# 8 DIGITAL FILTER IMPLEMENTATION

## *8.1 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 precission. And arithmetic operations are subjects to errors, since a computer word has only a finite number of bits. The operation of representing a number to a fixed precission (that is, by a fixed number of bits) is called quantization. Consider, for example, the digital filter

$$y(n) = -a(1)y(n-1) + b(0)x(n) + b(1)x(n-1) \tag{8.1}$$

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 know, A/D conversion gives rise to quantization errors, determining 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 (LSB) 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 $y(n)$, 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 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 to overflow.

6. If the output signal $y(n)$ is to be fed to the 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.

## *8.2 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 (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, many microprocessors designed for DSP applications have word lengths from 16 bits (about 4 decimal digits) and up to 24 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.

### 8.2.1 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 LSB of the computer, multiplied by the full-scale value of the parameter. For example, consider a second-order IIR filter

$$H(z) = \frac{b(0) + b(1)z^{-1} + b(2)z^{-2}}{1 + a(1)z^{-1} + a(2)z^{-2}} \tag{8.2}$$

Since we are dealing only with stable filters, we know that necessarily $|a(1)| < 2$ and $|a(2)| < 1$. Suppose we represent each coefficient by $B$ bits, including sign. Then, assuming we scale $a(1)$ so that the largest representable number is $\pm 2$, 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 $\pm 1$, the LSB for $a(1)$ will be $2^{-(B-1)}$, and the error $|\hat{a}(1) - a(1)|$ 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 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, since this will lead to important qualitative conclusions. Consider, for example, the poles of a second-order filter, given by

$$\alpha_{1,2} = -0.5\hat{a}(1) \pm \sqrt{\left(\hat{a}(2) - 0.25\hat{a}(1)^2\right)} \tag{8.3}$$

Since $\{\hat{a}(1), \hat{a}(2)\}$ can assume only a finite number of values each (equal to $2^B$), the poles $\alpha_{1,2}$ 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 1, in the case $B = 5$.

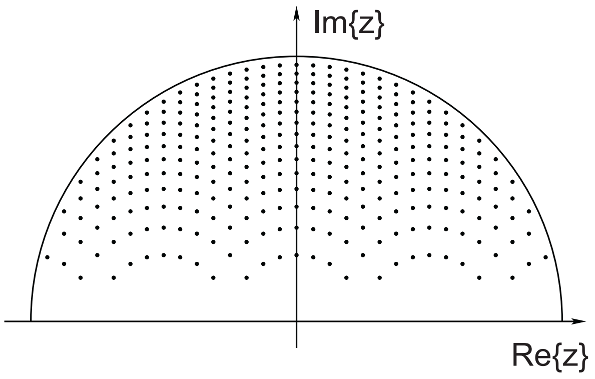

*Figure 1 Possible locations of complex stable poles of second-order digital filter in direct realization, number of bits $B = 5$*

As we see from Figure 1, the permissible locations of complex poles of the quantized filter are not distributed uniformly inside the unit circle. In particular, small 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, 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. 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 $\omega$ 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.

There is another realization of second –order section called coupled realization shown in Figure 2.

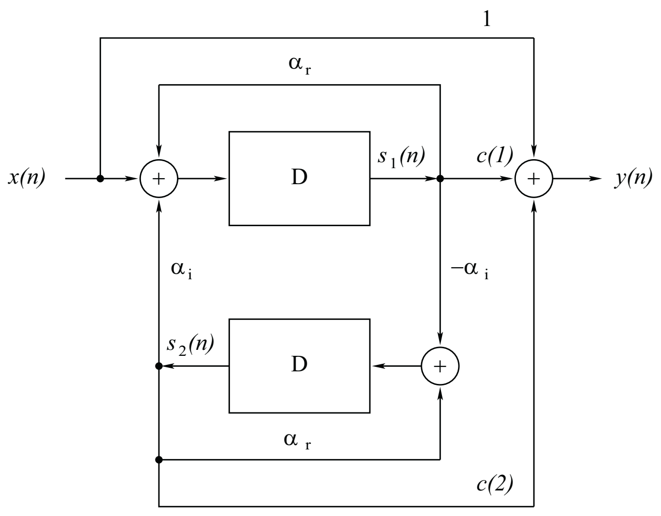

*Figure 2 A coupled second-order section for cascade realization*

The parameters $\alpha_r, \alpha_i$ are the real and imaginary parts of the complex pole of the second-order section that is

$$\alpha_r = \mathrm{Re}(\alpha), \quad \alpha_r^2 + \alpha_i^2 = |\alpha|^2 \tag{8.4}$$

$s_1(n)$, $s_2(n)$ are state-space variables and $c(1)$, $c(2)$ are coefficients necessary to obtain the right numerator of transfer function (8.2) (for $b(0)=1$). This realization is parametrized directly in the 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 3 for $B=5$. As we see, the density of permissible pole locations near $z=\pm 1$ is higher in this case.

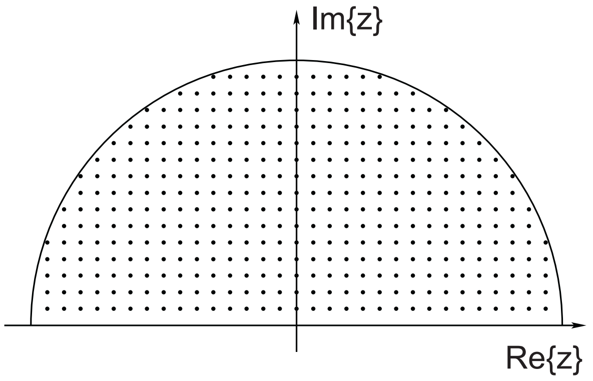

*Figure 3 Possible locations of complex stable poles of a second-order digital filter in coupled form, number of bits $B=5$*

The sensitivity of the poles of a digital IIR filter to denominator coefficient quantization usually increases with the filter order.

The effect of quantization of the numerator coefficients may seems, at first glance, to be the same as that of the denominator coefficients. This, however, is not the case. As we know, 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 $z = e^{\pm j\xi}$. In the former case, the coefficients are trivial ($\pm 1$ or $\pm 2$), so the problem of coefficient quantization does not arise. In the latter case, the numerator has the form $\left(1 - 2\cos(\xi) z^{-1} + z^{-2}\right)$ (up to a constant gain), so only the coefficient $2\cos(\xi)$ 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.

## 8.2.2 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 $H(j\omega)$ depends on a set of real parameters $\{x_k, 1 \le k \le K\}$. 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 $\{h(i), 0 \le i \le \lfloor 0.5N \rfloor\}$.

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

$$\left|\hat{H}(j\omega)\right| - \left|H(j\omega)\right| \approx \sum_{k=1}^{K} \frac{\partial \left|H(j\omega)\right|}{\partial x_k} (\hat{x}_k - x_k) \tag{8.5}$$

The partial derivatives on the right side of (8.5) are given by

$$\frac{\partial \left|H(j\omega)\right|}{\partial x_k} = \operatorname{Re}\left[\frac{\partial H(j\omega)}{\partial x_k} e^{-j\phi(\omega)}\right] \tag{8.6}$$

where $\phi(\omega)$ is the phase response of the filter. The approximate error formula (8.5) can be used for estimating the effect of coefficient quantization on the magnitude response in the following manner. Let $X_k$ denote the full scale used for the coefficient $x_k$. $X_k$ must be at least as large as $|x_k|$, 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

$$\left|\hat{x}_k - x_k\right| \le X_k Q \tag{8.7}$$

Substitution in (8.5) leads to the approximate inequality

$$\left\|\hat{H}(j\omega) - H(j\omega)\right\| \le Q \sum_{k=1}^{K} \left|\frac{\partial \left|H(j\omega)\right|}{\partial x_k}\right| x_k \tag{8.8}$$

The function

$$S(\omega) = \sum_{k=1}^{K} \left| \frac{\partial \left| H(j\omega) \right|}{\partial x_k} \right| x_k \tag{8.9}$$

is the **sensitivity bound** of the filter. As we see, the sensitivity bound depends on the realization. The realization determines the coefficients $x_k$, 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

$$\left\| \hat{H}(j\omega) - H(j\omega) \right\| \le 2^{-B} S(\omega) \tag{8.10}$$

A common way of choosing the full-scale values $X_k$ 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

$$X_{num} = MAX\left[ abs\left(b(i)\right), 0 \le i \le N \right] \tag{8.11}$$

$$X_{den} = MAX\left[ abs\left(a(i)\right), 1 \le i \le N \right] \tag{8.12}$$

This way, the coefficient of largest magnitude in each polynomial is represented by $\pm 1$. 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

$$X_{num} = 2^{\left\lceil \log_2\left( MAX\left[ abs\left(b(i)\right), 0 \le i \le N \right] \right) \right\rceil} \tag{8.13}$$

$$X_{den} = 2^{\left\lceil \log_2\left( MAX\left[ abs\left(a(i)\right), 1 \le i \le N \right] \right) \right\rceil} \tag{8.14}$$

This method yields larger $X_{num}$ and $X_{den}$ than the ones in (8.11) and (8.12), 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 $\delta_p$ be the minimum tolerance of all pass bands of the filter, and $\delta_s$ 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, (8.10) leads to

$$B \ge \log_2 \left( \frac{\max\left\{ S(\omega) : \omega \text{ in a pass band} \right\}}{0.1 \delta_{\mathrm{p}}} \right) \tag{8.15}$$

$$B \geq \log_2 \left( \frac{\max\{S(\omega) : \omega \text{ in a stop band}\}}{0.1\delta_s} \right) \tag{8.16}$$

Therefore, choosing $B$ according to the greater of the two right sides in (8.15) and (8.16) 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.

Matlab functions in the Appendix 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 $\partial|H(j\omega)|/\partial x_k$ in the matrix *dHmag*, and the sensitivity bound $S(\omega)$ in the vector *S*. These are computed at $K$ frequency points, in the interval specified by the input variable *theta*. The procedure **dhdirect**, **dhparal**, **dhcascad** return the partial derivatives $\partial H(j\omega)/\partial x_k$ of the respective IIR realizations, and the full-scale values $X_k$ of the coefficients. The full-scale values are computed by **scale2** according to (8.13) and (8.14).

## *8.3 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 signals values that exceed the maximum representable number. A problem of 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 filter 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 started in mathematical terms as follows. Suppose we wish to prevent the magnitude of the output signal $|y(n)|$ upper limit $y_{\max}$. Since the input and output are related through a convolution

$$y(n) = \{x * h\}(n) \tag{8.17}$$

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 $cH(z)$ 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 are very common, so familiarity with scaling methods and the effect of scaling on the properties of the filter is expedient.

### 8.3.1 TIME-DOMAIN SCALING

Consider a linear, time invariant filter whose impulse response and transfer function are $h(n)$ and $H(z)$, 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

$$y(n)=\sum_{m=0}^{\infty} h(m)x(n-m) \tag{8.18}$$

Therefore,

$$|y(n)| \le \sum_{m=0}^{\infty} |h(m)| \cdot |x(n-m)| \le x_{\max} \sum_{m=0}^{\infty} |h(m)| = x_{\max} \|h\|_1 \tag{8.19}$$

where

$$\|h\|_1 = \sum_{m=0}^{\infty} |h(m)| \tag{8.20}$$

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**

$$c=\frac{y_{\max}}{x_{\max}\|h\|_1} \tag{8.21}$$

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

**Example 1**
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

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

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 scaling is equivalent 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 (8.21) potentially **involves loss of accuracy** of the input signal.

## 8.3.2 FREQUENCY-DOMAIN SCALING

The scale factor given in (8.21) is often over-conservative 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-norm bound:**
We have, by the inverse Fourier transform formula,

$$y(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(j\omega)X(j\omega)e^{i\omega n}d\omega \tag{8.22}$$

Therefore,

$$|y(n)| \le \frac{1}{2\pi}\int_{-\pi}^{\pi} |H(j\omega)|\cdot|X(j\omega)|d\omega \tag{8.23}$$

Define

$$\|X\|_\infty = \max_{\omega}|X(j\omega)|, \quad \|H\|_1 = \frac{1}{2\pi}\int_{-\pi}^{\pi} |H(j\omega)d\omega| \tag{8.24}$$

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

$$c = \frac{y_{\max}}{\|H\|_1\|X\|_\infty} \tag{8.25}$$

**Infinity-norm bound:**
By reversing the roles of $H$ and $X$ in (8.25), we get the scale factor

$$c = \frac{y_{\max}}{\|H\|_\infty\|X\|_1} \tag{8.26}$$

This scale factor is applicable in case $\|X\|_1$, 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 (8.26) guarantees that the amplitude of the sinusoid at the output of the filter not exceed $y_{\max}$.

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

$$\|H\|_2 = \sqrt{\frac{1}{2\pi}\int_{-\pi}^{\pi} |H(j\omega)|^2 \, d\omega} \tag{8.27}$$

Note that (8.27) is identical to the square root of the noise gain of the filter. Let us assume that the input $x(n)$ is a wide-sense stationary signal having power spectral density $K_x(j\omega)$. Then, we can bound the variance of the output signal as follows:

$$\gamma_y = \frac{1}{2\pi}\int_{-\pi}^{\pi} K_y(j\omega)\, d\omega = \frac{1}{2\pi}\int_{-\pi}^{\pi} K_x(j\omega)|H(j\omega)|^2 \le \|K_x\|_\infty \|H\|_2^2 \tag{8.28}$$

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

$$c = \sqrt{\frac{\gamma_{y,\max}}{\|H\|_2^2 \|K_x\|_\infty}} \tag{8.29}$$

Bounding the variance of $y(n)$ **does not** guarantee boundedness of $|y(n)|$, and this is why we refer to (8.29) as probabilistic scaling. If, for example, we take $\gamma_{y,\max} \approx 0.1 y_{\max}^2$, the probability of $|y(n)|$ exceeding $y_{\max}$ is expected to be low[1].

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

$$\|H\|_1 \le \|H\|_2 \le \|H\|_\infty \le \|h\|_1 \tag{8.30}$$

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

## 8.3.3 MATLAB IMPLEMENTATION OF FILTER NORMS

Exact computation of norms we have presented is not possible in general for rational filters (with the exception of $\|H\|_2$). However, numerical approximations can be used instead. High accuracy is not required, since the scale factor resulting from those norms is usually[2] rounded to an integer power of 2. The procedure **filnorm** in the Appendix 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

[1] For example, if $y(n)$ has Gaussian probability density, this probability is about 0.0016.

[2] For some realizations scaling factors can be absorbed into the filter coefficients. In this case the scaling factors need not to be limited to an integer power of 2.

terms $\left|h(n)\right|$ until the relative error drops below a prescribed threshold. The norm $\|H\|_2$ is computed by calling **nsgain** and taking the square root of the result. The norm $\|H\|_\infty$ is approximated by the maximum of the magnitude response, computed on a dense grid. Finally, the norm $\|H\|_1$ is computed by evaluating the magnitude response on a dense grid and approximating the integral by Simpson's rule.

### 8.3.4 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 signal result. For example, suppose that $x_1 + x_2 + x_3$ does not overflow, and the computation is performed in the 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 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 2**
In the direct realization shown in Figure 4 (chapter 7) 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

$$\frac{U(z)}{X(z)} = \frac{1}{A(z)}$$

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

In the transposed direct realization shown in Figure 7 (chapter 7), only the output signal $y(n)$ is of the fourth type. Any overflow generated at one of the adders along the central 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.

## 8.3.5 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 the last example, 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[3].

Figure 4 illustrates a parallel connection of three second-order sections (delays are representable 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 specified section, but different factors are permitted at different sections. The $f_m$ shown in Figure 4 are assumed to be scaled already.
2. The coefficients $\lambda_k$ are computed to make 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 $\lambda_1$ must be larger than $\lambda_2$ 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 $\lambda_k$ 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 $\lambda_k$ 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 3**

Consider parallel realization of IIR filter with coefficients quantized to 16 bits with

[3] For some hardware implementations it make sense to use other structures. E.g. standard DSP architectures are typically more suitable for noncanonical realization. This will be demonstrated in the last exercise "Hardware Implementation of Digital Filters".

$$H(z) = -24984\times 2^{-17} + \frac{8909\times 2^{-20} - 22869\times 2^{-20} z^{-1}}{1 - 30365\times 2^{-14} z^{-1} + 15710\times 2^{-14} z^{-2}} +$$
$$+\frac{-30199\times 2^{-17} + 28813\times 2^{-17} z^{-1}}{1 - 28072\times 2^{-14} z^{-1} + 13038\times 2^{-14} z^{-2}} + \frac{19504\times 2^{-16}}{1 - 25279\times 2^{-15} z^{-1}}$$

Suppose we know that the input signal satisfies $\|K_x\|_\infty = 0.1$. 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 fo each section must be no larger than 1 after scaling.

The 2-norms of the tree 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 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 taking

$$\lambda_0 = 2, \quad \lambda_1 = 0.5, \quad \lambda_2 = \lambda_3 = 1 \tag{8.31}$$

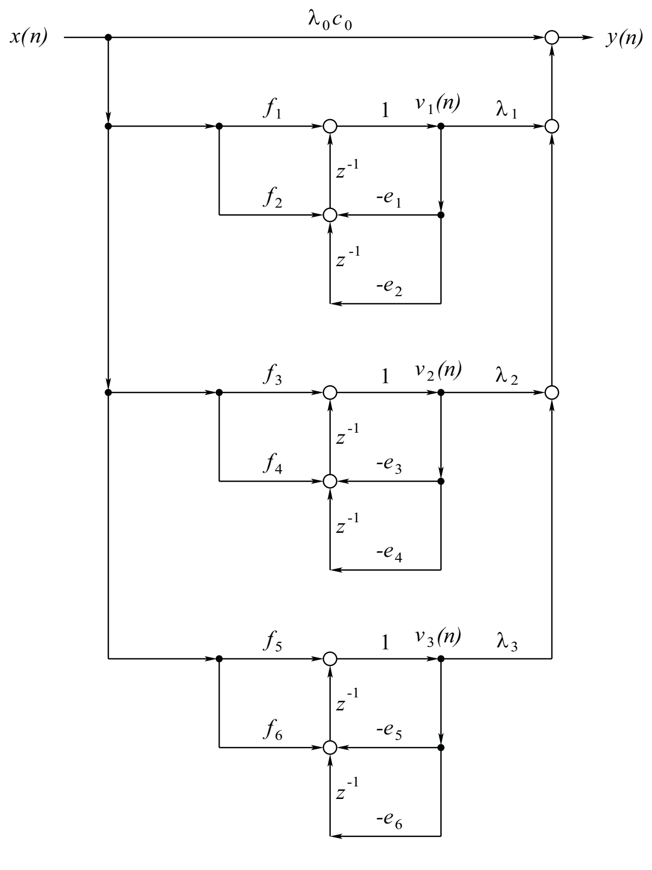

*Figure 4 Scaling for a parallel realization (second-order sections in a transposed direct realization).*

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 specified filter. Ordering rules have been proposed in the literature but are not discussed here. We use the order provided by the program **pairz**. 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 5 illustrates a cascade connection of three second-order sections. Let $H_k(z)$ 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 $\lambda_k$ to avoid overflow in the transfer function

$$\prod_{i=1}^{k} \lambda_i H_i(z)$$

At the *k*th stage, the factors $\{\lambda_i, 1 \le i \le k-1\}$ have already been determined, so only $\lambda_k$ 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 desired value $b(0)$. The final stage is therefore to decrease one of the $\lambda_k$ 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 $H(z)$ scaled by a power of 2, as in the case of parallel realization.

**Example 4**
We look again at the IIR filter discussed in Example 3. The cascade decomposition of the filter, with coefficients quantized to 16 bits, is

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

Assume, as in Example 3, that the input signal satisfies $\|K_x\|_\infty = 0.1$. 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:

$$\|H_1\|_2 = 1.8958, \quad \|H_1 H_2\|_2 = 1.8730, \quad \|H_1 H_2 H_3\|_2 = 11.5802$$

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 $\lambda_k$ to account for $b(0)$. We decide to decrease $\lambda_2$ from 1 to $29414 \times 2^{-15}$ $(\approx 0.8976)$. This gives an overall gain of $29414 \times 2^{-19}$, which is equal to $2b(0)$, quantized to 16 bits. In summary,

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

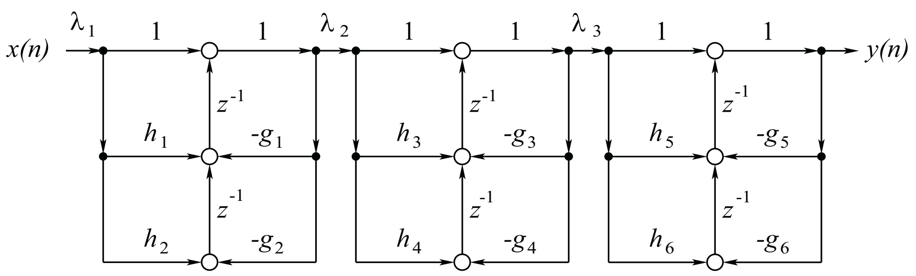

*Figure 5 Scaling for a cascade realization (second-order sections in a transposed direct realization).*

## *8.4 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. 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 mathematical formulas exactly. Quantization resulting from finite word length is a nonlinear operation, so stability properties of linear systems do not necessary 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 the 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. Oscilations 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 resulting signal may be audible.

Mathematical analysis of limit cycles is difficult except for first-order filters. We shall therefore omit most mathematical details, and restrict our discussion to some examples.

**Example 5**
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 $\omega_o=\pi$ and whose amplitude is 1/8.

**Example 6**
Next consider the same filter, but implemented with 4-bit truncation arithmetic. The sequence of values of the output signal will now be

$$y(n)=3/8,-1/4,1/8,-1/8,0,0,\ldots$$

As we see, truncation of $y(4)$ makes it zero, thus stopping the oscillations. This time there is no limit cycle.

**Example 7**
Consider the filter

$$y(n)=0.625y(n-1)+x(n)$$

with 4-bit rounding arithmetic. This time the output sequence is

$$y(n)=3/8,1/4,1/8,1/8,1/8,\ldots$$

As we see, the output reaches a constant nonzero value. This phenomenon is called **zero-frequency limit cycle**.

**Example 8**
For the same filter as in Example 7, but with truncation arithmetic and the same initial condition, $y(n)$ will reach zero as in Example 6.

**Example 9**
Consider the filter from Example 7, but with truncation arithmetic and initial condition $y(0)=-3/8$. Then

$$y(n)=-3/8,-1/4,-1/8,-1/8,-1/8,\ldots$$

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 in first-order filters depend on the filter in question, the quantization level, the method of quantization, and the initial conditions. 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 $\pi$. 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 on Figure 2 is less susceptible to limit cycles than a direct realization. The coupled realization is presented by the state-space equations. It is possible to 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 $\lfloor |a| \rfloor sign(a)$. This method is seldom implemented in hardware, so it usually requires special programming.
2. Magnitude truncation of each component of state $\mathbf{s}(n+1)$ is performed **after** both product (by coefficients $\alpha_r$ and $\alpha_i$) are formed and added (subtracted) in double precission.

# APPENDIX - MATLAB PROGRAMS

```
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));
```

```
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.

% Part of software package for the book:
% A Course in Digital Signal Processing
% by Boaz Porat, John Wiley & Sons, 1997

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
```

```
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)];
```

**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)];
```

**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)));
```

**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));
```

**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
```

**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));
```

# LITERATURE

[1] Porat, B: *A Course in Digital Signal Processing*. John Wiley & Sons, Inc. New York 1997, ISBN 0-471-14961-7.