Chapter 11

Digital Filter Realization and Implementation

In the preceding two chapters we learned how to design digital filters, both IIR and FIR. The end result of the design was the transfer function $H^z(z)$ of the filter or, equivalently, the difference equation it represents. We thus far looked at a filter as a black box, whose input-output relationships are well defined, but whose internal structure is ignored. Now it is time to look more closely at possible internal structures of digital filters, and to learn how to build such filters. It is convenient to break the task of building a digital filter into two stages:

1. Construction of a block diagram of the filter. Such a block diagram is called a *realization* of the filter. Realization of a filter at a block-diagram level is essentially a flow graph of the signals in the filter. It includes operations such as delays, additions, and multiplications of signals by constant coefficients. It ignores ordering of operations, accuracy, scaling, and the like. A given filter can be realized in infinitely many ways. Different realizations differ in their properties, and some are better than others.
2. Implementation of the realization, either in hardware or in software. At this stage we must concern ourselves with problems neglected during the realization stage: order of operations; signal scaling; accuracy of signal values; accuracy of coefficients; accuracy of arithmetic operations. We must analyze the effect of such imperfections on the performance of the filter. Finally, we must build the filter—either the hardware or the program code (or both, if the filter is a specialized combination of hardware and software).

In this chapter we cover the aforementioned subjects. We begin by presenting the most common filter realizations. We describe each realization by its block diagram and by a representative MATLAB code. This will naturally lead us to state-space representations of digital filters. State-space representations are a powerful tool in linear system theory. They are useful for analyzing realizations, performing block-diagram manipulations, computing transfer functions and impulse responses, and executing a host of other applications. State-space theory is rich and we cannot hope to do it justice in a few sections. We therefore concentrate on aspects of state-space theory useful for digital filters, mainly computational ones.

The remainder of this chapter is devoted to finite word length effects in filter implementation. We concentrate on fixed-point implementations, since floating-point implementations suffer less from problems caused by finite word length. We first discuss the effect of coefficient quantization, and explore its dependence on the realization of the filter. This will lead to important guidelines on choice of realizations for different uses. Then we explore the scaling problem, and present scaling procedures for various realizations. Our next topic is noise generated by quantization of multiplication operations, also called computation noise. We present procedures for analyzing the effect of computation noise for different realizations. Finally, we briefly discuss the phenomenon of limit cycle oscillations.

11.1 Realizations of Digital Filters

11.1.1 Building Blocks of Digital Filters

Any digital system that is linear, time invariant, rational, and causal can be realized using three basic types of element:

1. A unit delay: The purpose of this element is to hold its input for a unit of time (physically equal to the sampling interval T) before it is delivered to the output. Mathematically, it performs the operation

$$y[n] = x[n-1].$$

Unit delay is depicted schematically in Figure 11.1(a). The letter "D," indicating delay, sometimes is replaced by z^{-1}, which is the delay operator in the z domain. Unit delay can be implemented in hardware by a data register, which moves its input to the output when clocked. In software, it is implemented by a storage variable, which changes its value when instructed by the program.

2. An adder: The purpose of this element is to add two or more signals appearing at the input at a specific time. Mathematically, it performs the operation

$$y[n] = x_1[n] + x_2[n] + \cdots$$

An adder is depicted schematically in Figure 11.1(b).

3. A multiplier: The purpose of this element is to multiply a signal (a varying quantity) by a constant number. Mathematically,

$$y[n] = ax[n].$$

A multiplier is depicted schematically in Figure 11.1(c). We do not use a special graphical symbol for it, but simply put the constant factor above (or beside) the signal line. A physical multiplier (in hardware or software) can multiply two signals equally easily, but such an operation is not needed in LTI filters.

Example 11.1 Consider the first-order FIR filter

$$H_1^z(z) = b_0 + b_1 z^{-1}.$$

Figure 11.2(a) shows a realization of this filter using one delay element, two multipliers, and one adder. The input to the delay element at time n is $x[n]$, and its output is then $x[n-1]$. The output of the realization is therefore

$$y[n] = b_0 x[n] + b_1 x[n-1],$$

and this is exactly the time-domain expression for $H_1^z(z)$. □

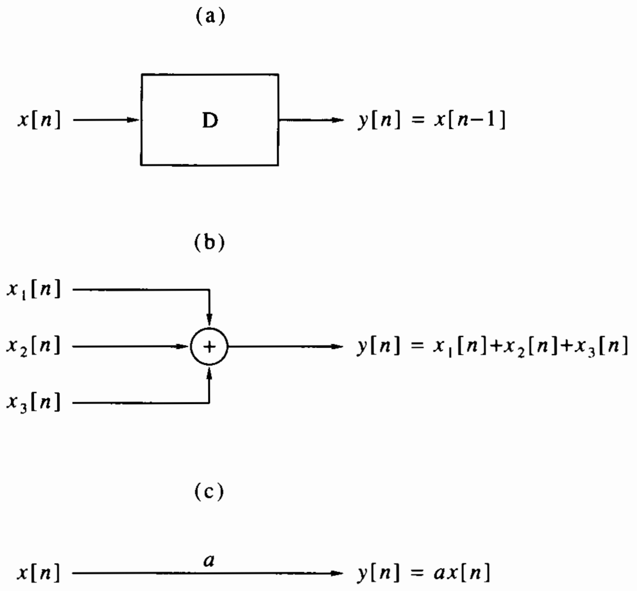

Figure 11.1 Basic building blocks for digital filter realizations: (a) unit delay; (b) adder; (c) multiplier by a constant.

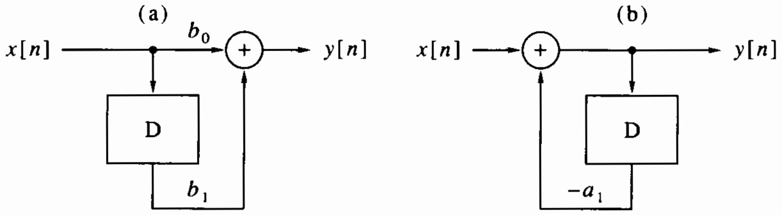

Figure 11.2 Realizations of first-order filters: (a) FIR; (b) IIR.

Example 11.2 Consider the first-order IIR filter

$$H_2^z(z) = \frac{1}{1 + a_1 z^{-1}}.$$

Figure 11.2(b) shows a realization of this filter using one delay element, one multiplier, and one adder. The input to the delay element at time n is $y[n]$, and its output is then $y[n-1]$. The output of the realization is therefore

$$y[n] = -a_1 y[n-1] + x[n],$$

and this is exactly the time-domain expression for $H_2^z(z)$. This realization is *recursive*: It builds the present value of $y[n]$ from its own past values and the input signal. Put another way, the realization uses *feedback*. The feedback enables the filter to have an infinite impulse response, although it has only one delay element. To see this, suppose that the output of the delay element has zero output at all $n < 0$, and we apply a unit-sample signal $\delta[n]$ at time $n = 0$. This causes $y[0]$ to be set to 1. At time $n = 1$, the value $y[0] = 1$ is transferred to the output of the delay element, is multiplied by $-a_1$, and is fed back to the input via the adder. The input $x[1]$ is now zero. Therefore, $y[1] = -a_1$. Continuing the same way, we see that $y[2] = a_1^2, y[3] = -a_1^3$, and in general $y[n] = (-a_1)^n$ for all $n \geq 0$. We get the impulse response of the filter at the output, as expected. □

11.1.2 Direct Realizations

Let $H^z(z)$ be a rational, causal, stable transfer function. We assume, for convenience, that the orders of the numerator and denominator polynomials of the filter transfer function are equal, that is, $p = q = N$. There is no loss of generality in this assumption since, for any p and q, we can always define $N = \max\{p, q\}$ and extend either $a(z)$ or $b(z)$ to degree N by adding zero-valued coefficients. Thus $H^z(z)$ is given by

$$\frac{Y^z(z)}{X^z(z)} = H^z(z) = \frac{b(z)}{a(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}}. \tag{11.1}$$

Let us introduce an auxiliary signal $u[n]$, related to the input and output signals $x[n]$ and $y[n]$ through

$$u[n] = -a_1 u[n-1] - \cdots - a_N u[n-N] + x[n], \tag{11.2}$$

$$y[n] = b_0 u[n] + b_1 u[n-1] + \cdots + b_N u[n-N], \tag{11.3}$$

or, in the z domain,

$$U^z(z) = \frac{1}{a(z)} X^z(z), \quad Y^z(z) = b(z) U^z(z). \tag{11.4}$$

Figure 11.3 shows a realization of (11.2) using the three types of building block (in the figure we have used $N = 3$ as an example). By passing $u[n]$ through a chain of N delay elements, we get the signals $\{u[n-1], u[n-2], \ldots, u[n-N]\}$. Then we can form $u[n]$ as a linear combination of the delayed signals, plus the input signal $x[n]$, as is expressed in (11.2). We need N multipliers for the coefficients $\{-a_1, \ldots, -a_N\}$, and N binary adders to form the sum. The realization uses feedback: It builds the present value of the signal $u[n]$ from its own past values and the present value of the input signal $x[n]$.

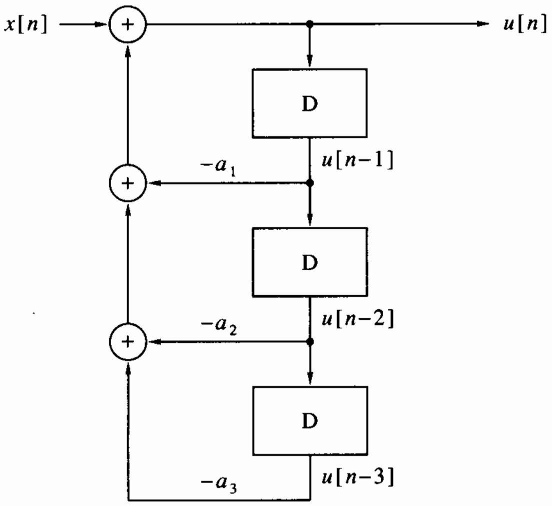

Figure 11.3 Realization of the auxiliary signal $u[n]$.

We can now use (11.3) for generating the output signal $y[n]$ from the auxiliary signal $u[n]$ and its delayed values. We do this by augmenting Figure 11.3 with $N + 1$

multipliers for the coefficients $\{b_0, \ldots, b_N\}$ and N adders. This results in the realization shown in Figure 11.4. Note that it is not necessary to increase the number of delay elements.

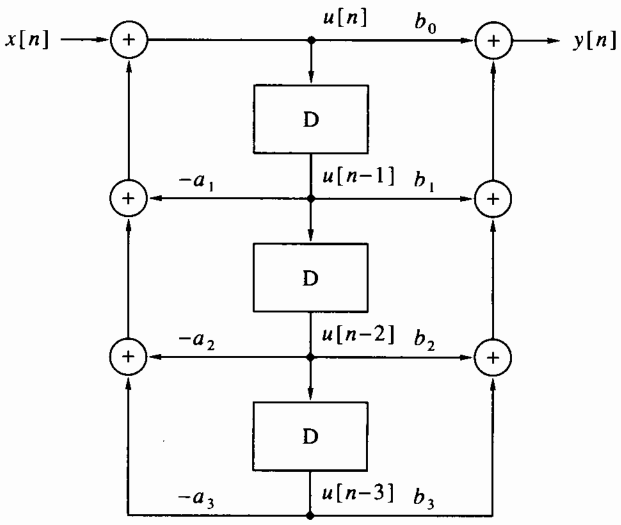

Figure 11.4 Direct realization of a digital IIR system.

The realization shown in Figure 11.4 is called a *direct realization*, or a *direct form*.[1] It has the following properties:

1. The number of delay elements N is the maximum of p and q. Assuming that the polynomials $a(z)$ and $b(z)$ have no common factor, this is the *minimum possible number of delays* in any realization of $H^z(z)$. The set of values at the outputs of the delay elements is called *the state* of the system.
2. There are $2N + 1$ multipliers and $2N$ adders. However, since some coefficients may be zero (e.g., if p and q are different), there may be a smaller number of adders and multipliers.
3. The realization is recursive, since it generates present values of its internal signals from past values of these signals. It can be shown that any realization of an IIR system must be recursive if it is required to include only a finite number of delay elements.
4. Assuming that the input signal $x[n]$ is causal, it is necessary to initialize the state components before feeding the input signal to the system. The state is usually initialized to zero. However, initialization to nonzero values is sometimes required, depending on the application.

We now explore an alternative to the direct realization. Instead of the variable $u[n]$, we introduce another auxiliary variable $v[n]$, such that

$$y[n] = -a_1 y[n-1] - \cdots - a_N y[n-N] + v[n], \tag{11.5}$$

$$v[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N], \tag{11.6}$$

or, in the z domain,

$$Y^z(z) = \frac{1}{a(z)} V^z(z), \quad V^z(z) = b(z) X^z(z). \tag{11.7}$$

Figure 11.5 shows a realization of (11.5) using the three types of building block (in the figure we use $N = 3$ as an example). The present value of $y[n]$ is built from its own past values and the auxiliary signal $v[n]$. To generate $y[n - N]$, we need N delay elements; these elements can be used for generating all intermediate delays, as shown in the figure. This realization effectively computes (11.5) in the z-domain form

$$Y^z(z) = (\cdots(-a_N z^{-1} - a_{N-1})z^{-1} - \cdots - a_1)z^{-1}Y^z(z) + V^z(z).$$

The state of this realization does not consist of pure delays of a single signal, but of linear combinations of different delays. As before, we need N multipliers for the coefficients $\{-a_1, \ldots, -a_N\}$, and N binary adders.

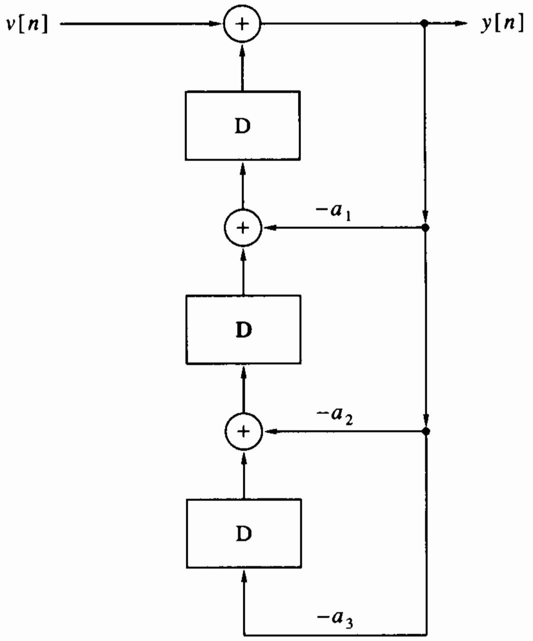

Figure 11.5 Realization of $y[n]$ from the auxiliary signal $v[n]$.

We can now use (11.6) for generating the auxiliary signal $v[n]$ from the input signal $x[n]$ and its delayed values. We do this by augmenting Figure 11.5 with $N+1$ multipliers for the coefficients $\{b_0, \ldots, b_N\}$ and N adders. This results in the realization shown in Figure 11.6. Note that it is not necessary to increase the number of delay elements, since the existing elements can take care of the necessary delays of the input signal.

The realization shown in Figure 11.6 is known as a *transposed direct realization* (or transposed direct form), for reasons explained next. The transposed direct realization shares the main properties of the direct realization. In particular, it has the same number of delays, multipliers, and binary adders. Note that, in Figure 11.6, there are $N - 1$ ternary adders and 2 binary adders, which are equivalent to $2N$ binary adders. However, the two realizations have different states. As long as the state is initialized to zero, this difference is inconsequential. However, when initialization to a nonzero state is necessary, the two realizations require different computations of the initial state.

Comparison of Figures 11.4 and 11.6 reveals that the latter can be obtained from the former by the following sequence of operations:

1. Reversal of the signal flow direction in all lines (i.e., reversal of all arrows).

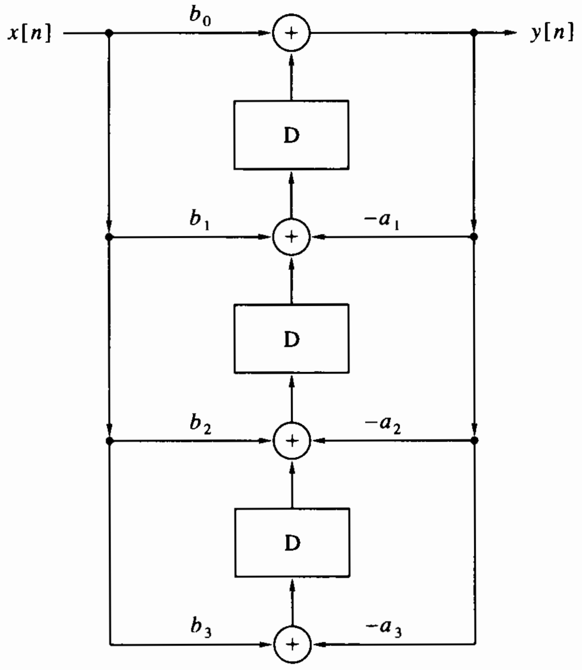

Figure 11.6 Transposed direct realization of a digital IIR system.

2. Replacing all adders by contact points, and all contact points by adders.
3. Interchanging the input and output.

This sequence of operations is called *transposition* of the realization. A known theorem in network theory states that transposition of a given realization leaves the transfer function of the realization invariant. Transposition of the transposed realization obviously gives back the original realization. Two such realizations are said to be *dual* to each other.

The procedure `direct` in Program 11.1 implements the two direct realizations of an IIR filter.[2] The MATLAB function `filter` performs the same computation more efficiently, since it is coded internally. Therefore, our program is intended for educational purposes, rather than for serious use.

11.1.3 Direct Realizations of FIR Filters

Realizations of digital FIR filters can be obtained from IIR filter realizations by specializing these realizations to the case $a(z) \equiv 1$. However, since FIR filters are usually either symmetric or antisymmetric, we can save about half the number of multiplications by exploiting symmetry. For an even-order filter we can write the filter's output as

$$y[n] = h[N/2]x[n - N/2] + \sum_{m=0}^{N/2-1} h[m](x[n - m] \pm x[n - N + m]), \tag{11.8}$$

whereas for an odd-order filter we put

$$y[n] = \sum_{m=0}^{(N-1)/2} h[m](x[n - m] \pm x[n - N + m]). \tag{11.9}$$

The plus sign is for a symmetric filter (type I or II), and the minus sign for an antisymmetric one (type III or IV). Figure 11.7 shows a realization of (11.8). This realization can be regarded as a specialization of the direct realization shown in Figure 11.4 to an FIR filter. As we see, symmetry (or antisymmetry) helps reducing the number of multiplications from $N + 1$ to $\lfloor N/2 \rfloor + 1$; the number of additions, however, is still N.

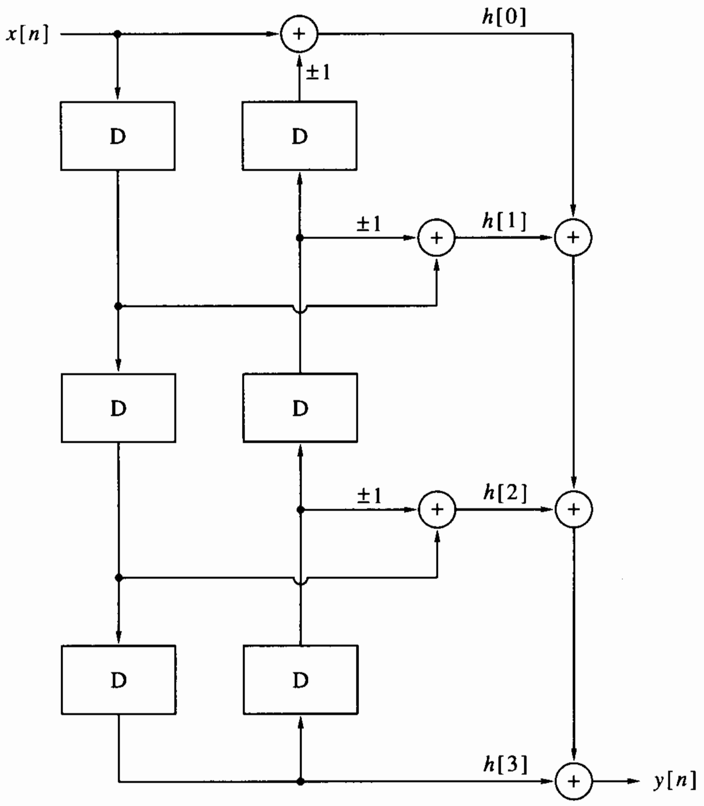

Figure 11.7 Direct realization of a symmetric or antisymmetric FIR filter.

By exploiting the rules of transposition of realizations, we get from Figure 11.7 the transposed direct realization shown in Figure 11.8.

11.1.4 Parallel Realization

Recall the partial fraction decomposition of a rational causal transfer function whose poles are simple:

$$H^z(z) = \frac{b(z)}{a(z)} = c_0 + \cdots + c_{q-p}z^{-(q-p)} + \sum_{i=1}^{p} \frac{A_i}{1 - \alpha_i z^{-1}}. \tag{11.10}$$

For most practical digital IIR filters $q \le p$, so the c coefficients beyond c_0 are absent from the right side. Also, if α_i is complex then A_i is complex as well, and the conjugate fraction $\bar{A}_i/(1 - \bar{\alpha}_i z^{-1})$ also appears on the right side. The two can be brought together

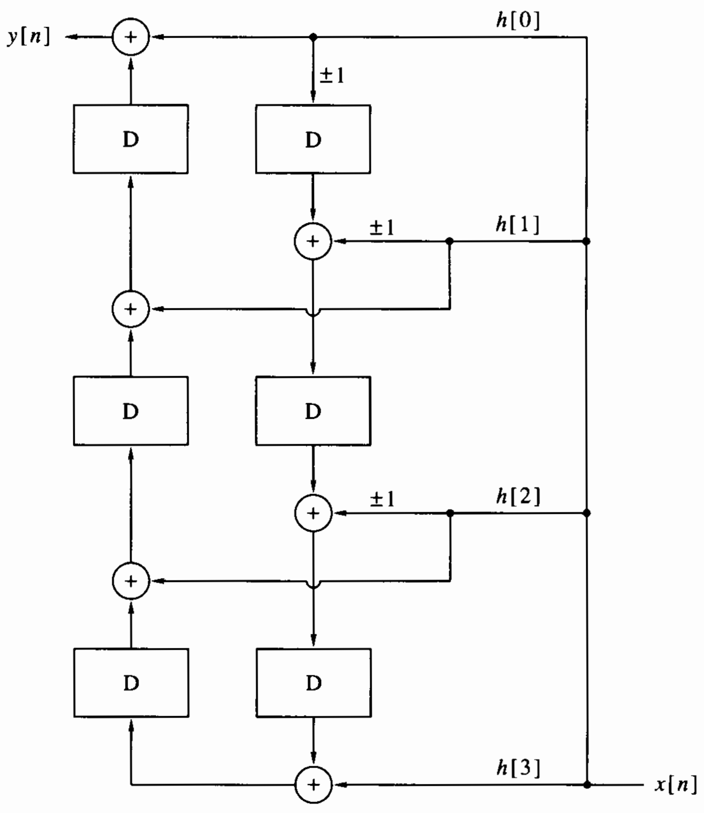

Figure 11.8 Transposed direct realization of a symmetric or antisymmetric FIR filter.

under a common denominator, yielding the real fraction

$$\frac{(A_i + \bar{A}_i) - (A_i\bar{\alpha}_i + \bar{A}_i\alpha_i)z^{-1}}{1 - (\alpha_i + \bar{\alpha}_i)z^{-1} + \alpha_i\bar{\alpha}_i z^{-2}}.$$

Let us denote the order of the filter by N instead of p, for notational consistency with the preceding section. Also, let N_1 be the number of real poles and $2N_2$ the number of complex poles, so $N = N_1 + 2N_2$. After joining complex conjugate fractions, we can bring (11.10) to the form

$$H^z(z) = c_0 + \sum_{i=1}^{N_1} \frac{f_i}{1 + e_i z^{-1}} + \sum_{i=1}^{N_2} \frac{f_{N_1+2i-1} + f_{N_1+2i}z^{-1}}{1 + e_{N_1+2i-1}z^{-1} + e_{N_1+2i}z^{-2}}, \tag{11.11}$$

where the real numbers $\{e_i, f_i,\ 1 \le i \le N\}$ depend on $\{A_i, \alpha_i,\ 1 \le i \le N\}$.

The sum on the right side of (11.11) corresponds to a parallel connection of the individual terms. Each of the first- and second-order terms can be implemented by a direct realization. The constant term c_0 is realized by simple multiplication. Figure 11.9 illustrates the result for $N_1 = 1$, $N_2 = 1$, $N = 3$. The realization thus obtained is called *parallel realization*.

Parallel realization requires the same number of delay elements as the direct realizations. If $q = p$, it also requires the same amount of additions and multiplications. If $q < p$, the direct realizations are more economical, since in this case they require only $p + q + 1$ multiplications and additions, whereas the parallel realization still requires $2p + 1$ operations of each kind (the reason is that all the coefficients f_i will be nonzero

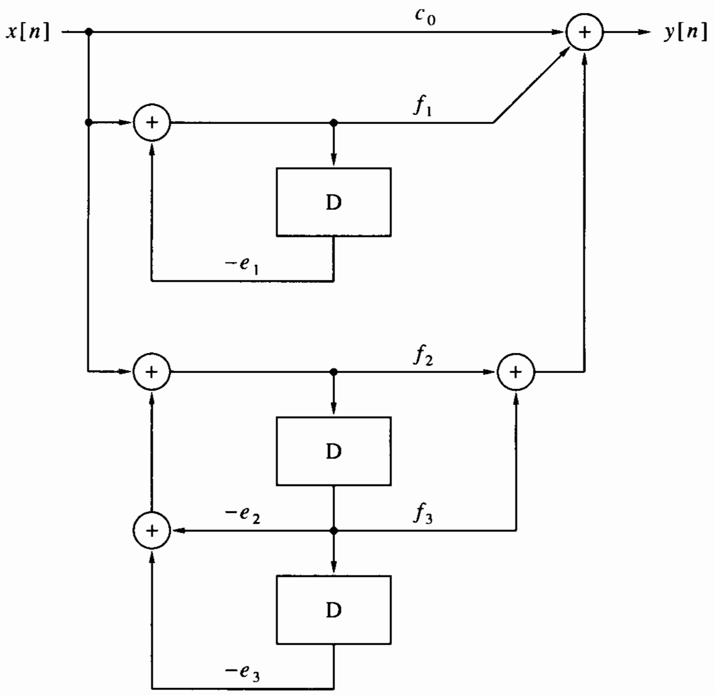

Figure 11.9 Parallel realization of a digital IIR system.

in general). As we have said, the parallel realization is limited to systems whose poles are simple. It can be extended to the case of multiple poles, but then a parallel realization is rarely used.[3] The advantages of parallel realization over direct realizations will become clear in Section 11.5, when we study the sensitivity of the frequency response of the filter to finite word length.

The procedure `tf2rpf` in Program 11.2 computes the parallel decomposition (11.11) of a digital IIR filter. The program first calls `tf2pf` (Program 7.1) to compute the complex partial fraction decomposition, then combines complex pairs to real second-order sections.

The procedure `parallel` in Program 11.3 implements the parallel realization of a digital IIR filter. It accepts the parameters computed by the program `tf2rpf`, computes the response to each second-order section separately, and adds the results, including the constant term. The program is slightly inefficient in that it treats first-order sections as second-order ones, so it is likely to perform redundant multiplications and additions of zero values. However, common IIR filters either do not have real poles (if the order is even), or they have a single real pole (if the order is odd). Therefore, the redundant operations do not amount to much. In real-time implementations, however, care should be taken to avoid them. We also reiterate that the MATLAB implementation is rather time consuming, due to the inefficiency of MATLAB in loop computations.

11.1.5 Cascade Realization

Let us assume again that $q = p = N$ for the digital filter in question. Assume for now that N is even. Recall the pole-zero factorization of the transfer function,

$$H^z(z) = b_0 \frac{\prod_{i=1}^{N}(1 - \beta_i z^{-1})}{\prod_{i=1}^{N}(1 - \alpha_i z^{-1})}. \tag{11.12}$$

Since N is even, we can always rewrite (11.12) as

$$H^z(z) = b_0 \prod_{i=1}^{N/2} \frac{(1 + h_{2i-1}z^{-1} + h_{2i}z^{-2})}{(1 + g_{2i-1}z^{-1} + g_{2i}z^{-2})}. \tag{11.13}$$

The second-order factors in the numerator and the denominator are obtained by expanding conjugate pairs of zeros or poles; hence the coefficients $\{g_i, h_i, 1 \leq i \leq N\}$ are real. In general, some poles or zeros may be real. Since we have assumed that N is even, their number must be even, so we can expand them in pairs as well. Thus in general, the second-order terms in (11.13) may correspond to either real or complex pairs of poles or zeros.

A product of transfer functions represents a cascade connection of the factors. Therefore, we can implement (11.13) as a cascade connection of $N/2$ sections, each of order 2. Each section can be realized by either of the two direct realizations. Figure 11.10 illustrates this connection for $N = 4$. This is called a *cascade realization*. Note that the constant gain b_0 can appear anywhere along the cascade (in Figure 11.10 it appears in the middle between the two sections). The second-order sections are also called *bi-quads*.

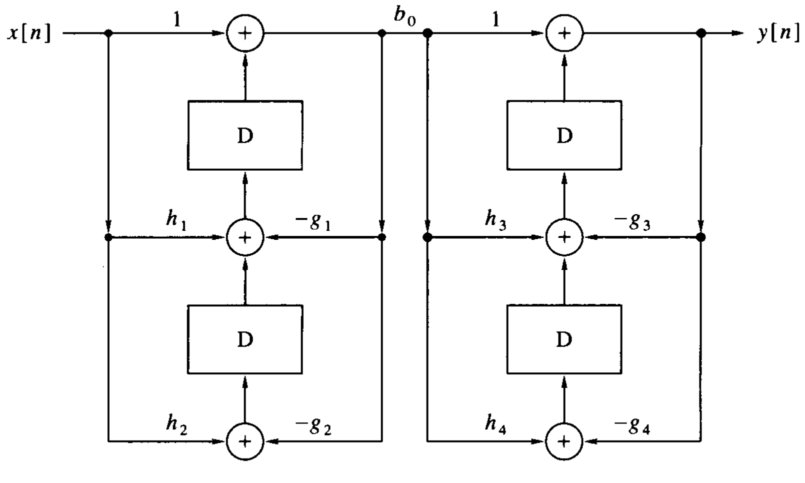

Figure 11.10 Cascade realization of a digital IIR system.

Remarks

1. Although we have assumed that N is even, the realization can be easily extended to the case of odd N. In this case there is an extra first-order term, so we must add a first-order section in cascade.
2. Although we have assumed that $p = q$, this condition is not necessary. Extra poles can be represented by sections with zero values for the h coefficients, whereas extra zeros can be represented by sections with zero values for the g coefficients.

3. The realization is minimal in terms of number of delays, additions and multiplications (with the understanding that zero-valued coefficients save the corresponding multiplications and additions).
4. The realization is nonunique, since:
   (a) There are multiple ways of pairing each second-order term in the denominator with one in the numerator. In Section 11.1.6 we discuss the pairing problem in detail.
   (b) There are multiple ways of ordering the sections in the cascade connection.
   (c) There are multiple ways of inserting the constant gain factor b_0.
5. Contrary to the parallel realization, the cascade realization is not limited to simple poles. Moreover, it does not require the condition $q \le p$. Cascade realization is applicable to FIR filters, although its use for such filters is relatively uncommon.

11.1.6 Pairing in Cascade Realization

When cascade realization is implemented in floating point and at a high precision (such as in MATLAB), the pairing of poles and zeros to second-order sections is of little importance. However, in fixed-point implementations and short word lengths, it is advantageous to pair poles and zeros to produce a frequency response for each section that is as flat as possible (i.e., such that the ratio of the maximum to the minimum magnitude response is close to unity). We now describe a pairing procedure that approximately achieves this goal. We consider only digital filters obtained from one of the four standard filter types (Butterworth, Chebyshev-I, Chebyshev-II, elliptic) through an analog frequency transformation followed by a bilinear transform. Such filters satisfy the following three properties:

1. The number of zeros is equal to the number of poles. If the underlying analog filter has more poles than zeros, the extra zeros of the digital filter are all at $z = -1$.
2. The number of complex poles is never smaller than the number of complex zeros.
3. The number of real poles is not larger than 2. A low-pass filter has one real pole if its order is odd, and this pole may be transformed to two real poles or to a pair of complex poles by either a low-pass to band-pass or a low-pass to band-stop transformation. Except for those, all poles of the analog filter are complex, hence so are the poles of the digital filters.

The basic idea is to pair each pole with a zero as close to it as possible. By what we have learned in Section 7.6, this makes the magnitude response of the pole-zero pair as flat as possible. The pairing procedure starts at the pair of complex poles nearest to the unit circle (i.e., those with the largest absolute value) and pairs them with the nearest complex zeros. It then removes these two pairs from the list and proceeds according to the same rule. When all the complex zeros are exhausted, pairing continues with the real zeros according to the same rule. Finally, there may be left up to two real poles, and these are paired with the remaining real zeros.

The procedure `pairpz` in Program 11.4 implements this algorithm. It receives the vectors of poles and zeros, supplied by the program `iirdes` (Program 10.8) and supplies arrays of second-order numerator and denominator polynomials (a first-order pair, if any, is represented as a second-order pair with zero coefficient of z^{-2}). The routine `cplxpair` is a built-in MATLAB function that orders the poles (or zeros) in conjugate pairs, with real ones (if any) at the end. The program then selects one representative of each conjugate pair and sorts them in decreasing order of magnitude. Next the program loops over the complex poles and, for each one, finds the nearest complex zero. Every paired zero is removed from the list. The polynomials of the corresponding second-order section are computed and stored. When the complex zeros are exhausted, the remaining complex poles are paired with the real zeros using the same search procedure. Finally, the real poles are paired with the remaining real zeros.

The procedure `cascade` in Program 11.5 implements the cascade realization of a digital IIR filter. It accepts the parameters computed by the program `pairpz`. The input sequence is fed to the first section, the output is fed to the second section, and so forth. Finally, the result is multiplied by the constant gain.

The cascade realization is usually considered as the best of all those we have discussed, for reasons to be explained in Section 11.5; therefore, it is the most widely used.

11.1.7 A Coupled Cascade Realization

Direct realization of second-order sections used in cascade realization may be unsatisfactory, especially if the word length is short and the filter has poles near $z = 1$ or $z = -1$. Detailed explanation of this phenomenon is deferred until later in this chapter. We now present an alternative realization of second-order sections, which offers an improvement over a direct realization in case of a short word length. This so-called *coupled realization* is shown in Figure 11.11. The parameters α_r, α_i are the real and imaginary parts of the complex pole α of the second-order section, that is,

$$g_1 = -2\alpha_r = -2\Re\{\alpha\}, \quad g_2 = \alpha_r^2 + \alpha_i^2 = |\alpha|^2. \tag{11.14}$$

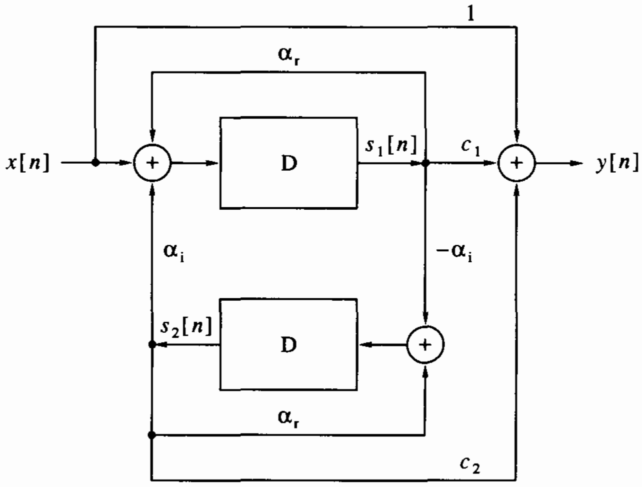

Figure 11.11 A coupled second-order section for cascade realization.

We now derive the relationship between the outputs $s_1[n]$ and $s_2[n]$ of the delay elements and the input $x[n]$. This will enable us to prove that the transfer function from $x[n]$ to $y[n]$ has the right denominator and will provide the values of the coefficients c_1, c_2 necessary to obtain the right numerator. We do the calculation in the z domain. We have from the figure,

$$zS_1^z(z) = X^z(z) + \alpha_r S_1^z(z) + \alpha_i S_2^z(z), \tag{11.15a}$$

$$zS_2^z(z) = \alpha_r S_2^z(z) - \alpha_i S_1^z(z). \tag{11.15b}$$

Or, in a matrix form,

$$\begin{bmatrix} z - \alpha_r & -\alpha_i \\ \alpha_i & z - \alpha_r \end{bmatrix} \begin{bmatrix} S_1^z(z) \\ S_2^z(z) \end{bmatrix} = \begin{bmatrix} X^z(z) \\ 0 \end{bmatrix}. \tag{11.16}$$

Therefore,

$$\begin{bmatrix} S_1^z(z) \\ S_2^z(z) \end{bmatrix} = \frac{X^z(z)}{z^2 - 2\alpha_r z + \alpha_r^2 + \alpha_i^2} \begin{bmatrix} z - \alpha_r \\ -\alpha_i \end{bmatrix} = \frac{X^z(z)}{z^2 + g_1 z + g_2} \begin{bmatrix} z - \alpha_r \\ -\alpha_i \end{bmatrix}. \tag{11.17}$$

We see that the denominator has the correct form. We can now express the output $Y^z(z)$ in terms of $X^z(z)$, $S_1^z(z)$, $S_2^z(z)$ and substitute (11.17) to get

$$Y^z(z) = X^z(z) + c_1 S_1^z(z) + c_2 S_2^z(z) = \frac{z^2 + (g_1 + c_1)z + (g_2 - c_1\alpha_r - c_2\alpha_i)}{z^2 + g_1 z + g_2} X^z(z). \tag{11.18}$$

Therefore, by choosing

$$c_1 = h_1 - g_1, \quad c_2 = \frac{g_2 - h_2 - c_1\alpha_r}{\alpha_i}, \tag{11.19}$$

we get the desired transfer function.

It follows from the preceding derivation that the section shown in Figure 11.11 can be constructed only for a pair of complex poles and does not apply to pairs of real poles. A pair of real poles must be realized by a second-order direct realization or by a cascade of two first-order sections. Each section in a coupled realization requires six multiplications and five additions, as compared with four multiplications and four additions for a direct realization. Despite the extra computations, the coupled realization is sometimes preferred to a direct realization of second-order sections, for reasons explained in Section 11.5.

11.1.8 FFT-Based Realization of FIR Filters

In Section 5.6 we showed how to use the fast Fourier transform for efficient convolution of a fixed-length sequence by a potentially long sequence. The overlap-add algorithm described there (or the overlap-save algorithm developed in Problem 5.23) can be used for FIR filter realization. The impulse response $h[n]$ is taken as the fixed-length sequence, and the input $x[n]$ as the long sequence. Table 5.2 gave the optimal FFT length as a function of the order of the filter (with N_2 in the table corresponding to $N+1$ here). FFT-based FIR realization requires more memory than direct realization and is performed blockwise (as opposed to point by point), but is more efficient when the order of the filter is at least 18, so it is a serious candidate to consider in some applications.

11.2 State-Space Representations of Digital Filters

11.2.1 The State-Space Concept

A difference equation expresses the present output of the system in terms of past outputs, and present and past inputs. In discussing realizations of difference equations, we noted that the number of delay elements needed in any realization of the difference equation is equal to $\max\{p, q\}$. The delay elements represent the *memory* of the system, in the sense that their inputs must be stored and remembered from one time point to the next. Until now, the outputs of the delay elements were of no interest to us by themselves, only as auxiliary variables.

In this section we study a different way of representing rational LTI systems. In this representation, the outputs of the delay elements play a central role. Collected together, they are called the *state vector* of the system. Correspondingly, the representations we are going to present are called *state-space representations*. A state-space representation comprises two equations: The *state equation* expresses the time evolution of the state vector as a function of its own past and the input signal; the *output equation* expresses the output signal as a function of the state vector and the input signal.

To motivate the state-space concept, consider again the direct realization shown in Figure 11.4; introduce the notation

$$s_1[n] = u[n-1], \quad s_2[n] = u[n-2], \quad s_3[n] = u[n-3].$$

In other words, the signal $s_k[n]$ is the output of the kth delay element (starting from the top) at time n. Then we can read the following relationships directly from the figure:

$$s_1[n+1] = x[n] - a_1 s_1[n] - a_2 s_2[n] - a_3 s_3[n], \tag{11.20a}$$

$$s_2[n+1] = s_1[n], \tag{11.20b}$$

$$s_3[n+1] = s_2[n], \tag{11.20c}$$

$$y[n] = b_0 x[n] + (b_1 - b_0 a_1)s_1[n] + (b_2 - b_0 a_2)s_2[n] + (b_3 - b_0 a_3)s_3[n]. \tag{11.20d}$$

A compact way of writing (11.20) is

$$\begin{bmatrix} s_1[n+1] \\ s_2[n+1] \\ s_3[n+1] \end{bmatrix} = \begin{bmatrix} -a_1 & -a_2 & -a_3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} s_1[n] \\ s_2[n] \\ s_3[n] \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} x[n], \tag{11.21}$$

$$y[n] = \begin{bmatrix} c_1 & c_2 & c_3 \end{bmatrix} \begin{bmatrix} s_1[n] \\ s_2[n] \\ s_3[n] \end{bmatrix} + b_0 x[n], \tag{11.22}$$

where

$$c_k = b_k - b_0 a_k, \quad 1 \le k \le 3. \tag{11.23}$$

These equations can be easily extended to any order N as follows. Define

$$\boldsymbol{s}[n] = \begin{bmatrix} s_1[n] \\ s_2[n] \\ \vdots \\ s_N[n] \end{bmatrix}, \ \boldsymbol{A}_1 = \begin{bmatrix} -a_1 & -a_2 & \dots & -a_{N-1} & -a_N \\ 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}, \ \boldsymbol{B}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{11.24}$$

$$c_k = b_k - b_0 a_k, \quad \boldsymbol{C}_1 = \begin{bmatrix} c_1 & c_2 & \dots & c_N \end{bmatrix}, \quad \boldsymbol{D}_1 = b_0. \tag{11.25}$$

Then

$$\boldsymbol{s}[n+1] = \boldsymbol{A}_1 \boldsymbol{s}[n] + \boldsymbol{B}_1 x[n], \tag{11.26a}$$

$$y[n] = \boldsymbol{C}_1 \boldsymbol{s}[n] + \boldsymbol{D}_1 x[n]. \tag{11.26b}$$

Equations (11.26a,b), together with the definitions (11.24), (11.25), are the state-space representation of the direct realization of the transfer function (11.1). The first is the *state equation*, and the second is the *output equation*. Together they describe the same input-output behavior as the difference equation

$$y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{N} b_k x[n-k]. \tag{11.27}$$

Remember that, as we did for (11.1), here too we define $N = \max\{p, q\}$ and extend the sequences $\{a_k\}$, $\{b_k\}$ by zeros as necessary.

It is clear from the preceding construction that every difference equation has a corresponding state-space representation. The procedure `tf2ss` in Program 11.6 computes the matrices A_1, B_1, C_1, D_1 given in (11.24), (11.25). These matrices are called, respectively, the *state, input, output*, and *direct transmission* matrices. The first is $N \times N$, the second is $N \times 1$, the third is $1 \times N$, and the fourth is 1×1 (i.e., it is a scalar). A matrix such as A_1, whose first row is arbitrary, whose entries along the first subdiagonal are 1, and whose other entries are 0, is called a *top-row companion matrix*.

State-space representation is not limited to the direct realization. Let us now illustrate the state-space construction procedure for the transposed direct realization shown in Figure 11.6. As before, we denote the outputs of the delay elements by $\{s_k[n],\ 1 \le k \le N\}$, from top to bottom. The state vector $s[n]$ is the column vector of the $s_k[n]$. This time we define the state-space matrices as

$$A_2 = \begin{bmatrix} -a_1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ -a_{N-1} & 0 & \dots & 0 & 1 \\ -a_N & 0 & \dots & 0 & 0 \end{bmatrix}, \quad c_k = b_k - b_0 a_k, \quad B_2 = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix}. \tag{11.28}$$

$$C_2 = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}, \quad D_2 = b_0. \tag{11.29}$$

Inspecting the connections in Figure 11.6 in relation to the definitions in (11.28), (11.29), we see that the transposed direct realization obeys the equations

$$s[n+1] = A_2 s[n] + B_2 x[n], \tag{11.30a}$$

$$y[n] = C_2 s[n] + D_2 x[n]. \tag{11.30b}$$

Equations (11.30a,b), together with the definitions (11.28), (11.29), are the state-space representation of the transposed direct realization of the transfer function (11.1). We remark that the state vector $s[n]$ in (11.30) is *different* from the one in (11.26), although we have used the same notation for convenience.

It is now clear that a given rational LTI system has more than one state-space representation, since we have demonstrated the existence of at least two such representations. We shall soon see that the number of state-space representations of a given rational LTI system is infinite. For now we note that the two representations we have derived are related as follows:

$$A_2 = A_1', \quad B_2 = C_1', \quad C_2 = B_1', \quad D_2 = D_1. \tag{11.31}$$

The two representations are said to be *dual*, or *transpose* of each other. In Section 11.1.2 we explained the sequence of operations that must be performed on a given realization to obtain its transpose. Now we see the corresponding operations on their state-space matrices: Transpose the state matrix, interchange and transpose the input and output matrices, and leave the direct transmission matrix (which is scalar in fact) unchanged.

Example 11.3 The signals $s_1[n], s_2[n]$ in Figure 11.11 are the state of the coupled realization of a second-order section. The following state equations describe this representation:

$$s[n+1] = \begin{bmatrix} \alpha_r & \alpha_i \\ -\alpha_i & \alpha_r \end{bmatrix} s[n] + \begin{bmatrix} 1 \\ 0 \end{bmatrix} x[n], \tag{11.32a}$$

$$y[n] = \begin{bmatrix} c_1 & c_2 \end{bmatrix} s[n] + x[n], \tag{11.32b}$$

where α_r, α_i are the real and imaginary parts of the complex pole α. □

11.2.2 Similarity Transformations

We now prove that a given difference equation (or a rational transfer function) has an infinite number of state-space representations. Let T be an arbitrary $N \times N$ nonsingular matrix. Define

$$\tilde{s}[n] = T^{-1}s[n], \tag{11.33}$$

and

$$\tilde{A} = T^{-1}A_1T, \quad \tilde{B} = T^{-1}B_1, \quad \tilde{C} = C_1T, \quad \tilde{D} = D_1. \tag{11.34}$$

Note that, whereas both s and $\tilde{s}$ depend on n, the matrix T is fixed (independent of n). Now go back to (11.26a) and premultiply it by T^{-1} to get

$$T^{-1}s[n+1] = T^{-1}A_1s[n] + T^{-1}B_1x[n] = T^{-1}A_1TT^{-1}s[n] + T^{-1}B_1x[n]. \tag{11.35}$$

Therefore, substituting (11.33) and (11.34), we get

$$\tilde{s}[n+1] = \tilde{A}\tilde{s}[n] + \tilde{B}x[n], \tag{11.36}$$

and similarly for (11.26b),

$$y[n] = C_1TT^{-1}s[n] + D_1x[n] = \tilde{C}\tilde{s}[n] + \tilde{D}x[n]. \tag{11.37}$$

The state-space representation (11.36), (11.37) is said to be *similar* to (11.26), and the relationships (11.33), (11.34) are called *similarity transformations*. Since there is an infinite number of $N \times N$ nonsingular matrices, there is an infinite number of state-space representations similar to (11.26). Although the state vectors of similar representations are different, the input-output relationships they describe are identical. A direct proof of this property is discussed in Problem 11.8.

11.2.3 Applications of State Space

State-space representations are of great importance in linear system theory. Here we describe only few of their many uses. We concentrate on uses relevant to topics dealt with earlier in this book. In particular, we show how state-space representations of a given system are related to its impulse response, input-output response, and transfer function.

Let

$$s[n+1] = As[n] + Bx[n], \tag{11.38a}$$

$$y[n] = Cs[n] + Dx[n], \tag{11.38b}$$

be a state-space representation of a given LTI system. Suppose the system is initially at rest, that is, $s[n] = 0$ for $n \leq 0$ and $x[n] = 0$ for $n < 0$. Now suppose we input a unit-sample signal at $n = 0$. Then the output of the system at all $n \geq 0$ is, by definition, the impulse response of the system. Let us determine the impulse response from (11.38). We have from (11.38a)

$$\boldsymbol{s}[0] = 0,\ \boldsymbol{s}[1] = \boldsymbol{B},\ \boldsymbol{s}[2] = \boldsymbol{AB},\ \boldsymbol{s}[3] = \boldsymbol{A}^2\boldsymbol{B}, \dots \tag{11.39}$$

and in general

$$\boldsymbol{s}[n] = \begin{cases} 0, & n \le 0, \\ \boldsymbol{A}^{n-1}\boldsymbol{B}, & n > 0. \end{cases} \tag{11.40}$$

Therefore, we have from (11.38b)

$$h[n] = \begin{cases} 0, & n < 0, \\ \boldsymbol{D}, & n = 0, \\ \boldsymbol{CA}^{n-1}\boldsymbol{B}, & n > 0. \end{cases} \tag{11.41}$$

Observe how easy it is to obtain the impulse response from the state-space representation. In particular, there is no need to resort to z-transforms and partial fraction decompositions, as we did when we started from the difference equation.

Next we derive the system's response to an arbitrary (but causal) input signal. As before, we assume that the system is initially at rest. Then we get from (11.38a)

$$\boldsymbol{s}[0] = 0,\ \boldsymbol{s}[1] = \boldsymbol{B}x[0],\ \boldsymbol{s}[2] = \boldsymbol{AB}x[0] + \boldsymbol{B}x[1], \tag{11.42a}$$

$$\boldsymbol{s}[3] = \boldsymbol{A}^2\boldsymbol{B}x[0] + \boldsymbol{AB}x[1] + \boldsymbol{B}x[2], \dots \tag{11.42b}$$

and, in general,

$$\boldsymbol{s}[n] = \begin{cases} 0, & n \le 0, \\ \sum_{k=0}^{n-1} \boldsymbol{A}^{n-k-1}\boldsymbol{B}x[k], & n > 0. \end{cases} \tag{11.43}$$

Therefore, we get from (11.38b),

$$y[n] = \begin{cases} 0, & n < 0, \\ \boldsymbol{D}x[0], & n = 0, \\ \sum_{k=0}^{n-1} \boldsymbol{CA}^{n-k-1}\boldsymbol{B}x[k] + \boldsymbol{D}x[n], & n > 0. \end{cases} \tag{11.44}$$

Again, we were able to compute the system's response to an arbitrary input without using z-transforms and partial fraction decompositions. Observe also from (11.41) that the relationship (11.44) is $y[n] = \{h * x\}[n]$, as expected.

The transfer function of a system described in state-space form is derived as follows. Take the z-transform of (11.38a) to obtain

$$z\boldsymbol{S}^z(z) = \boldsymbol{AS}^z(z) + \boldsymbol{B}X^z(z), \tag{11.45}$$

hence

$$(z\boldsymbol{I}_N - \boldsymbol{A})\boldsymbol{S}^z(z) = \boldsymbol{B}X^z(z) \implies \boldsymbol{S}^z(z) = (z\boldsymbol{I}_N - \boldsymbol{A})^{-1}\boldsymbol{B}X^z(z). \tag{11.46}$$

Now substitute in the z-transform of (11.38b) to obtain

$$Y^z(z) = \boldsymbol{C}(z\boldsymbol{I}_N - \boldsymbol{A})^{-1}\boldsymbol{B}X^z(z) + \boldsymbol{D}X^z(z). \tag{11.47}$$

Therefore, the transfer function of the system is

$$H^z(z) = \frac{Y^z(z)}{X^z(z)} = \boldsymbol{C}(z\boldsymbol{I}_N - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}. \tag{11.48}$$

Direct use of (11.48) for computing the transfer function $H^z(z)$ requires the inversion of a $N \times N$ matrix containing the nonnumerical variable z. This can be done by hand if N is small, but it is tedious for large N. A computer program can be written for this purpose, but this is not straightforward either. It turns out that $H^z(z)$ can be computed without explicit inversion of $(z\boldsymbol{I}_N - \boldsymbol{A})$, as we now explain.

We observe that

$$(z\boldsymbol{I}_N - \boldsymbol{A})^{-1} = \frac{\operatorname{adj}(z\boldsymbol{I}_N - \boldsymbol{A})}{\det(z\boldsymbol{I}_N - \boldsymbol{A})}, \tag{11.49}$$

where *det* denotes the determinant of the argument, and *adj* denotes the adjugate matrix (matrix of cofactors) of the argument. Therefore,

$$H^z(z) = \frac{\boldsymbol{C}\operatorname{adj}(z\boldsymbol{I}_N - \boldsymbol{A})\boldsymbol{B}}{\det(z\boldsymbol{I}_N - \boldsymbol{A})} + \boldsymbol{D}. \tag{11.50}$$

The elements of the adjugate matrix are polynomials in positive powers of z of degree $N-1$ at most. Since $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ are constant matrices, $\boldsymbol{C}\operatorname{adj}(z\boldsymbol{I}_N - \boldsymbol{A})\boldsymbol{B}$ is a polynomial of degree $N-1$ at most.

The denominator polynomial $a(z)$ of the transfer function is the characteristic polynomial of the matrix $\boldsymbol{A}$, so it can be computed directly from $\boldsymbol{A}$. To compute the numerator polynomial $b(z)$ of the transfer function, observe that

$$b(z) = a(z)H^z(z), \tag{11.51}$$

or

$$\begin{aligned} &b_0 + b_1 z^{-1} + \cdots + b_N z^{-N} \\ &= (1 + a_1 z^{-1} + \cdots + a_N z^{-N})(h[0] + h[1]z^{-1} + \cdots + h[N]z^{-N} + \cdots). \end{aligned} \tag{11.52}$$

Equating powers of z^{-k} for $0 \le k \le N$ in this equation gives

$$\begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_N \end{bmatrix} = \begin{bmatrix} h[0] & 0 & \dots & 0 \\ h[1] & h[0] & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ h[N] & h[N-1] & \dots & h[0] \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ \vdots \\ a_N \end{bmatrix}. \tag{11.53}$$

The impulse response terms $\{h[n],\ 0 \le n \le N\}$ can be obtained using (11.41). Therefore, all the parameters on the right side of (11.53) can be computed, thus enabling the computation of $b(z)$.

The procedure `ss2tf` in Program 11.7 implements the computation. The characteristic polynomial is computed by the MATLAB built-in function `poly`. The terms $h[n]$ are computed using (11.41). Finally, the numerator polynomial is computed using (11.53).

11.3 General Block-Diagram Manipulation

The realizations we have seen are specific examples of discrete-time LTI networks. Since these realizations are relatively simple, we could relate them to the transfer functions they represent merely by inspection. For more complex networks, inspection may not be sufficient and more general tools are required.

A general discrete-time LTI network consists of blocks, which are LTI themselves, interconnected to make the complete system LTI. This restriction leaves us with only simple possible connections: The input to a block must be a linear combination of outputs of other blocks, with constant coefficients. In addition, some blocks may be fed from one or more external inputs. The output, or outputs, of the network are linear combinations of outputs of blocks, and possibly of the external inputs.

The preceding description can be expressed in mathematical terms as follows; see Figure 11.12.

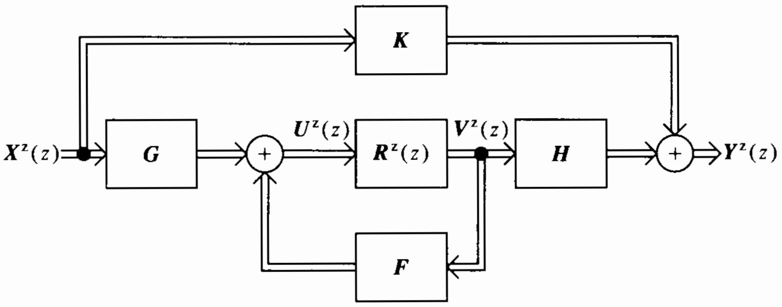

Figure 11.12 A general discrete-time LTI network.

1. Suppose we are given L LTI blocks, and let $U_l^z(z)$, $R_l^z(z)$, $V_l^z(z)$ be the input, the transfer function, and the output of the Lth block, respectively. Then we have
$$V_l^z(z) = R_l^z(z)U_l^z(z), \quad 1 \le l \le L. \tag{11.54}$$
We can write these equations collectively as
$$\boldsymbol{V}^z(z) = \boldsymbol{R}^z(z)\boldsymbol{U}^z(z), \tag{11.55}$$
where $\boldsymbol{U}^z(z)$, $V_l^z(z)$ are L-dimensional vectors whose components are functions of z, and $\boldsymbol{R}^z(z)$ is an $L \times L$ diagonal matrix, with the functions $R_l^z(z)$ along its main diagonal.

2. Suppose we are given M external inputs whose z-transforms are $X_m^z(z)$, and P external outputs whose z-transforms are $Y_p^z(z)$. We denote the vectors built from these functions by $\boldsymbol{X}^z(z)$ (of dimension M) and $\boldsymbol{Y}^z(z)$ (of dimension P), respectively.

3. Let the block inputs be related to the block outputs and the external inputs by
$$\boldsymbol{U}^z(z) = \boldsymbol{F}\boldsymbol{V}^z(z) + \boldsymbol{G}\boldsymbol{X}^z(z), \tag{11.56}$$
where
   (a) $\boldsymbol{F}$ is an $L \times L$ matrix of constants, called the *interconnection matrix*. The element $f_{i,j}$ of the matrix represents the multiplication factor of the connection from the output of the jth block to the input of the ith block. If there is no such connection, then $f_{i,j} = 0$.
   (b) $\boldsymbol{G}$ is an $L \times M$ matrix of constants, called the *input matrix*. The element $g_{i,j}$ of the matrix represents the multiplication factor of the connection from the jth external input to the input of the ith block. If there is no such connection, then $g_{i,j} = 0$.

4. Let the external outputs be related to the block outputs and the external inputs by
$$\boldsymbol{Y}^z(z) = \boldsymbol{H}\boldsymbol{V}^z(z) + \boldsymbol{K}\boldsymbol{X}^z(z), \tag{11.57}$$
where
   (a) $\boldsymbol{H}$ is a $P \times L$ matrix of constants, called the *output matrix*. The element $h_{i,j}$ of the matrix represents the multiplication factor of the connection from the output of the jth block to the ith external output. If there is no such connection, then $h_{i,j} = 0$.
   (b) $\boldsymbol{K}$ is a $P \times M$ matrix of constants, called the *direct connection matrix*. The element $k_{i,j}$ of the matrix represents the multiplication factor of the

connection from the jth external input to the ith external output. If there is no such connection, then $k_{i,j} = 0$.

5. Substitution of (11.56) in (11.55) gives

$$\boldsymbol{V}^z(z) = \boldsymbol{R}^z(z)\boldsymbol{F}\boldsymbol{V}^z(z) + \boldsymbol{R}^z(z)\boldsymbol{G}\boldsymbol{X}^z(z), \tag{11.58}$$

so,

$$[\boldsymbol{I}_L - \boldsymbol{R}^z(z)\boldsymbol{F}]\boldsymbol{V}^z(z) = \boldsymbol{R}^z(z)\boldsymbol{G}\boldsymbol{X}^z(z), \tag{11.59}$$

hence

$$\boldsymbol{V}^z(z) = [\boldsymbol{I}_L - \boldsymbol{R}^z(z)\boldsymbol{F}]^{-1}\boldsymbol{R}^z(z)\boldsymbol{G}\boldsymbol{X}^z(z). \tag{11.60}$$

6. Finally, substitution of (11.60) in (11.57) gives

$$\boldsymbol{Y}^z(z) = \{\boldsymbol{H}[\boldsymbol{I}_L - \boldsymbol{R}^z(z)\boldsymbol{F}]^{-1}\boldsymbol{R}^z(z)\boldsymbol{G} + \boldsymbol{K}\}\boldsymbol{X}^z(z). \tag{11.61}$$

The matrix

$$\boldsymbol{T}^z(z) = \boldsymbol{H}[\boldsymbol{I}_L - \boldsymbol{R}^z(z)\boldsymbol{F}]^{-1}\boldsymbol{R}^z(z)\boldsymbol{G} + \boldsymbol{K} \tag{11.62}$$

is the aggregate of transfer functions from the external inputs to the external outputs; that is, its element $T^z_{i,j}(z)$ relates the jth input to the ith output via

$$Y^z_i(z) = T^z_{i,j}(z)X^z_j(z), \tag{11.63}$$

provided all inputs except for the jth one are set to zero. This matrix is well defined if and only if the determinant of the matrix $[\boldsymbol{I}_L - \boldsymbol{R}^z(z)\boldsymbol{F}]$ is nonzero. If this determinant is zero (by this we mean *identically zero*, rather than just for particular values of z), the network is said to be *singular* and its response is undefined.

Example 11.4 Consider the network shown in Figure 11.13. The matrices of this network are

$$\boldsymbol{R}^z(z) = \begin{bmatrix} R^z_1(z) & 0 & 0 \\ 0 & R^z_2(z) & 0 \\ 0 & 0 & R^z_3(z) \end{bmatrix}, \quad \boldsymbol{F} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \quad \boldsymbol{G} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \tag{11.64a}$$

$$\boldsymbol{H} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{K} = 0. \tag{11.64b}$$

Therefore,

$$\boldsymbol{I}_3 - \boldsymbol{R}^z(z)\boldsymbol{F} = \begin{bmatrix} 1 & 0 & R^z_1(z) \\ 0 & 1 & 0 \\ -R^z_3(z) & -R^z_3(z) & 1 \end{bmatrix}, \tag{11.65}$$

$$[\boldsymbol{I}_3 - \boldsymbol{R}^z(z)\boldsymbol{F}]^{-1} = \frac{1}{1 + R^z_1(z)R^z_3(z)} \begin{bmatrix} 1 & -R^z_1(z)R^z_3(z) & -R^z_1(z) \\ 0 & 1 + R^z_1(z)R^z_3(z) & 0 \\ R^z_3(z) & R^z_3(z) & 1 \end{bmatrix}, \tag{11.66}$$

and finally from (11.62),

$$T^z(z) = \frac{Y^z(z)}{X^z(z)} = \frac{[R^z_1(z) + R^z_2(z)]R^z_3(z)}{1 + R^z_1(z)R^z_3(z)}. \tag{11.67}$$

□

As we have seen, formula (11.62) is a convenient means of computing the input-output relationships of LTI networks. However, when the network has a large number of blocks with many interconnections, this formula becomes difficult to manipulate by

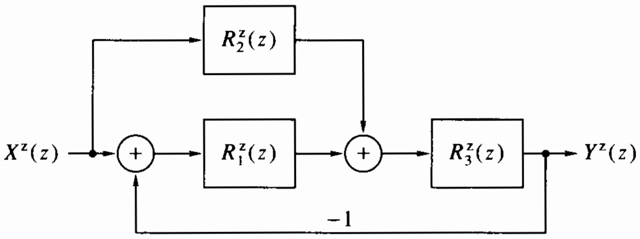

Figure 11.13 The network in Example 11.4.

hand. A computer program for computing this formula can be written, but this is not an easy task either, since the formula involves functions of z. On the other hand, state-space representations enable us to compute the matrix of transfer functions $T^z(z)$ without using (11.62) explicitly. This is done as follows.

1. We first represent each of the blocks $R_l^z(z)$ by state-space equations, say

$$s_l[n+1] = A_l s_l[n] + B_l u_l[n], \qquad (11.68a)$$

$$v_l[n] = C_l s_l[n] + D_l u_l[n], \qquad (11.68b)$$

so

$$R_l^z(z) = C_l(zI_{N_l} - A_l)^{-1}B_l + D_l. \qquad (11.69)$$

Here N_l denotes the dimension of the state-space representation of the lth block. Any of the realizations we have studied can be used for this purpose.

2. Collecting (11.68) for all $1 \le l \le L$, we can write

$$s[n+1] = As[n] + Bu[n], \qquad (11.70a)$$

$$v[n] = Cs[n] + Du[n]. \qquad (11.70b)$$

Each of the matrices A, B, C, D is block diagonal, with blocks A_l, B_l, C_l, D_l on the respective main diagonals. The vector $s[n]$ is the state vector of the complete network at time n, consisting of the components of the state vectors of all blocks. The dimension of $s[n]$ is $\sum_{l=1}^{L} N_l$. The vectors $u[n]$, $v[n]$ are, respectively, the block inputs and outputs at time n.

3. Recall (11.56) and write it in the time domain as

$$u[n] = Fv[n] + Gx[n]. \qquad (11.71)$$

Then substitute (11.71) in (11.70b) to get

$$u[n] = FCs[n] + FDu[n] + Gx[n], \qquad (11.72)$$

so

$$u[n] = EFCs[n] + EGx[n], \quad \text{where} \quad E = (I_L - FD)^{-1}. \qquad (11.73)$$

This relationship holds provided $I_L - FD$ is nonsingular. If this matrix is singular then the network is singular and its response is not defined.

4. Now substitute (11.73) in (11.70) to get

$$s[n+1] = (A + BEFC)s[n] + BEGx[n], \qquad (11.74)$$

$$v[n] = (I_L + DEF)Cs[n] + DEGx[n]. \qquad (11.75)$$

5. Finally substitute (11.75) in the time-domain form of (11.57) to get

$$y[n] = H(I_L + DEF)Cs[n] + (K + HDEG)x[n]. \qquad (11.76)$$

Equations (11.74) and (11.76) provide a state-space representation of the network.

The procedure `network` in Program 11.8 implements the construction in MATLAB. It accepts the four connection matrices $\boldsymbol{F}, \boldsymbol{G}, \boldsymbol{H}, \boldsymbol{K}$ and the coefficients of the transfer functions $R_l^z(z)$ as inputs. Its outputs are the state-space matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$.

11.4 The Finite Word Length Problem*

In discussing filter realizations, we have so far assumed that all variables can be represented exactly in the computer, and all arithmetic operations can be performed to an infinite precision. In practice, numbers can be represented only to a finite precision, and arithmetic operations are subject to errors, since a computer word has only a finite number of bits. The operation of representing a number to a fixed precision (that is, by a fixed number of bits) is called *quantization*. Consider, for example, the digital filter

$$y[n] = -a_1 y[n-1] + b_0 x[n] + b_1 x[n-1].$$

In implementing this filter, we must deal with the following problems:

1. The input signal $x[n]$ may have been obtained by converting a continuous-time signal $x(t)$. As we saw in Section 3.5, A/D conversion gives rise to quantization errors, determined by the number of bits of the A/D.
2. The constant coefficients a_1, b_0, b_1 cannot be represented exactly, in general; the error in each of these coefficients can be up to the least significant bit of the computer word. Because of these errors, the digital filter we implement differs from the desired one: Its poles and zeros are not in the desired locations, and its frequency response is different from the desired one.
3. When we form the product $a_1 y[n-1]$, the number of bits in the result is the sum of the numbers of bits in a_1 and $y[n-1]$. It is unreasonable to keep increasing the number of bits of $y[n]$ as n increases. We must assign a fixed number of bits to $y[n]$, and quantize $a_1 y[n-1]$ to this number. Such quantization leads to an error each time we update $y[n]$ (i.e., at every time point).
4. If $x[n]$ and $y[n]$ are represented by the same number of bits, we must quantize the products $b_0 x[n]$, $b_1 x[n-1]$ every time they are computed. It is possible to avoid this error if we assign to $y[n]$ a number of bits equal to or greater than that of these products. In practice, this usually means representation of $y[n]$ in double precision.
5. The range of values of $y[n]$ that can be represented in fixed point is limited by the word length. Large input values can cause $y[n]$ to *overflow*, that is, to exceed its full scale. To avoid overflow, it may be necessary to scale down the input signal. Scaling always involves trade-off: On one hand we want to use as many significant bits as possible, but on the other hand we want to eliminate or minimize the possibility of overflow.
6. If the output signal $y[n]$ is to be fed to a D/A converter, it sometimes needs to be further quantized, to match the number of bits in the D/A. Such quantization is another source of error.

In the remaining sections of this chapter we shall study these problems, analyze their effects, and learn how they can be solved or at least mitigated.

11.5 Coefficient Quantization in Digital Filters*

When a digital filter is designed using high-level software, the coefficients of the designed filter are computed to high accuracy. MATLAB, for example, gives the coefficients to 15 decimal digits. With such accuracy, the specifications are usually met exactly (or even exceeded in some bands, because the order of the filter is typically rounded upward). When the filter is to be implemented, there is usually a need to quantize the coefficients to the word length used for the implementation (whether in software or in hardware). Coefficient quantization changes the transfer function and, consequently, the frequency response of the filter. As a result, the implemented filter may fail to meet the specifications. This was a difficult problem in the past, when computers had relatively short word lengths. Today (mid-1990s), many microprocessors designed for DSP applications have word lengths from 16 bits (about 4 decimal digits) and up to 24 bits in some (about 7 decimal digits). In the future, even longer words are likely to be in use, and floating-point arithmetic may become commonplace in DSP applications. However, there are still many cases in which finite word length is a problem to be dealt with, whether because of hardware limitations, tight specifications of the filter in question, or both. We therefore devote this section to the study of coefficient quantization effects. We first consider the effect of quantization on the poles and the zeros of the filter, and then its effect on the frequency response.

11.5.1 Quantization Effects on Poles and Zeros

Coefficient quantization causes a replacement of the exact parameters $\{a_k, b_k\}$ of the transfer function by corresponding approximate values $\{\hat{a}_k, \hat{b}_k\}$. The difference between the exact and approximate values of each parameter can be up to the least significant bit (LSB) of the computer, multiplied by the full-scale value of the parameter. For example, consider a second-order IIR filter

$$H^z(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}.$$

Since we are dealing only with stable filters, we know that necessarily $|a_1| < 2$ and $|a_2| < 1$ (cf. Figure 7.3). Suppose we represent each coefficient by B bits, including sign. Then, assuming we scale a_1 so that the largest representable number is ± 2, the LSB for a_1 will be $2^{-(B-2)}$, and the error $|\hat{a}_1 - a_1|$ can be up to $2^{-(B-1)}$. Similarly, assuming we scale a_2 so that the largest representable number is ± 1, the LSB of a_2 will be $2^{-(B-1)}$, and the error $|\hat{a}_2 - a_2|$ can be up to 2^{-B}.

Replacement of the exact parameters by the quantized values causes the poles and zeros of the filter to shift from their desired locations, and the frequency response to deviate from its desired shape. Usually, we are not as much interested in the deviation of the poles and zeros as in the deviation of the frequency response. However, it is instructive to explore the former first, since this will lead to important qualitative conclusions. Consider, for example, the poles of a second-order filter, given by

$$\hat{\alpha}_{1,2} = -0.5\hat{a}_1 \pm j(\hat{a}_2 - 0.25\hat{a}_1^2)^{1/2}.$$

Since $\{\hat{a}_1, \hat{a}_2\}$ can assume only a finite number of values each (equal to 2^B), the poles $\hat{\alpha}_{1,2}$ can assume only a finite number of values. Of particular interest are the possible locations of complex stable poles. These are depicted by the dots in Figure 11.14, in the case $B = 5$.

As we see from Figure 11.14, the permissible locations of complex poles of the quantized filter are not distributed uniformly inside the unit circle. In particular, small

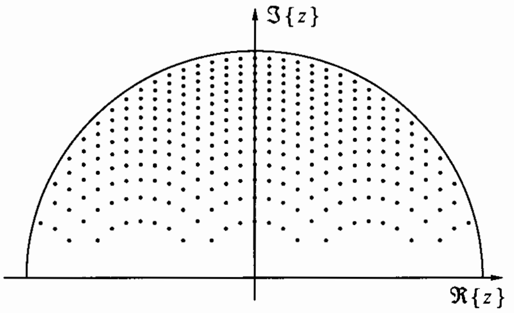

Figure 11.14 Possible locations of complex stable poles of a second-order digital filter in direct realization; number of bits: $B = 5$.

values of the imaginary part are virtually excluded. The low density of permissible pole locations in the vicinity of $z = 1$ and $z = -1$ is especially troublesome. Narrow-band, low-pass filters must have complex poles in the neighborhood of $z = 1$, whereas narrow-band, high-pass filters must have complex poles in the neighborhood of $z = -1$. We therefore conclude that high coefficient accuracy is needed to accurately place the poles of such filters.[4] Another conclusion is that high sampling rates are undesirable from the point of view of sensitivity of the filter to coefficient quantization. A high sampling rate means that the frequency response in the θ domain is pushed toward the low-frequency band; correspondingly, the poles are pushed toward $z = 1$ and, as we have seen, this increases the word length necessary for accurate representation of the coefficients.

The coupled realization of a second-order section, introduced in Section 11.1.7, can be better appreciated now. Recall that this realization is parameterized in terms of $\{\alpha_r, \alpha_i\}$, the real and imaginary parts of the complex pole. Therefore, if each of these two parameters is quantized to 2^B levels in the range $(-1, 1)$, the permissible pole locations will be distributed uniformly in the unit circle. This is illustrated in Figure 11.15 for $B = 5$. As we see, the density of permissible pole locations near $z = \pm 1$ is higher in this case.

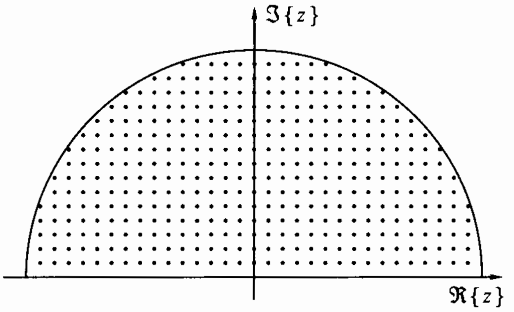

Figure 11.15 Possible locations of complex stable poles of a second-order digital filter in coupled form; number of bits: $B = 5$.

The effect of quantization of the numerator coefficients may seem, at first glance, to be the same as that of the denominator coefficients. This, however, is not the case. As we saw in Chapter 10, the zeros of an analog filter of the four common classes (Butterworth, Chebyshev-I, Chebyshev-II, elliptic) are always on the imaginary axis. Consequently, the zeros of a digital IIR filter obtained from an analog filter by a bilinear transform are always on the unit circle. Thus, the numerator of a second-order IIR filter has either real zeros at $z = \pm 1$ or a conjugate complex pair at $z = e^{\pm j\zeta}$. In the former case, the coefficients are trivial (± 1 or ± 2), so the problem of coefficient quantization does not arise. In the latter case, the numerator has the form $(1 - 2\cos\zeta z^{-1} + z^{-2})$ (up to a constant gain), so only the coefficient $2\cos\zeta$ is subject to quantization. When this coefficient undergoes a small perturbation, the zeros will shift but will stay on the unit circle. The effect of such a shift on the behavior of the filter is usually minor.

The sensitivity of the poles of a digital IIR filter to denominator coefficient quantization usually increases with the filter's order. Pole maps such as the one in Figure 11.14 are difficult to generate for orders higher than 2. An alternative way to analyze the sensitivity is to express the perturbation in a specific pole α_m as an approximate linear combination of the perturbations $\{\hat{a}_k - a_k\}$ of the coefficients, assuming that these perturbations are small and using the chain rule for partial differentiation. This analysis is carried out in Problem 11.17. The resulting formula is

$$\hat{\alpha}_m - \alpha_m \approx -\sum_{k=1}^{p} \frac{\alpha_m^{p-k}(\hat{a}_k - a_k)}{\prod_{l \neq m}(\alpha_m - \alpha_l)}. \tag{11.77}$$

As we see, poles α_l, α_m that are close to each other lead to small values of $\alpha_l - \alpha_m$, so they decrease the denominator of (11.77) and increase the error $\hat{\alpha}_m - \alpha_m$. Narrow-band filters, in particular, have their poles tightly spaced, so we can expect high sensitivity of the poles to coefficient quantization in such filters. We can already see, at least qualitatively, why parallel and cascade realizations are less sensitive to coefficient quantization than direct realizations. In parallel and cascade realizations, the second-order sections are isolated from each other, so formula (11.77) holds for each section separately. For a single section, only the distance between the two conjugate poles affects the error. Unless the conjugate poles are close to $z = \pm 1$, the sensitivity is not high. This qualitative behavior will be verified below by different means.

11.5.2 Quantization Effects on the Frequency Response

Next we examine the sensitivity of the magnitude of the filter's frequency response to coefficient quantization. We assume that $H^{\mathrm{f}}(\theta)$ depends on a set of real parameters $\{p_k,\ 1 \le k \le K\}$. In case of a direct realization, the parameters are $\{a_i,\ 1 \le i \le N\}$ and $\{b_i,\ 0 \le i \le N\}$; in case of a parallel realization, they are c_0 and $\{e_i, f_i,\ 1 \le i \le N\}$; in case of a cascade realization, they are b_0 and $\{g_i, h_i,\ 1 \le i \le N\}$. For all IIR filter realizations, $K = 2N + 1$. In case of a direct realization of a linear phase FIR filter, the parameters are $\{h[n],\ 0 \le n \le \lfloor 0.5N \rfloor\}$.

Let us assume that the perturbations in the parameters resulting from quantization are small, and approximate the perturbation in the magnitude response $|H^{\mathrm{f}}(\theta)|$ by a first-order Taylor series:

$$|\hat{H}^{\mathrm{f}}(\theta)| - |H^{\mathrm{f}}(\theta)| \approx \sum_{k=1}^{K} \frac{\partial |H^{\mathrm{f}}(\theta)|}{\partial p_k}(\hat{p}_k - p_k). \tag{11.78}$$

The partial derivatives on the right side of (11.78) are given by[5]

$$\frac{\partial |H^{\mathrm{f}}(\theta)|}{\partial p_k} = \Re\left\{\frac{\partial H^{\mathrm{f}}(\theta)}{\partial p_k} e^{-j\psi(\theta)}\right\}, \tag{11.79}$$

where $\psi(\theta)$ is the phase response of the filter. We note that, since

$$a(e^{j\theta}) = \sum_{k=0}^{p} a_k e^{-j\theta k}, \quad b(e^{j\theta}) = \sum_{k=0}^{q} b_k e^{-j\theta k}, \tag{11.80}$$

we have

$$\frac{\partial a(e^{j\theta})}{\partial a_k} = e^{-j\theta k}, \quad \frac{\partial a(e^{j\theta})}{\partial b_k} = 0, \tag{11.81a}$$

$$\frac{\partial b(e^{j\theta})}{\partial b_k} = e^{-j\theta k}, \quad \frac{\partial b(e^{j\theta})}{\partial a_k} = 0. \tag{11.81b}$$

The partial derivatives $\partial H^{\mathrm{f}}(\theta)/\partial p_k$ are computed as follows for the realizations we have studied:

1. For a direct realization we have

$$H^{\mathrm{f}}(\theta) = \frac{b(e^{j\theta})}{a(e^{j\theta})}, \tag{11.82}$$

therefore,

$$\frac{\partial H^{\mathrm{f}}(\theta)}{\partial b_k} = \frac{e^{-j\theta k}}{a(e^{j\theta})}, \quad \frac{\partial H^{\mathrm{f}}(\theta)}{\partial a_k} = -\frac{e^{-j\theta k} b(e^{j\theta})}{a^2(e^{j\theta})}. \tag{11.83}$$

2. The frequency response of a parallel realization has the form

$$H^{\mathrm{f}}(\theta) = c_0 + \sum_{l=1}^{L} H_l^{\mathrm{f}}(\theta) = c_0 + \sum_{l=1}^{L} \frac{b_l(e^{j\theta})}{a_l(e^{j\theta})}, \tag{11.84}$$

where the $H_l^{\mathrm{f}}(\theta)$ are of first or second order. Therefore,

$$\frac{\partial H^{\mathrm{f}}(\theta)}{\partial c_0} = 1, \tag{11.85}$$

and, if p_k is a parameter appearing in a certain $H_l^{\mathrm{f}}(\theta)$, then

$$\frac{\partial H^{\mathrm{f}}(\theta)}{\partial p_k} = \frac{\partial H_l^{\mathrm{f}}(\theta)}{\partial p_k}. \tag{11.86}$$

The right side of (11.86) can be computed as in (11.83).

3. The frequency response of a cascade realization has the form

$$H^{\mathrm{f}}(\theta) = b_0 \prod_{l=1}^{L} H_l^{\mathrm{f}}(\theta) = b_0 \prod_{l=1}^{L} \frac{b_l(e^{j\theta})}{a_l(e^{j\theta})}, \tag{11.87}$$

where the $H_l^{\mathrm{f}}(\theta)$ are of first or second order. Therefore,

$$\frac{\partial H^{\mathrm{f}}(\theta)}{\partial b_0} = \prod_{l=1}^{L} H_l^{\mathrm{f}}(\theta), \tag{11.88}$$

and, if p_k is a parameter appearing in a certain $H_l^{\mathrm{f}}(\theta)$, then

$$\frac{\partial H^{\mathrm{f}}(\theta)}{\partial p_k} = b_0 \frac{\partial H_l^{\mathrm{f}}(\theta)}{\partial p_k} \prod_{\substack{m=1 \\ m \neq l}}^{L} H_m^{\mathrm{f}}(\theta). \tag{11.89}$$

The partial derivatives on the right side of (11.89) can be computed as in (11.83).

4. For a linear phase FIR filter in direct realization we have

$$\frac{\partial H^{\mathrm{f}}(\theta)}{\partial h[n]} = \begin{cases} e^{-j\theta n} \pm e^{-j\theta(N-n)}, & n \neq 0.5N, \\ e^{-j0.5\theta N}, & n = 0.5N. \end{cases} \tag{11.90}$$

where the positive sign is for types I and II, and the negative sign is for types III and IV.

The approximate error formula (11.78) can be used for estimating the effect of coefficient quantization on the magnitude response in the following manner. Let P_k denote the full scale used for the coefficient p_k. P_k must be at least as large as $|p_k|$, and is often larger, for reasons explained below. Let q be half the quantization level relative to full scale, so $q = 2^{-B}$ (where B is the word length in bits). Then we have

$$|\hat{p}_k - p_k| \le P_k q. \tag{11.91}$$

Substitution in (11.78) leads to the approximate inequality

$$\big||\hat{H}^{\mathrm{f}}(\theta)| - |H^{\mathrm{f}}(\theta)|\big| \le q \sum_{k=1}^{K} \left| \frac{\partial |H^{\mathrm{f}}(\theta)|}{\partial p_k} \right| P_k. \tag{11.92}$$

The function

$$S(\theta) = \sum_{k=1}^{K} \left| \frac{\partial |H^{\mathrm{f}}(\theta)|}{\partial p_k} \right| P_k, \tag{11.93}$$

is the ***sensitivity bound*** of the filter. As we see, the sensitivity bound depends on the realization. The realization determines the coefficients p_k, their full-scale values, and the partial derivatives of the magnitude response with respect to the coefficients. In summary, the error in the magnitude response at any given frequency is approximately bounded by

$$\big||\hat{H}^{\mathrm{f}}(\theta)| - |H^{\mathrm{f}}(\theta)|\big| \le 2^{-B} S(\theta). \tag{11.94}$$

A common way of choosing the full-scale values P_k is as follows. For each of the numerator and denominator polynomials of a direct realization, we find the coefficient having the largest magnitude and use it as the full-scale value for *all* the coefficients of the polynomial in question. The corresponding formulas are

$$P_{\mathrm{num}} = \max\{\mathrm{abs}(b_i),\ 0 \le i \le N\}, \tag{11.95a}$$

$$P_{\mathrm{den}} = \max\{\mathrm{abs}(a_i),\ 1 \le i \le N\}. \tag{11.95b}$$

This way, the coefficient of largest magnitude in each polynomial is represented by ±1. For parallel and cascade realizations, we scale each section separately in the same way.

Another way of choosing the full-scale values is to round the largest magnitude coefficient of each polynomial upward to the nearest integer power of 2 and take the rounded value as the full scale of all the coefficients of the polynomial. The corresponding formulas are

$$P_{\mathrm{num}} = 2^{\lceil \log_2(\max\{\mathrm{abs}(b_i),\ 0 \le i \le N\}) \rceil}, \tag{11.96a}$$

$$P_{\mathrm{den}} = 2^{\lceil \log_2(\max\{\mathrm{abs}(a_i),\ 1 \le i \le N\}) \rceil}. \tag{11.96b}$$

This method yields larger P_{num} and P_{den} than the ones in (11.95), but is more convenient, since the scaled coefficients are related to the true ones by simple shifts.

The sensitivity bound is useful for determining a suitable word length for implementing a given filter in a given realization. Let δ_{p} be the minimum tolerance of all pass bands of the filter, and δ_{s} the minimum tolerance of all stop bands. It is reasonable to require that the maximum deviation from the tolerance caused by quantization be no more than 10 percent of the tolerance itself (a different percentage can be chosen, depending on the application). With this requirement, (11.94) leads to

$$B \ge \log_2 \left(\frac{\max\{S(\theta) : \theta \text{ in a pass band}\}}{0.1\delta_{\mathrm{p}}} \right), \tag{11.97a}$$

$$B \ge \log_2 \left(\frac{\max\{S(\theta) : \theta \text{ in a stop band}\}}{0.1\delta_{\mathrm{s}}} \right). \tag{11.97b}$$

Therefore, choosing B according to the greater of the two right sides in (11.97) guarantees that the error in the magnitude response resulting from quantization will deviate only slightly (if at all) from the permitted bounds in all frequency bands.

Programs 11.9 through 11.14 compute the sensitivity bounds for the standard realizations. The procedure `sensiir` is for IIR filters in direct, parallel, and cascade realizations, whereas `sensfir` is for linear-phase FIR filters in a direct realization. Each of the two procedures returns the individual sensitivities $\partial|H^{\mathrm{f}}(\theta)|/\partial p_k$ in the matrix `dHmag`, and the sensitivity bound $S(\theta)$ in the vector `S`. These are computed at `K` frequency points, in the interval specified by the input variable `theta`. The procedures `dhdirect`, `dhparal`, `dhcascad` return the partial derivatives $\partial H^{\mathrm{f}}(\theta)/\partial p_k$ of the respective IIR realizations, and the full-scale values P_k of the coefficients. The full-scale values are computed by `scale2` according to (11.96).

Example 11.5 Consider the fifth-order, low-pass elliptic filter designed in Example 10.14. We compute the sensitivity bounds of this filter for the three standard realizations. Figure 11.16 shows the results. Parts a and b show the bounds of the pass band and the stop band, respectively, for the direct realization. The maximum sensitivity in the pass band is about 9×10^4, and that in the stop band is about 1100. Recall that the tolerances of this filter are $\delta_{\mathrm{p}} \approx 0.011$, $\delta_{\mathrm{s}} = 0.05$. The criterion (11.97) gives

$$\text{Pass-band requirement:} \quad B \geq \log_2\left(\frac{9 \cdot 10^4}{0.0011}\right) \approx 26.3,$$

$$\text{Stop-band requirement:} \quad B \geq \log_2\left(\frac{1100}{0.005}\right) \approx 17.7.$$

Therefore, 27 bits are sufficient for a direct realization of the filter.

Parts c and d of Figure 11.16 show the bounds of the pass band and the stop band, respectively, for the parallel realization. The maximum sensitivity at the pass band is about 15.5, and that at the stop band is about 14. The criterion (11.97) gives

$$\text{Pass-band requirement:} \quad B \geq \log_2\left(\frac{15.5}{0.0011}\right) \approx 13.8,$$

$$\text{Stop-band requirement:} \quad B \geq \log_2\left(\frac{14}{0.005}\right) \approx 11.5.$$

Therefore, 14 bits are sufficient for a parallel realization of the filter.

Parts e and f of Figure 11.16 show the bounds of the pass band and the stop band, respectively, for the cascade realization. The maximum sensitivity at the pass band is about 57, and that at the stop band is about 1.3. The criterion (11.97) gives

$$\text{Pass-band requirement:} \quad B \geq \log_2\left(\frac{57}{0.0011}\right) \approx 15.7,$$

$$\text{Stop-band requirement:} \quad B \geq \log_2\left(\frac{1.3}{0.005}\right) \approx 8.0.$$

Therefore, 16 bits are sufficient for a cascade realization of the filter.

In summary, the parallel realization is the least sensitive to coefficient quantization in this example. At present (mid-1990s), 16-bit fixed-point arithmetic, which is the most common in commercial DSP microprocessors, is suitable for both parallel and cascade realizations but is far from being sufficient for a direct realization. Even 24-bit DSP microprocessors may be insufficient for direct realizations in certain cases. □

The phenomenon seen in Example 11.5 is typical. Parallel realizations are usually best in the pass band (*best* meaning least sensitive to coefficient quantization), whereas

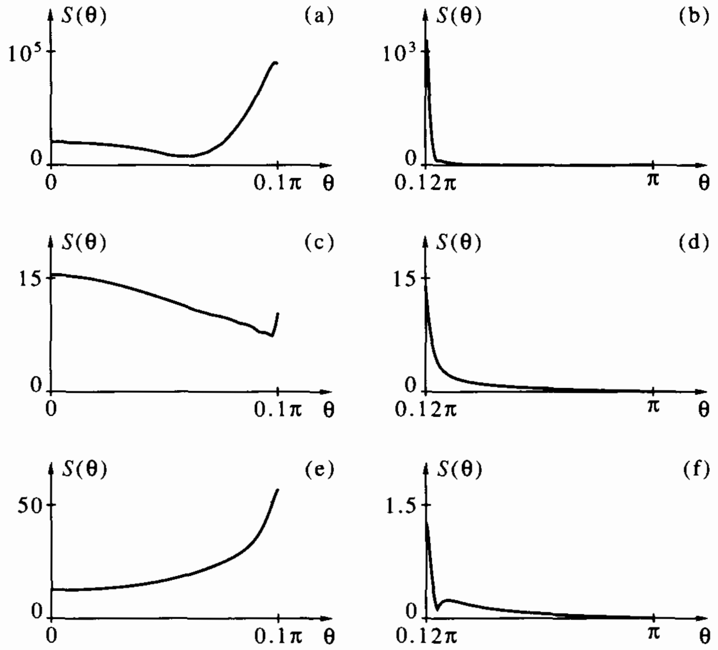

Figure 11.16 Sensitivity bounds for the filter in Example 11.5: (a) direct realization, pass band; (b) direct realization, stop band; (c) parallel realization, pass band; (d) parallel realization, stop band; (e) cascade realization, pass band; (f) cascade realization, stop band (graphs are drawn to different scales).

cascade realizations are usually best in the stop band. It is common for a practical filter to have δ_s much smaller than δ_p (the filter in Example 11.5 is an exception). Therefore, the stop-band requirement (11.97b) usually wins over the pass-band requirement (11.97a). Consequently, cascade realizations are the most commonly used. Direct realizations of IIR filters are highly sensitive to coefficient quantization and should be avoided as a rule, unless the order of the filter is low (3 at most).

Example 11.6 Consider the equiripple FIR filter of order $N = 146$, designed in Example 9.16. This filter has the same frequency bands and tolerances as the IIR filter in Example 11.5. Figure 11.17 shows the sensitivity bounds of the pass band and the stop band, respectively, for a direct realization of the filter. The maximum sensitivity is about 19 at both pass and stop bands. The tolerances of this filter are $\delta_p \approx 0.011$, $\delta_s = 0.05$. The criterion (11.97) gives

$$\text{Pass-band requirement:} \quad B \geq \log_2\left(\frac{19}{0.0011}\right) \approx 14.1,$$

$$\text{Stop-band requirement:} \quad B \geq \log_2\left(\frac{19}{0.005}\right) \approx 11.9.$$

Therefore, 15 bits are sufficient for a direct realization of this filter. □

The phenomenon seen in Example 11.6 is typical. FIR filters, even in a direct realization, are generally much less sensitive to coefficient quantization than IIR filters meeting the same specifications. Word length of 16 bits often is sufficient. Because of this property, cascade realization is seldom used for FIR filters. We reiterate that parallel realization is not applicable at all to FIR filters.

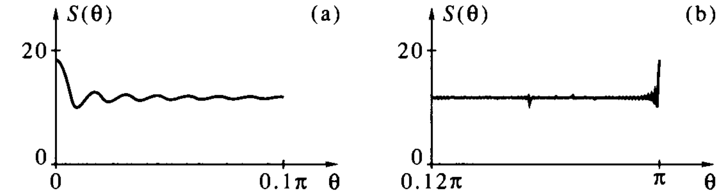

Figure 11.17 Sensitivity bounds for the filter in Example 11.6: (a) pass band; (b) stop band.

The sensitivity analysis we have presented is useful for preliminary analysis of errors due to coefficient quantization in a filter's magnitude of frequency response, and for choosing scaling and word length to represent the coefficients. After the coefficients have been quantized to the selected word length, it is good practice to compute the frequency response of the actual filter, and verify that it is acceptable. The MATLAB procedure qfrqresp, listed in Program 11.15, performs this computation. The program accepts the numerator and denominator coefficients of the filter, the desired realization (direct, parallel, or cascade), the number of bits, and the desired frequency points (number and range). For an FIR filter, the coefficients are entered in the numerator polynomial, and 1 is entered for the denominator polynomial. The program finds the coefficients of the desired realization and their scaling. It then quantizes the coefficients, and finally computes the frequency response by calling frqresp as needed.

Example 11.7 We test the filters discussed in Examples 11.5 and 11.6 with the word length found there for each case. Figure 11.18 shows the results. Only the pass-band response is shown in each case, since the stop-band response is much less sensitive to coefficient quantization for these filters. As we see, the predicted word length is indeed suitable in all cases, except for slight deviation of the response of the parallel realization at the band edge. The obvious remedy in this case is to use a word length of 16 bits, which is more practical than 14 bits anyway. We emphasize again that the response of the FIR filter, although this filter is implemented in only 15 bits, is well within the tolerance. □

11.6 Scaling in Fixed-Point Arithmetic*

When implementing a digital filter in fixed-point arithmetic, it is necessary to scale the input and output signals, as well as certain inner signals, to avoid signal values that exceed the maximum representable number. A problem of a similar nature arises in active analog filters: There, it is required to limit the signals to voltages below the saturation levels of the operational amplifiers. However, there is an important difference between the analog and the digital cases: When an analog signal exceeds its permitted value, its magnitude is limited but its polarity is preserved. When a digital signal exceeds its value, we call it an *overflow*. An overflow in two's-complement arithmetic leads to polarity reversal: A number slightly larger than 1 changes to a number slightly larger than -1. Therefore, overflows in digital filters are potentially more harmful than in analog filters, and care is necessary to prohibit them, or to treat them properly when they occur.

The scaling problem can be stated in mathematical terms as follows. Suppose we wish to prevent the magnitude of the output signal $|y[n]|$ from exceeding a certain

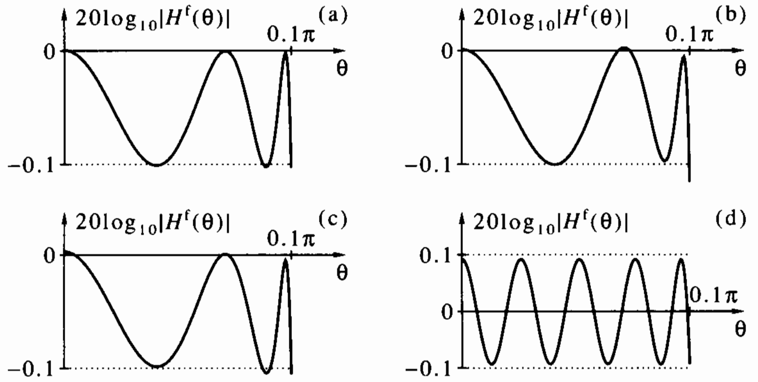

Figure 11.18 Pass-band magnitude responses of the filters in Examples 11.5, 11.6 after coefficient quantization: (a) IIR filter, direct realization, 27 bits; (b) IIR filter, parallel realization, 14 bits; (c) IIR filter, cascade realization, 16 bits; (d) FIR filter, direct realization, 15 bits.

upper limit $y_{\max}$. Since the input and output signals are related through a convolution $y[n] = \{x * h\}[n]$, we may be able to achieve this goal by properly limiting the input signal $x[n]$. This, however, may be undesirable because limiting is a nonlinear operation that distorts the signal. A more attractive solution is to scale the impulse response by a proportionality factor c, chosen in a way that will prevent the magnitude of $c\{x * h\}[n]$ from exceeding the maximum value $y_{\max}$. Such a proportionality factor may also be useful in the opposite case: If the dynamic range of $x[n]$ is such that $y[n]$ is much smaller in magnitude than $y_{\max}$, we can adjust c to increase the range of $y[n]$ as needed. In either case, the transfer function $cH^z(z)$ is called a *scaled transfer function.* Our task is to find a scale factor c that will cause $y[n]$ to have as large dynamic range as possible without exceeding $y_{\max}$ in magnitude. Judicious choice of a scale factor requires knowledge of certain parameters of the input signal; accordingly, there exist several scaling methods, differing in their assumptions on the nature and properties of the input signal.

The scaling problem is obviated if floating-point arithmetic is used. However, fixed-point implementations of digital filters are very common, so familiarity with scaling methods and the effects of scaling on the properties of the filter is expedient.

11.6.1 Time-Domain Scaling

Consider an LTI filter whose impulse response and transfer function are $h[n]$ and $H^z(z)$, respectively. Assume that the input signal $x[n]$ is known to be bounded in magnitude by $x_{\max}$, and the output signal is required to be bounded in magnitude by $y_{\max}$. Both values are relative to the maximum value representable by the computer, usually taken as 1. The output signal is related to the input by the convolution formula

$$y[n] = \sum_{m=0}^{\infty} h[m]x[n-m]. \tag{11.98}$$

Therefore,

$$|y[n]| \le \sum_{m=0}^{\infty} |h[m]| \cdot |x[n-m]| \le x_{\max} \sum_{m=0}^{\infty} |h[m]| = x_{\max}\|h\|_1, \tag{11.99}$$

where

$$\|h\|_1 = \sum_{m=0}^{\infty} |h[m]|. \tag{11.100}$$

Stability of the filter $h[n]$ implies that $\|h\|_1$ is finite. As we see, the requirement $|y[n]| \le y_{\max}$ will be achieved if we use a scale factor

$$c = \frac{y_{\max}}{x_{\max}\|h\|_1} \tag{11.101}$$

In practice, it may be convenient to round c downward to an integer power of 2, since this facilitates scaling by bit shifting.

Example 11.8 Suppose that an analog signal is sampled by a 12-bit A/D converter, then fed to a 16-bit filter whose transfer function is

$$H^z(z) = \frac{1}{1 - 0.98z^{-1}}.$$

If we place the sampled signal $x[n]$ in the lower bits of computer word (with sign extension), the corresponding $x_{\max}$ will be $1/16$. Suppose we wish to scale the output signal to $y_{\max} = 1$. We have

$$\|h\|_1 = \sum_{m=0}^{\infty} 0.98^m = \frac{1}{1 - 0.98} = 50.$$

Therefore, the maximum scale factor is $c = 16/50 = 0.32$ in this case. Rounding downward to a power of 2, we get a scale factor of 0.25. Such a scaling is tantamount to shifting $x[n]$ by 2 bits to the right, thus losing 2 out of the 12 available bits. The lesson from this example is that scaling using c as in (11.101) potentially involves loss of accuracy of the input signal. □

11.6.2 Frequency-Domain Scaling

The scale factor given in (11.101) is often overconservative in its use of the dynamic range of the computer word. By this we mean that the values assumed by the output signal are usually small in magnitude compared with the full scale. Less conservative scaling is provided by frequency-domain bounds, as follows.

1. 1-norm bound: We have, by the inverse Fourier transform formula,

$$y[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H^f(\theta)X^f(\theta)e^{j\theta n}d\theta. \tag{11.102}$$

   Therefore,

$$|y[n]| \le \frac{1}{2\pi}\int_{-\pi}^{\pi} |H^f(\theta)| \cdot |X^f(\theta)|d\theta. \tag{11.103}$$

   Define

$$\|X^f\|_\infty = \max_\theta |X^f(\theta)|, \quad \|H^f\|_1 = \frac{1}{2\pi}\int_{-\pi}^{\pi} |H^f(\theta)|d\theta. \tag{11.104}$$

   The quantity $\|X^f\|_\infty$ is called the *infinity norm* of $X^f(\theta)$, and the quantity $\|H^f\|_1$ is called the *1-norm* of $H^f(\theta)$. If $\|X^f\|_\infty$ is finite, we can guarantee that $|y[n]| \le y_{\max}$ by using the scale factor

$$c = \frac{y_{\max}}{\|H^f\|_1\|X^f\|_\infty}. \tag{11.105}$$

2. Infinity-norm bound: By reversing the roles of H^{f} and X^{f} in (11.105), we get the scale factor

$$c = \frac{y_{\max}}{\|H^{\mathrm{f}}\|_\infty \|X^{\mathrm{f}}\|_1}. \tag{11.106}$$

This scale factor is applicable in case $\|X^{\mathrm{f}}\|_1$, the 1-norm of the input signal, is finite. It is particularly useful when the input signal is narrow band. For example, if $x[n]$ is known to be sinusoidal, the scale factor (11.106) guarantees that the amplitude of the sinusoid at the output of the filter not exceed $y_{\max}$.

3. 2-norm bound: Unlike the preceding bounds, the bound we introduce now is probabilistic. Define

$$\|H^{\mathrm{f}}\|_2 = \left[\frac{1}{2\pi}\int_{-\pi}^{\pi} |H^{\mathrm{f}}(\theta)|^2 d\theta\right]^{1/2}. \tag{11.107}$$

Note that (11.107) is identical to the square root of the noise gain of the filter [cf. (2.137)]. Let us assume that the input $x[n]$ is a wide-sense stationary signal having power spectral density $K_x^{\mathrm{f}}(\theta)$. Then, using (2.134), we can bound the variance of the output signal as follows:

$$\gamma_y = \frac{1}{2\pi}\int_{-\pi}^{\pi} K_y^{\mathrm{f}}(\theta)d\theta = \frac{1}{2\pi}\int_{-\pi}^{\pi} K_x^{\mathrm{f}}(\theta)|H^{\mathrm{f}}(\theta)|^2 d\theta \le \|K_x^{\mathrm{f}}\|_\infty \|H^{\mathrm{f}}\|_2^2. \tag{11.108}$$

Therefore, if we know the maximum possible value of the power spectral density of the input signal, and we require that the variance of the output signal not exceed $\gamma_{y,\max}$, we get the scale factor

$$c = \left[\frac{\gamma_{y,\max}}{\|H^{\mathrm{f}}\|_2^2 \|K_x^{\mathrm{f}}\|_\infty}\right]^{1/2}. \tag{11.109}$$

Bounding the variance of $y[n]$ *does not* guarantee boundedness of $|y[n]|$, and this is why we refer to (11.109) as probabilistic scaling. If, for example, we take $\gamma_{y,\max} \approx 0.1y_{\max}^2$, the probability of $|y[n]|$ exceeding $y_{\max}$ is expected to be low.[6]

The four norms we have introduced are known to satisfy the relationships

$$\|H^{\mathrm{f}}\|_1 \le \|H^{\mathrm{f}}\|_2 \le \|H^{\mathrm{f}}\|_\infty \le \|h\|_1. \tag{11.110}$$

This, however, does not imply that $\|H^{\mathrm{f}}\|_1$ always provides the best (largest) scale factor, since the different scaling rules also depend on $x_{\max}$, $y_{\max}$, $\gamma_{y,\max}$, $\|X^{\mathrm{f}}\|_\infty$, $\|X^{\mathrm{f}}\|_1$, and $\|K_x^{\mathrm{f}}\|_\infty$. Therefore, judicious choice of scaling method requires knowledge about the nature of the input signal and its bounds.

11.6.3 MATLAB Implementation of Filter Norms

Exact computation of norms we have presented is not possible in general for rational filters (with the exception of $\|H^{\mathrm{f}}\|_2$). However, numerical approximations can be used instead. High accuracy is not required, since the scale factor resulting from those norms is usually rounded to an integer power of 2. The procedure `filnorm` in Program 11.16 implements the computation of the four filter norms. The time-domain norm $\|h\|_1$ is computed by calling the function `filter` with a unit-sample input, and adding terms $|h[n]|$ until the relative error drops below a prescribed threshold. The norm $\|H^{\mathrm{f}}\|_2$ is computed by calling `nsgain` (see Program 7.4) and taking the square root of the result. The norm $\|H^{\mathrm{f}}\|_\infty$ is approximated by the maximum of the magnitude response, computed on a dense grid. Finally, the norm $\|H^{\mathrm{f}}\|_1$ is computed by evaluating the magnitude response on a dense grid and approximating the integral by Simpson's rule.

11.6.4 Scaling of Inner Signals

Our discussion so far has concentrated on input scaling of a transfer function to avoid overflow of the output. In realizing a filter, inner (or intermediate) signals are always present, so we need to concern ourselves with potential overflow problems in such signals as well. There are four types of inner signal:

1. A signal resulting from delaying another signal. Such a signal can never overflow if the signal at the input of the delay does not.
2. A signal resulting from multiplying a signal by a constant. Assuming that both factors in the product are scaled below 1, the product is also less than 1 in magnitude, so it cannot overflow.
3. A signal resulting from adding two signals to form a partial sum, which is later used as an operand in another sum. We assume that such a signal is not used for any other purpose in the filter or outside it. Addition of two signals can potentially lead to overflow. However, two's-complement arithmetic has the following important property: If a sum of n numbers ($n > 2$) does not overflow, overflows in partial sums cancel out and do not affect the final result. For example, suppose that $x_1 + x_2 + x_3$ does not overflow, and the computation is performed in the order $(x_1 + x_2) + x_3$. Then, even if $x_1 + x_2$ overflows and the overflow is ignored, the final result will still be correct. The reason is that two's-complement is a special case of modular arithmetic [i.e., a number x is represented by $(x \bmod 2)$, assuming that the binary point is immediately to the right of the sign bit], hence standard rules of modular addition apply to it. The conclusion is that, if the filter is implemented in two's-complement arithmetic, overflows in partial sums can be ignored.
4. A signal resulting from adding two signals to form a final sum, which is needed elsewhere in the filter or outside it. Such a signal must be scaled similarly to the output signal, since its overflow is potentially harmful. This is done by computing the transfer function from the input to the signal in question and using one of the scaling methods described earlier. It is also recommended, in most applications, to detect overflows of such signals and replace the overflowed signal by the corresponding saturation value. For example, if it is detected that $x \geq 1$, x should be replaced by $1 - 2^{-(B-1)}$; if it is detected that $x < -1$, x should be replaced by -1. Most DSP microprocessors have a *saturation mode*, in which this operation is performed automatically after the addition.

Example 11.9 In the direct realization shown in Figure 11.4 there are 15 inner signals, in addition to the output signal. However, only the signals $u[n]$ and $y[n]$ are of the fourth type: The former because it is used more than once inside the filter, and the latter because it is used outside it. The transfer function from $x[n]$ to $u[n]$ is

$$\frac{U^z(z)}{X^z(z)} = \frac{1}{a(z)},$$

so one of the scaling methods explained previously should be used for it.

In the transposed direct realization shown in Figure 11.6, only the output signal $y[n]$ is of the fourth type. Any overflow generated at one of the adders along the center lane will propagate through the delays and will eventually reach $y[n]$. However, if the input is properly scaled to avoid overflow in $y[n]$, the entire realization will be insensitive to intermediate overflows. In this respect, the transposed direct realization offers an advantage over the direct realization. □

11.6.5 Scaling in Parallel and Cascade Realization

We now discuss in detail scaling procedures for parallel and cascade realizations for IIR filters. As we saw in Example 11.9, the transposed direct realization has an advantage over the direct realization, in that there is no need to scale inner signals. We therefore consider only parallel and cascade realizations in which the second-order sections are in a transposed direct realization.

Figure 11.19 illustrates a parallel connection of three second-order sections in skeletal form (delays are represented by factors z^{-1}, and summing junctions by open circles). The scaling procedure is as follows:

1. The input coefficients f_m at each section are scaled to prevent overflow of $v_k[n]$, the output signal of the section. As we have explained, this scaling depends on the assumptions made on the input signal, and on the norm used. The same scale factor should be applied to both coefficients of a specific section, but different factors are permitted at different sections. The f_m shown in Figure 11.19 are assumed to be scaled already.
2. The coefficients λ_k are computed to make the total gains of all sections (including the branch of constant gain c_0) equal to the corresponding gains in the unscaled transfer function, up to a common proportionality factor. For example, if f_1, f_2 where scaled down by 2 compared with f_3, f_4, then λ_1 must be larger than λ_2 by 2.
3. The dynamic range of $y[n]$ is checked using the scaled transfer function. If it is found that $y[n]$ can overflow, all λ_k must be decreased by a common factor, to prevent (or decrease the probability of) this overflow. If it is found that the dynamic range of $y[n]$ is small relative to full scale, all λ_k may be increased by a common factor.
4. In the preceding steps, all factors are usually taken as integer powers of 2. This way, factors greater than 1 can be implemented by left shifts, and factors smaller than 1 can be implemented by right shifts.

Example 11.10 Consider again the IIR filter discussed in Example 11.5. The parallel decomposition of the filter, with coefficients quantized to 16 bits, is

$$H^z(z) = -24984 \times 2^{-17} + \frac{8909 \times 2^{-20} - 22869 \times 2^{-20}z^{-1}}{1 - 30365 \times 2^{-14}z^{-1} + 15710 \times 2^{-14}z^{-2}}$$

$$+ \frac{-30199 \times 2^{-17} + 28813 \times 2^{-17}z^{-1}}{1 - 28072 \times 2^{-14}z^{-1} + 13038 \times 2^{-14}z^{-2}} + \frac{19504 \times 2^{-16}}{1 - 25279 \times 2^{-15}z^{-1}}.$$

Suppose we know that the input signal satisfies $\|K_x^{\mathrm{f}}\|_\infty = 0.1$. We decide to use 2-norm scaling such that the variance of the signal at the output of each section, as well as at the output of the complete filter, will not exceed 0.1. This implies that the 2-norm of each section must be no larger than 1 after scaling.

The 2-norms of the three unscaled sections are computed by the program `filnorm` and found to be 0.1528, 0.3805, and 0.4677. Therefore, the first section can be scaled by 4, the second by 2, and the third by 2. This is done by doubling the numerator coefficients of the second and the third sections, and quadrupling the numerator coefficients of the first.

The 2-norm of the complete filter is 0.3248, so the transfer function can be scaled by 2 to increase the dynamic range. We therefore complete the scaling procedure by taking

$$\lambda_0 = 2, \quad \lambda_1 = 0.5, \quad \lambda_2 = \lambda_3 = 1. \qquad \square$$

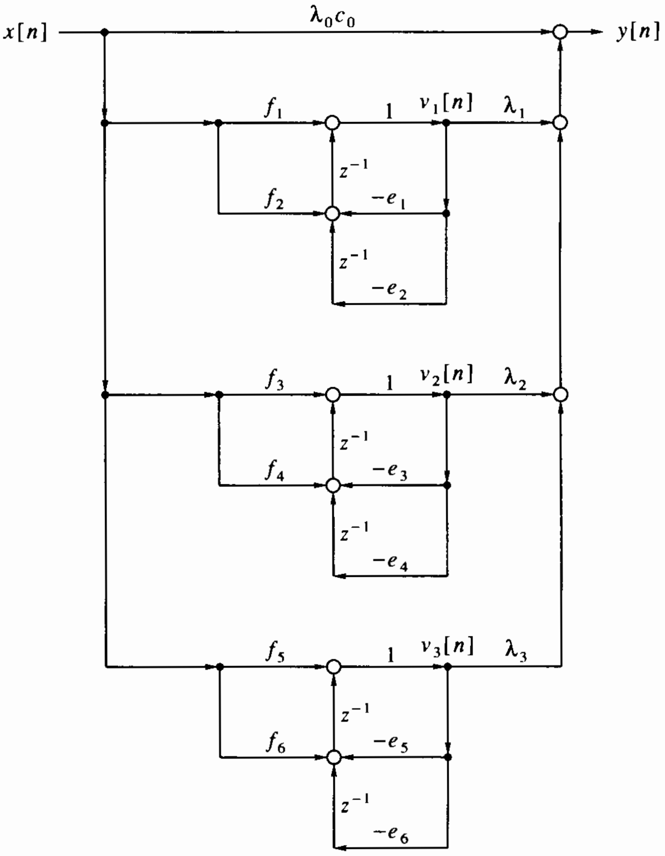

Figure 11.19 Scaling for a parallel realization (second-order sections in a transposed direct realization).

We now turn our attention to scaling of cascade realizations. Unlike a parallel realization, here we must choose the order of the sections in the cascade, since scaling depends on this order. There is no simple rule for choosing the best order in cascade realization: It depends on the nature of the input signal, the norm used for scaling, and the specific filter. Ordering rules have been proposed in the literature but are not discussed here. We use the order provided by the program pairpz. We reiterate that this program orders the complex poles in decreasing order of closeness to the unit circle (i.e., the poles closest to the unit circle appear first, and so on).

Figure 11.20 illustrates a cascade connection of three second-order sections. Let $H_k^z(z)$ denote the second-order sections from left to right (not including the scale factors). The scaling procedure is as follows. For each k, starting from 1 and continuing until all sections are exhausted, choose λ_k to avoid overflow in the transfer function $\prod_{i=1}^{k} \lambda_i H_i^z(z)$. At the kth stage, the factors $\{\lambda_i,\ 1 \le i \le k-1\}$ have already been determined, so only λ_k is free to choose. After this procedure has been completed, the output $y[n]$ will be free of overflow, but the total gain will be $\prod_k \lambda_k$, instead of the desired value b_0. The final stage is therefore to decrease one of the λ_k such that $\prod_k \lambda_k$ will be equal to b_0 times an integer power of 2. This way, the input-output transfer function will be the desired $H^z(z)$ scaled by a power of 2, as in the case of parallel realization.

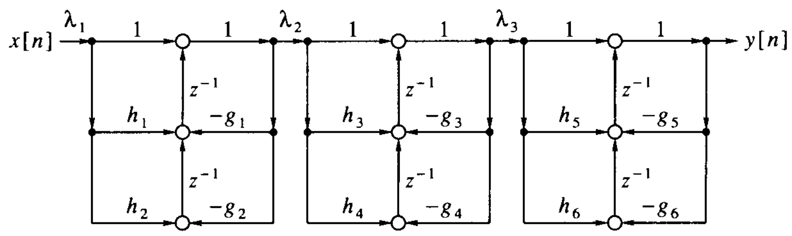

Figure 11.20 Scaling for a cascade realization (second-order sections in a transposed direct realization).

Example 11.11 We look again at the IIR filter discussed in Examples 11.5 and 11.10. The cascade decomposition of the filter, with coefficients quantized to 16 bits, is

$$H^{z}(z) = 29414 \times 2^{-20} \cdot \frac{1 - 28587 \times 2^{-14}z^{-1} + z^{-2}}{1 - 30365 \times 2^{-14}z^{-1} + 15710 \times 2^{-14}z^{-2}}$$
$$\cdot \frac{1 - 30493 \times 2^{-14}z^{-1} + z^{-2}}{1 - 28072 \times 2^{-14}z^{-1} + 13038 \times 2^{-14}z^{-2}} \cdot \frac{1 + z^{-1}}{1 - 25279 \times 2^{-15}z^{-1}}.$$

Assume, as in Example 11.10, that the input signal satisfies $\|K_x^{\mathrm{f}}\|_\infty = 0.1$. We decide to use 2-norm scaling such that the variance of the signal at the output of each section will not exceed 0.1.

The program filnorm gives the following 2-norms:

$$\|H_1^{\mathrm{f}}\|_2 = 1.0656, \quad \|H_1^{\mathrm{f}}H_2^{\mathrm{f}}\|_2 = 1.8730, \quad \|H_1^{\mathrm{f}}H_2^{\mathrm{f}}H_3^{\mathrm{f}}\|_2 = 11.5802.$$

From the first value we get that $\lambda_1 = 0.5$, from the second that $\lambda_1\lambda_2 = 0.5$, and from the third that $\lambda_1\lambda_2\lambda_3 = 2^{-4}$. Therefore, $\lambda_2 = 1$ and $\lambda_3 = 2^{-3}$. Finally, we must decrease one of the λ_k to account for b_0. We decide to decrease λ_2 from 1 to $29414 \times 2^{-15} (\approx 0.8976)$. This gives an overall gain of 29414×2^{-19}, which is equal to $2b_0$, quantized to 16 bits. In summary,

$$\lambda_1 = 0.5, \quad \lambda_2 = 29414 \times 2^{-15}, \quad \lambda_3 = 2^{-3}. \qquad \square$$

11.7 Quantization Noise*

11.7.1 Modeling of Quantization Noise

Addition of two fixed-point numbers represented by the same number of bits involves no loss of precision. It can, as we have seen, lead to overflow, but careful scaling usually prevents this from happening. The situation is different for multiplication. Every time we multiply two numbers of B bits each (of which 1 bit is for sign and $B - 1$ bits for magnitude) we get a result of $2B - 1$ bits (1 bit for sign and $2B - 2$ bits for magnitude). It is common, in fixed-point implementations, to immediately drop the $B - 1$ least significant bits of the product, retaining B bits (including sign) for subsequent computations. In two's-complement arithmetic, this truncation leads to a negative error, which can be up to $2^{-(B-1)}$ in magnitude. A slightly more sophisticated operation is to round the product either upward or downward so that the error does not exceed 2^{-B} in magnitude.

The error generated at any given multiplication depends on the operands. In LTI filters, one of the operands is always a constant number, whereas the other is always an instantaneous signal value. If the filter is well scaled, the instantaneous signal value will be a sizable fraction of the full scale most of the time. If, in addition, the number of bits is fairly large (say 12 or more), the error resulting from truncation (or rounding) of the product will appear random. By "appear random" we mean that it will not be random in the mathematical sense of the term, but will behave approximately as though it were random. The probability density function of the error will be approximately uniform, as shown in Figure 11.21. In case of truncation, the range of the error will be $(-2^{-(B-1)}, 0]$ [Figure 11.21(a)], whereas in case of rounding it will be $(-2^{-B}, 2^{-B}]$ [Figure 11.21(b)]. In both cases, the variance of the error will be

$$2^{B-1}\int_{-2^{-B}}^{2^{-B}} x^2 dx = \frac{2^{B-1}\times 2\times 2^{-3B}}{3} = \frac{2^{-2B}}{3}. \tag{11.111}$$

The mean of the error will be zero in case of rounding, and -2^{-B} in case of truncation.

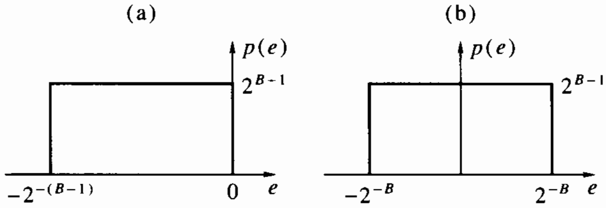

Figure 11.21 Probability density functions of quantization noise: (a) truncation; (b) rounding.

Experience shows that truncation or rounding errors at the output of a given multiplier at different times are usually uncorrelated, or approximately so. Furthermore, errors at different multipliers are usually uncorrelated as well because quantization errors, being very small fractions of multiplicands, are sufficiently random. Therefore, each error signal appears as discrete-time white noise, uncorrelated with the other error signals. This leads to the following:

Quantization noise model for a digital filter in fixed-point implementation: The error at the output of every multiplier is represented as additive discrete-time white noise. The total number of noise sources is equal to the number of multipliers, and they are all uncorrelated. The variance of each noise source is $2^{-2B}/3$. The mean is zero in case of rounding, and -2^{-B} in case of truncation.

It is common to express the mean of the noise in units of least significant bit (LSB), and the variance in units of LSB^2. Since the least significant bit value is $2^{-(B-1)}$, the mean is -0.5 LSB for truncation and 0 for rounding; the variance is $(1/12)$ LSB^2 in either case. From now on, for convenience, we shall omit the negative sign of the mean in the case of truncation.

The noise generated at any point in a filter is propagated through the filter and finally appears at the output. Since the filter is linear, the output noise is added to the output signal. The additive noise at the output is undesirable, since it degrades the performance of the system. For example, if the output signal is used for detecting a sinusoidal component in noise (as discussed in Section 6.5), quantization noise decreases the SNR. If the filter is used for audio signals (e.g., speech or music), the noise may be audible and become a nuisance. It is therefore necessary to analyze the noise at the output of the filter quantitatively, and verify that it does not exceed the permitted level. If it does, the word length must be increased to decrease the mean and the variance of the quantization error at the multipliers outputs.

The effect of noise generated at a particular point in a filter on the output signal can be computed in the following manner. Let $e[n]$ be the noise at the point of interest and let μ_e, γ_e be its mean and variance, respectively. Let $G^z(z)$ be the transfer function from $e[n]$ to the filter output $y[n]$, say

$$G^z(z) = \frac{\sum_{k=0}^{N} c_k z^{-k}}{\sum_{k=0}^{N} d_k z^{-k}}. \tag{11.112}$$

The noise mean at the output is the product of μ_e and the DC gain of $G^z(z)$, that is,

$$\mu_y = G^z(1)\mu_e = \frac{\mu_e \sum_{k=0}^{N} c_k}{\sum_{k=0}^{N} d_k}. \tag{11.113}$$

The noise variance at the output is the product of γ_e and the noise gain of $G^z(z)$, that is,

$$\gamma_y = \mathrm{NG}\,\gamma_e = \frac{\gamma_e}{2\pi}\int_{-\pi}^{\pi} |G^{\mathrm{f}}(\theta)|^2 d\theta. \tag{11.114}$$

The noise gain of $G^z(z)$ can be computed from its coefficients as explained in Section 7.4.5.

Since different noise sources are uncorrelated, the noise mean at the output is the sum of the individual means, and the noise variance at the output is the sum of the individual variances. If the filter has L noise sources $\{e_l[n],\ 1 \le l \le L\}$, with corresponding transfer functions $\{G_l^z(z),\ 1 \le l \le L\}$ to the output, then the mean and the variance of the noise at the output are

$$\mu_y = \sum_{l=1}^{L} G_l^z(1)\mu_{e,l}, \quad \gamma_y = \sum_{l=1}^{L} \frac{\gamma_{e,l}}{2\pi}\int_{-\pi}^{\pi} |G_l^{\mathrm{f}}(\theta)|^2 d\theta. \tag{11.115}$$

In the following we compute the output noise parameters (mean and variance) for standard filter realizations. We assume that truncation arithmetic is used. In case of rounding arithmetic, all means are zero.

11.7.2 Quantization Noise in Direct Realizations

Consider the transposed direct realization of a second-order filter. Such a realization contains five multipliers, as shown in Figure 11.22(a). We represent the error caused by each multiplication by an additive signal $e_k[n]$, $1 \le k \le 5$, at the output of the corresponding multiplier. Each of the $e_k[n]$ is white noise, and they are all uncorrelated.

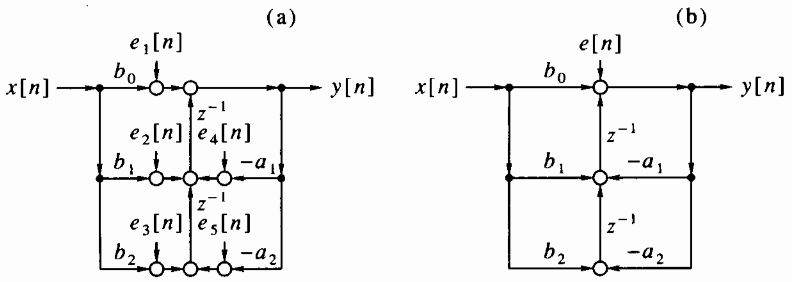

Figure 11.22 Quantization noise in a transposed direct realization: (a) separate noise sources; (b) combined noise source.

All five noise sources can be combined to a single equivalent noise source at the top adder of the realization, as shown in Figure 11.22(b). The equivalent noise is given by

$$e[n] = e_1[n] + e_2[n-1] + e_3[n-2] + e_4[n-1] + e_5[n-2]. \tag{11.116}$$

Because of the absence of correlation between the noise sources, the equivalent noise $e[n]$ has the following properties:

1. Its mean is the sum of the means, that is, 2.5 LSB.
2. Its variance is the sum of the variances, that is, $(5/12)\,\text{LSB}^2$.

Therefore, it is not necessary to consider the individual sources for further noise computations, only the equivalent source.

As we see from Figure 11.22(b), the transfer function from the noise to the output is

$$\frac{Y^z(z)}{E^z(z)} = \frac{1}{a(z)}. \tag{11.117}$$

Therefore,

1. The mean of the output is equal to the mean of the noise times the DC gain of the transfer function, that is,
$$\mu_y = \frac{2.5}{a(1)}\,\text{LSB} = \frac{2.5}{1 + a_1 + a_2}\,\text{LSB}. \tag{11.118}$$
2. The variance of the output is equal to the variance of the noise times the noise gain of the transfer function, that is,
$$\gamma_y = \frac{5}{12 \times 2\pi}\int_{-\pi}^{\pi} \frac{d\theta}{|a(e^{j\theta})|^2}\,\text{LSB}^2. \tag{11.119}$$

This method of analysis applies to an IIR filter of any order in a transposed direct realization. The number of noise sources adding up to the equivalent noise is $p + q + 1$. Therefore, we need only to multiply the aforementioned formulas by $(p + q + 1)/5$.

In the case of a linear-phase FIR filter, the number of multipliers is $\lfloor 0.5N \rfloor + 1$. The denominator polynomial is $a(z) = 1$. Therefore we get in this case,[7]

$$\mu_y = \frac{\lfloor 0.5N \rfloor + 1}{2}\,\text{LSB}, \quad \gamma_y = \frac{\lfloor 0.5N \rfloor + 1}{12}\,\text{LSB}^2. \tag{11.120}$$

Next consider the direct realization of a second-order filter. Such a realization contains five multipliers, as shown in Figure 11.23(a). As before, we represent the error caused by each multiplication by an additive signal $e_k[n]$, $1 \le k \le 5$, at the output of the corresponding multiplier. Each of the $e_k[n]$ is white noise, and they are all uncorrelated.

This time we cannot combine all five noise sources into a single equivalent source, but we can form two equivalent sources, as shown in Figure 11.23(b). The equivalent noise sources are given by

$$e_a[n] = e_1[n] + e_2[n], \quad e_b[n] = e_3[n] + e_4[n] + e_5[n]. \tag{11.121}$$

As we see from Figure 11.23(b), the transfer functions from the noise sources to the output are

$$\frac{Y^z(z)}{E_a^z(z)} = \frac{b(z)}{a(z)}, \quad \frac{Y^z(z)}{E_b^z(z)} = 1. \tag{11.122}$$

Therefore,

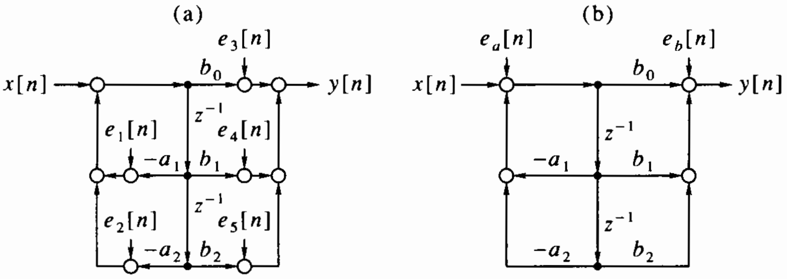

Figure 11.23 Quantization noise in a direct realization: (a) separate noise sources; (b) combined noise sources.

1. The mean of the output is

$$\mu_y = \left(\frac{b(1)}{a(1)} + \frac{3}{2}\right) \text{LSB} = \left(\frac{b_0 + b_1 + b_2}{1 + a_1 + a_2} + \frac{3}{2}\right) \text{LSB}. \tag{11.123}$$

2. The variance of the output is

$$\gamma_y = \left(\frac{2}{12 \cdot 2\pi} \int_{-\pi}^{\pi} \frac{|b(e^{j\theta})|^2}{|a(e^{j\theta})|^2} d\theta + \frac{3}{12}\right) \text{LSB}^2. \tag{11.124}$$

This method of analysis applies to an IIR filter of any order in a direct realization with obvious modifications: Instead of the factor 2 we use p, and instead of the factor 3 we use $q + 1$.

In the case of a linear-phase FIR filter, we get

$$\mu_y = \frac{\lfloor 0.5N \rfloor + 1}{2} \text{LSB}, \quad \gamma_y = \frac{\lfloor 0.5N \rfloor + 1}{12} \text{LSB}^2. \tag{11.125}$$

11.7.3 Quantization Noise in Parallel and Cascade Realizations

Figure 11.24 shows a scaled parallel realization, with second-order sections in transposed direct realization, and noise sources added. We observe the following:

1. There are four multipliers in each section, so the mean of the equivalent noise source $e_k[n]$ is 2 LSB and the variance is $(1/3)\,\text{LSB}^2$.
2. If the realization also contains a first-order section (there can be one such section at most), the mean of its equivalent noise source is 1 LSB and the variance is $(1/6)\,\text{LSB}^2$.
3. The scale factors λ_k are usually integer powers of 2, so they generate no extra noise (the corresponding left or right shifts are usually performed prior to truncation).
4. The amplified noise components add up at the output of the filter, as a result of the parallel connection. Also adding up at this point is the noise resulting from the constant branch c_0.

Example 11.12 Consider again the scaled parallel realization of the IIR filter discussed in Example 11.10. We wish to compute the mean and variance of the output noise to two decimal digits. This accuracy is sufficient for most practical purposes. The DC gains of the three sections (from the corresponding noise inputs) are 9.5, 12, and 4.4.

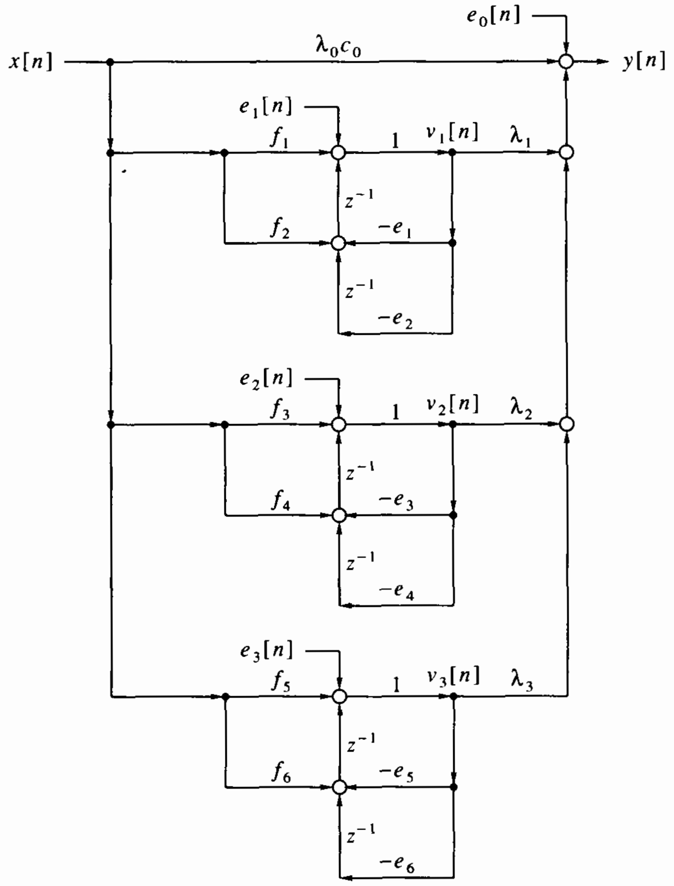

Figure 11.24 Quantization noise in a parallel realization (second-order sections in a transposed direct realization).

The noise gains are 118, 30, and 2.5. Taking into account the scale factors λ_k and adding the effect of the constant branch, we get

$$\mu_y = [2(0.5 \times 9.5 + 12) + 4.4 + 0.5]\,\text{LSB} \approx 39\,\text{LSB},$$
$$\gamma_y = [0.33(0.25 \times 118 + 30) + (2.5 \times 0.17) + 0.083]\,\text{LSB}^2 \approx 20\,\text{LSB}^2.$$

As we see, the mean output noise is about 39 least significant bits, and the standard deviation is about 4.5 least significant bits. □

Figure 11.25 shows a scaled cascade realization, with second-order sections in a transposed direct realization, and noise sources added. We observe the following:

1. There are five multipliers in each second-order section. However, one of those is identically 1, so it is not subject to quantization. Furthermore, because of the special properties of the zeros of a digital IIR filter derived from a standard analog filter, the numerator coefficients h_{2k-1}, h_{2k} are not arbitrary: h_{2k} is either 1 or -1, and h_{2k-1} is either general (if the zeros are complex) or ± 2 (if the zeros are real). Therefore, the effective number of noise sources in a section is either two or three. Correspondingly, the mean of the equivalent noise source $e_k[n]$ is 1 LSB or 1.5 LSB and the variance is $(1/6)\,\text{LSB}^2$ or $(1/4)\,\text{LSB}^2$.

2. If the realization also contains a first-order section (there can be one such section at most), this section will have only one noise source, since the numerator coefficients are ±1, and are not subject to quantization. Therefore, the mean of the equivalent noise source is 0.5 LSB and the variance is $(1/12)\,\text{LSB}^2$.
3. One of the scale factors along the cascade is a general number, so it adds a noise source. The other factors are integer powers of 2, so they are not subject to quantization.
4. The transfer function from the noise source $e_k[n]$ to the output is not just $1/g_k(z)$, but $1/g_k(z)$ times the product of transfer functions of the sections further down the cascade. Therefore, the noise gain computations are more laborious than in the case of parallel realization.

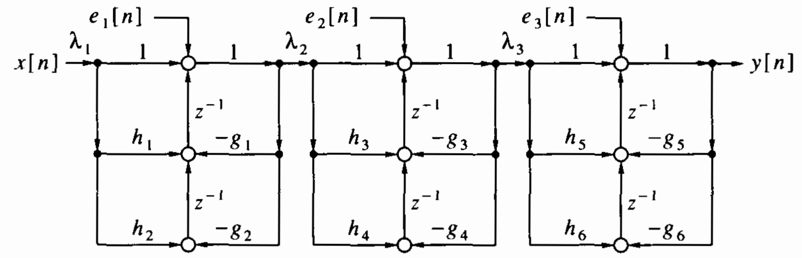

Figure 11.25 Quantization noise in a cascade realization (second-order sections in a transposed direct realization).

Example 11.13 Consider again the scaled cascade realization of the IIR filter discussed in Example 11.11. We wish to compute the mean and variance of the output noise to two decimal digits. In this example, the first two sections are second order and the third is first order. The multiplier λ_2 is subject to quantization. The DC gains from $e_1[n]$, $e_2[n]$, $e_3[n]$, and λ_2 are 29, 13, 4.4, and 3.4, respectively. The noise gains are 302, 21, 2.5, and 1.0. We get

$$\mu_y = [1.5(29 + 13) + 0.5(4.4 + 3.4)]\,\text{LSB} \approx 67\ \text{LSB},$$
$$\gamma_y = [0.25(302 + 13) + 0.083(2.5 + 1.0)]\,\text{LSB}^2 \approx 81\ \text{LSB}^2.$$

As we see, the mean output noise is about 67 LSB and the standard deviation is about 9 LSB. In this case, the noise level of the cascade realization is about twice that of the parallel realization. Different ordering of the sections will lead in general to a different (and possibly lower) noise level.

Recall, for comparison, that an FIR filter meeting the same specifications has order 146 (Example 9.16). We get from (11.120) that the mean of the output noise of the FIR filter is 37 LSB and the standard deviation is about 2.5 LSB. Therefore, the FIR filter is better than both IIR filter realizations in its output noise level. □

11.7.4 Quantization Noise in A/D and D/A Converters

In Section 3.5.2 we introduced A/D converters, and we noted that they provide quantized signals, either truncated or rounded. The tools we developed here enable us to complete the discussion on A/D quantization and analyze this effect quantitatively. We assume, as in the case of arithmetical quantization, that the quantization level is small and the input signal is large and fast varying. Under these assumptions, A/D quantization noise can be modeled as discrete-time white noise. The mean of the noise is 0 in case of rounding, and half the least significant bit of the A/D in case of truncation. The variance is $1/12$ of the square of the least significant bit of the A/D. The mean is multiplied by the DC gain of the filter to yield the mean of the output. The variance is multiplied by the noise gain of the filter to yield the variance of the output. The noise and DC gains of the filter depend only on its total transfer function and are independent of the realization. The alignment of the least significant bit of the A/D word relative to the computer word determines the quantization level used for noise computations. Suppose, for example, that the computer word length is 16 bits and that of the A/D is 12 bits. Then the quantization level of the A/D is 2^{-15} if its output is placed at the low bits, but is 2^{-11} if placed at the high bits. However, since this placement similarly determines the dynamic range of the output signal, the *relative noise level* at the output (the signal-to-noise ratio) depends on the number of bits of the A/D, but not on their placement.

D/A converters are not, by themselves, subject to quantization. However, sometimes the word length of the D/A is shorter than that of the filter. In such cases, the signal $y[n]$ at the output of the filter must be further quantized (truncated or rounded) before it is passed to the D/A converter. We can regard this quantization as yet another source of white noise in the system. However, contrary to the preceding effects, this noise is not processed by the digital filter. Therefore, it is neither amplified nor attenuated, but appears at the output as is.

11.8 Zero-Input Limit Cycles in Digital Filters*

When a stable linear filter receives no input, and its internal state has nonzero initial conditions, its output decays to zero asymptotically. This follows because the response of each pole of the filter to initial conditions is a geometric series with parameter less than 1 in magnitude (Sections 7.5, 7.7). However, the analysis that leads to this conclusion is based on the assumption that signals in the filter are represented to infinite precision, so they obey the mathematical formulas exactly. Quantization resulting from finite word length is a nonlinear operation, so stability properties of linear systems do not necessarily hold for a filter subject to quantization. Indeed, digital filters can exhibit sustained oscillations when implemented with finite word length. Oscillations resulting from nonlinearities are called *limit cycles.*

Limit cycle phenomena are different from the noiselike behavior caused by quantization. Quantization effects are noiselike when the signal level is large and relatively fast varying, rendering the quantization error at any given time nearly independent of the errors at past times. When the signal level is low, errors caused by quantization become correlated. When the input signal is zero, randomness disappears, and the error behavior becomes completely deterministic. Oscillations in the absence of input are called *zero-input limit cycles.* Such oscillations are periodic, but not necessarily sinusoidal. They are likely to appear whenever there is feedback in the filter. Digital IIR filters always have inner feedback paths, so they are susceptible to limit cycle oscillations. On the other hand, FIR filters are feedback free, so they are immune to limit cycles. This is yet another advantage of FIR filters over IIR filters. Limit cycles may be troublesome in applications such as speech and music, because the resulting signal may be audible.

In this section we briefly discuss zero-input limit cycles in IIR filter structures. Since practical IIR implementations are almost always in parallel or cascade, we concentrate on first- and second-order sections. Mathematical analysis of limit cycles is difficult, except for first-order filters. We shall therefore omit most mathematical details, and restrict our discussion to practical techniques for analysis and remedy of limit cycle phenomena.

Example 11.14 Let us illustrate the limit-cycle phenomenon by a few simple examples.

1. Consider the first-order filter
$$y[n] = -0.625y[n-1] + x[n].$$
Suppose that we implement the filter with 4-bit rounding arithmetic, so the least significant bit is 1/8. Let the input signal $x[n]$ be zero and $y[0] = 3/8$. Then $y[1] = -15/64$, and this will be rounded to $y[1] = -1/4$. Next, $y[2] = 5/32$, and this will be rounded to $y[2] = 1/8$. Next, $y[3] = -5/64$, and this will be rounded to $y[3] = -1/8$. Next, $y[4] = 5/64$, and this will be rounded to $y[4] = 1/8$. From this point on we will have $y[n] = 1/8$ for all even n and $y[n] = -1/8$ for all odd n. We get a constant amplitude oscillation whose frequency is $\theta_0 = \pi$ and whose amplitude is 1/8.
2. Next consider the same filter, but implemented with 4-bit truncation arithmetic. The sequence of values of the output signal will now be
$$y[n] = 3/8,\ -1/4,\ 1/8,\ -1/8,\ 0,\ 0,\ \dots\ .$$
As we see, truncation of $y[4]$ makes it zero, thus stopping the oscillations. This time there is no limit cycle.
3. Consider the filter
$$y[n] = 0.625y[n-1] + x[n],$$
with 4-bit rounding arithmetic. This time the output sequence is
$$y[n] = 3/8,\ 1/4,\ 1/8,\ 1/8,\ 1/8,\ \dots\ .$$
As we see, the output reaches a constant nonzero value. This phenomenon is called *zero-frequency limit cycle.*
4. For the same filter as in part 3, but with truncation arithmetic and the same initial condition, $y[n]$ will reach zero as in part 2.
5. Consider the filter of part 3, but with truncation arithmetic and initial condition $y[0] = -3/8$. Then,
$$y[n] = -3/8,\ -1/4,\ -1/8,\ -1/8,\ -1/8,\ \dots\ .$$
We get zero-frequency limit cycle again, this time with a negative constant value.

The lesson from this example is that the existence and nature of limit cycles depend on the filter in question, the quantization level, the method of quantization, and the initial conditions. □

Zero-input limit cycles in first-order filters are relatively easy to analyze. Let $y[n] = -ay[n-1] + x[n]$ be the filter in question, and assume that $x[n] = 0$. Then:

1. Zero-frequency limit cycle is possible in rounding arithmetic only if
$$\text{round}\{-2^{B-1}ay\} = 2^{B-1}y, \tag{11.126}$$
where y is the steady-state value of the output. This is equivalent to (Problem 11.25)
$$|2^{B-1}(1+a)y| \le 0.5, \tag{11.127}$$

or

$$|y| \le \frac{2^{-B}}{1+a}. \tag{11.128}$$

This condition imposes an upper limit on the magnitude that a zero-frequency limit cycle can attain, as a function of the word length and the coefficient of the filter. The permissible magnitude range is called the *dead band* of the filter. It follows from (11.128) that limit cycle of this kind is possible only if $-1 < a \le -0.5$. Note that the dead band increases as a approaches -1.

2. Limit cycle at frequency π is possible in rounding arithmetic only if

$$\text{round}\{-2^{B-1}ay\} = -2^{B-1}y. \tag{11.129}$$

This is equivalent to

$$|2^{B-1}(1-a)y| \le 0.5, \tag{11.130}$$

or

$$|y| \le \frac{2^{-B}}{1-a}. \tag{11.131}$$

This condition again imposes an upper limit on the magnitude that a limit cycle at frequency π can attain. In this case limit cycle is possible only if $0.5 \le a < 1$. Note that the dead band increases as a approaches 1.

3. Zero-frequency limit cycle is possible in truncation arithmetic only if

$$\lfloor -2^{B-1}ay \rfloor = 2^{B-1}y. \tag{11.132}$$

This leads to the condition

$$-\frac{2^{-(B-1)}}{1+a} < y < 0. \tag{11.133}$$

This kind of limit cycle is possible only if $-1 < a < 0$.

4. Limit cycle at frequency π is possible in truncation arithmetic only if

$$\lfloor -2^{B-1}ay \rfloor = -2^{B-1}y. \tag{11.134}$$

This is equivalent to

$$0 < y < \frac{2^{-(B-1)}}{1-a}. \tag{11.135}$$

This leads to contradiction, since y cannot oscillate and remain nonnegative at all times. The conclusion is that limit cycles at frequency π are not possible in first-order filters if truncation arithmetic is used.[8]

Limit cycles in second-order filters are much more difficult to analyze. As in the case of first-order filters, limit cycles are possible at both frequencies 0 and π. However, second-order filters can also sustain limit cycles at other frequencies. The frequency and amplitude of the limit cycle (if it exists) depend on the coefficients, initial conditions, quantization method, and word length. However, in case of a second-order section they also depend on the realization. In particular, the coupled realization presented in Section 11.1.7 is less susceptible to limit cycles than a direct realization. The coupled realization is represented by the state-space equations (11.32). We now show that the coupled realization can be made completely free of zero-input limit cycles if two precautions are taken in its implementation:

1. *Magnitude truncation*, also called *rounding toward zero*, is used instead of rounding or truncation. The magnitude truncation of a number a is the number $\lfloor |a| \rfloor \text{sign}(a)$. This method of truncation is seldom implemented in hardware, so it usually requires special programming.

2. Magnitude truncation of each component of $s[n+1]$ is performed *after* both products (by α_r and α_i) are formed and added (subtracted) in double precision.

To show the coupled realization is free of zero-input limit cycles under these conditions, observe from (11.32) that

$$\begin{aligned}(s_1[n+1])^2 + (s_2[n+1])^2 &= (\alpha_r s_1[n] + \alpha_i s_2[n])^2 + (-\alpha_i s_1[n] + \alpha_r s_2[n])^2 \\ &= (\alpha_r^2 + \alpha_i^2)\{(s_1[n])^2 + (s_2[n])^2\} = |\alpha|^2\{(s_1[n])^2 + (s_2[n])^2\} \\ &< (s_1[n])^2 + (s_2[n])^2,\end{aligned} \tag{11.136}$$

since $|\alpha|^2 < 1$. Furthermore, magnitude quantization reduces the left side still further, because it reduces each of $|s_1[n+1]|$ and $|s_2[n+1]|$ individually. The conclusion is that the sum of squares of the state-vector components strictly decreases as a function of n at least as fast as a geometric series with parameter $|\alpha|^2$. Therefore, there must come a time n for which both $|s_1[n]|$ and $|s_2[n]|$ be less than $2^{-(B-1)}$. At this point they are truncated to zero, and the filter comes to a complete rest.

Coupled realization, implemented with magnitude truncation after multiplication and addition at each stage, is therefore free of zero-input limit cycles, so it is recommended whenever such limit cycles must be prevented. However, the following drawbacks of this method should not be overlooked:

1. The realization is costly in number of operations: There are 6 multiplications and 5 additions per section (compared with 4 and 4 for direct realization), and magnitude truncation requires additional operations.
2. Magnitude truncation doubles the standard deviation of the quantization noise (compared with rounding or truncation).

Since limit cycles in second-order sections are difficult to analyze mathematically, it is helpful to develop alternative tools for exploring them. The MATLAB procedure `lc2sim`, listed in Program 11.17, is such a tool. This procedure tests a given second-order realization for limit cycles by simulating its zero-input response. The procedure accepts the following parameters: `qtype` indicates the quantization type—rounding, truncation, or magnitude truncation; `when` indicates whether quantization is to be performed before or after the addition; `rtype` enables choosing between direct and coupled realizations; `apar` is a vector of two parameters, either a_1, a_2 or α_r, α_i; `B` is the number of bits; `s0` is the initial state vector; finally, `n` is the maximum number of points to be simulated. The program implements the realization in state space, and performs the necessary quantizations by calling the auxiliary procedure `quant`, listed in Program 11.18. At the end of each step, the program performs two tests. First it tests whether the state vector is identically zero. If so, the realization is necessarily free of zero-input limit cycles, since zero state at any time implies zero state at all subsequent times. Next it tests whether the state vector is the same as in the preceding step. If so, it means that zero-frequency limit cycle has been reached. If the program reaches the end of the simulation without either of these conditions being met, it declares that a limit cycle exists at nonzero frequency. The program outputs its decision in the variable `flag`, and also the history of the output signal $y[n]$.

The procedure `lcdrive` in Program 11.19 is a driver program for `lc2sim`. Note that it is a script file, rather than a function. Therefore, the variables must be declared in the MATLAB environment prior to running the file. The program determines the maximum duration of the simulation as a function of the magnitude of the poles. It then performs `M` simulations (where `M` is entered by the user), each time choosing the initial state at random. This is necessary because absence of a limit cycle from a specific initial state does not guarantee absence of limit cycles from other initial states. If, at any of the M simulations, the existence of limit cycle is observed, the program stops and reports occurrence. If no limit cycles are observed, the program reports that the realization appears to be free of limit cycles. The reliability of this test increases with the value assigned to M.

Example 11.15 We repeat the test of the IIR low-pass filter used in the earlier examples in this chapter. We examine each of its two sections for possible limit cycles in all 12 combinations of quantization method, quantization before or after the addition, and direct or coupled realization. Performing 100 simulations for each case, we find the following:

1. Both second-order sections are free of limit cycles if coupled realization with magnitude truncation is used, regardless of whether truncation is performed before or after the addition.
2. Both second-order sections are also free of limit cycles if a direct realization with magnitude truncation is used, provided that truncation is performed *after* the addition.
3. In all other cases, limit cycles may occur, at either zero or nonzero frequency. □

11.9 Summary and Complements

11.9.1 Summary

This chapter was devoted to three different, but related topics: digital filter realization, state-space representations of digital systems, and finite word length effects in digital filters.

Realization of a digital filter amounts to constructing a block diagram of the internal structure of the filter. Such a block diagram shows the elementary blocks of which the filter is constructed, their interconnections, and the numerical parameters associated with them. For LTI filters, only three block types are needed: delays, adders, and multipliers. Standard digital filter realizations include the direct (of which there are two forms), the parallel, and the cascade realizations. They all have the same number of delays, adders, and multipliers. However, they differ in their behavior when implemented in finite word length.

The standard realizations are special cases of the general concept of state-space representation. A state-space representation describes the time evolution of the memory variables (i.e., the state) of the filter, and the dependence of the output signal on the state and on the input signal. To a given LTI filter, there correspond an infinite number of similar state-space representations. State space is a useful tool for performing computations related to digital filters, for example, transfer function and impulse response computations. State-space representations of complex digital networks can be constructed using well-defined procedures.

Finite word length implementation affects a digital filter in several respects:

1. It causes the filter coefficients to deviate from their ideal values. Consequently, the poles, zeros, and frequency response are changed, and the filter may fail to meet the specifications.

2. The dynamic range of the various signals (input, output, and internal) becomes a problem (in fixed-point implementation), and may lead to overflows or saturations. Hence there is a need to scale the signals at various points of the filter, to maximize the dynamic range with little or no danger of overflow.
3. Quantization of multiplication operations leads to computational noise. The noise is propagated to the output and added to the output signal.
4. Digital filter structures may develop self oscillations, called limit cycles, in the absence of input.

It turns out that different realizations have different sensitivities to these effects. Direct realizations are the worst on almost all accounts. Both parallel and cascade realizations are much better, and their sensitivities are comparable. The parallel realization typically has a better pass-band behavior, whereas the cascade realization typically has a better stop-band behavior. The cascade realization is more general and more flexible than the parallel realization; therefore it is preferred in most applications of IIR filters. For FIR filters, on the other hand, direct realizations are the most common, since their sensitivity to finite word length effects, although worse than that of a cascade realization, is usually acceptable.

11.9.2 Complements

1. [p. 393] Some books use the name "direct realization II" for the structure appearing in Figure 11.4 and the name "direct realization I" for the nonminimal structure discussed in Problem 11.2; we avoid this terminology.
2. [p. 395] Programs 11.1, 11.3, and 11.5 get the input signal $x[n]$ in its entirety as a single object. In real-time applications, $x[n]$ is supplied sequentially, one value per sampling interval. In such cases it is necessary to store the state vector u, either internally as a *static* variable, or externally. For example, the MATLAB function `filter` optionally accepts the state vector as input and optionally returns its updated value. This facilitates external storage of the state.
3. [p. 398] Parallel realization of a system with multiple poles has a mixed parallel/series structure, see Kailath [1980] for details.
4. [p. 413] There are software schemes that greatly alleviate this problem. Consider, for example, the case of complex poles at $z = \rho e^{\pm j\zeta}$, close to $z = 1$. Then ρ is nearly 1 and ζ is nearly 0. Since $a_1 = -2\rho\cos\zeta$ and $a_2 = \rho^2$, a_1 is nearly -2 and a_2 is nearly 1. Write the difference equation for the auxiliary variable $u[n]$ of the direct realization as [cf. (11.2)]

$$\begin{aligned} u[n] = x[n] + u[n-1] + (u[n-1] - u[n-2]) \\ - (a_1+2)u[n-1] - (a_2-1)u[n-2]. \end{aligned}$$

Now the modified coefficients (a_1+2) and (a_2-1) are both small numbers, so we can scale them up by a few bits, thus retaining a few of their lower order bits. For instance, if the modified coefficients are around 0.05, we can gain 4 significant bits this way. The products $(a_1+2)u[n-1]$ and $(a_2-1)u[n-2]$ are generated in double precision by default in most computers. These products are shifted to the right by the same number of bits used for scaling the modified coefficients, truncated to the basic word length, and added to $u[n]$. This scheme is more accurate than truncating the coefficients prior to multiplication. The addition of $u[n-1]$ and $(u[n-1]-u[n-2])$ can be done exactly, with no loss in precision. Since the filter is necessarily low pass, the signal $u[n]$ changes slowly in time, so

$|u[n-1]-u[n-2]|$ is typically small compared with the full scale of $u[n]$. If very high accuracy is required, $u[n]$ can be stored in double precision. In this case all the additions on the right side of the equation should be in double precision as well.

5. [p. 414] The differentiation that takes place here may be unfamiliar. Suppose that $x(t)$ is a complex-valued function of a real variable t. Then $y(t) = |x(t)|$ is a real-valued function of t. If $x(t)$ is differentiable, then $y(t)$ is differentiable at any point t for which $x(t) \neq 0$. The derivative of $y(t)$ can be obtained as follows. We first observe that
$$y(t) = (|x(t)|^2)^{1/2},$$
hence
$$\frac{dy(t)}{dt} = 0.5(|x(t)|^2)^{-1/2}\frac{d|x(t)|^2}{dt} = 0.5|x(t)|^{-1}\frac{d|x(t)|^2}{dt}.$$
Then we observe that
$$|x(t)|^2 = x(t)\bar{x}(t), \quad \text{hence} \quad \frac{d|x(t)|^2}{dt} = \frac{dx(t)}{dt}\bar{x}(t) + \frac{d\bar{x}(t)}{dt}x(t).$$
Therefore,
$$\frac{dy(t)}{dt} = 0.5|x(t)|^{-1}\left[\frac{dx(t)}{dt}\bar{x}(t) + \frac{d\bar{x}(t)}{dt}x(t)\right].$$
If we express $x(t)$ as $|x(t)|e^{j\psi(t)}$, where $\psi(t)$ is the phase of $x(t)$, we will get that
$$|x(t)|^{-1}x(t) = e^{j\psi(t)}, \quad |x(t)|^{-1}\bar{x}(t) = e^{-j\psi(t)}.$$
Therefore, we can express the derivative of $y(t)$ as
$$\frac{dy(t)}{dt} = 0.5\left[\frac{dx(t)}{dt}e^{-j\psi(t)} + \frac{d\bar{x}(t)}{dt}e^{j\psi(t)}\right] = \Re\left\{\frac{dx(t)}{dt}e^{-j\psi(t)}\right\}. \tag{11.137}$$
This is the formula used for obtaining (11.79).

6. [p. 422] For example, if $y[n]$ has Gaussian probability density, this probability is about 0.0016.

7. [p. 429] Quantization noise in FIR filters can be greatly reduced by performing the summation
$$y[n] = \sum_{k=0}^{N} h[k]x[n-k]$$
in double precision. In this case, the products $h[k]x[n-k]$ are not truncated at all, only the final value of $y[n]$ is truncated. This reduces the factor in (11.120) from $(\lfloor 0.5N \rfloor + 1)$ to 1. Such an implementation is recommended when the order of the filter is large and noise is a major concern (as in audio applications).

8. [p. 435] It can be shown that first-order filters can sustain limit cycles only at frequencies 0 or π.

11.10 MATLAB Programs

Program 11.1 Direct realizations of digital IIR filters.

```
function y = direct(typ,b,a,x);
% Synopsis: direct(typ,b,a,x).
% Direct realizations of rational transfer functions.
% Input parameters:
% typ: 1 for direct realization, 2 for transposed
% b, a: numerator and denominator polynomials
% x: input sequence.
% Output:
% y: output sequence.

p = length(a)-1; q = length(b)-1; pq = max(p,q);
a = a(2:p+1); u = zeros(1,pq); % u: the internal state
if (typ == 1),
   for i = 1:length(x),
      unew = x(i)-sum(u(1:p).*a);
      u = [unew,u];
      y(i) = sum(u(1:q+1).*b);
      u = u(1:pq);
   end
elseif (typ == 2),
   for i = 1:length(x),
      y(i) = b(1)*x(i)+u(1);
      u = [u(2:pq),0];
      u(1:q) = u(1:q) + b(2:q+1)*x(i);
      u(1:p) = u(1:p) - a*y(i);
   end
end
```

Program 11.2 Computation of the parallel decomposition of a digital IIR filter.

```
function [c,nsec,dsec] = tf2rpf(b,a);
% Synopsis: [c,nsec,dsec] = tf2rpf(b,a).
% Real partial fraction decomposition of b(z)/a(z). The polynomials
% are in negative powers of z. The poles are assumed distinct.
% Input parameters:
% a, b: the input polynomials
% Output parameters:
% c: the free polynomial; empty if deg(b) < deg(a)

nsec = []; dsec = []; [c,A,alpha] = tf2pf(b,a);
while (length(alpha) > 0),
  if (imag(alpha(1)) ~= 0),
    dsec = [dsec; [1,-2*real(alpha(1)),abs(alpha(1))^2]];
    nsec = [nsec; [2*real(A(1)),-2*real(A(1)*conj(alpha(1)))]];
    alpha(1:2) = []; A(1:2) = [];
  else,
    dsec = [dsec; [1,-alpha(1),0]]; nsec = [nsec; [real(A(1)),0]];
    alpha(1) = []; A(1) = [];
  end
end
```

Program 11.3 Parallel realization of a digital IIR filter.

```
function y = parallel(c,nsec,dsec,x);
% Synopsis: y = parallel(c,nsec,dsec,x).
% Parallel realization of an IIR digital filter.
% Input parameters:
% c: the free term of the filter.
% nsec, dsec: numerators and denominators of second-order sections
% x: the input sequence.
% Output:
% y: the output sequence.

[n,m] = size(dsec); dsec = dsec(:,2:3);
u = zeros(n,2); % u: the internal state
for i = 1:length(x),
   y(i) = c*x(i);
   for k = 1:n,
      unew = x(i)-sum(u(k,:).*dsec(k,:)); u(k,:) = [unew,u(k,1)];
      y(i) = y(i) + sum(u(k,:).*nsec(k,:));
   end
end
```

Program 11.4 Pairing of poles and zeros to real second-order sections.

```
function [nsec,dsec] = pairpz(v,u);
% Synopsis: [nsec,dsec] = pairpz(v,u).
% Pole-zero pairing for cascade realization.
% Input parameters:
% v, u: the vectors of poles and zeros, respectively.
% Output parameters:
% nsec: matrix of numerator coefficients of second-order sections
% dsec: matrix of denom. coefficients of second-order sections.

if (length(v) ~= length(u)),
  error('Different numbers of poles and zeros in PAIRPZ'); end
u = reshape(u,1,length(u)); v = reshape(v,1,length(v));
v = cplxpair(v); u = cplxpair(u);
vc = v(find(imag(v) > 0)); uc = u(find(imag(u) > 0));
vr = v(find(imag(v) == 0)); ur = u(find(imag(u) == 0));
[temp,ind] = sort(abs(vc)); vc = vc(fliplr(ind));
[temp,ind] = sort(abs(vr)); vr = vr(fliplr(ind));
nsec = []; dsec = [];
for n = 1:length(vc),
   dsec = [dsec; [1,-2*real(vc(n)),abs(vc(n))^2]];
   if (length(uc) > 0),
      [temp,ind] = min(abs(vc(n)-uc)); ind = ind(1);
      nsec = [nsec; [1,-2*real(uc(ind)),abs(uc(ind))^2]];
      uc(ind) = [];
   else,
      [temp,ind] = min(abs(vc(n)-ur)); ind = ind(1);
      tempsec = [1,-ur(ind)]; ur(ind) = [];
      [temp,ind] = min(abs(vc(n)-ur)); ind = ind(1);
      tempsec = conv(tempsec,[1,-ur(ind)]); ur(ind) = [];
      nsec = [nsec; tempsec];
   end
end
if (length(vr) == 0), return
elseif (length(vr) == 1),
   dsec = [dsec; [1,-vr,0]]; nsec = [nsec; [1,-ur,0]];
elseif (length(vr) == 2),
   dsec = [dsec; [1,-vr(1)-vr(2),vr(1)*vr(2)]];
   nsec = [nsec; [1,-ur(1)-ur(2),ur(1)*ur(2)]];
else
   error('Something wrong in PAIRPZ, more than 2 real zeros');
end
```

Program 11.5 Cascade realization of a digital IIR filter.

```
function y = cascade(C,nsec,dsec,x);
% Synopsis: y = cascade(C,nsec,dsec,x).
% Cascade realization of an IIR digital filter.
% Input parameters:
% C: the constant gain of the filter.
% nsec, dsec: numerators and denominators of second-order sections
% x: the input sequence.
% Output:
% y: the output sequence.

[n,m] = size(dsec);
u = zeros(n,2); % u: the internal state
dsec = dsec(:,2:3); nsec = nsec(:,2:3);
for i = 1:length(x),
   for k = 1:n,
      unew = x(i)-sum(u(k,:).*dsec(k,:));
      x(i) = unew + sum(u(k,:).*nsec(k,:));
      u(k,:) = [unew,u(k,1)];
   end
   y(i) = C*x(i);
end
```

Program 11.6 Computation of the state-space matrices corresponding to a given transfer function.

```
function [A,B,C,D] = tf2ss(b,a);
% Synopsis: [A,B,C,D] = tf2ss(b,a).
% Converts a transfer function to direct state-space realization.
% Inputs:
% b, a: the numerator and denominator polynomials.
% Outputs:
% A, B, C, D: the state-space matrices

p = length(a)-1; q = length(b)-1; N = max(p,q);
if (N > p), a = [a,zeros(1,N-p)]; end
if (N > q), b = [b,zeros(1,N-q)]; end
A = [-a(2:N+1); [eye(N-1), zeros(N-1,1)]];
B = [1; zeros(N-1,1)];
C = b(2:N+1) - b(1)*a(2:N+1);
D = b(1);
```

Program 11.7 Computation of the transfer function corresponding to given state-space matrices.

```
function [b,a] = ss2tf(A,B,C,D);
% Synopsis: [b,a] = tf2ss(A,B,C,D).
% Converts a state-space realization to a transfer function.
% Inputs:
% A, B, C, D: the state-space matrices
% Outputs:
% b, a: the numerator and denominator polynomials.

a = poly(A); N = length(a)-1; h = zeros(1,N+1); h(1) = D; tmp = B;
for i = 1:N, h(i+1) = C*tmp; tmp = A*tmp; end
b = a*toeplitz([h(1);zeros(N,1)],h);
```

Program 11.8 Construction of a state-space representation of a digital network.

```
function [A,B,C,D] = network(F,G,H,K,Rnum,Rden);
% Synopsis: [A,B,C,D] = network(F,G,H,K,Rnum,Rden).
% Builds a state-space representation of a digital network.
% Input parameters:
% F, G, H, K: network connection matrices
% Rnum: rows contain numerator coefficients of blocks
% Rden: rows contain denominator coefficients of blocks
% Output parameters:
% A, B, C, D: state-space matrices.

[L,Nnum] = size(Rnum); [L,Nden] = size(Rden);
A = []; B = []; C = []; D = []; N = 0;
for l = 1:L,
   rnum = Rnum(l,:); rden = Rden(l,:);
   while (rnum(length(rnum)) == 0),
      rnum = rnum(1:length(rnum)-1); end
   while (rden(length(rden)) == 0),
      rden = rden(1:length(rden)-1); end
   [At,Bt,Ct,Dt] = tf2ss(rnum,rden); Nt = length(Bt);
   A = [A,zeros(N,Nt); zeros(Nt,N),At];
   B = [B,zeros(N,1); zeros(Nt,l-1),Bt];
   C = [C,zeros(l-1,Nt); zeros(1,N),Ct];
   D = [D,zeros(l-1,1); zeros(1,l-1),Dt];
   N = N + Nt;
end
E = eye(L)-F*D;
if (rank(E) < L), error('Network is singular'), end
E = inv(E); A = A + B*E*F*C; B = B*E*G;
C = H*(eye(L) + D*E*F)*C; D = K + H*D*E*G;
```

Program 11.9 Sensitivity bound for the magnitude response of an IIR filter to coefficient quantization.

```
function [dHmag,S] = sensiir(typ,b,a,K,theta);
% Synopsis: [dHmag,S] = sensiir(typ,b,a,K,theta).
% Computes the sensitivity bound for the magnitude response of
% an IIR filter to coefficient quantization.
% Input parameters:
% typ: 'd' for direct realization
%      'p' for parallel realization
%      'c' for cascade realization
% b, a: numerator and denominator polynomials
% K: number of frequency points
% theta: frequency interval (2-element vector).
% Output parameters:
% dHmag: the partial derivative matrix, M by K, where M is the
%        number of coefficients in the realization
% S: the sensitivity bound, 1 by K.

Hangle = exp(-j*angle(frqresp(b,a,K,theta)));
if (typ == 'd'),
   [dH,sc] = dhdirect(b,a,K,theta);
elseif (typ == 'p'),
   [c,nsec,dsec] = tf2rpf(b,a);
   [dH,sc] = dhparal(nsec,dsec,c,K,theta);
elseif (typ == 'c'),
   c = b(1); v = roots(a); u = roots(b);
   [nsec,dsec] = pairpz(v,u);
   [dH,sc] = dhcascad(nsec,dsec,c,K,theta);
end
[M,junk] = size(dH);
dHmag = real(dH.*(ones(M,1)*Hangle));
S = sum(abs((sc*ones(1,K)).*dHmag));
```

Program 11.10 Partial derivatives of the frequency response of an IIR filter in direct realization with respect to the coefficients.

```
function [dH,sc] = dhdirect(b,a,K,theta);
% Synopsis: [dH,sc] = dhdirect(b,a,K,theta).
% Computes the derivatives of the magnitude response of an
% IIR filter in direct realization with respect to the
% parameters, and a scaling vector for the parameters.
% Input parameters:
% b, a: the numerator and denominator polynomials
% K: number of frequency points
% theta: frequency interval (2-element vector).
% Output parameters:
% dH: matrix of partial derivatives of |H(theta)|
% sc: a scaling vector.

dHn = []; dHd = []; scn = []; scd = [];
H = frqresp(b,a,K,theta);
for k = 0:length(b)-1,
   dHn = [dHn; frqresp([zeros(1,k),1],a,K,theta)];
end
for k = 1:length(a)-1,
   dHd = [dHd; -frqresp([zeros(1,k),1],a,K,theta).*H]; end
   scn = scale2(b)*ones(length(b),1);
   scd = scale2(a)*ones(length(a)-1,1);
   dH = [dHn; dHd]; sc = [scn; scd];
end
```

Program 11.11 Partial derivatives of the frequency response of an IIR filter in parallel realization with respect to the coefficients.

```
function [dH,sc] = dhparal(nsec,dsec,c,K,theta);
% Synopsis: [dH,sc] = dhparal(nsec,dsec,c,K,theta).
% Computes the derivatives of the magnitude response of an
% IIR filter in parallel realization with respect to the
% parameters, and a scaling vector for the parameters.
% Input parameters:
% nsec, dsec, c: parameters of the parallel realization
% K: number of frequency points
% theta: frequency interval (2-element vector).
% Output parameters:
% dH: matrix of partial derivatives of |H(theta)|
% sc: a scaling vector.

dHn = []; dHd = []; scn = []; scd = [];
[M,junk] = size(nsec);
for k = 1:M,
   if (dsec(k,3) == 0),
      [dHt,sct] = dhdirect(nsec(k,1),dsec(k,1:2),K,theta);
      dHn = [dHn; dHt(1,:)];  dHd = [dHd; dHt(2,:)];
      scn = [scn; sct(1)]; scd = [scd; sct(2)];
   else,
      [dHt,sct] = dhdirect(nsec(k,:),dsec(k,:),K,theta);
      dHn = [dHn; dHt(1:2,:)]; dHd = [dHd; dHt(3:4,:)];
      scn = [scn; sct(1)*ones(2,1)];
      scd = [scd; sct(2)*ones(2,1)];
   end
end
dH = [dHn; dHd; ones(1,K)]; sc = [scn; scd; scale2(c)];
```

Program 11.12 Partial derivatives of the frequency response of an IIR filter in cascade realization with respect to the coefficients.

```
function [dH,sc] = dhcascad(nsec,dsec,c,K,theta);
% Synopsis: [dH,sc] = cascad(nsec,dsec,c,K,theta).
% Computes the derivatives of the magnitude response of an
% IIR filter in cascade realization with respect to the
% parameters, and a scaling vector for the parameters.
% Input parameters:
% nsec, dsec, c: parameters of the cascade realization
% K: number of frequency points
% theta: frequency interval (2-element vector).
% Output parameters:
% dH: matrix of partial derivatives of |H(theta)|
% sc: a scaling vector.

dHn = []; dHd = []; scn = []; scd = [];
cntd = 0; cntn = 0;
[M,junk] = size(nsec); H = ones(1,K);
for k = 1:M,
   if (nsec(k,3) ~= 0 & abs(nsec(k,2)) ~= 2),
      Ht = frqresp(nsec(k,:),dsec(k,:),K,theta);
      [dHt,sct] = dhdirect(nsec(k,:),dsec(k,:),K,theta);
      H = Ht.*H;
      dHn = [dHn; dHt(2,:)./Ht]; cntn = cntn+1;
      dHd = [dHd; dHt(4:5,:)./(ones(2,1)*Ht)];
      cntd = cntd+2;
      scn = [scn; sct(2,1)];
      scd = [scd; sct(4:5,1)];
   end
end
dHn = c*(ones(cntn,1)*H).*dHn; dHd = c*(ones(cntd,1)*H).*dHd;
dH = [dHn; dHd; H]; sc = [scn; scd; scale2(c)];
```

Program 11.13 Full scale of a vector of coefficients in fixed-point filter implementation.

```
function s = scale2(a);
% Synopsis: s = scale2(a).
% Finds a power-of-2 full scale for the vector a.

s =  exp(log(2)*ceil(log(max(abs(a)))./log(2)));
```

Program 11.14 Sensitivity bound for the magnitude response of a linear-phase FIR filter to coefficient quantization.

```
function [dHmag,S] = sensfir(h,K,theta);
% Synopsis: [dHmag,S] = sensfir(h,K,theta).
% Computes the sensitivity bound for the magnitude response of
% a linear-phase FIR filter to coefficient quantization.
% Input parameters:
% h: vector of coefficients
% K: number of frequency points
% theta: frequency interval (2-element vector).
% Output parameters:
% dHmag: the partial derivative matrix, M by K, where M is the
%        number of coefficients in the realization
% S: the sensitivity bound, 1 by K.

Hangle = exp(-j*angle(frqresp(h,1,K,theta)));
N = length(h) - 1; dH = [];
if (sign(h(1))==sign(h(N+1))), pm = 1; else, pm = -1; end
for k = 0:floor((N-1)/2),
   dH = [dH; frqresp( ...
   [zeros(1,k),1,zeros(1,N-1-2*k),pm,zeros(1,k)],1,K,theta)];
end
if (rem(N,2) == 0),
   dH = [dH; frqresp([zeros(1,N/2),1,zeros(1,N/2)],1,K,theta)];
end
sc = scale2(h);
[M,junk] = size(dH);
dHmag = real(dH.*(ones(M,1)*Hangle));
S = sc*sum(abs(dHmag));
```

Program 11.15 Frequency response of a filter subject to coefficient quantization.

```
function H = qfrqresp(typ,B,b,a,K,theta);
% Synopsis: H = qfrqresp(typ,B,b,a,K,theta).
% Computes the frequency response of a filter subject
% to coefficient quantization.
% Input parameters:
% typ: 'd' for direct, 'p' for parallel, 'c' for cascade
% b, a: numerator and denominator polynomials
% K: number of frequency points
% theta: frequency interval (2-element vector).
% Output parameters:
% H: the frequency response.

if (typ == 'd'),
   scn = (2^(B-1))/scale2(b); b = (1/scn)*round(scn*b);
   scd = (2^(B-1))/scale2(a); a = (1/scd)*round(scd*a);
   H = frqresp(b,a,K,theta);
elseif (typ == 'p'),
   [c,nsec,dsec] = tf2rpf(b,a);
   sc = (2^(B-1))/scale2(c); c = (1/sc)*round(sc*c);
   [M,junk] = size(nsec); H = c;
   for k = 1:M,
      nt = nsec(k,:); dt = dsec(k,:);
      if (dt(3) == 0), dt = dt(1:2); nt = nt(1); end
      scn = (2^(B-1))/scale2(nt); nt = (1/scn)*round(scn*nt);
      scd = (2^(B-1))/scale2(dt); dt = (1/scd)*round(scd*dt);
      H = H + frqresp(nt,dt,K,theta);
   end
elseif (typ == 'c'),
   c = b(1); v = roots(a); u = roots(b);
   [nsec,dsec] = pairpz(v,u);
   sc = (2^(B-1))/scale2(c); c = (1/sc)*round(sc*c);
   [M,junk] = size(nsec); H = c;
   for k = 1:M,
      nt = nsec(k,:); dt = dsec(k,:);
      if (dt(3) == 0), dt = dt(1:2); nt = nt(1:2); end
      scn = (2^(B-1))/scale2(nt); nt = (1/scn)*round(scn*nt);
      scd = (2^(B-1))/scale2(dt); dt = (1/scd)*round(scd*dt);
      H = H.*frqresp(nt,dt,K,theta);
   end
end
```

Program 11.16 The four norms of a rational filter.

```
function [h1,H1,H2,Hinf] = filnorm(b,a);
% Synopsis: [h1,H1,H2,Hinf] = filnorm(b,a).
% Computes the four norms of a rational filter.
% Input parameters:
% b, a: the numerator and denominator polynomials.
% Output parameters:
% h1: sum of absolute values of the impulse response
% H1: integral of absolute value of frequency response
% H2: integral of magnitude-square of frequency response
% Hinf: maximum magnitude response.

[h,Z] = filter(b,a,[1,zeros(1,99)]);
h1 = sum(abs(h)); n = 100;  h1p = 0;
while((h1-h1p)/h1 > 0.00001),
   [h,Z] = filter(b,a,zeros(1,n),Z);
   h1p = h1; h1 = h1 + sum(abs(h)); n = 2*n;
end

H2 = sqrt(nsgain(b,a));

N = 2 .^ ceil(log(max(length(a),length(b))-1)/log(2));
N = max(16*N,512)+1; temp = abs(frqresp(b,a,N));
Hinf = max(temp);
temp = [1,kron(ones(1,(N-1)/2-1),[4,2]),4,1].*temp;
H1 = sum(temp)/(3*(N-1));
```

Program 11.17 Zero-input limit cycle simulation for a second-order filter.

```
function [flag,y] = lc2sim(qtype,when,rtype,apar,B,s0,n);
% Synopsis: [flag,y] = lc2sim(qtype,when,rtype,apar,B,s0,n).
% Zero-input limit cycle simulation for a second-order filter.
% Input parameters:
% qtype: 't': truncate, 'r': round, 'm': magnitude truncate
% when:  'b': quantize before summation, 'a': after
% rtype: 'd': direct realization, 'c': coupled realization
% apar:  [a1, a2] for direct, [alphar, alphai] for coupled
% B:     number of bits, s0:    initial state
% n:     maximum number of time points to simulate.
% Output parameters:
% flag:  0: no LC, 1: DC LC, 2: other LC
% y:     the output signal.

s = [quant(s0(1),qtype,B), quant(s0(2),qtype,B)]; sp = s;
apar(2) = quant(apar(2),'r',B);
if (abs(apar(1)) >= 1), apar(1) = 2*quant(apar(1)/2,'r',B);
else, apar(1) = quant(apar(1),'r',B); end
y = zeros(1,n); flag = 2;
for i = 1:n,
if (rtype == 'd'),
temp1 = -apar(1)*s(1); temp2 = -apar(2)*s(2); s(2) = s(1);
if (when == 'b'),
   s(1) = quant(temp1,qtype,B) + quant(temp2,qtype,B);
else, s(1) = quant(temp1+temp2,qtype,B); end; y(i) = s(1);
else,
temp1 = apar(1)*s(1); temp2 = apar(2)*s(2);
temp3 = -apar(2)*s(1); temp4 = apar(1)*s(2);
if (when == 'b'),
   s(1) = quant(temp1,qtype,B) + quant(temp2,qtype,B);
   s(2) = quant(temp3,qtype,B) + quant(temp4,qtype,B);
else,
   s(1) = quant(temp1+temp2,qtype,B);
   s(2) = quant(temp3+temp4,qtype,B);
end; y(i) = s(1);
end
if (s(1) == 0 & s(2) == 0),
   flag = 0; y = y(1:i); break; end
if (s(1) == sp(1) & s(2) == sp(2)),
   flag = 1; y = y(1:i); break; end
sp = s; end
```

Program 11.18 Quantization by rounding, truncation, or magnitude truncation.

```
function aq = quant(a,qtype,B);
% Synopsis: aq = quant(a,qtype,B).
% Quantizes a number.
% Input parameters:
% a: the input number, assumed a fraction.
% qtype: 't': truncation, 'r': rounding,
%        'm': magnitude truncation
% B:     number of bits

fs = 2^(B-1);
aq = a*fs;
if (qtype == 't'),
   aq = floor(aq)/fs;
elseif (qtype == 'r'),
   aq = round(aq)/fs;
elseif (qtype == 'm'),
   aq = (sign(aq)*floor(abs(aq)))/fs;
else
   error('Unrecognized qtype in QUANT')
end
```

Program 11.19 A driver program for `lc2sim`.

```
disp('Make sure qtype, when, rtype, a1, a2, B, M are defined');
r = roots([1,a1,a2]);
if (max(abs(r)) >= 1),
   disp('Input filter is unstable'); return, end
if (imag(r(1)) == 0),
   disp('Poles are real'); return, end
if (rtype == 'c'), apar = [real(r(1)),imag(r(1))];
else, apar = [a1,a2]; end
n = ceil(-2*B*log(2)/log(abs(r(1))));
for i = 1:M,
   s0 = rand(1,2)-0.5*ones(1,2);
   flag = lc2sim(qtype,when,rtype,apar,B,s0,n);
   if (flag == 1),
      disp('DC limit cycle exists!'); return
   elseif (flag == 2),
      disp('Non-DC limit cycle exists!'); return
   end
end
disp('Apparently limit cycle free!');
```

11.11 Problems

11.1 Figure 11.26 shows a realization of a digital filter.

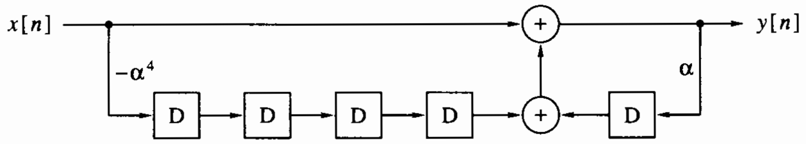

Figure 11.26 Pertaining to Problem 11.1.

(a) Find the impulse response of the filter.

(b) Draw a realization of the filter having the minimum possible number of delays.

(c) For what values of α will the filter have a linear phase (exact or generalized)?

(d) Compute the response of the filter to the input

$$x[n] = \begin{cases} 0, & n < 0, \\ 1, & n \geq 0. \end{cases}$$

11.2 Construct a realization of an Nth-order IIR filter using $2N$ delay elements, N for the input signal $x[n]$, and N for the output signal $y[n]$.

11.3 It is required to build a digital filter having the impulse response

$$h[n] = \begin{cases} \alpha^m, & n = mM, \; m \geq 0, \\ 0, & \text{otherwise,} \end{cases}$$

where M is a fixed positive integer, $M > 1$, and $|\alpha| < 1$. Construct a realization of the filter having a minimum number of delays. How many operations per time point are needed in this realization for a general input signal $x[n]$?

11.4 The filter

$$H^z(z) = 1 + z^{-1} + \cdots + z^{-(N-1)}$$

is called a *moving-average* filter, or a *boxcar* filter. It is an FIR filter whose coefficients are all 1. Direct realization of this filter requires $N - 1$ additions and no multiplications per time point. Derive a realization of this filter that requires only two additions and no multiplications per time point, but N delay elements. Draw a block diagram of this realization. Hint: The realization looks like IIR, but it is FIR.

11.5 Consider the digital system shown in Figure 11.27. Assume that the input of the top delay is fed with the constant positive number A at time $n = 0$ (which also causes $y[0]$ to be equal to A), and that of the bottom delay is fed with zero. At later times the system receives no input.

(a) Derive a closed-form mathematical expression for the output signal $y[n]$. Hint: If you work in the z domain, consult Table 7.1.

(b) Suggest a possible application of this system and specify the main advantage of generating $y[n]$ this way.

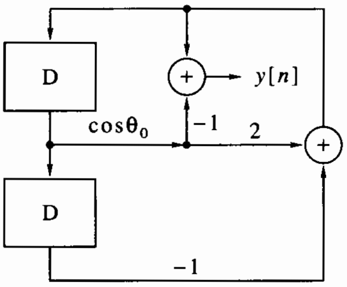

Figure 11.27 Pertaining to Problem 11.5.

11.6 It is required to design a tunable band-pass filter whose pass band is from $\theta_0 - \Delta\theta$ to $\theta_0 + \Delta\theta$, where $\Delta\theta$ is fixed and θ_0 is adjustable externally. The filter is to be type-I FIR, designed by the windowing method.

Show how to implement the filter using $N + 1$ multipliers, N adders, and $0.5N$ trigonometric function generators. The inputs to the filter are the input signal $x[n]$ and the frequency θ_0. Hint: Use the trigonometric identity

$$\sin\alpha - \sin\beta = 2\sin[0.5(\alpha - \beta)]\cos[0.5(\alpha + \beta)].$$

11.7 Let $\{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}\}$ be the matrices of a state-space realization, and $H^z(z)$ the corresponding transfer function. Prove that the transposed realization, whose matrices are $\{\boldsymbol{A}', \boldsymbol{C}', \boldsymbol{B}', \boldsymbol{D}\}$, has the same transfer function.

11.8 Let $\{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}\}$ be the matrices of a state-space realization, and $\{\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}}, \tilde{\boldsymbol{C}}, \tilde{\boldsymbol{D}}\}$ those of a similar realization [as defined in (11.33) and (11.34)]. Use either (11.41) or (11.48) and show that both realizations have the same transfer function.

11.9 This problem discusses state-space representation of a parallel realization.

(a) Let the two LTI systems $H_1^z(z)$, $H_2^z(z)$ be connected in parallel. Let $\{\boldsymbol{A}_i, \boldsymbol{B}_i, \boldsymbol{C}_i, \boldsymbol{D}_i,\ i = 1, 2\}$ be state-space representations of the two systems. Construct a state-space representation for the parallel connection.

(b) Suppose we wish to construct a state-state representation (11.74), (11.76) for the parallel realization of an IIR filter. We take $\{\boldsymbol{A}_l, \boldsymbol{B}_l, \boldsymbol{C}_l, \boldsymbol{D}_l, 1 \le l \le N_1 + N_2\}$ as the state-space matrices of the individual first- or second-order sections. What are the matrices $\boldsymbol{F}, \boldsymbol{G}, \boldsymbol{H}, \boldsymbol{K}$ needed in the construction discussed in Section 11.3?

11.10 This problem discusses state-space representation of a cascade realization.

(a) Repeat Problem 11.9(a) for two systems $H_1^z(z)$, $H_2^z(z)$ connected in cascade.

(b) Suppose we wish to construct a state-state representation for the cascade realization of an IIR filter. We take $\{\boldsymbol{A}_l, \boldsymbol{B}_l, \boldsymbol{C}_l, \boldsymbol{D}_l, 1 \le l \le \lceil 0.5N \rceil\}$ as the state-space matrices of the individual second-order sections. What are the matrices $\boldsymbol{F}, \boldsymbol{G}, \boldsymbol{H}, \boldsymbol{K}$ needed in the construction discussed in Section 11.3?

11.11 Write the state-space matrices of two different state-space representations of an FIR filter

$$H^z(z) = h[0] + h[1]z^{-1} + \cdots + h[N]z^{-N}.$$

11.12 We are given the state-space matrices

$$\boldsymbol{A} = \begin{bmatrix} 3 & -0.25 \\ 5 & 0 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{C} = \begin{bmatrix} 2 & -1 \end{bmatrix}, \quad \boldsymbol{D} = 0.$$

(a) Draw a block diagram of the state-space representation corresponding to these matrices.

(b) Write down the difference equation of the system.

(c) Compute the impulse response of the system.

(d) Suggest a realization having the same transfer function as the that of the given system, but simpler than the one you drew in part a.

11.13 The purpose of this problem is to derive a state-space version of the bilinear transform. The derivation is based on the trapezoidal integration interpretation of the bilinear transform, introduced in Problem 10.32.

(a) Suppose we have an analog filter expressed in the state-space form

$$\frac{d\boldsymbol{s}(t)}{dt} = \boldsymbol{A}\boldsymbol{s}(t) + \boldsymbol{B}x(t),$$
$$y(t) = \boldsymbol{C}\boldsymbol{s}(t) + \boldsymbol{D}x(t).$$

Use the trapezoidal integration derived in Problem 10.32 for the approximation

$$\boldsymbol{s}(nT + T) = \boldsymbol{s}(nT) + 0.5T[\boldsymbol{A}\boldsymbol{s}(nT + T) + \boldsymbol{B}x(nT + T) + \boldsymbol{A}\boldsymbol{s}(nT) + \boldsymbol{B}x(nT)],$$
$$y(nT) = \boldsymbol{C}\boldsymbol{s}(nT) + \boldsymbol{D}x(nT).$$

(b) Define

$$\tilde{\boldsymbol{A}} = (\boldsymbol{I}_N - 0.5T\boldsymbol{A})^{-1}(\boldsymbol{I}_N + 0.5T\boldsymbol{A}), \quad \tilde{\boldsymbol{B}} = 0.5T(\boldsymbol{I}_N - 0.5T\boldsymbol{A})^{-1}\boldsymbol{B}.$$

Also define the discrete-time, state-space vector as

$$\tilde{\boldsymbol{s}}(nT) = \boldsymbol{s}(nT) - \tilde{\boldsymbol{B}}x(nT).$$

Show that $x(nT), y(nT), \tilde{\boldsymbol{s}}(nT)$ have the discrete-time, state-space representation

$$\tilde{\boldsymbol{s}}(nT + T) = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{s}}(nT) + (\boldsymbol{I}_N + \tilde{\boldsymbol{A}})\tilde{\boldsymbol{B}}x(nT),$$
$$y(nT) = \boldsymbol{C}\tilde{\boldsymbol{s}}(nT) + (\boldsymbol{D} + \boldsymbol{C}\tilde{\boldsymbol{B}})x(nT).$$

(c) Write a MATLAB program `bilinss` that implements the bilinear transformation in state-space form. The inputs to the program are the numerator and denominator polynomials of the analog filter. The program should convert those polynomials to continuous-time, state-space form, using `tf2ss`; compute the discrete-time, state-space matrices, as found in part b; convert to a z-domain transfer function, using `ss2tf`.

11.14 In this problem we introduce *lattice* realizations of minimum-phase FIR filters, by way of a second-order example.

Let

$$a(z) = 1 + a_1 z^{-1} + a_2 z^{-2},$$

and define

$$\rho_1 = -\frac{a_1}{1 + a_2}, \quad \rho_2 = -a_2.$$

(a) Show that the roots of $a(z)$ are inside the unit circle if and only if $|\rho_1| < 1$ and $|\rho_2| < 1$. Hint: Recall Example 7.2.

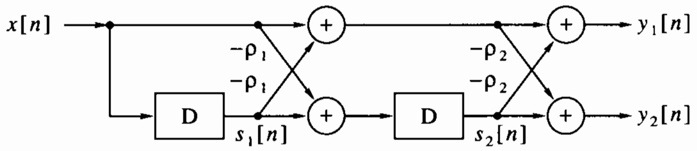

Figure 11.28 A second-order FIR lattice.

(b) Write down the state equations of the realization shown in Figure 11.28. Take the state vector as shown in the figure.

(c) Use the state equations you have found in part b and show that the transfer function from $x[n]$ to $y_1[n]$ is

$$\frac{Y_1^z(z)}{X^z(z)} = a(z).$$

(d) Find the transfer function from $x[n]$ to $y_2[n]$, and relate it to $a(z)$.

The structure in Figure 11.28 is called an *FIR lattice.* In Section 13.4.4 we will extend this structure to a minimum-phase FIR filter of any order.

11.15 In this problem we introduce a lattice realization of an all-pole IIR filter (i.e., an IIR filter that has no zeros), again by way of a second-order example. Let $a(z), \rho_1, \rho_2$ be as in Problem 11.14.

(a) Write down the state equations of the realization shown in Figure 11.29. Take the state vector as shown in the figure. Certain elements of the matrix A require special care; account for all interconnections. Obtaining the correct result in part b will serve as a verification of the correctness of your solution to this part.

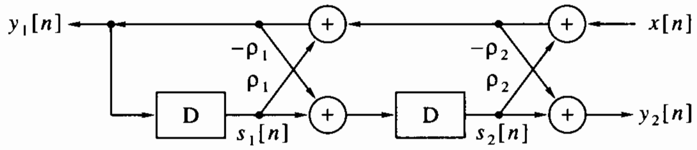

Figure 11.29 A second-order all-pole IIR lattice.

(b) Use the state equations you have found in part b and show that the transfer function from $x[n]$ to $y_1[n]$ is

$$\frac{Y_1^z(z)}{X^z(z)} = \frac{1}{a(z)}.$$

(c) Find the transfer function from $x[n]$ to $y_2[n]$. What special property does this transfer function have? Hint: Recall Problem 7.29.

The structure in Figure 11.29 is called an *all-pole IIR lattice.* In Section 13.4.4 we will extend this structure to an all-pole IIR filter of any order.

11.16 This problem explores a realization of a second-order section of an IIR filter based on a lattice structure. It can serve as an alternative to the coupled realization introduced in Section 11.1.7. The realization is shown in Figure 11.30. It is obtained from the all-pole IIR lattice of Figure 11.29 by adding a chain of multipliers/adders to form a new output $y[n]$.

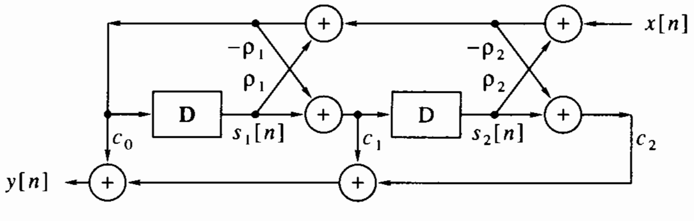

Figure 11.30 A second-order pole-zero IIR lattice.

(a) Find the transfer function from $x[n]$ to $y[n]$ in Figure 11.30.

(b) Suppose we wish to realize the second-order transfer function

$$H^{z}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}.$$

We already know from Problem 11.15 how to choose ρ_1, ρ_2 to get the required denominator. Compute the coefficients c_0, c_1, c_2 to get the required numerator.

The structure in Figure 11.30 is called a *pole-zero IIR lattice.* It can be extended to filters $b(z)/a(z)$ of any order, see Oppenheim and Schafer [1989] for details. Here, however, we are interested in it only as a candidate for a second-order section in a cascade realization.

11.17* The purpose of this problem is to derive the sensitivity of a root of a polynomial to small perturbations in the coefficients of the polynomial. Let

$$a(z) = \sum_{k=0}^{p} a_k z^{-k} = \prod_{i=1}^{p}(1 - \alpha_i z^{-1}).$$

Suppose that all roots α_k are distinct.

(a) Find $\partial a(z)/\partial a_k$ from the sum representation of $a(z)$, and $\partial a(z)/\partial \alpha_l$ from the product representation.

(b) Use the result of part a and show that

$$\left.\frac{\partial a(z)}{\partial \alpha_l}\right|_{z=\alpha_m} = \begin{cases} -\alpha_m^{-1} \prod_{i \neq m}(1 - \alpha_i \alpha_m^{-1}), & l = m, \\ 0, & l \neq m. \end{cases}$$

(c) Substitute $z = \alpha_m$ in the chain rule

$$\frac{\partial a(z)}{\partial a_k} = \sum_{l=1}^{p} \frac{\partial a(z)}{\partial \alpha_l} \frac{\partial \alpha_l}{\partial a_k}$$

and use part b for deriving an expression for $\partial \alpha_m / \partial a_k$.

(d) Assuming that the differences $\hat{a}_k - a_k$ are small, show that the deviation in the pole α_m is given by the approximate formula (11.77).

11.18* Let $H^{z}(z) = b(z)/a(z)$.

(a) If p_k is a coefficient appearing only in $a(z)$, show that

$$\frac{\partial |H^{f}(\theta)|}{\partial p_k} = -|H^{f}(\theta)| \Re \left\{ \frac{1}{a(e^{j\theta})} \cdot \frac{\partial a(e^{j\theta})}{\partial p_k} \right\}.$$

(b) If p_k is a coefficient appearing only in $b(z)$, show that

$$\frac{\partial |H^{f}(\theta)|}{\partial p_k} = |H^{f}(\theta)| \Re \left\{ \frac{1}{b(e^{j\theta})} \cdot \frac{\partial b(e^{j\theta})}{\partial p_k} \right\}.$$

11.19* Recall that the coupled realization of a second-order section, as presented in Section 11.1.7, as a denominator polynomial

$$a(z) = 1 - 2\alpha_r z^{-1} + (\alpha_r^2 + \alpha_i^2)z^{-2},$$

where α_r, α_i are the real and imaginary parts of the complex pole. The purpose of this problem is to compare the sensitivity of $|H^f(\theta)|$ to the parameters α_r, α_i with the sensitivity to a_1, a_2, in a direct realization of a second-order section.

(a) Compute $\partial a(e^{j\theta})/\partial\alpha_r$ and $\partial a(e^{j\theta})/\partial\alpha_i$.

(b) By Problem 11.18, it is sufficient to compare the two realizations by examining the corresponding values of $\Re\{[1/a(e^{j\theta})][\partial a(e^{j\theta})/\partial p_k]\}$. Derive these expressions for $p_k = a_1$, $p_k = a_2$ in a direct realization, and for $p_k = \alpha_r$, $p_k = \alpha_i$ in the coupled realization.

(c) Compute the four expressions in part b for $\theta = 0$. Assuming that the poles of the second-order section are in the vicinity of $z = 1$, draw your conclusions about the relative sensitivity of the two realizations to coefficient quantization at low frequencies.

11.20* Explain why coefficient quantization in a linear-phase FIR filter preserves the linear-phase property.

11.21* Discuss potential finite word length effects in the realization you have obtained in Problem 11.4.

11.22* Write a MATLAB procedure `qtf` that converts a transfer function to either parallel or cascade realization, then scales and quantizes the coefficients to a desired number of bits. The calling syntax of the function should be

```
[c,nsec,dsec,sc,sn,sd] = qtf(b,a,typ,B);
```

The input parameters are as follows:

- `b`, `a`: the numerator and denominator polynomial coefficients,
- `typ`: 'c' for cascade, 'p' for parallel,
- `B`: number of bits.

The output parameters are as follows:

- `c`: the constant coefficient,
- `nsec`: matrix of numerators of second-order sections,
- `dsec`: matrix of denominators of second-order sections,
- `sc`: scale factor for `c`,
- `sn`: scale factors for numerators,
- `sd`: scale factors for denominators.

The procedure should use the procedures `tf2rpf`, `scale2`, and `pairpz`, described in this chapter.

11.23* Use MATLAB for computing the possible pole locations of a second-order pole-zero lattice filter, assuming that the parameters ρ_1, ρ_2 are quantized to $B = 5$ bits. Note that these parameters can assume values only in the range $(-1, 1)$. Draw a diagram in the style of Figures 11.14 and 11.15, and interpret the result.

11.24* Consider the digital system described in Problem 11.5 and shown in Figure 11.27. Assume the same initial conditions and input as in Problem 11.5.

(a) Discuss possible problems resulting from (i) quantization of $\cos\theta_0$ to a finite word length and (ii) quantization of the multiplier's output.

(b) Simulate the system in MATLAB, using 10-bit fixed-point arithmetic with truncation. Use the function `quant` for this purpose. Take $A = 0.875$. For θ_0 examine two cases: one such that $\cos\theta_0 = 0.9375$, and one such that $\cos\theta_0 = 0.9375 + 2^{-10}$.

(c) Let the simulated system run for $0 \le n \le 5000$ and store the output signal $y[n]$. Compute the theoretical waveform $y[n]$ as found in part a. Plot the error between the theoretical waveform and the one obtained from the simulation. Repeat for the two values of θ_0 specified in part b. Report your results and conclusions.

11.25* Derive (11.127) from (11.126). Hint: In general,

$$\text{round}\{x\} = m \quad \text{if and only if} \quad m - 0.5 \le x \le m + 0.5.$$

11.26* Modify the scheme suggested in item 4 of Section 11.9.2 to the case of a second-order filter whose complex poles are near $z = -1$.