. These numbers are known as *Wolfer's sunspot data*; they are available in the file **sunspot.dat** (see page vi for information on how to download this file). The quasi-periodicity of the sequence is evident from the figure.

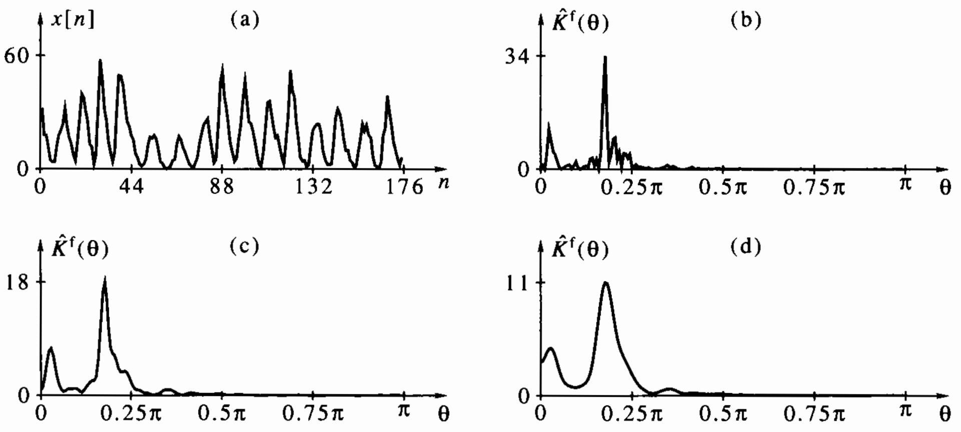

Figure 13.8 Wolfer's sunspot data: (a) annual sunspot numbers, 1749-1924; (b) periodogram; (c) smoothed periodogram, $M = 88$; (d) smoothed periodogram, $M = 44$ (parts b, c, d are drawn to different scales; maximum values shown are in thousands).

The periodogram (i.e., square magnitude of the Fourier transform) of the sunspot numbers is shown in Figure 13.8(b). The randomness of the data leads to randomness of the periodogram, but a dominant peak is apparent, at a frequency slightly below $(1/11)\,\text{year}^{-1}$. The periodogram was first used for this purpose by Schuster [1906a]. We show two smoothed periodograms of the sequences [Figure 13.8(c, d)], both using a Hamming window, with $M = 88, 44$, respectively (i.e., $1/2$ and $1/4$ of the data length). As we see, a shorter window leads to a smoother periodogram. In both cases, the dominant peak has a frequency of about $(1/11)\,\text{year}^{-1}$.

The three periodograms exhibit, in addition to the main 11-year periodicity, a second periodicity with a period of about 70 years. Since the number of data is 176, there are only slightly over two full periods of this apparent periodicity. It is therefore not entirely clear whether the additional periodicity represents a true physical phenomenon or whether it is an artifact. □

13.3 Rational Parametric Models of Random Signals

The spectrum estimation methods presented in the preceding two sections do not rely on specific assumptions on the signal $x[n]$, except that it is wide-sense stationary. Such methods are said to be *nonparametric.* A second class of spectrum estimation methods relies on a *parametric model* of the signal in question. For example, the sinusoids-in-noise model (6.74), studied in Section 6.5.3, is a parametric signal model. The parameters of this model are the amplitudes, frequencies, and initial phases $\{A_k, \theta_k, \phi_k,\ 1 \le k \le M\}$ of the sinusoidal components.

A general parametric model for a discrete-time, zero-mean, WSS random signal $x[n]$ is its expression as the output of a difference equation excited by discrete-time white noise, that is, as

$$x[n] = -a_1x[n-1] - \cdots - a_px[n-p] + b_0v[n] + b_1v[n-1] + \cdots + b_qv[n-q], \tag{13.16}$$

where $v[n]$ is zero-mean white noise with variance γ_v. The linear system represented by (13.16) is assumed to be causal and stable. The random variable $v[n]$ is assumed to be uncorrelated with past values of $x[n]$, that is,

$$E(v[n]x[n-m]) = 0 \quad \text{for all } n \text{ and all } m > 0. \tag{13.17}$$

The white noise $v[n]$ is called the *innovation* of $x[n]$.

The model (13.16) is called *rational*, or *pole-zero*, for obvious reasons. The orders p, q and the parameters $\{a_1, \dots, a_p, b_0, \dots, b_q, \gamma_v\}$ characterize the model. Two important special cases of this model are as follows:

1. When $p = 0$ we get

$$x[n] = b_0 v[n] + b_1 v[n-1] + \cdots + b_q v[n-q]. \tag{13.18}$$

   This is called an *all-zero*, or a *moving-average* model, or MA.

2. When $q = 0$ we get

$$x[n] = -a_1 x[n-1] - \cdots - a_p x[n-p] + v[n]. \tag{13.19}$$

   This is called an *all-pole*, or an *autoregressive* model, or AR.[4]

The general pole-zero model (13.16) is also called an *autoregressive moving-average* model, or ARMA.

As we recall from (2.135), the power spectral density of $x[n]$ satisfying the ARMA model (13.16) is given by

$$K_x^{\mathrm{f}}(\theta) = \gamma_v \left| \frac{b(e^{j\theta})}{a(e^{j\theta})} \right|^2, \tag{13.20}$$

where $a(z)$ and $b(z)$ are, respectively, the denominator and numerator polynomials of the transfer function corresponding to the difference equation (13.16). In the special case of a moving-average model, $a(z) = 1$, whereas in the special case of an autoregressive model, $b(z) = 1$. Because of the stability requirement on (13.16), the poles [the roots of $a(z)$] are necessarily inside the unit circle. It can be shown that, because of the requirement (13.17), the zeros [the roots of $b(z)$] must also be inside the unit circle. In other words, the rational transfer function $b(z)/a(z)$ is minimum phase, as defined in Section 8.4.7.

The *ARMA modeling problem* for a discrete-time, zero-mean WSS random signal can be stated as follows: Given a finite-length segment of the signal, say $\{x[n], 0 \le n \le N-1\}$, find a set of parameters in the difference equation (13.16) that will describe the given signal, subject to the stability and minimum-phase requirements on $a(z)$ and $b(z)$, with $v[n]$ being white noise. The MA and AR modeling problems are likewise defined. For a WSS signal whose mean μ_x is nonzero, the model in question (ARMA, AR, or MA) applies to (or is sought for) $x[n] - \mu_x$.

The following important point should be borne in mind: When modeling a physical random signal by means of a model such as (13.16), (13.18), or (13.19), usually we do not mean that the signal necessarily obeys the model exactly. More often, the model provides, or aims at providing, only an approximate description of the signal. The "goodness" of such an approximation can be measured by various criteria. A common criterion is the accuracy to which the true PSD of the signal is approximated by the rational formula (13.20). On the other hand, when *deriving properties* of a model, we always assume that the model holds exactly for the signal in question. Thus, an ARMA signal is a signal obeying (13.16) exactly, an MA signal is one obeying (13.18) exactly, and an AR signal is one obeying (13.19) exactly. ARMA, MA, and AR signals are almost never encountered as physical signals, but serve as approximate models for such signals.

Both ARMA and MA modeling are relatively complicated subjects, which are not treated further in this book. AR modeling, on the other hand, is much simpler; we discuss AR modeling in detail in the next section.

13.4 Autoregressive Signals

The autoregressive signal model (13.19) has a unique place among parametric models for discrete-time random signals. The theory of AR signals is rich and elegant, but also simple and easy to utilize. In this section we present basic methods for AR signal modeling, algorithms related to these methods, and their implementation. An extended discussion of autoregressive signals can be found in Porat [1994, Chapters 2, 5, 6, and 8], including proofs of results and facts given in this section without proofs. To avoid repetition, we will not refer to Porat [1994] explicitly each time we state a fact without a proof. Extended discussions of the material in this section can also be found in Marple [1987], Kay [1988], Therrien [1992], and Hayes [1996].

In autoregressive modeling of a random signal, we have to distinguish between two cases: (1) when we are given the covariance sequence $\kappa_x[m]$ of the signal; (2) when we are given a set of measured values of the signal, but not its covariances. We will first treat the former case and then proceed to the latter.

13.4.1 The Yule-Walker Equations

The Yule-Walker equations establish the relationships between the covariance sequence $\kappa_x[m]$ of $x[n]$ and the parameters $\{\gamma_v, a_1, \ldots, a_p\}$. Yule [1927] first introduced the AR model and applied it to Wolfer's sunspot data; see Example 13.2. Walker [1931] extended Yule's work. The Yule-Walker equations are derived as follows. Multiply (13.19) by $x[n-m]$, where $m > 0$, and take the expectation of both sides. Recall that $\kappa_x[m] = E(x[n]x[n-m])$, cf. (2.119), and use (13.17) to get

$$\kappa_x[m] = -a_1\kappa_x[m-1] - \cdots - a_p\kappa_x[m-p], \quad m > 0. \tag{13.21}$$

In particular, by collecting the equalities (13.21) for $1 \le m \le p$ and expressing them in a matrix-vector notation, we get

$$\begin{bmatrix} \kappa_x[0] & \kappa_x[-1] & \ldots & \kappa_x[-p+1] \\ \kappa_x[1] & \kappa_x[0] & \ldots & \kappa_x[-p+2] \\ \vdots & \vdots & \ldots & \vdots \\ \kappa_x[p-1] & \kappa_x[p-2] & \ldots & \kappa_x[0] \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = -\begin{bmatrix} \kappa_x[1] \\ \kappa_x[2] \\ \vdots \\ \kappa_x[p] \end{bmatrix}. \tag{13.22}$$

When we repeat the operation leading to (13.21) for $m = 0$, we get

$$\kappa_x[0] = -a_1\kappa_x[-1] - \cdots - a_p\kappa_x[-p] + E(v[n]x[n]). \tag{13.23}$$

The cross-correlation $E(v[n]x[n])$ is nonzero. To find it, multiply (13.19) by $v[n]$ and use (13.17) to get

$$E(v[n]x[n]) = E(v[n])^2 = \gamma_v. \tag{13.24}$$

Substitution of (13.24) in (13.23) gives

$$\gamma_v = \kappa_x[0] + a_1\kappa_x[-1] + \cdots + a_p\kappa_x[-p]. \tag{13.25}$$

Recalling that the sequence $\kappa_x[m]$ is symmetric, we can rewrite these equations as

$$\begin{bmatrix} \kappa_x[0] & \kappa_x[1] & \dots & \kappa_x[p-1] \\ \kappa_x[1] & \kappa_x[0] & \dots & \kappa_x[p-2] \\ \vdots & \vdots & \dots & \vdots \\ \kappa_x[p-1] & \kappa_x[p-2] & \dots & \kappa_x[0] \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = - \begin{bmatrix} \kappa_x[1] \\ \kappa_x[2] \\ \vdots \\ \kappa_x[p] \end{bmatrix}, \tag{13.26a}$$

$$\gamma_v = \kappa_x[0] + a_1\kappa_x[1] + \cdots + a_p\kappa_x[p]. \tag{13.26b}$$

Equations (13.26) are the Yule-Walker equations. The first enables the computation of the parameters $\{a_1, \dots, a_p\}$ as a function of the covariance sequence values $\{\kappa_x[0], \dots, \kappa_x[p]\}$. The second enables the computation of the innovation variance γ_v. The procedure yw in Program 13.3 solves the Yule-Walker equations.

Example 13.3 Let us solve the Yule-Walker equations for a first-order AR model. We get from (13.26a)

$$a_1 = -\frac{\kappa_x[1]}{\kappa_x[0]} \tag{13.27}$$

and

$$\gamma_v = \kappa_x[0] + a_1\kappa_x[1] = \kappa_x[0] - \frac{\kappa_x^2[1]}{\kappa_x[0]}. \tag{13.28}$$

□

13.4.2 Linear Prediction with Minimum Mean-Square Error

The preceding derivation of the Yule-Walker equations relies only on the AR model (13.19) and on the property (13.17) of the innovation $v[n]$. We now examine the AR model from a different point of view—that of linear prediction—and introduce an optimality criterion that also leads to the Yule-Walker equations.

Suppose we wish to predict the random variable $x[n]$ from the past p values of the signal $\{x[n-k],\ 1 \le k \le p\}$. For example, suppose we wish to predict today's market value of a certain stock from its values in the last p days. Let us restrict ourselves to a *linear prediction* strategy, that is, to an expression of the form

$$\hat{x}[n] = -a_1x[n-1] - \cdots - a_px[n-p], \tag{13.29}$$

where $\hat{x}[n]$ denotes the predicted value of $x[n]$ and $\{-a_1, \dots, -a_p\}$ are the coefficients of the linear combination. Let $v[n]$ be the error between the true $x[n]$ and its predicted value $\hat{x}[n]$. Then we get from (13.29)

$$v[n] = x[n] - \hat{x}[n] = \sum_{k=0}^{p} a_kx[n-k], \quad \text{where} \quad a_0 \overset{\Delta}{=} 1. \tag{13.30}$$

We propose the following criterion for the predictor $\hat{x}[n]$: Choose $\{a_1, \dots, a_p\}$ to minimize $E(v[n])^2$, the mean-square prediction error. The predictor thus obtained is called *linear MMSE predictor*, where MMSE stands for *minimum mean-square error*. The number p of past values used for prediction is called the *order* of the predictor. Each order has a set of linear MMSE predictor coefficients associated with it.

The problem of finding the values of $\{a_1, \dots, a_p\}$ that minimize $E(v[n])^2$ is similar to the one solved in Section 9.5, and is likewise solved by differentiation and equating the partial derivatives to zero. We get

$$\frac{\partial E(v[n])^2}{\partial a_l} = 2E\left(\frac{\partial v[n]}{\partial a_l}v[n]\right) = 2E\left(x[n-l]\sum_{k=0}^{p} a_k x[n-k]\right)$$
$$= 2\sum_{k=0}^{p} a_k E(x[n-l]x[n-k]) = 2\sum_{k=0}^{p} a_k \kappa_x[l-k] = 0, \quad 1 \le l \le p. \tag{13.31}$$

As we see, the resulting set of equations is identical to (13.22). Furthermore, the minimum value of $E(v[n])^2$ is given by

$$E(v[n])^2 = E\left(\sum_{k=0}^{p} a_k x[n-k]\right)\left(\sum_{l=0}^{p} a_l x[n-l]\right)$$
$$= \sum_{k=0}^{p}\sum_{l=0}^{p} a_k a_l E(x[n-k]x[n-l]) = \sum_{k=0}^{p}\sum_{l=0}^{p} a_k a_l \kappa_x[l-k]$$
$$= \sum_{k=0}^{p} a_k \kappa_x[-k] + \sum_{l=1}^{p} a_l\left(\sum_{k=0}^{p} a_k \kappa_x[l-k]\right). \tag{13.32}$$

However, by (13.31), the second term on the right side of (13.32) is zero. Therefore,

$$E(v[n])^2 = \gamma_v = \sum_{k=0}^{p} a_k \kappa_x[-k]. \tag{13.33}$$

In summary, (13.31) and (13.33) are identical to (13.22) and (13.25), respectively. The conclusion we have reached is thus:

The linear MMSE predictor of order p is characterized by coefficients $\{a_1, \ldots, a_p\}$ and error variance γ_v satisfying the Yule-Walker equations. Therefore, if the signal $x[n]$ is AR of order p, the linear predictor coefficients coincide with the model parameters, and the prediction error $v[n]$ coincides with the innovation.

We emphasize the following point: The linear MMSE predictor is defined for *any* discrete-time WSS signal, whether autoregressive or not, and is *always* obtained by solving the Yule-Walker equations. When $x[n]$ is a true AR signal of order p, and only then, the error $v[n]$ is white noise and satisfies (13.17).

The solution of the Yule-Walker equations can be used to construct a polynomial

$$a(z) = 1 + a_1 z^{-1} + \cdots + a_p z^{-p}. \tag{13.34}$$

The transfer function from the error signal $v[n]$ to $x[n]$ is

$$\frac{X^z(z)}{V^z(z)} = \frac{1}{a(z)}. \tag{13.35}$$

An important property of $a(z)$ is that it is *always stable*, whether $x[n]$ is a true AR signal of order p or not. We will not prove this property here.

13.4.3 The Levinson-Durbin Algorithm

A numerical solution of p equations in p unknowns, as needed in solving the Yule-Walker equations, requires about p^3 multiply-add operations. As we have said, solutions of different orders provide linear MMSE predictors of different orders. Often we are interested in solving the Yule-Walker equations of all orders from 1 up to a certain maximum order p. This enables us to compare different predictors and decide which one is the best for the specific application. Such a solution requires $\sum_{i=1}^{p} i^3 \approx p^4/4$

multiply-add operations. The Levinson-Durbin algorithm, which we derive in this section, solves the Yule-Walker equations of all orders $1 \le i \le p$ in approximately p^2 multiply-add operations. This is considerably less than p^3 (or $p^4/4$) if p is large. For example, if $p = 10$, the number of operations is reduced by a factor of 10 if only the tenth-order solution is required, and by a factor of 25 if the solutions of all orders up to 10 are required. Beside reducing the computational requirements, the Levinson-Durbin algorithm leads to some interesting results and applications in stability theory and digital filter realizations. We shall discuss these applications after presenting the basic theory of the Levinson-Durbin algorithm. Levinson [1947] first presented the algorithm (in a more general setting than described here), and Durbin [1960] derived the version presented here.

Since we are now dealing with AR models of different orders, let us rewrite the definition of an ith-order AR model as [cf. (13.19)]

$$x[n] = -a_{i,1}x[n-1] - \cdots - a_{i,i}x[n-i] + v_i[n], \tag{13.36}$$

where $\{a_{i,l},\ 1 \le l \le i\}$ are the coefficients of the ith-order model, and $v_i[n]$ is the ith-order prediction error. We will also use the following definitions:

1. $\boldsymbol{K}_i$: the symmetric Toeplitz matrix shown in (13.26a), but of dimensions $i \times i$ (i.e., containing values of the covariance sequence up to $\kappa_x[i-1]$).
2. $\boldsymbol{a}_i = [a_{i,1}, \ldots, a_{i,i}]'$: the solution of the ith-order Yule-Walker equation (13.26a).
3. $\boldsymbol{\kappa}_i = [\kappa_x[1], \ldots, \kappa_x[i]]'$: the right side of the ith-order Yule-Walker equation (without the minus sign).
4. $\tilde{\boldsymbol{a}}_i$: the vector $\boldsymbol{a}_i$ in reversed order.
5. $\tilde{\boldsymbol{\kappa}}_i$: the vector $\boldsymbol{\kappa}_i$ in reversed order.
6. $\boldsymbol{a}_{i+1}^* = [a_{i+1,1}, \ldots, a_{i+1,i}]'$: the vector consisting of the first i components of the solution of the $(i+1)$st-order Yule-Walker equation.
7. $a_{i+1,i+1}$: the last component of the solution of the $(i+1)$st-order Yule-Walker equation.
8. s_i: the variance $E(v_i[n])^2$ of the ith-order prediction error, given by (13.26b).

The ith-order Yule-Walker equations can be expressed as

$$\boldsymbol{K}_i\boldsymbol{a}_i = -\boldsymbol{\kappa}_i, \quad \boldsymbol{K}_i\tilde{\boldsymbol{a}}_i = -\tilde{\boldsymbol{\kappa}}_i, \quad s_i = \kappa_x[0] + \boldsymbol{\kappa}_i'\boldsymbol{a}_i = \kappa_x[0] + \tilde{\boldsymbol{\kappa}}_i'\tilde{\boldsymbol{a}}_i. \tag{13.37}$$

The first is the Yule-Walker equation for the ith-order predictor. The second is a direct consequence of the first: It follows from the fact that reversing the order of the rows and columns of the matrix $\boldsymbol{K}_i$ leaves this matrix unchanged. The third is a rewriting of (13.26b).

The Yule-Walker equation for the $(i+1)$st-order predictor is

$$\begin{bmatrix} \boldsymbol{K}_i & \tilde{\boldsymbol{\kappa}}_i \\ \tilde{\boldsymbol{\kappa}}_i' & \kappa_x[0] \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_{i+1}^* \\ a_{i+1,i+1} \end{bmatrix} = -\begin{bmatrix} \boldsymbol{\kappa}_i \\ \kappa_x[i+1] \end{bmatrix}. \tag{13.38}$$

Let us assume that we have solved the Yule-Walker equations of order i, and we wish to solve (13.38) for $\boldsymbol{a}_{i+1}^*$ and $a_{i+1,i+1}$. We can expand this equation to

$$\boldsymbol{K}_i\boldsymbol{a}_{i+1}^* + \tilde{\boldsymbol{\kappa}}_i a_{i+1,i+1} = -\boldsymbol{\kappa}_i, \tag{13.39a}$$

$$\tilde{\boldsymbol{\kappa}}_i'\boldsymbol{a}_{i+1}^* + \kappa_x[0]a_{i+1,i+1} = -\kappa_x[i+1]. \tag{13.39b}$$

Solving (13.39a) for $\boldsymbol{a}_{i+1}^*$ and using the first two equalities in (13.37) gives

$$\boldsymbol{a}_{i+1}^* = -\boldsymbol{K}_i^{-1}\boldsymbol{\kappa}_i - a_{i+1,i+1}\boldsymbol{K}_i^{-1}\tilde{\boldsymbol{\kappa}}_i = \boldsymbol{a}_i + a_{i+1,i+1}\tilde{\boldsymbol{a}}_i. \tag{13.40}$$

Substituting (13.40) in (13.39b) and solving for $a_{i+1,i+1}$ gives

$$a_{i+1,i+1} = -\frac{\kappa_x[i+1] + \tilde{\boldsymbol{\kappa}}_i' \boldsymbol{a}_i}{\kappa_x[0] + \tilde{\boldsymbol{\kappa}}_i' \tilde{\boldsymbol{a}}_i} = -\frac{\kappa_x[i+1] + \tilde{\boldsymbol{\kappa}}_i' \boldsymbol{a}_i}{s_i}. \tag{13.41}$$

The parameter s_i is updated as follows:

$$\begin{aligned} s_{i+1} &= \kappa_x[0] + \boldsymbol{\kappa}_{i+1}' \boldsymbol{a}_{i+1} = \kappa_x[0] + \boldsymbol{\kappa}_i' \boldsymbol{a}_{i+1}^* + \kappa_x[i+1] a_{i+1,i+1} \\ &= \kappa_x[0] + \boldsymbol{\kappa}_i' \boldsymbol{a}_i + a_{i+1,i+1} \boldsymbol{\kappa}_i' \tilde{\boldsymbol{a}}_i + \kappa_x[i+1] a_{i+1,i+1} \\ &= s_i + a_{i+1,i+1} \{\kappa_x[i+1] + \boldsymbol{\kappa}_i' \tilde{\boldsymbol{a}}_i\} = s_i(1 - a_{i+1,i+1}^2). \end{aligned} \tag{13.42}$$

The parameter

$$\rho_{i+1} = -a_{i+1,i+1} = \frac{\kappa_x[i+1] + \tilde{\boldsymbol{\kappa}}_i' \boldsymbol{a}_i}{s_i} = \frac{\kappa_x[i+1] + \sum_{l=1}^{i} a_{i,l} \kappa_x[i+1-l]}{s_i} \tag{13.43}$$

is called the *partial correlation coefficient*, or the *reflection coefficient*, of order $i+1$, and is defined for all $i \geq 0$. Equations (13.41), (13.40), (13.42), in this order of evaluation, comprise the ith step of the Levinson-Durbin algorithm. The parameter s_0 is equal to $\kappa_x[0]$, since $\hat{x}[n] = 0$ and $v_0[n] = x[n]$ for a zero-order predictor. For convenience, we rewrite the complete Levinson-Durbin algorithm, including initialization.

1. Define

$$s_0 = \kappa_x[0]. \tag{13.44}$$

2. For $i = 0, 1, \ldots, p-1$ do the following:

$$\rho_{i+1} = \frac{\kappa_x[i+1] + \sum_{l=1}^{i} a_{i,l} \kappa_x[i+1-l]}{s_i}, \tag{13.45a}$$

$$s_{i+1} = s_i(1 - \rho_{i+1}^2), \tag{13.45b}$$

$$a_{i+1,l} = a_{i,l} - \rho_{i+1} a_{i,i+1-l}, \quad 1 \leq l \leq i, \tag{13.45c}$$

$$a_{i+1,i+1} = -\rho_{i+1}. \tag{13.45d}$$

The procedure `levdur` in Program 13.4 implements the Levinson-Durbin algorithm.

The number of operations needed for the ith step of the algorithm is determined as follows: In (13.45a) we have i additions and i multiplications and likewise in (13.45c). In (13.45a) we also have one division, and in (13.45b) we have two multiplications and one addition. In total, ith step requires $2i+2$ multiplications, $2i+1$ additions, and one division. The complete algorithm thus requires p^2+p multiplications, p^2 additions, and p divisions. As we said in the beginning of this section, the computational efficiency of the Levinson-Durbin algorithm is the main reason for its importance to AR modeling.

The parameters s_i and ρ_{i+1} deserve special attention. The former is, by definition, the variance of the prediction error of the ith-order AR model. The product $s_i\rho_{i+1}$ is the covariance between $y[n-i-1]$ and the prediction error of the ith-order AR model at time n; see Problem 13.9. By its definition, s_i is nonnegative for all i. It therefore follows from (13.45b) that $|\rho_{i+1}| \leq 1$ for all i. Furthermore, s_i is monotonically decreasing in i, that is, $s_{i+1} \leq s_i$ for all i. This is not surprising, since increasing the model order implies better ability to predict the signal's present value from its past, hence a smaller variance of the prediction error. Two extreme cases are as worth noting:

1. If $x[n]$ is a true AR signal of order p, then $s_p = \gamma_v$, and also $s_i = \gamma_v$ for all $i > p$. Therefore, it follows from (13.45b) that in this case $\rho_{i+1} = 0$ for all $i \geq p$. In other words, for a true AR signal of order p, the Levinson-Durbin algorithm stops naturally after p steps: The parameters s_i and $a_{i,l}$ cease to change, and the reflection coefficients become identically zero for $i \geq p$. On the other hand, if

$x[n]$ is not a true AR signal of any order, the algorithm does not stop naturally, but s_i continues to decrease and the $a_{i,l}$ continue to change as i increases.

2. If $\rho_{i+1} = \pm 1$ for some i, then s_{i+1}, the prediction error of order $i+1$, becomes zero. Therefore, in this case, $x[n]$ becomes exactly predictable from its $i+1$ past values. Random signals for which this happens are called *deterministic*. Problem 13.12 provides an example of a signal that is deterministic according to this definition. Most physical random signals are not deterministic; that is, they are not exactly predictable from their past, no matter how large the order of the predictor.

13.4.4 Lattice Filters

The FIR filter $a(z)$, used to obtain the innovation $v[n]$ from $x[n]$, can be realized by either of the two direct realizations described in Section 11.1 (note that this filter is neither symmetric nor antisymmetric, so the realizations described in Section 11.1.3 are not applicable). Likewise, the IIR filter $1/a(z)$, used to obtain $x[n]$ from $v[n]$, can be realized by any of the realizations described in Section 11.1: direct, parallel, or cascade. In this section we use the Levinson-Durbin algorithm to derive new realizations of these filters, called *lattice realizations*.

Let us define

$$v_i[n] = \sum_{l=0}^{i} a_{i,l}x[n-l], \quad \tilde{v}_i[n] = \sum_{l=0}^{i} a_{i,i-l}x[n-l], \quad a_{i,0} = 1, \tag{13.46}$$

$$a_i(z) = \sum_{l=0}^{i} a_{i,l}z^{-l}, \quad \tilde{a}_i(z) = \sum_{l=0}^{i} a_{i,i-l}z^{-l}. \tag{13.47}$$

The signal $v_i[n]$ is the result of passing $x[n]$ through the FIR filter obtained from the ith-order linear MMSE predictor. Similarly, the signal $\tilde{v}_i[n]$ is the result of passing $x[n]$ through the reversed-order filter. We can use (13.45c,d) to express the $(i+1)$st-order signals $v_{i+1}[n]$ and $\tilde{v}_{i+1}[n]$ in terms of the ith-order signals as follows:

$$\begin{aligned} v_{i+1}[n] &= \sum_{l=0}^{i+1} a_{i+1,l}x[n-l] = x[n] + \sum_{l=1}^{i}(a_{i,l} - \rho_{i+1}a_{i,i+1-l})x[n-l] - \rho_{i+1}x[n-i-1] \\ &= v_i[n] - \rho_{i+1}\tilde{v}_i[n-1], \end{aligned} \tag{13.48}$$

$$\begin{aligned} \tilde{v}_{i+1}[n] &= \sum_{l=0}^{i+1} a_{i+1,i+1-l}x[n-l] \\ &= -\rho_{i+1}x[n] + \sum_{l=1}^{i}(a_{i,i+1-l} - \rho_{i+1}a_{i,l})x[n-l] + x[n-i-1] \\ &= \tilde{v}_i[n-1] - \rho_{i+1}v_i[n]. \end{aligned} \tag{13.49}$$

Figure 13.9(a) illustrates the dependence of the $(i+1)$st-order signals on the ith-order ones. The signal $\tilde{v}_i[n]$ is delayed by one time unit and, together with $v_i[n]$, fed to a butterfly-like section that performs the operations expressed in (13.48) and (13.49). Figure 13.9(b) depicts this operation as a black box. The matrix $\boldsymbol{R}_{i+1}(z)$ is the 2×2 z-domain matrix

$$\boldsymbol{R}_{i+1}(z) = \begin{bmatrix} 1 & -\rho_{i+1}z^{-1} \\ -\rho_{i+1} & z^{-1} \end{bmatrix}. \tag{13.50}$$

The inputs to the section $\boldsymbol{R}_1(z)$, $v_0[n]$ and $\tilde{v}_0[n]$, are both identically equal to $x[n]$. When fed to this section, they yield $v_1[n]$ and $\tilde{v}_1[n]$. When these are fed to the

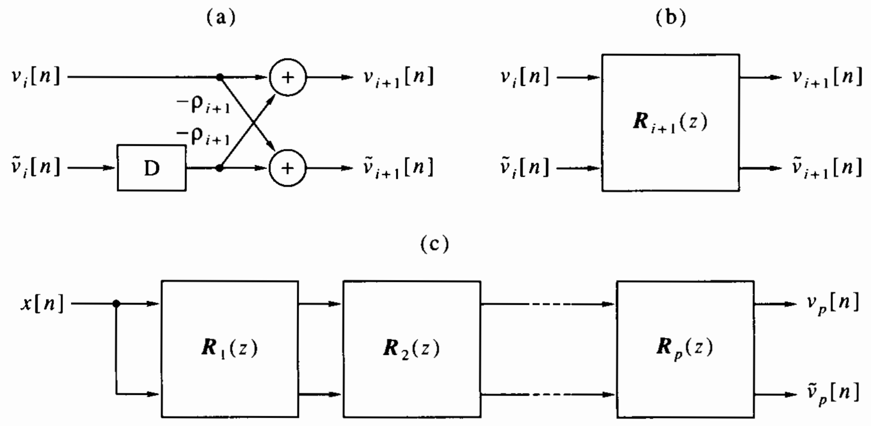

Figure 13.9 An FIR lattice filter: (a) a typical section; (b) schematic drawing of a typical section; (c) the complete lattice.

section $R_2(z)$, they yield $v_2[n]$ and $\tilde{v}_2[n]$, and so on. By connecting p such sections in cascade, as shown in Figure 13.9(c), we get $v_p[n]$ and $\tilde{v}_p[n]$. Therefore, the transfer function from $x[n]$ to $v_p[n]$ in the realization shown in Figure 13.9(c) is precisely $a_p(z)$. Likewise, the transfer function from $x[n]$ to $\tilde{v}_p[n]$ is $\tilde{a}_p(z)$, the reversed-order filter. The realization shown in Figure 13.9(c) is called an *FIR lattice*, owing to its visual appearance. We already encountered a special case of this realization in Problem 11.14.

The FIR lattice shown in Figure 13.9(c) is obviously of order p, since it contains p delay elements; it is FIR because it contains no feedback paths. It has the following properties:

1. The lattice structure is *nested*, in the sense that all intermediate-order FIR filters $a_i(z)$ are embedded in the pth-order filter. When we feed the signal $x[n]$ at the input, we get the ith-order error signal $v_i[n]$ at the output of the ith section for all $1 \le i \le p$. We can therefore change the desired filter's order simply by picking the desired $v_i[n]$, and there is no need to modify the filter's structure or its coefficients.
2. The internal gains ρ_{i+1} are less than 1 in magnitude, so the filter can be conveniently implemented in fixed-point arithmetic, and there is no need to scale the gains.
3. The lattice structure is known to have low sensitivity to quantization noise (we will not elaborate on this topic here).
4. The realization requires $2p$ multiplications and additions, which is twice the necessary minimum. Therefore, it is wasteful compared with the direct realizations. This drawback may make the lattice realization problematic in time-critical applications.

To derive a lattice realization for $1/a_p(z)$, we use (13.48) to express $v_i[n]$ in terms of $v_{i+1}[n]$ and $\tilde{v}_i[n-1]$:

$$v_i[n] = v_{i+1}[n] + \rho_{i+1}\tilde{v}_i[n-1]. \tag{13.51}$$

Figure 13.10(a) illustrates the operations expressed in (13.49) and (13.51). Figure 13.10(b) depicts these operations as a black box. The matrix $S_{i+1}(z)$ expresses the z-domain dependence of $v_i[n]$ and $\tilde{v}_{i+1}[n]$ on $v_{i+1}[n]$ and $\tilde{v}_i[n]$. To derive this matrix, substitute (13.51) in (13.49) to get

$$\tilde{v}_{i+1}[n] = (1 - \rho_{i+1}^2)\tilde{v}_i[n-1] - \rho_{i+1}v_{i+1}[n]. \tag{13.52}$$

Therefore,

$$S_{i+1}(z) = \begin{bmatrix} 1 & \rho_{i+1}z^{-1} \\ -\rho_{i+1} & (1-\rho_{i+1}^2)z^{-1} \end{bmatrix} \tag{13.53}$$

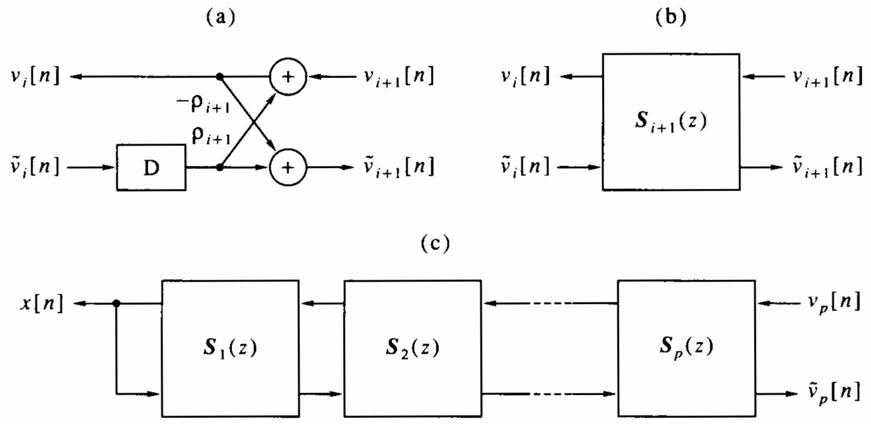

Figure 13.10 An IIR lattice filter: (a) a typical section; (b) schematic drawing of a typical section; (c) the complete lattice.

The upper output of the section $S_1(z)$, $v_0[n]$, is short-circuited to the input $\tilde{v}_0[n]$, since both are identically equal to $x[n]$. The p sections are connected as shown in Figure 13.10(c). When $v_p[n]$ is fed at the upper right input, we get $x[n]$ at the upper left output and $\tilde{v}_p[n]$ at the lower right output. Therefore, the transfer function from $v_p[n]$ to $x[n]$ in the realization shown in Figure 13.10(c) is precisely $1/a_p(z)$. Also, the transfer function from $v_p[n]$ to $\tilde{v}_p[n]$ is $\tilde{a}_p(z)/a_p(z)$.

The realization shown in Figure 13.10(c) is called an *IIR lattice*. We already encountered a special case of this realization in Problem 11.15. The realization is of order p, since it contains p delay elements; it is IIR because it contains feedback paths. It has properties similar to those of the FIR lattice. However, the nesting property does not hold for the IIR filter as for the FIR filter. To realize the intermediate-order all-pole filter $1/a_i(z)$, we must disable the higher order sections by resetting the coefficients $\{\rho_{i+1}, \dots, \rho_p\}$ to zero. Stability of the IIR lattice is guaranteed as long as all ρ_{i+1} are less than 1 in magnitude.

Both FIR and IIR lattice realizations can be constructed for any stable polynomial $a(z)$ and without regard to any relation to a random signal. For this, we need to compute the set of reflection coefficients $\{\rho_{i+1},\ 0 \le i \le p-1\}$ associated with $a(z)$. These coefficients can be computed by carrying out the Levinson-Durbin in reversed order. The reversed-order Levinson-Durbin algorithm is derived as follows: It is obvious from the construction of the FIR lattice that

$$\begin{bmatrix} a_{i+1}(z) \\ \tilde{a}_{i+1}(z) \end{bmatrix} = \boldsymbol{R}_{i+1}(z) \begin{bmatrix} a_i(z) \\ \tilde{a}_i(z) \end{bmatrix}, \tag{13.54}$$

so

$$\begin{bmatrix} a_i(z) \\ \tilde{a}_i(z) \end{bmatrix} = [\boldsymbol{R}_{i+1}(z)]^{-1} \begin{bmatrix} a_{i+1}(z) \\ \tilde{a}_{i+1}(z) \end{bmatrix} = \frac{1}{(1-\rho_{i+1}^2)z^{-1}} \begin{bmatrix} z^{-1} & \rho_{i+1}z^{-1} \\ \rho_{i+1} & 1 \end{bmatrix} \begin{bmatrix} a_{i+1}(z) \\ \tilde{a}_{i+1}(z) \end{bmatrix}. \tag{13.55}$$

In particular,

$$a_i(z) = (1-\rho_{i+1}^2)^{-1}[a_{i+1}(z) + \rho_{i+1}\tilde{a}_{i+1}(z)], \tag{13.56}$$

where $\rho_{i+1} = -a_{i+1,i+1}$. The reversed-order Levinson-Durbin algorithm is initialized by setting $i+1 = p$ and $a_p(z) = a(z)$, the given stable polynomial. The iteration proceeds downward, until $i = 0$ is reached. Then we have all the coefficients ρ_{i+1}, needed to construct the (FIR or IIR) lattice filter. Equation (13.56) is seen to be identical to (7.61); see Section 7.4.4. In fact, the reversed-order Levinson-Durbin algorithm is identical to the Schur-Cohn stability test. Stability of $a(z)$ guarantees that $|\rho_{i+1}| < 1$ for all $0 \le i \le p-1$.

13.4.5 The Schur Algorithm

The Levinson-Durbin algorithm can be used to obtain the sequence of reflection coefficients from the sequence of covariances, computing the intermediate-order model coefficients along the way. The *Schur algorithm* [Schur, 1917, 1918] enables the computation of the reflection coefficients from the covariance sequence without explicitly computing the AR coefficients.

To derive the Schur algorithm, let us input the sequence $\kappa_x[n]$ to the FIR lattice shown in Figure 13.9(c) (note that this input sequence is not causal). Denote the upper and lower output sequences of the $(i+1)$st section by $c_{i+1}[n]$ and $\tilde{c}_{i+1}[n]$, respectively. Then,

$$c_{i+1}[n] = \kappa_x[n] + \sum_{l=1}^{i+1} a_{i+1,l}\kappa_x[n-l], \tag{13.57a}$$

$$\tilde{c}_{i+1}[n] = \sum_{l=0}^{i} a_{i+1,i+1-l}\kappa_x[n-l] + \kappa_x[n-i-1]. \tag{13.57b}$$

It follows from the $(i+1)$st-order Yule-Walker equation that

$$c_{i+1}[n] = 0, \quad 1 \le n \le i+1, \tag{13.58a}$$

$$\tilde{c}_{i+1}[n] = 0, \quad 0 \le n \le i. \tag{13.58b}$$

Furthermore, it follows from (13.37) that

$$\tilde{c}_{i+1}[i+1] = \sum_{l=0}^{i} a_{i+1,i+1-l}\kappa_x[i+1-l] + \kappa_x[0] = s_{i+1}. \tag{13.59}$$

Observing the lattice structure and letting $\kappa_x[n]$ be the input sequence, we see that the sequences $c_i[n]$ and $\tilde{c}_i[n]$ obey the relationships

$$c_{i+1}[n] = c_i[n] - \rho_{i+1}\tilde{c}_i[n-1], \tag{13.60a}$$

$$\tilde{c}_{i+1}[n] = \tilde{c}_i[n-1] - \rho_{i+1}c_i[n]. \tag{13.60b}$$

Since, by (13.58a), $c_{i+1}[i+1] = 0$, we get from (13.60a)

$$\rho_{i+1} = \frac{c_i[i+1]}{\tilde{c}_i[i]}. \tag{13.61}$$

The initial conditions of the Schur algorithm are

$$c_0[n] = \tilde{c}_0[n] = \kappa_x[n], \quad \text{for all } n. \tag{13.62}$$

The algorithm then iterates (13.61) and (13.60) for all $0 \le i \le p-1$. Note that, because of (13.58), (13.60a) needs to be computed only for $i+2 \le n \le p$ and (13.60b) needs to be computed only for $i+1 \le n \le p$. At the end of the iteration, s_p can be obtained from (13.59). For convenience, we rewrite the complete Schur algorithm, including initialization.

1. Define

$$c_0[n] = \tilde{c}_0[n] = \kappa_x[n], \quad 0 \le n \le p. \tag{13.63}$$

2. For $i = 0, 1, \dots, p-1$ do the following:

$$\rho_{i+1} = \frac{c_i[i+1]}{\tilde{c}_i[i]}, \tag{13.64a}$$

$$c_{i+1}[n] = c_i[n] - \rho_{i+1}\tilde{c}_i[n-1], \quad i+2 \le n \le p, \tag{13.64b}$$

$$\tilde{c}_{i+1}[n] = \tilde{c}_i[n-1] - \rho_{i+1}c_i[n], \quad i+1 \le n \le p. \tag{13.64c}$$

3. Set

$$s_p = \tilde{c}_p[p]. \tag{13.65}$$

We leave it to the reader to develop a MATLAB procedure for the Schur algorithm (Problem 13.15).

The Schur algorithm has the same computational complexity as the Levinson-Durbin algorithm. However, it avoids the operation (13.45a), which requires i additions to complete. In contrast, the operations in (13.64) require only one addition each. This facilitates parallel computation of these operations, hence the Schur algorithm can be advantageously implemented on a parallel computer; see Hayes [1996, Sec. 5.2.6] for further discussion of this subject.

13.4.6 AR Modeling from Measured Data

The autoregressive model obtained by solving the Yule-Walker equations requires knowledge of the covariance sequence $\kappa_x[m]$. In reality, this sequence is seldom known, at least not exactly. Typically, we are given only a finite set of measurements, say $\{x[n],\ 0 \le n \le N-1\}$. If we wish to use these measurements for AR modeling of $x[n]$, an obvious approach is to compute a set of estimated covariances first, say $\{\hat{\kappa}_x[m],\ 0 \le m \le p\}$, then use the estimated covariances in the Yule-Walker equations, in lieu of the true covariances $\{\kappa_x[m],\ 0 \le m \le p\}$. The estimated covariances can be computed using the formula

$$\hat{\kappa}_x[m] \triangleq N^{-1} \sum_{i=0}^{N-1-|m|} x[i]x[i+|m|], \quad 0 \le m \le p \tag{13.66}$$

[cf. (13.12) and recall we are assuming that the signal has zero mean]. The procedure `kappahat` in Program 13.5 computes the sequence $\hat{\kappa}_x[m]$. The corresponding AR model coefficients will be denoted by $\{\hat{a}_1, \dots, \hat{a}_p\}$. We can also input the estimated covariances to the Levinson-Durbin algorithm and obtain AR models of increasing orders, with coefficients $\{\hat{a}_{i,l},\ 1 \le i \le p,\ 1 \le l \le i\}$.

The polynomial $a_i(z)$ obtained from the solution of the ith-order Yule-Walker equation is always stable, because $\kappa_x[m]$ is the true covariance sequence of a WSS signal (whether autoregressive or not). Since the estimated covariances $\hat{\kappa}_x[m]$ differ from the true covariances $\kappa_x[m]$, it is not obvious whether the polynomial $\hat{a}_i(z)$ obtained by solving the Yule-Walker equation with $\hat{\kappa}_x[m]$ is necessarily stable. The following result is of considerable importance:

Theorem 13.1 If the estimated covariances $\hat{\kappa}_x[m]$ are computed using (13.66) and substituted in the ith-order Yule-Walker equations, then the polynomial

$$\hat{a}_i(z) \triangleq 1 + \hat{a}_{i,1}z^{-1} + \cdots + \hat{a}_{i,i}z^{-i} \tag{13.67}$$

obtained from the solution of these equations is stable for any i in the range $1 \le i \le N-1$. □

A formal proof of this theorem is difficult and will not be given here. However, we give a heuristic argument that is almost as good as a proof, as follows: A fundamental property of the covariance sequence $\kappa_x[m]$, besides being real and symmetric, is that its Fourier transform $K_x^{\mathrm{f}}(\theta)$ is nonnegative; see (2.125). Now, the sequence $\{\hat{\kappa}_x[m],\ -(N-1) \le m \le N-1\}$ is real and symmetric, and according to (13.11) it also has a nonnegative Fourier transform, namely $N^{-1}|X^{\mathrm{f}}(\theta)|^2$. Therefore, this sequence is (or at least *can be*) a bona fide covariance sequence of *some* WSS signal, albeit not necessarily the signal $x[n]$ itself. Therefore, $\hat{a}_i(z)$ obtained from this sequence is also a linear MMSE predictor for *some* WSS signal, hence is stable.

If we wish to model the given signal by an IIR lattice filter, we need estimates of the reflection coefficients, say $\hat{\rho}_{i+1}$. These can be obtained by the Schur algorithm and using the estimated covariances. Since $\{\hat{\kappa}_x[m],\ -(N-1) \le m \le N-1\}$ is a valid covariance sequence, the estimated reflection coefficients are guaranteed to be less than 1 in magnitude.

The estimated polynomial $\hat{a}_i(z)$, together with the estimate $\hat{s}_i$ of the input-signal variance, given by

$$\hat{s}_i = \sum_{k=0}^{i} \hat{a}_{i,k}\hat{\kappa}_x[k]. \tag{13.68}$$

can be used to construct an estimate of the PSD of $x[n]$ [cf. (13.20)]:

$$\hat{K}_x^{\mathrm{f}}(\theta) = \hat{s}_i \left| \frac{1}{\hat{a}_i(e^{j\theta})} \right|^2. \tag{13.69}$$

This is called an *autoregressive spectrum estimate* of $x[n]$. Note that $\hat{K}_x^{\mathrm{f}}(\theta)$ is not unique, since it depends on the order i of the linear MMSE predictor. Therefore, we get in fact a family of AR spectrum estimates, one for each i.

Example 13.4 We continue to examine Wolfer's sunspot data we met in Example 13.2, this time fitting an AR model to these data. We feed the data (after subtracting the mean value and computing the estimated covariances) to the Levinson-Durbin algorithm and run the algorithm up to order $p = 50$. Figure 13.11(a) shows the parameters s_i for $2 \le i \le 50$. We recall that s_i measures the variance of the ith-order prediction error. As we see, between a second-order model and a 50th-order one, there is only a moderate decrease of the error variance, from 250 to 170.

Figure 13.11(b) shows the PSD computed from (13.69), using a second-order AR model. This was the model used by Yule in his 1927 paper. As we see, the estimated PSD is a very smooth, hence probably is a very crude depiction of the true PSD. Nonetheless, it displays a peak at frequency of about $(1/11)\,\text{year}^{-1}$, as obtained from the smoothed periodograms in Example 13.2.

Figure 13.11(c) shows the PSD computed from (13.69), using a 50th-order AR model. Such a model was beyond anyone's ability to compute in Yule's day but can be easily computed today. As we see, the estimated PSD shows many details not shown in Figure 13.11(b). Some of these details may be random artifacts, but some may be real.

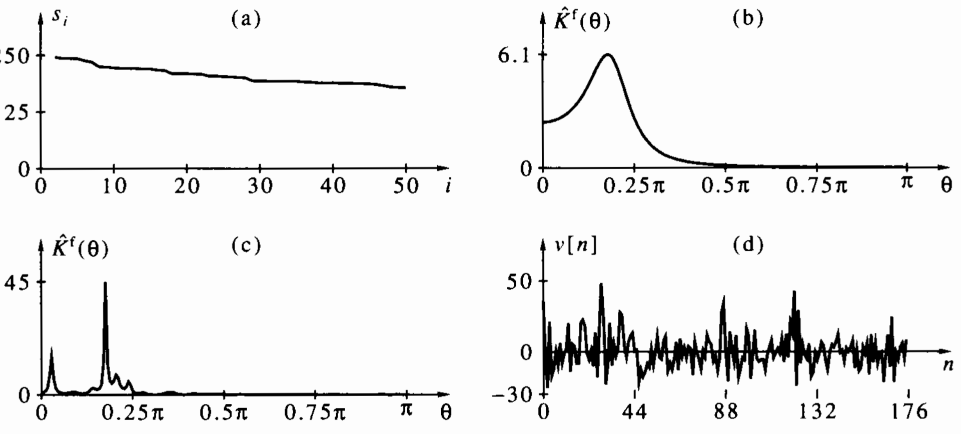

Figure 13.11 AR modeling of Wolfer's sunspot data: (a) the prediction-error variance as a function of the model order; (b) AR spectrum of order 2; (c) AR spectrum of order 50; (d) the prediction error for a 50th-order model (parts b, c are drawn to different scales; maximum values shown are in thousands).

In particular, the 70-year period found in the smoothed periodograms is seen here also.

Figure 13.11(d) shows the prediction error $v[n]$ corresponding to the 50th-order model. Since $s_{50} \approx 170$, the standard deviation of $v[n]$ is about 13. As seen from the figure, the peak error reaches about 50 occasionally. The practical implication of these numbers is: Based on the AR model, we may not be able to predict next year's sunspot number to an accuracy better than ± 13 on the average, and the error may be up to 50 in extreme cases. □

13.4.7 AR Modeling by Least Squares

An alternative to the Yule-Walker method for estimating AR parameters from given measurements of the signal is provided by the *least-squares* method. Let us assume again that the values $\{x[n],\ 0 \le n \le N-1\}$ are given. Then we can form the prediction-error equation for a pth-order model

$$v[n] = x[n] - \hat{x}[n] = \sum_{k=0}^{p} a_k x[n-k], \quad \text{where} \quad a_0 \triangleq 1, \ p \le n \le N-1. \tag{13.70}$$

Note that the specified range of n makes use of all available data. Now define the average square error

$$V = \frac{1}{N-p} \sum_{n=p}^{N-1} v^2[n]. \tag{13.71}$$

The *least-squares* AR model of order p is the set of parameters $\{a_k,\ 1 \le k \le p\}$ for which V is minimum. The least-squares criterion is similar to the MMSE criterion, presented in Section 13.4.2. However, instead of minimizing the *expected* square error, which relies on the unknown covariance sequence $\kappa_x[m]$, it seeks to minimize the *actual* square error, which depends only on the given data.

To solve the problem of minimizing V, we proceed as in Section 13.4.2: We differentiate with respect to the unknown parameters and equate the partial derivatives to zero. We get

$$\frac{\partial V}{\partial a_l} = \frac{2}{N-p}\sum_{n=p}^{N-1}\frac{\partial v[n]}{\partial a_l}v[n] = \frac{2}{N-p}\sum_{n=p}^{N-1}x[n-l]\left(\sum_{k=0}^{p}a_k x[n-k]\right)$$
$$= \frac{2}{N-p}\sum_{k=0}^{p}a_k\left(\sum_{n=p}^{N-1}x[n-l]x[n-k]\right) = 0, \quad 1 \le l \le p. \tag{13.72}$$

Define

$$\xi_{l,k} = \frac{1}{N-p}\sum_{n=p}^{N-1}x[n-l]x[n-k]. \tag{13.73}$$

Let Ξ_p be the $p \times p$ matrix constructed from $\{\xi_{l,k},\ 1 \le l,k \le p\}$ and let $\boldsymbol{\xi}_p$ be the p-dimensional vector constructed from $\{\xi_{l,0},\ 1 \le l \le p\}$. Then we can rewrite (13.72) as

$$\Xi_p \boldsymbol{a}_p + \boldsymbol{\xi}_p = 0 \implies \boldsymbol{a}_p = -\Xi_p^{-1}\boldsymbol{\xi}_p. \tag{13.74}$$

The vector of parameters $\boldsymbol{a}_p$ obtained by solving (13.74) is the pth-order, least-squares estimate for the measured data $\{x[n],\ 0 \le n \le N-1\}$.

The minimum value of V is given by

$$V_{\min} = \frac{1}{N-p}\sum_{n=p}^{N-1}\left(\sum_{k=0}^{p}a_k x[n-k]\right)\left(\sum_{l=0}^{p}a_l x[n-l]\right) = \sum_{k=0}^{p}\sum_{l=0}^{p}a_k a_l \xi_{l,k}$$
$$= \sum_{k=0}^{p}a_k\xi_{0,k} + \sum_{l=1}^{p}a_l\left(\sum_{k=0}^{p}a_k\xi_{l,k}\right). \tag{13.75}$$

However, by (13.74), the second term on the right side of (13.75) is zero. Therefore,

$$V_{\min} = \sum_{k=0}^{p}a_k\xi_{0,k}. \tag{13.76}$$

The value of $V_{\min}$ can serve as an estimate of γ_v.

The least-squares solution usually provides a more accurate AR model than the Yule-Walker solution when N, the number of data, is relatively small. Therefore, it is often preferred to the Yule-Walker solution in such cases. However, the least-squares solution suffers from two major drawbacks:

1. Contrary to the polynomial obtained from the solution of the Yule-Walker equation, the polynomial obtained from the least-squares solution (13.74) is not guaranteed to be stable, although the occurrence of instability is rare. Stability of the polynomial can be verified, if needed, using the Schur-Cohn test.
2. The matrix Ξ_p is not Toeplitz, so the solution of (13.74) normally requires p^3 operations, rather than p^2 (as required by the Levinson-Durbin algorithm). Efficient algorithms have been derived for solving (13.74) in a number of operations proportional to p^2 [Friedlander et al., 1979; Porat et al., 1982]. However, these algorithms are somewhat complicated and not easy to program, therefore they are not in common use. Since the computational complexity of the least-squares method is higher than that of the Levinson-Durbin method, the former is seldom used in time-critical applications.

We finally remark that the Yule-Walker (or Levinson-Durbin) solution to the AR modeling problem is also called the *autocorrelation method*, and the least-squares solution is also called the *covariance method*. These names are common in speech modeling applications; in Section 14.2 we shall study such an application.

13.5 Joint Signal Modeling

The autoregressive model, discussed in the preceding section, is a way of modeling a single signal. In numerous signal processing applications, we are interested in modeling the relationship between a pair of signals, say $x[n]$, $y[n]$. The signals may be the input and output of a system whose nature and parameters are unknown. For example, $x[n]$ may represent the input of a communication channel and $y[n]$ the output of the channel, and we may wish to model the dynamic behavior of the channel. More generally, the signals may have been generated by some common mechanism. For example, $x[n]$ and $y[n]$ may represent, respectively, a neurological signal (so-called *electromyogram*, or EMG) and a brain signal (so-called *electroencephalogram*, or EEG) measured in response to some stimulus.

Two discrete-time WSS random signals $x[n]$, $y[n]$ are said to be *jointly wide-sense stationary* if each of them is WSS and their cross-covariance $E\{(y[n+m]-\mu_y)(x[n]-\mu_x)\}$ depends only on the difference m between their respective time indices. The corresponding *cross-covariance sequence* is then defined as [cf. (2.119)]

$$\kappa_{yx}[m] = E\{(y[n+m]-\mu_y)(x[n]-\mu_x)\} = E(y[n+m]x[n]) - \mu_y\mu_x. \tag{13.77}$$

In particular, when the signals are zero mean, we have $\kappa_{yx}[m] = E(y[n+m]x[n])$. Note that, in contrast with the covariance sequence $\kappa_x[m]$, the cross-covariance sequence is not symmetric in m.

In this section we study a simple model for a pair of jointly WSS signals: that of a linear, causal, FIR filter. We assign to $x[n]$ the role of input and to $y[n]$ the role of output, although these roles are not necessarily implied by physical considerations. Our model is thus

$$y[n] = b_0x[n] + b_1x[n-1] + \cdots + b_qx[n-q]. \tag{13.78}$$

The parameters of the model are the order q and the coefficients $\{b_0, b_1, \dots, b_q\}$. We note that (13.78) is not a moving-average model, despite being similar to (13.18), because we do not assume here that $x[n]$ is white noise. As in the case of an autoregressive model, we initially assume that the covariances and cross-covariances of the two signals are known, and later consider the case where they are unknown. Based on knowledge of these sequences, we wish to compute the parameters of the model.

Let us proceed in the same way as we derived the Yule-Walker equation: Multiply (13.78) by $x[n-m]$, $m \geq 0$, and take expected values of both sides to obtain

$$\kappa_{yx}[m] = b_0\kappa_x[m] + b_1\kappa_x[m-1] + \cdots + b_q\kappa_x[m-q]. \tag{13.79}$$

Now collect $q+1$ such equations, for $0 \leq m \leq q$, and arrange them in a matrix-vector form:

$$\begin{bmatrix} \kappa_x[0] & \kappa_x[1] & \dots & \kappa_x[q] \\ \kappa_x[1] & \kappa_x[0] & \dots & \kappa_x[q-1] \\ \vdots & \vdots & \dots & \vdots \\ \kappa_x[q] & \kappa_x[q-1] & \dots & \kappa_x[0] \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_q \end{bmatrix} = \begin{bmatrix} \kappa_{yx}[0] \\ \kappa_{yx}[1] \\ \vdots \\ \kappa_{yx}[q] \end{bmatrix}. \tag{13.80}$$

Equation (13.80) is known as the *Wiener equation*, or sometimes the *Wiener-Hopf equation*, and its solution—the Wiener solution to the modeling problem (13.78).[5] The Wiener solution can also be derived using a minimum mean-square error approach as in Section 13.4.2; see Problem 13.16. The procedure `wiener` in Program 13.6 solves the Wiener equation.

The Wiener equation can be solved iteratively, in a number of operations proportional to q^2, like the Levinson-Durbin algorithm. In fact, Levinson's work of 1947 concerned the solution of the Wiener equation (13.80) rather than the Yule-Walker equations (13.26). We now derive this solution; we call the resulting algorithm the *joint Levinson algorithm.* We introduce the following notations, in addition to the ones defined in Section 13.4.3 for the derivation of the Levinson-Durbin algorithm.

1. $\boldsymbol{b}_{i-1} = [b_{i-1,0},\ldots,b_{i-1,i-1}]'$: the solution of the $(i-1)$st-order Wiener equation.
2. $\boldsymbol{b}_i^* = [b_{i,0},\ldots,b_{i,i-1}]'$: the vector consisting of the first i components of the solution of the ith-order Wiener equation.
3. $b_{i,i}$: the last component of the solution of the ith-order Wiener equation.
4. $\boldsymbol{\lambda}_{i-1} = [\kappa_{yx}[0],\ldots,\kappa_{yx}[i-1]]'$: the right side of the $(i-1)$st-order Wiener equation.

The ith-order Wiener equation can be written as

$$\begin{bmatrix} \boldsymbol{K}_i & \tilde{\boldsymbol{\kappa}}_i \\ \tilde{\boldsymbol{\kappa}}_i' & \kappa_x[0] \end{bmatrix} \begin{bmatrix} \boldsymbol{b}_i^* \\ b_{i,i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\lambda}_{i-1} \\ \kappa_{yx}[i] \end{bmatrix}. \tag{13.81}$$

Let us assume that we have solved the Wiener equation of order i, and we wish to solve (13.81) for $\boldsymbol{b}_i^*$ and $b_{i,i}$. We can expand this equation to

$$\boldsymbol{K}_i \boldsymbol{b}_i^* + \tilde{\boldsymbol{\kappa}}_i b_{i,i} = \boldsymbol{\lambda}_{i-1}, \tag{13.82a}$$

$$\tilde{\boldsymbol{\kappa}}_i' \boldsymbol{b}_i^* + \kappa_x[0] b_{i,i} = \kappa_{yx}[i]. \tag{13.82b}$$

Solving (13.82a) for $\boldsymbol{b}_i^*$ and using the second equality in (13.37) (the solution of the reversed ith-order Yule-Walker equation) gives

$$\boldsymbol{b}_i^* = \boldsymbol{K}_i^{-1}\boldsymbol{\lambda}_{i-1} - \boldsymbol{K}_i^{-1}\tilde{\boldsymbol{\kappa}}_i b_{i,i} = \boldsymbol{b}_{i-1} + b_{i,i}\tilde{\boldsymbol{a}}_i. \tag{13.83}$$

Substituting (13.83) in (13.82b) and solving for $b_{i,i}$ gives

$$b_{i,i} = \frac{\kappa_{yx}[i] - \tilde{\boldsymbol{\kappa}}_i'\boldsymbol{b}_{i-1}}{\kappa_x[0] + \tilde{\boldsymbol{\kappa}}_i'\tilde{\boldsymbol{a}}_i} = \frac{\kappa_{yx}[i] - \tilde{\boldsymbol{\kappa}}_i'\boldsymbol{b}_{i-1}}{s_i}. \tag{13.84}$$

A convenient way for implementing the joint Levinson algorithm is to add it on the top of the Levinson-Durbin algorithm, because the latter computes the parameters $\tilde{\boldsymbol{a}}_i$ and s_i needed in (13.83), (13.84). We add the computation of $b_{0,0}$ to the initialization step, using

$$b_{0,0} = \frac{\kappa_{yx}[0]}{\kappa_x[0]}. \tag{13.85}$$

We then rewrite (13.83), (13.84) for $i+1$ and add them at the ith step, right after (13.45d):

$$b_{i+1,i+1} = \frac{\kappa_{yx}[i+1] - \sum_{l=0}^{i} b_{i,l}\kappa_x[i+1-l]}{s_{i+1}}, \tag{13.86a}$$

$$b_{i+1,l} = b_{i,l} + b_{i+1,i+1}a_{i+1,i+1-l}, \quad 0 \le l \le i. \tag{13.86b}$$

The joint Levinson algorithm should be iterated for $0 \le i \le q-1$, where q is the desired filter's order. The procedure `jlev` in Program 13.7 implements the joint Levinson algorithm.

When the covariances $\kappa_x[m]$ and $\kappa_{yx}[m]$ are not available, only measured sequences $x[n]$ and $y[n]$, we can use these sequences to construct estimates of the covariances. The sequence $\hat{\kappa}_x[m]$ can be constructed as in (13.66) and the sequence $\hat{\kappa}_{yx}[m]$ by

$$\hat{\kappa}_{yx}[m] \triangleq N^{-1}\sum_{i=0}^{N-1-m} x[i]y[i+m], \quad 0 \le m \le q. \tag{13.87}$$

The corresponding model coefficients will be denoted by $\{\hat{b}_0, \hat{b}_1, \ldots, \hat{b}_q\}$. A least-squares model can also be computed in lieu of the Wiener solution, as done in Section 13.4.7 for the autoregressive model. The details of the least-squares approach are left as a problem; see Problem 13.17.

Example 13.5 Consider the problem of transferring digital information through a base-band channel, that is, a channel capable of transmitting unmodulated signals. Such channels are rare in practice; however, real modulated channels have equivalent base-band models, so the application we present here is applicable to modulated channels equally well. The signaling method we discuss in this example is *pulse amplitude modulation* (PAM). In this method, digital information is represented by symbols that can take a finite number of real values. For example, possible symbol values are ± 1 in binary PAM, $\{\pm 1, \pm 3\}$ in quaternary PAM, etc. To each symbol we assign a pulse whose waveform is fixed and whose amplitude is proportional to the symbol value. The pulses are then transmitted in a sequence. Denoting the symbol sequence by $u[m]$, the symbol interval by T, and the pulse waveform by $p(t)$, the transmitted waveform is

$$s(t) = \sum_{m=-\infty}^{\infty} u[m]p(t - mT). \tag{13.88}$$

In the simplest case, the waveform $p(t)$ is a rectangle of duration T, but other waveforms are also used.

Let us assume that the channel is linear time invariant, and denote its impulse response by $h(t)$. Then the received signal is (disregarding noise and other effects)

$$x(t) = \{h * s\}(t) = \sum_{m=-\infty}^{\infty} u[m] \int_{-\infty}^{\infty} h(\tau)p(t - \tau - mT)d\tau. \tag{13.89}$$

Sampling the received signal at interval T gives the discrete-time signal

$$\begin{aligned} x[n] = x(nT) &= \sum_{m=-\infty}^{\infty} u[m] \int_{-\infty}^{\infty} h(\tau)p(nT - \tau - mT)d\tau \\ &= \sum_{m=-\infty}^{\infty} u[m]g[n - m], \end{aligned} \tag{13.90}$$

where

$$g[n] = \int_{-\infty}^{\infty} h(\tau)p(nT - \tau)d\tau, \quad n \in \mathbb{Z}. \tag{13.91}$$

The sequence $g[n]$ is the equivalent discrete-time impulse response of the channel. If this sequence were equal to a unit sample $\delta[n]$, we would get $x[n] = u[n]$, so we would be able to recover the transmitted symbol sequence exactly. In reality, $g[n]$ is almost never equal to $\delta[n]$. Therefore, $x[n]$ is affected by symbol values at other times, not just by $u[n]$. This phenomenon is called *intersymbol interference* (ISI), because symbols belonging to times $m \neq n$ interfere with the symbol at time n and hinder our ability to detect it faithfully. ISI is a major limiting factor of the performance of narrow-band communication channels.

An *equalizer* is a linear time invariant system designed to invert, or cancel, the convolution operation (13.90). The transfer function of an ideal equalizer should be equal to the inverse of $G^z(z)$, the transfer function of the equivalent discrete-time channel. An ideal equalizer is not realizable, except in rare cases, for the following reasons:

1. The impulse response $h(t)$ of the channel is usually unknown (or at least not exactly known), so its discrete-time equivalent $g[n]$ is not available.

2. Even if $g[n]$ is known, $G^z(z)$ is not necessarily rational, so its inverse is not necessarily rational either.
3. $G^z(z)$ does not necessarily have a causal and stable inverse $1/G^z(z)$. This is true even when $G^z(z)$ itself is rational, causal, and stable. If you have solved Problem 8.12, you know the reason: $G^z(z)$ may be nonminimum phase; that is, it may contain zeros outside the unit circle. In such a case, $G^z(z)$ may possess a stable inverse, but this inverse will not be causal. Therefore, to determine $u[n]$, an ideal equalizer must use future values of $x[n]$ (potentially infinite in number).

A common alternative to an ideal, unrealizable, equalizer is an approximate FIR equalizer. Such an equalizer is defined by the relationship

$$y[n] = b_0x[n] + b_1x[n-1] + \cdots + b_qx[n-q]. \tag{13.92}$$

The order q is usually chosen to be even, say $q = 2N$, and the equalizer's coefficients are chosen to satisfy

$$\{b * g\}[n] \approx \delta[n-N]. \tag{13.93}$$

The approximation here means that the coefficient $\{b * g\}[N]$ should be close to unity and all other coefficients should be close to zero. This equalizer "looks into the future" in the sense that its output at time n, $y[n]$, is approximately equal to $u[n-N]$, so the reconstructed symbol value corresponding to time $n-N$ depends on the received signal values $x[n-N+1]$ through $x[n]$, which are in its future (it also depends on the past and present signal values $x[n-2N]$ through $x[n-N]$).

Practical equalizers need to estimate the coefficients $\{b_0, b_1, \ldots, b_q\}$, since these coefficients depend on the unknown impulse response of the channel. In a *minimum mean-square error equalizer*, the coefficients are obtained by solving the Wiener equation (13.80). A common way to estimate the covariances $\kappa_x[m]$ and $\kappa_{yx}[m]$ is as follows: During initial operation, a fixed sequence of symbols $u[n]$, known to the receiver, is transmitted. The equalizer sets $y[n] = u[n-N]$; it then collects the received sequence $x[n]$ corresponding to the known transmitted sequence, and uses the two sequences to construct $\hat{\kappa}_x[m]$ and $\hat{\kappa}_{yx}[m]$ as in (13.66), (13.87). The estimated covariances are substituted in (13.80) and this equation is solved to yield $\{\hat{b}_0, \hat{b}_1, \ldots, \hat{b}_q\}$. An alternative procedure is to use the two sequences in a *least-squares equalizer*, as discussed in Problem 13.17. The fixed sequence $u[n]$ is called a *training sequence*, because it serves to train the equalizer. Figure 13.12 shows a block diagram of the equalizer. The left position of the switch corresponds to a training mode.

After the training sequence has been exhausted, the equalizer is normally switched to a *decision-directed mode*, indicated by the right position of the switch in Figure 13.12. In this mode, the equalizer's output $\hat{y}[n] = \{b * x\}[n]$ is fed to a decision circuit, whose task is to detect the transmitted symbols. The detected symbol corresponding to time $n-N$, denoted by $\hat{u}[n-N]$, is obtained by rounding $\hat{y}[n]$ to the nearest legal symbol value (e.g., one of $\pm 1, \pm 3$ for a quaternary PAM). The sequence of detected symbols is used to construct the equalizer's reference sequence $y[n]$ according to the rule $y[n] = \hat{u}[n-N]$. Then, the equalizer continues to update the coefficients $\{\hat{b}_0, \hat{b}_1, \ldots, \hat{b}_q\}$. In reality, *adaptive equalization* is normally employed, but we do not discuss this subject here; see Gitlin et al. [1992, Chapter 8]. □

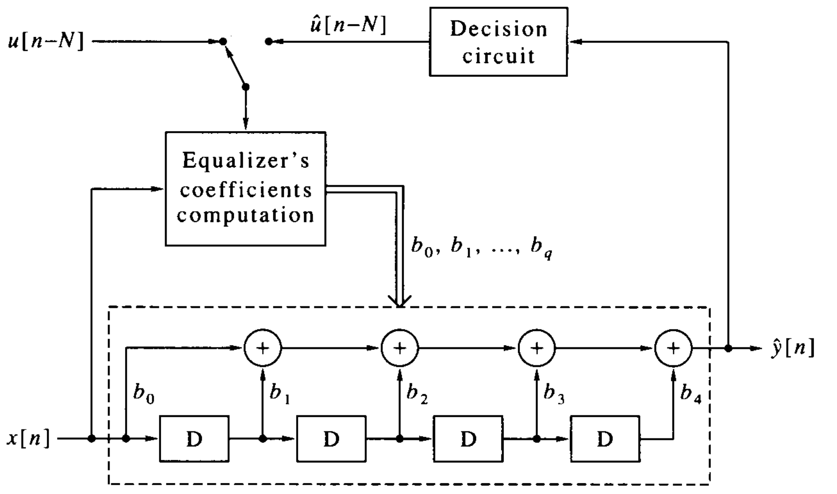

Figure 13.12 Channel equalizer, illustrated for $q = 4$, $N = 2$; switch in left position: training mode; switch in right position: decision-directed mode.

13.6 Summary and Complements

13.6.1 Summary

In this section we discussed spectrum estimation methods and parametric modeling methods for WSS random signals. Simple short-time spectral analysis is of limited use in the case of random signals, because of the random appearance of the Fourier transform of such signals. Randomness can be smoothed by averaging the square magnitudes of the DFTs of successive segments. The amount of smoothing depends on the number of segments being averaged. This number, in turn, is limited by the length of time during which the signal can be assumed stationary. Among the various methods of averaging, the most popular is the one by Welch. An alternative to the Welch periodogram is the smoothed periodogram, useful mainly when the data length is too short for effective averaging.

Rational parametric models for WSS random signals were introduced next, in particular autoregressive models. The parameters of an AR model are related to the covariance sequence of the signal by the Yule-Walker equations. The Levinson-Durbin algorithm facilitates efficient solution of the Yule-Walker equations. This algorithm also leads to lattice realizations of the FIR filter $a(z)$ and the IIR filter $1/a(z)$ corresponding to the AR model. When the covariance sequence of the signal is not known, it can be estimated from actual measurements of the signal, and used in the Levinson-Durbin algorithm in place of the true covariances. An alternative approach is to obtain the model parameters by minimizing the sum of squares of the measured prediction error.

Rational models are also useful in joint modeling of two random signals. We examined the special case of an FIR model, and derived the Wiener solution for this model. The joint Levinson algorithm facilitates an efficient solution to the Wiener equation.

The parametric techniques presented in this chapter naturally lead to the field of *adaptive signal processing.* This field is concerned with real-time, time-varying modeling of signals and systems. For a comprehensive, up-to-date exposition of adaptive signal processing, see Haykin [1996].

13.6.2 Complements

1. [p. 515] The term *periodogram* was introduced by Schuster in his landmark paper [1906b]; it is derived from *a diagram of periods*, since its most common use is for finding periodic components in a signal.
2. [p. 515] The limit (13.2) holds for random signals that are *ergodic in the mean square*. We do not define such signals here, nor do we deal with their mathematical theory, but we implicitly assume that all random signals mentioned in this book are ergodic in the mean square.
3. [p. 523] Expressing a signal as a linear combination of other signals plus noise (or error) is called *regression* in statistics. Equation (13.19) is an expression of $x[n]$ as a linear combination of *its own past values* and the noise $v[n]$, therefore it is called *autoregression*.
4. [p. 521] Radio communication at frequencies 3 to 30 megahertz (so-called "short waves") is made mainly through the ionosphere: The waves are transmitted from the ground upward and are reflected by the ionosphere back to the ground. Communication ranges of thousands of kilometers are made possible this way and are used mainly by radio amateurs all over the world. High sunspot activity increases the range of frequencies that can be transmitted via the ionosphere and improves the quality of communication. Therefore, radio amateurs need to be aware of the current sunspot cycle status.
5. [p. 537] The name *Wiener equation* for (13.80), though widespread, diminishes the contribution of Norbert Wiener, which was incomparably deeper than the straightforward solution (13.80) to the simple modeling problem (13.78). Wiener solved the problem of computing the *causal* minimum mean-square error filter in full generality (originally in continuous time). His work, performed as part of the Second World War effort, was published in a classified report in 1942 and later reprinted [Wiener, 1949]. The name also does injustice to A. N. Kolmogorov, who was the first to develop a mean-square prediction theory for discrete-time WSS signals [Kolmogorov, 1941].

13.7 MATLAB Programs

Program 13.1 Short-time spectral analysis.

```
function X = stsa(x,N,K,L,w,opt,M,theta0,dtheta);
% Synopsis: X = stsa(x,N,K,L,w,opt,M,theta0,dtheta).
% Short-time spectral analysis.
% Input parameters:
% x: the input vector
% N: segment length
% K: number of overlapping points in adjacent segments
% L: number of consecutive DFTs to average
% w: the window (a row vector of length N)
% opt: an optional parameter for nonstandard DFT:
%      'zp' for zero padding
%      'chirpf' for chirp Fourier transform
% M: length of DFT if zero padding or chirp was selected
% theta0, dtheta: parameters for chirp FT.
% Output:
% X: a matrix whose rows are the DFTs of the segments
%    (or averaged segments).

lx = length(x); nsec = ceil((lx-N)/(N-K)) + 1;
x = [reshape(x,1,lx), zeros(1,N+(nsec-1)*(N-K)-lx)];
nout = N; if (nargin > 5), nout = M; else, opt = 'n'; end
X = zeros(nsec,nout);
for n = 1:nsec,
   temp = w.*x((n-1)*(N-K)+1:(n-1)*(N-K)+N);
   if (opt(1) == 'z'), temp = [temp, zeros(1,M-N)]; end
   if (opt(1) == 'c'), temp = chirpf(temp,theta0,dtheta,M);
   else, temp = fftshift(fft(temp)); end
   X(n,:) = abs(temp).^2;
end
if (L > 1),
   nsecL = floor(nsec/L);
   for n = 1:nsecL, X(n,:) = mean(X((n-1)*L+1:n*L,:)); end
   if (nsec == nsecL*L+1),
      X(nsecL+1,:) = X(nsecL*L+1,:); X = X(1:nsecL+1,:);
   elseif (nsec > nsecL*L),
      X(nsecL+1,:) = mean(X(nsecL*L+1:nsec,:));
      X = X(1:nsecL+1,:);
   else, X = X(1:nsecL,:); end
end
```

Program 13.2 A smoothed periodogram.

```
function s = smooper(x,w);
% Synopsis: s = smooper(x,w).
% Computes the smoothed periodogram of the data vector x.
% Input parameters:
% x: the data vector
% w: the window; must have odd length.
% Output:
% s: the smoothed periodogram, of length equal to that of x.

if (rem(length(w),2) == 0),
   error('Window in SMOOPER must have an odd length');
end
x = reshape(x,1,length(x));
x = x - mean(x);
kappa = (1/length(x))*conv(x,fliplr(x));
n = 0.5*(length(kappa)-length(w));
s = fft([zeros(1,n),w,zeros(1,n)].*kappa);
s = abs(s(1:length(x)));
```

Program 13.3 Solution of the Yule-Walker equations.

```
function [a,gammav] = yw(kappa);
% Synopsis: [a,gammav] = yw(kappa).
% Solves the Yule-Walker equations.
% Input:
% kappa: the covariance sequence value from 0 to p.
% Output parameters:
% a: the AR polynomial, with leading entry 1.
% gammav: the innovation variance

p = length(kappa)-1;
kappa = reshape(kappa,p+1,1);
a = [1; -toeplitz(kappa(1:p,1))\kappa(2:p+1,1)]';
gammav = a*kappa;
```

Program 13.4 The Levinson-Durbin algorithm.

```
function [a,rho,s] = levdur(kappa);
% Synopsis: [a,rho,s] = levdur(kappa).
% The Levinson-Durbin algorithm.
% Input:
% kappa: the covariance sequence values from 0 to p.
% Output parameters:
% a: the AR polynomial, with leading entry 1
% rho: the set of p reflection coefficients
% s: the innovation variance.

p = length(kappa)-1;
kappa = reshape(kappa,p+1,1);
a = 1; s = kappa(1); rho = [];
for i = 1:p,
   rhoi = (a*kappa(i+1:-1:2))/s; rho = [rho,rhoi];
   s = s*(1-rhoi^2);
   a = [a,0]; a = a - rhoi*fliplr(a);
end
```

Program 13.5 Computation of the estimated covariance sequence.

```
function kappa = kappahat(x,p);
% Synopsis: kappa = kappahat(x,p).
% Generate estimated covariance values of a data sequence.
% Input parameters:
% x: the data vector
% p: maximum order of covariance.
% Output parameters:
% kappa: the vector of kappahat from 0 through p.

x = x - mean(x); N = length(x);
kappa = sum(x.*x);
for i = 1:p,
   kappa = [kappa, sum(x(1:N-i).*x(i+1:N))];
end
kappa = (1/N)*kappa;
```

Program 13.6 Solution of the Wiener equation.

```
function b = wiener(kappax,kappayx);
% Synopsis: b = wiener(kappax,kappayx).
% Solves the Wiener equation.
% Input parameters:
% kappax: the covariance sequence of x from 0 to q
% kappayx: the joint covariance sequence of y and x from 0 to q.
% Output:
% b: the Wiener filter.

q = length(kappax)-1;
kappax = reshape(kappax,q+1,1);
kappayx = reshape(kappayx,q+1,1);
b = (toeplitz(kappax)\kappayx)';
```

Program 13.7 The joint Levinson algorithm.

```
function b = jlev(kappax,kappayx);
% Synopsis: b = jlev(kappax,kappayx).
% The joint Levinson algorithm.
% Input parameters:
% kappax: the covariance sequence of x from 0 to q
% kappayx: the joint covariance sequence of y and x from 0 to q.
% Output:
% b: the Wiener filter.

q = length(kappax)-1;
kappax = reshape(kappax,q+1,1);
kappayx = reshape(kappayx,q+1,1);
a = 1; s = kappax(1); b = kappayx(1)/kappax(1);
for i = 1:q,
   rho = (a*kappax(i+1:-1:2))/s;
   s = s*(1-rho^2);
   a = [a,0]; a = a - rho*fliplr(a);
   bii = (kappayx(i+1) - b*kappax(i+1:-1:2))/s;
   b = [b+bii*fliplr(a(2:i+1)), bii];
end
```

13.8 Problems

13.1 Write the Welch periodogram formula (13.4) in case the overlap is not 50 percent, but a given number of points K (where $K < N$).

13.2 Extend the idea in Problem 6.17 to the case where $x(t)$ is a complex OQPSK signal, as defined in Example 13.1. Suggest an operation on $y(nT)$ which will enable estimation of ω_0 using DFT, determine the sampling interval, and find the loss in output SNR.

13.3 For a signal $x[n]$ of length N, define

$$S^{\mathrm{d}}[k] = N^{-1}|X^{\mathrm{d}}[k]|^2, \quad 0 \le k \le N-1.$$

(a) Find the relationship between $s[n]$, the inverse DFT of $S^{\mathrm{d}}[k]$, and the sequence $\hat{\kappa}_x[m]$ defined in (13.12). Hint: Use (13.11).

(b) Obtain $x_{\mathrm{a}}[n]$ from $x[n]$ by zero padding to length $M \ge 2N-1$, and let

$$S_{\mathrm{a}}^{\mathrm{d}}[k] = N^{-1}|X_{\mathrm{a}}^{\mathrm{d}}[k]|^2, \quad 0 \le k \le M-1.$$

Find the relationship between $s_{\mathrm{a}}[n]$, the inverse DFT of $S_{\mathrm{a}}^{\mathrm{d}}[k]$, and the sequence $\hat{\kappa}_x[m]$.

(c) Suggest a replacement for **kappahat** that uses the result of part b.

13.4 Solve the Yule-Walker equation (13.26a) for $p = 2$ and obtain an explicit solution for a_1, a_2.

13.5 Write the Yule-Walker equations (13.26a,b) explicitly for $p = 1$. Now assume that, in these equations, a_1 and γ_v are known and solve explicitly for $\kappa_x[0], \kappa_x[1]$.

13.6 Repeat Problem 13.5 for $p = 2$ and solve for $\kappa_x[0], \kappa_x[1], \kappa_x[2]$ as a function of γ_v, a_1, a_2. Remark: This is quite tedious, but the final expressions are not overly complicated.

13.7 Generalize Problems 13.5 and 13.6 to an arbitrary order p as follows: Assume that γ_v and $\{a_1, \ldots, a_p\}$ are known, and write down a set of linear expressions for the unknown variables $\{\kappa_x[0], \ldots, \kappa_x[p]\}$. Do not attempt to solve this system of equations explicitly, since this is quite complicated. Hint: You may find it useful to take $0.5\kappa_x[0]$ as an unknown variable, rather than $\kappa_x[0]$.

13.8 Write a MATLAB procedure that solves the set of equations you have obtained in Problem 13.7. Test your procedure by computing $\{\kappa_x[0], \ldots, \kappa_x[p]\}$ for a given pth-order polynomial $a(z)$ (ensure that this polynomial is stable!) and a given positive constant γ_v. Feed the result to the procedure **yw**, and verify that you get back the polynomial $a(z)$ and the constant γ_v.

13.9 Show that the numerator of the reflection coefficient ρ_{i+1}, defined in (13.43), is the covariance between $x[n-i-1]$ and $v_i[n]$, the prediction error of the ith-order AR model at time n.

13.10 Show that the variance of $\tilde{v}_i[n]$, defined in (13.46), is equal to the variance of $v_i[n]$.

13.11 Show that ρ_{i+1}, defined in (13.43), is the correlation coefficient between $v_i[n]$ and $\tilde{v}_i[n-1]$.

13.12 Let

$$x[n] = \sin(\theta_0 n + \phi_0),$$

where ϕ_0 is a random variable whose distribution is uniform in the range $[-\pi, \pi)$, and θ_0 is fixed.

(a) Prove that the mean of $x[n]$ is zero.

(b) Prove that the covariance sequence of $x[n]$ is

$$\kappa_x[m] = 0.5\cos(\theta_0 m),$$

hence $x[n]$ is a WSS signal. Hint for parts a, b: The distributions of both $\sin\phi_0$ and $\cos\phi_0$ are symmetric about 0; therefore their means are zero.

(c) Prove that $x[n]$ satisfies the identity

$$x[n] - 2\cos\theta_0 x[n-1] + x[n-2] = 0.$$

(d) Explain why part c implies that $x[n]$ is deterministic in the sense defined in Section 13.4.3.

(e) Derive formulas for ρ_1 and ρ_2 obtained by the Levinson-Durbin algorithm when fed with the covariance sequence $\kappa_x[m]$ given in part b. What is $|\rho_2|$ in this case? What can you conclude about s_2? Reconcile this result with the one in part d.

13.13 Explain how to use the IIR lattice for realizing an all-pass filter. Hint: Recall Problem 7.29.

13.14 Generate the following signal in MATLAB:

$$x[n] = \sin(2\pi 0.17n) + 0.5\sin(2\pi 0.18n), \quad 0 \le n \le 127.$$

(a) Is it possible to measure the frequencies of the two sinusoids using the windowed DFT technique studied in Chapter 6? Answer based on theoretical arguments and, if necessary, experiment with MATLAB.

(b) Use the procedures `kappahat` and `yw` to generate a 50th-order AR model to $x[n]$. Compute and plot the magnitude response of the AR model, using `frqresp` (use 1000 points for the plot). Is it possible to measure the frequencies from the AR magnitude response?

(c) Add noise to $x[n]$, that is, form

$$y[n] = x[n] + v[n],$$

where $v[n]$ is generated using `s*randn(1,128)`. Choose `s` as 0.1, then 0.2, and finally 0.4. Conclude whether it is possible to measure the frequencies from the AR magnitude response at each of the three noise levels.

(d) State your conclusions from this experiment.

13.15 Write a MATLAB procedure for the Schur algorithm; test it by comparing its output with that of the procedure `levdur`.

13.16 In the model (13.78), let $e[n]$ denote the error between the two sides. Find the parameters $\{b_0, b_1, \dots, b_q\}$ that minimize $E(e^2[n])$. Show that this leads to the Wiener equation (13.80).

13.17 Develop a least-squares solution to the modeling problem (13.78). Assume that the sequences $\{x[n], y[n],\ 0 \le n \le N-1\}$ are given. Define $e[n]$ as the difference between the two sides of (13.78). Let

$$V = \frac{1}{N-q} \sum_{n=q}^{N-1} e^2[n]. \qquad (13.94)$$

Finally, find the values of $\{b_0, b_1, \dots, b_q\}$ that minimize V, similarly to the derivation in Section 13.4.7.