CHAPTER 9

Sampling Rate Conversion

In many practical applications of digital signal processing, one is faced with the problem of changing the sampling rate of a signal, either increasing it or decreasing it by some amount. The process of converting a signal from a given rate to a different rate is called *sampling rate conversion*. In turn, systems that employ multiple sampling rates in the processing of digital signals are called *multirate digital signal processing systems*. In this chapter we describe sampling rate conversion and multirate signal processing in the digital domain.

As an example, consider the system shown in Figure 9.1 in which an analog signal $x_{\mathrm{a}}(t)$ is sampled using the sampling rate of $F_{\mathrm{s}} = \frac{1}{T}$ samples/second. The resulting digital signal $x(n)$ is subsequently filtered using a lowpass filter (LPF) with a cutoff frequency of ω_{c}.

Thus, the output signal $y(n)$ has all its energy in the band $0 \leq \omega \leq \omega_{\mathrm{c}} = 2\pi f_{\mathrm{c}}$. According to the sampling theorem, such a signal may be represented by the rate of $2f_{\mathrm{c}}/T$ samples/second instead of its existing rate of $F_{\mathrm{s}} = 1/T$. Note that $|f_c| \leq 0.5$. However, if $f_{\mathrm{c}} \ll 0.5$, then $2f_{\mathrm{c}}/T \ll F_{\mathrm{s}}$. Hence it would seem more advantageous to lower the sampling frequency to a value closer to $2f_{\mathrm{c}}/T$ and perform signal processing operations at this lower rate.

Other applications include the need for an optimal interpolation in computer tomography and efficient multistage designs of narrowband lowpass filters.

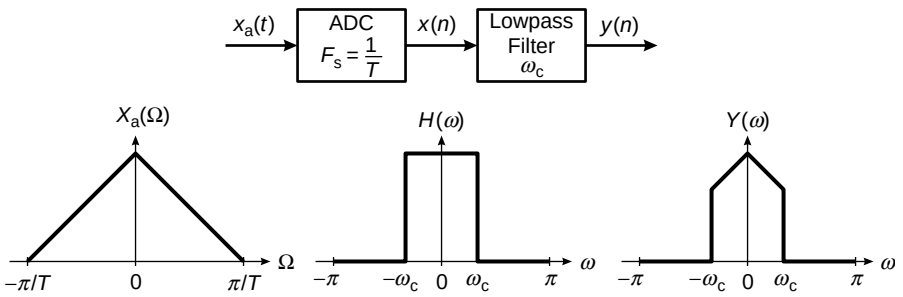


FIGURE 9.1 *A typical signal processing system*

9.1 INTRODUCTION

The idea of interpolation is a very familiar concept to most of us and has its origin in numerical analysis. Typically, interpolation is performed on a table of numbers representing a mathematical function. Such a table may be printed in a handbook or stored in a computer memory device. The interpolation, often simply linear (or straight line) approximation, creates an error called the *interpolation error*. The main difference between interpolation in digital signal processing and interpolation in numerical analysis is that we will assume that the given data is bandlimited to some band of frequencies and develop schemes that are optimal on this basis, whereas a numerical analyst typically assumes that the data consists of samples of polynomials (or very nearly so) and develops schemes to minimize the resulting error.

To motivate this concept of interpolation in signal processing, it is helpful to think of an underlying (or original) analog signal $x_a(t)$ that was sampled to produce the given discrete signal $x(n)$. If the $x_a(t)$ was sampled at the minimum required rate, then, according to the sampling theorem, it can be recovered completely from the samples $x(n)$. If we now sample this recovered analog signal, at say twice the old rate, we have succeeded in doubling the sampling rate or interpolating by a factor of 2 with zero interpolation error. Specifically, we have:

$$\text{Original discrete signal:} \quad x(n) = x_a(nT) \tag{9.1}$$

$$\text{Reconstructed analog signal:} \quad x_a(t) = \sum_k x_a(kT) \frac{\sin[\pi(t - kT)/T]}{\pi(t - kT)/T} \tag{9.2}$$

$$\text{Resampled analog signal:} \quad x_{\mathrm{a}}\left(m\frac{T}{2}\right) = \sum_k x_{\mathrm{a}}(kT)\frac{\sin\left[\pi\left(m\frac{T}{2}-kT\right)/T\right]}{\pi\left(m\frac{T}{2}-kT\right)/T}$$

$$= \sum_k x_{\mathrm{a}}(kT)\frac{\sin\left[\pi\left(\frac{m}{2}-k\right)\right]}{\pi\left(\frac{m}{2}-k\right)} \tag{9.3}$$

$$\text{resulting in } \textit{high-rate discrete signal}: \quad y(m)\stackrel{\triangle}{=}x_{\mathrm{a}}\left(m\frac{T}{2}\right) \tag{9.4}$$

In this formulation of ideal interpolation, the discrete signal was converted to the analog signal and then back to the discrete signal at twice the rate. In the subsequent sections we will study how to avoid this roundabout approach and perform sampling rate conversion completely in the digital domain.

The process of sampling rate conversion in the digital domain can be viewed as a linear filtering operation, as illustrated in Figure 9.2a. The input signal $x(n)$ is characterized by the sampling rate $F_x = 1/T_x$, and the output signal $y(m)$ is characterized by the sampling rate $F_y = 1/T_y$, where T_x and T_y are the corresponding sampling intervals. In our treatment, the ratio F_y/F_x is constrained to be rational

$$\frac{F_y}{F_x} = \frac{I}{D} \tag{9.5}$$

where D and I are relatively prime integers. We shall show that the linear filter is characterized by a time-variant impulse response, denoted

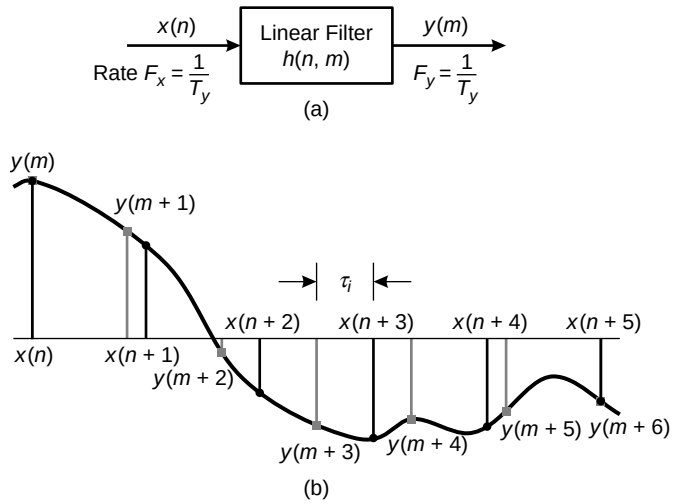


FIGURE 9.2 *Sampling rate conversion viewed as a linear filtering process*

as $h(n, m)$. Hence the input $x(n)$ and the output $y(m)$ are related by the superposition summation for time-variant systems.

The sampling rate conversion process can also be understood from the point of view of digital resampling of the same analog signal. Let $x_{\mathrm{a}}(t)$ be the analog signal that is sampled at the first rate F_x to generate $x(n)$. The goal of rate conversion is to obtain another sequence $y(m)$ directly from $x(n)$, which is equal to the sampled values of $x_{\mathrm{a}}(t)$ at a second rate F_y. As is depicted in Figure 9.2b, $y(m)$ is a time-shifted version of $x(n)$. Such a time shift can be realized by using a linear filter that has a flat magnitude response and a linear phase response (i.e., it has a frequency response of $\mathrm{e}^{-\jmath\omega\tau_i}$, where τ_i is the time delay generated by the filter). If the two sampling rates are not equal, the required amount of time shifting will vary from sample to sample, as shown in Figure 9.2b. Thus, the rate converter can be implemented using a set of linear filters that have the same flat magnitude response but generate different time delays.

Before considering the general case of sampling rate conversion, we shall consider two special cases. One is the case of sampling rate reduction by an integer factor D, and the second is the case of a sampling rate increase by an integer factor I. The process of reducing the sampling rate by a factor D (downsampling by D) is called *decimation*. The process of increasing the sampling rate by an integer factor I (upsampling by I) is called *interpolation*.

9.2 DECIMATION BY A FACTOR D

The basic operation required in decimation is the downsampling of the *high-rate* signal $x(n)$ into a *low-rate* signal $y(m)$. We will develop the time- and frequency-domain relationships between these two signals to understand the frequency-domain aliasing in $y(m)$. We will then study the condition needed for error-free decimation and the system structure required for its implementation.

9.2.1 THE DOWNSAMPLER

Note that the downsampled signal $y(m)$ is obtained by selecting one out of D samples of $x(n)$ and throwing away the other $(D-1)$ samples out of every D samples—i.e.,

$$y(m) = x(n)|_{n=mD} = x(mD); \quad n, m, D \in \{\text{integers}\} \tag{9.6}$$

The block diagram representation of (9.6) is shown in Figure 9.3. This downsampling element changes the rate of processing and thus is fundamentally different from other block diagram elements that we have used

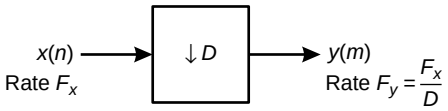


FIGURE 9.3 *A downsampling element*

previously. In fact, we can show that a system containing a downsampling element is shift varying. However, this fact does not prohibit the frequency-domain analysis of $y(m)$ in terms of $x(n)$ as we shall see later.

□ **EXAMPLE 9.1** Using $D = 2$ and $x(n) = \{\underset{\uparrow}{1}, 2, 3, 4, 3, 2, 1\}$ verify that the downsampler is time varying.

Solution The downsampled signal is $y(m) = \{\underset{\uparrow}{1}, 3, 3, 1\}$. If we now delay $x(n)$ by one sample, we get $x(n-1) = \{0, \underset{\uparrow}{1}, 2, 3, 4, 3, 2, 1\}$. The corresponding downsampled signal is $y_1(m) = \{\underset{\uparrow}{0}, 2, 4, 2\}$, which is different from $y(m-1)$. □

MATLAB Implementation MATLAB provides the function `[y] = downsample(x,D)` that downsamples input array `x` into output array `y` by keeping every `D`-th sample starting with the first sample. An optional third parameter "`phase`" specifies the sample offset which must be an integer between `0` and `(D-1)`. For example,

```
>> x = [1,2,3,4,3,2,1]; y = downsample(x,2)
y =
     1     3     3     1
```

downsamples by a factor of 2 starting with the first sample. However,

```
>> x = [1,2,3,4,3,2,1]; y = downsample(x,2,1)
y =
     2     4     2
```

produces an entirely different sequence by downsampling, starting with the second sample (i.e., offset by 1).

The frequency-domain representation of the downsampled signal $y(m)$ We now express $Y(\omega)$ in terms of $X(\omega)$ using z-transform relations. Toward this we introduce a high-rate sequence $\bar{x}(n)$, which is

given by

$$\bar{x}(n) \triangleq \begin{cases} x(n), & n = 0, \pm D, \pm 2D, \ldots \\ 0, & \text{elsewhere} \end{cases} \tag{9.7}$$

Clearly, $\bar{x}(n)$ can be viewed as a sequence obtained by multiplying $x(n)$ with a periodic train of impulses $p(n)$, with period D, as illustrated in Figure 9.4. The discrete Fourier series representation of $p(n)$ is

$$p(n) \triangleq \begin{cases} 1, & n = 0, \pm D, \pm 2D, \ldots \\ 0, & \text{elsewhere.} \end{cases} = \frac{1}{D} \sum_{\ell=0}^{D-1} e^{j\frac{2\pi}{D}\ell n} \tag{9.8}$$

Hence we can write

$$\bar{x}(n) = x(n)p(n) \tag{9.9}$$

and

$$y(m) = \bar{x}(mD) = x(mD)p(mD) = x(mD) \tag{9.10}$$

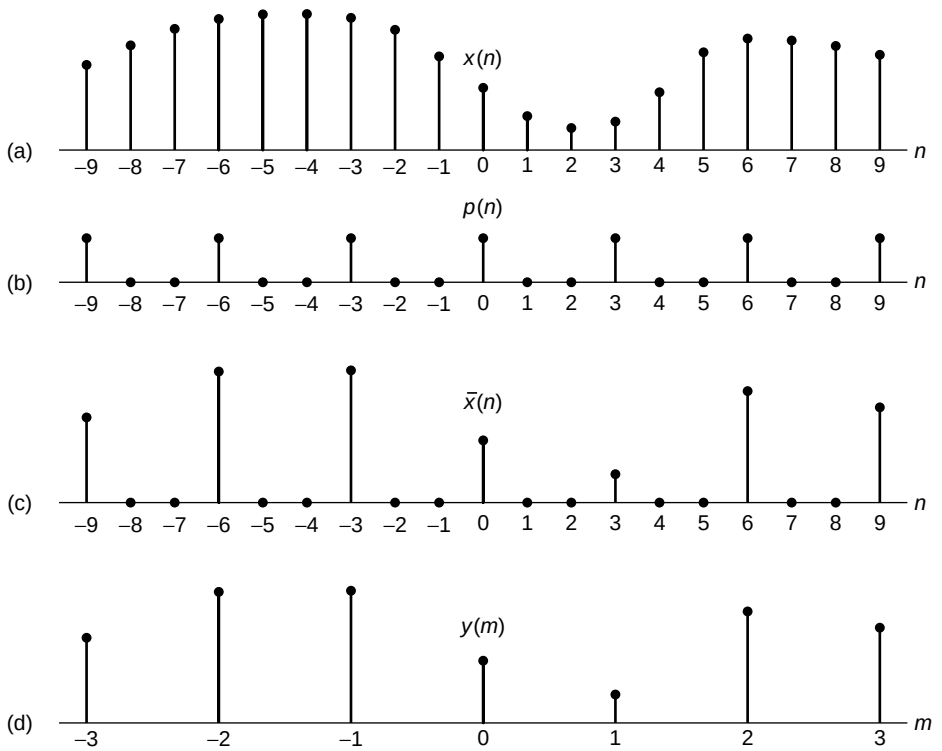


FIGURE 9.4 *Operation of downsampling: (a) original signal $x(n)$, (b) periodic impulse train $p(n)$ with period $D = 3$, (c) multiplication of $x(n)$ with $p(n)$, and (d) downsampled signal $y(n)$*

as shown in (9.6). Figure 9.4 shows an example of sequences $x(n)$, $\bar{x}(n)$, and $y(m)$ defined in (9.7)–(9.10).

Now the z-transform of the output sequence $y(m)$ is

$$\begin{aligned} Y(z) &= \sum_{m=-\infty}^{\infty} y(m)z^{-m} = \sum_{m=-\infty}^{\infty} \bar{x}(mD)z^{-m} \\ Y(z) &= \sum_{m=-\infty}^{\infty} \bar{x}(m)z^{-m/D} \end{aligned} \tag{9.11}$$

where the last step follows from the fact that $\bar{x}(m) = 0$, except at multiples of D. By making use of the relations in (9.7) and (9.8) in (9.11), we obtain

$$\begin{aligned} Y(z) &= \sum_{m=-\infty}^{\infty} x(m) \left[\frac{1}{D} \sum_{k=0}^{D-1} \mathrm{e}^{\jmath 2\pi mk/D} \right] z^{-m/D} \\ &= \frac{1}{D} \sum_{k=0}^{D-1} \sum_{m=-\infty}^{\infty} x(m) \left(\mathrm{e}^{-\jmath 2\pi k/D} z^{1/D} \right)^{-m} \\ &= \frac{1}{D} \sum_{k=0}^{D-1} X\left(\mathrm{e}^{-\jmath 2\pi k/D} z^{1/D} \right) \end{aligned} \tag{9.12}$$

The key steps in obtaining the z-transform representation (9.12), for the $(D \downarrow 1)$ downsampler, are as follows:

- the introduction of the high-rate sequence $\bar{x}(n)$, which has $(D-1)$ zeros in between the retained values $x(nD)$, and
- the impulse-train representation (9.8) for the periodic sampling series that relates $x(n)$ to $\bar{x}(n)$.

By evaluating $Y(z)$ on the unit circle, we obtain the spectrum of the output signal $y(m)$. Since the rate of $y(m)$ is $F_y = 1/Ty$, the frequency variable, which we denote as ω_y, is in radians and is relative to the sampling rate F_y,

$$\omega_y = \frac{2\pi F}{F_y} = 2\pi F T_y \tag{9.13}$$

Since the sampling rates are related by the expression

$$F_y = \frac{F_x}{D} \tag{9.14}$$

it follows that the frequency variables ω_y and

$$\omega_x = \frac{2\pi F}{F_x} 2\pi F T_x \tag{9.15}$$

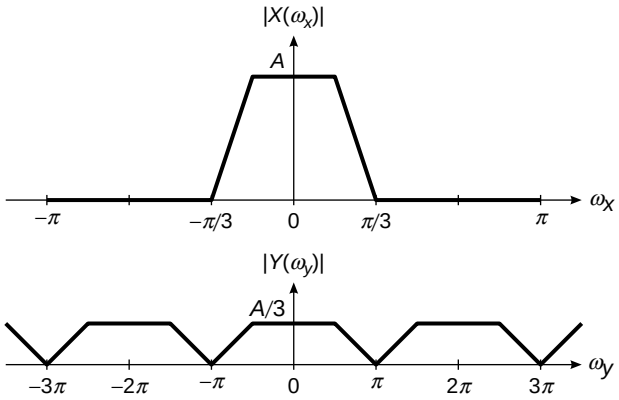


FIGURE 9.5 *Spectra of $x(n)$ and $y(m)$ in no-aliasing case*

are related by

$$\omega_y = D\omega_x \tag{9.16}$$

Thus, as expected, the frequency range $0 \leq |\omega_x| \leq \pi/D$ is stretched into the corresponding frequency range $0 \leq |\omega_y| \leq \pi$ by the downsampling process.

We conclude that the spectrum $Y(\omega_y)$, which is obtained by evaluating (9.12) on the unit circle, can be expressed as[1]

$$Y(\omega_y) = \frac{1}{D}\sum_{k=0}^{D-1} X\left(\frac{\omega_y - 2\pi k}{D}\right) \tag{9.17}$$

which is an aliased version of the spectrum $X(\omega_x)$ of $x(n)$. To avoid aliasing error, one needs the spectrum $X(\omega_x)$ to be less than full band or bandlimited (note that this bandlimitedness is in the digital frequency domain). In fact we must have

$$X(\omega_x) = 0 \quad \text{for} \quad \frac{\pi}{D} \leq |\omega_x| \leq \pi \tag{9.18}$$

Then,

$$Y(\omega_y) = \frac{1}{D}X\left(\frac{\omega_y}{D}\right), |\omega_y| \leq \pi \tag{9.19}$$

and no aliasing error is present. An example of this for $D = 3$ is shown in Figure 9.5.

[1]In this chapter, we will make a slight change in our notation for the DTFT. We will use $X(\omega)$ to denote the spectrum of $x(n)$ instead of the previously used notation $X(e^{j\omega})$. Although this change does conflict with the z-transform notation, the meaning should be clear from the context. This change is made for the sake of clarity and visibility of variables.

Comments:

1. The sampling theorem interpretation for (9.19) is that the sequence $x(n)$ was originally sampled at D times higher rate than required; therefore, downsampling by D simply reduces the effective sampling rate to the minimum required to prevent aliasing.
2. Equation (9.18) expresses the requirement for *zero decimation error* in the sense that no information is lost—i.e., there is no irreversible aliasing error in the frequency domain.
3. The argument $\frac{\omega_y}{D}$ occurs because in our notation ω is expressed in rad/sample. Thus the frequency of $y(m)$ expressed in terms of the higher-rate sequence $x(n)$ must be divided by D to account for the slower rate of $y(m)$.
4. Note that there is a factor $\frac{1}{D}$ in (9.19). This factor is required to make the inverse Fourier transform work out properly and is entirely consistent with the spectra of the sampled analog signals.

9.2.2 THE IDEAL DECIMATOR

In general, (9.18) will not be exactly true, and the $(D \downarrow 1)$ downsampler would cause irreversible aliasing error. To avoid aliasing, we must first reduce the bandwidth of $x(n)$ to $F_{x,\max} = F_x/2D$ or, equivalently, to $\omega_{x,\max} = \pi/D$. Then we may downsample by D and thus avoid aliasing.

The decimation process is illustrated in Figure 9.6. The input sequence $x(n)$ is passed through a lowpass filter, characterized by the impulse response $h(n)$ and a frequency response $H_{\mathrm{D}}(\omega_x)$, which ideally satisfies the condition

$$H_{\mathrm{D}}(\omega_x) = \begin{cases} 1, & |\omega_x| \le \pi/D \\ 0, & \text{otherwise} \end{cases} \tag{9.20}$$

Thus, the filter eliminates the spectrum of $X(\omega_x)$ in the range $\pi/D < \omega_x < \pi$. Of course, the implication is that only the frequency components of $x(n)$ in the range $|\omega_x| \le \pi/D$ are of interest in further processing of the signal.

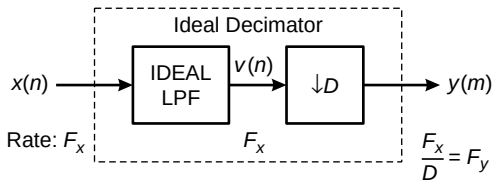


FIGURE 9.6 *Ideal decimation by a factor D*

The output of the filter is a sequence $v(n)$ given as

$$v(n) \stackrel{\triangle}{=} \sum_{k=0}^{\infty} h(k)x(n-k) \tag{9.21}$$

which is then downsampled by the factor D to produce $y(m)$. Thus,

$$y(m) = v(mD) = \sum_{k=0}^{\infty} h(k)x(mD-k) \tag{9.22}$$

Although the filtering operation on $x(n)$ is linear and time invariant, the downsampling operation in combination with the filtering results also in a time-variant system.

The frequency-domain characteristics of the output sequence $y(m)$ obtained through the filtered signal $v(n)$ can be determined by following the analysis steps given before—i.e., by relating the spectrum of $y(m)$ to the spectrum of the input sequence $x(n)$. Using these steps, we can show that

$$Y(z) = \frac{1}{D}\sum_{k=0}^{D-1} H\left(\mathrm{e}^{-\jmath 2\pi k/D} z^{1/D}\right) X\left(\mathrm{e}^{-\jmath 2\pi k/D} z^{1/D}\right) \tag{9.23}$$

or that

$$Y(\omega_y) = \frac{1}{D}\sum_{k=0}^{D-1} H\left(\frac{\omega_y - 2\pi k}{D}\right) X\left(\frac{\omega_y - 2\pi k}{D}\right) \tag{9.24}$$

With a properly designed filter $H_{\mathrm{D}}(\omega)$, the aliasing is eliminated and, consequently, all but the first term in (9.24) vanish. Hence,

$$Y(\omega_y) = \frac{1}{D} H_{\mathrm{D}}\left(\frac{\omega_y}{D}\right) X\left(\frac{\omega_y}{D}\right) = \frac{1}{D} X\left(\frac{\omega_y}{D}\right) \tag{9.25}$$

for $0 \leq |\omega_y| \leq \pi$. The spectra for the sequences $x(n)$, $h(n)$, $v(n)$, and $y(m)$ are illustrated in Figure 9.7.

MATLAB Implementation MATLAB provides the function `y = decimate(x,D)` that resamples the sequence in array `x` at `1/D` times the original sampling rate. The resulting resampled array `y` is `D` times shorter—i.e., `length(y) = length(x)/D`. An ideal lowpass filter given in (9.20) is not possible in the MATLAB implementation; however, fairly accurate approximations are used. The default lowpass filter used in the function is an 8th-order Chebyshev type-I lowpass filter with the cutoff frequency of $0.8\pi/D$. Using additional optional arguments, the filter order can be changed or an FIR filter of specified order and cutoff frequency can be used.

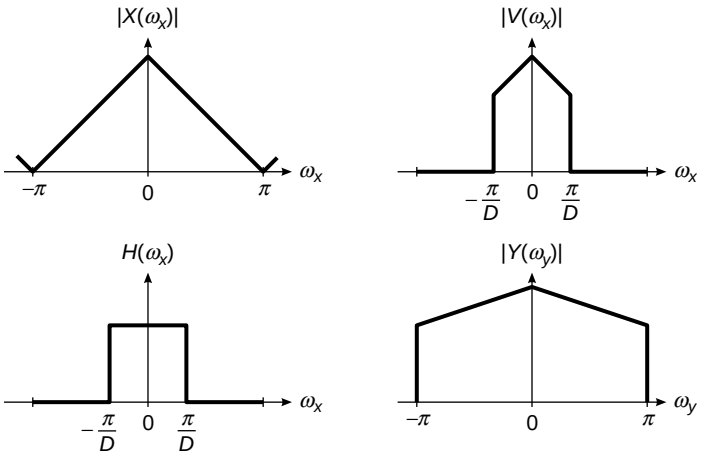


FIGURE 9.7 *Spectra of signals in the decimation of $x(n)$ by a factor D*

□ **EXAMPLE 9.2** Let $x(n) = \cos(0.125\pi n)$. Generate a large number of samples of $x(n)$ and decimate them using $D = 2$, 4, and 8 to show the results of decimation.

Solution

We will plot the middle segments of the signals to avoid end-effects due to the default lowpass filter in the `decimate` function. The following MATLAB script shows details of these operations, and Figure 9.7 shows the plots of the sequences.

```
n = 0:2048; k1 = 256; k2 = k1+32; m = 0:(k2-k1);
Hf1 = figure('units','inches','position',[1,1,6,4],...
    'paperunits','inches','paperposition',[0,0,6,4]);

% (a) Original signal
x = cos(0.125*pi*n); subplot(2,2,1);
Ha = stem(m,x(m+k1+1),'g','filled'); axis([-1,33,-1.1,1.1]);
set(Ha,'markersize',2); ylabel('Amplitude');
title('Original Sequence x(n)','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);

% (b) Decimation by D = 2
D = 2; y = decimate(x,D); subplot(2,2,2);
Hb = stem(m,y(m+k1/D+1),'c','filled');  axis([-1,33,-1.1,1.1]);
set(Hb,'markersize',2); ylabel('Amplitude');
title('Decimated by D = 2','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);
```

```
% (c) Decimation by D = 4
D = 4; y = decimate(x,D); subplot(2,2,3);
Hc = stem(m,y(m+k1/D+1),'r','filled');  axis([-1,33,-1.1,1.1]);
set(Hc,'markersize',2); ylabel('Amplitude');
title('Decimated by D = 4','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);
xlabel('n');

% (d) Decimation by D = 8
D = 8; y = decimate(x,D); subplot(2,2,4);
Hd = stem(m,y(m+k1/D+1),'m','filled');  axis([-1,33,-1.1,1.1]);
set(Hd,'markersize',2); ylabel('Amplitude');
title('Decimated by D = 8','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);
xlabel('n');
```

From Figure 9.8, we observe that the decimated sequences for $D = 2$ and $D = 4$ are correct and represent the original sinusoidal sequence $x(n)$ at lower sampling rates. However, the sequence for $D = 8$ is almost zero because the

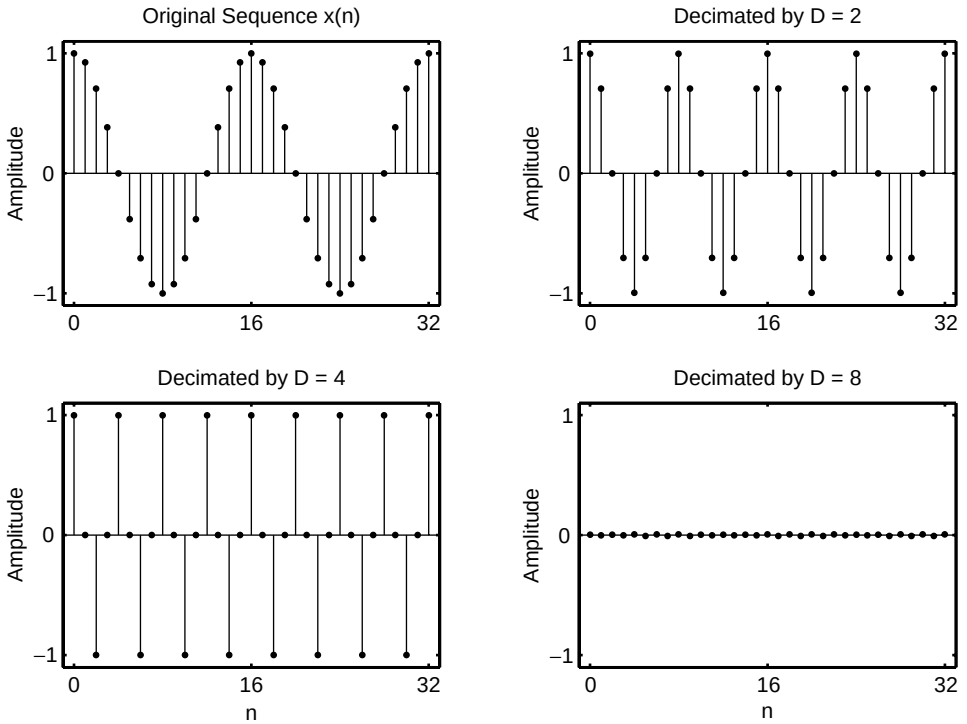


FIGURE 9.8 *Original and decimated signals in Example 9.2*

lowpass filter has attenuated $x(n)$ prior to downsampling. Recall that the cutoff frequency of the lowpass filter is set to $0.8\pi/D = 0.1\pi$ which eliminates $x(n)$. If we had used the downsampling operation on $x(n)$ instead of decimation, the resulting sequence would be $y(m) = 1$, which is an aliased signal. Thus, the lowpass filtering is necessary. □

9.3 INTERPOLATION BY A FACTOR *I*

An increase in the sampling rate by an integer factor of I—i.e., $F_y = IF_x$—can be accomplished by interpolating $I - 1$ new samples between successive values of the signal. The interpolation process can be accomplished in a variety of ways. We shall describe a process that preserves the spectral shape of the signal sequence $x(n)$. This process can be accomplished in two steps. The first step creates an intermediate signal at the high rate F_y by interlacing zeros in between nonzero samples in an operation called *upsampling*. In the second step, the intermediate signal is filtered to "fill in" zero-interlaced samples to create the interpolated high-rate signal. As before, we will first study the time- and frequency-domain characteristics of the upsampled signal and then introduce the interpolation system.

9.3.1 THE UPSAMPLER

Let $v(m)$ denote the intermediate sequence with a rate $F_y = IF_x$, which is obtained from $x(n)$ by adding $I - 1$ zeros between successive values of $x(n)$. Thus,

$$v(m) = \begin{cases} x(m/I), & m = 0, \pm I, \pm 2I, \ldots \\ 0, & \text{otherwise} \end{cases} \tag{9.26}$$

and its sampling rate is identical to the rate of $v(m)$. The block diagram of the upsampler is shown in Figure 9.9. Again, any system containing the upsampler is a time-varying system (Problem P9.1).

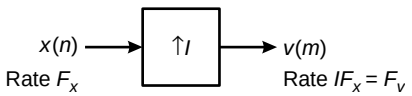


FIGURE 9.9 *An upsampling element*

□ **EXAMPLE 9.3** Let $I = 2$ and $x(n) = \{\underset{\uparrow}{1}, 2, 3, 4\}$. Verify that the upsampler is time varying.

Solution The upsampled signal is $v(m) = \{\underset{\uparrow}{1}, 0, 2, 0, 3, 0, 4, 0\}$. If we now delay $x(n)$ by one sample, we get $x(n-1) = \{0, \underset{\uparrow}{1}, 2, 3, 4\}$. The corresponding upsampled signal is $v_1(m) = \{\underset{\uparrow}{0}, 0, 1, 0, 2, 0, 3, 0, 4, 0\} = v(m-2)$ and not $v(m-1)$. □

MATLAB Implementation MATLAB provides the function `[v] = upsample(x,I)` that upsamples input array `x` into output `v` by inserting `(I-1)` zeros between input samples. An optional third parameter, "`phase`," specifies the sample offset, which must be an integer between `0` and `(I-1)`. For example,

```
>> x = [1,2,3,4]; v = upsample(x,3)
v =
     1     0     0     2     0     0     3     0     0     4     0     0
```

upsamples by a factor of 2 starting with the first sample. However,

```
>> v = upsample(x,3,1)
v =
     0     1     0     0     2     0     0     3     0     0     4     0
>> v = upsample(x,3,2)
v =
     0     0     1     0     0     2     0     0     3     0     0     4
```

produces two different signals by upsampling, starting with the second and the third sample (i.e., offset by 1), respectively. Note that the lengths of the upsampled signals are I times the length of the original signal.

The frequency-domain representation of the upsampled signal $y(m)$ The sequence $v(m)$ has a z-transform

$$V(z) = \sum_{m=-\infty}^{\infty} v(m)z^{-m} = \sum_{m=-\infty}^{\infty} v(m)z^{-mI} = X(z^I) \tag{9.27}$$

The corresponding spectrum of $v(m)$ is obtained by evaluating (9.27) on the unit circle. Thus

$$V(\omega_y) = X(\omega_y I) \tag{9.28}$$

where ω_y denotes the frequency variable relative to the new sampling rate F_y (i.e., $\omega_y = 2\pi F/F_y$). Now the relationship between sampling rates is $F_y = IF_x$, and hence the frequency variables ω_x and ω_y are related according to the formula

$$\omega_y = \frac{\omega_x}{I} \tag{9.29}$$

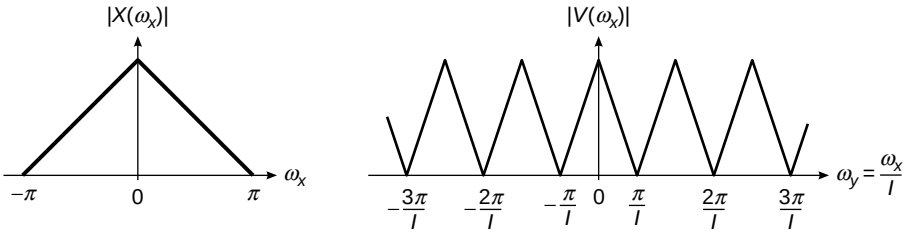


FIGURE 9.10 *Spectra of $x(n)$ and $v(m)$ where $V(\omega_y) = X(\omega_y I)$*

The spectra $X(\omega_x)$ and $V(\omega_y)$ are illustrated in Figure 9.10. We observe that the sampling rate increase, obtained by the addition of $I - 1$ zero samples between successive values of $x(n)$, results in a signal whose spectrum $V(\omega_y)$ is an I-fold periodic repetition of the input signal spectrum $X(\omega_x)$.

9.3.2 THE IDEAL INTERPOLATOR

Since only the frequency components of $x(n)$ in the range $0 \leq \omega_y \leq \pi/I$ are unique, the images of $X(\omega)$ above $\omega_y = \pi/I$ should be rejected by passing the sequence $v(m)$ through a lowpass filter with a frequency response $H_{\mathrm{I}}(\omega_y)$ that ideally has the characteristic

$$H_{\mathrm{I}}(\omega_y) = \begin{cases} C, & 0 \leq |\omega_y| \leq \pi/I \\ 0, & \text{otherwise} \end{cases} \tag{9.30}$$

where C is a scale factor required to properly normalize the output sequence $y(m)$. Consequently, the output spectrum is

$$Y(\omega_y) = \begin{cases} CX(\omega_y I), & 0 \leq |\omega_y| \leq \pi/I \\ 0, & \text{otherwise} \end{cases} \tag{9.31}$$

The scale factor C is selected so that the output $y(m) = x(m/I)$ for $m = 0, \pm I, \pm 2I, \ldots$. For mathematical convenience, we select the point $m = 0$. Thus,

$$y(0) = \frac{1}{2\pi} \int_{-\pi}^{\pi} Y(\omega_y) d\omega_y = \frac{C}{2\pi} \int_{-\pi/I}^{\pi/I} X(\omega_y I) d\omega_y \tag{9.32}$$

Since $\omega_y = \omega_x / I$, (9.32) can be expressed as

$$y(0) = \frac{C}{I} \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega_x) d\omega_x = \frac{C}{I} x(0) \tag{9.33}$$

therefore, $C = I$ is the desired normalization factor.

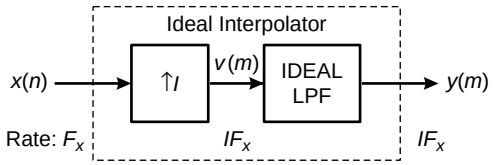


FIGURE 9.11 *Ideal interpolation by a factor I*

Finally, we indicate that the output sequence $y(m)$ can be expressed as a convolution of the sequence $v(n)$ with the unit sample response $h(n)$ of the lowpass filter. Thus

$$y(m) = \sum_{k=-\infty}^{\infty} h(m-k)v(k) \tag{9.34}$$

Since $v(k) = 0$ except at multiples of I, where $v(kI) = x(k)$, (9.34) becomes

$$y(m) = \sum_{k=-\infty}^{\infty} h(m-kI)x(k) \tag{9.35}$$

The ideal interpolator is shown in Figure 9.11.

MATLAB Implementation MATLAB provides the function `[y,h] = interp(x,I)` that resamples the signal in array `x` at `I` times the original sampling rate. The resulting resampled array `y` is `I` times longer—i.e., `length(y) = I*length(x)`. The ideal lowpass filter given in (9.30) is approximated by a symmetric filter impulse response, `h`, which is designed internally. It allows the original samples to pass through unchanged and interpolates between so that the mean square error between them and their ideal values is minimized. The third optional parameter `L` specifies the length of the symmetric filter as `2*L*I+1`, and the fourth optional parameter `cutoff` specifies the cutoff frequency of the input signal in π units. The default values are `L = 5` and `cutoff = 0.5`. Thus, if `I = 2`, then the length of the symmetric filter is 21 for the default `L = 5`.

□ **EXAMPLE 9.4** Let $x(n) = \cos(\pi n)$. Generate samples of $x(n)$ and interpolate them using $I = 2$, 4, and 8 to show the results of interpolation.

Solution We will plot the middle segments of the signals to avoid end-effects due to the default lowpass filter in the `interp` function. The following MATLAB script shows details of these operations, and Figure 9.12 shows the plots of the sequences.

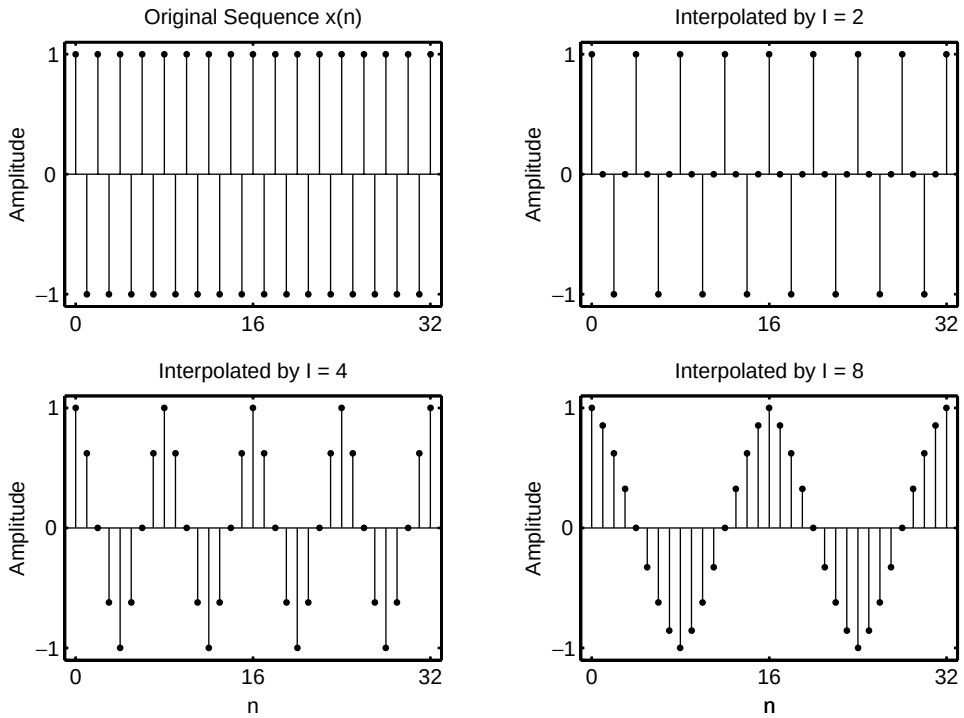


FIGURE 9.12 *Original and interpolated signals in Example 9.4*

```
n = 0:256; k1 = 64; k2 = k1+32; m = 0:(k2-k1);
Hf1 = figure('units','inches','position',[1,1,6,4],...
    'paperunits','inches','paperposition',[0,0,6,4]);

% (a) Original signal
x = cos(pi*n); subplot(2,2,1);
Ha = stem(m,x(m+k1+1),'g','filled'); axis([-1,33,-1.1,1.1]);
set(Ha,'markersize',2); ylabel('Amplitude');
title('Original Sequence x(n)','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);

% (b) Interpolation by I = 2
I = 2; y = interp(x,I); subplot(2,2,2);
Hb = stem(m,y(m+k1*I+1),'c','filled');  axis([-1,33,-1.1,1.1]);
set(Hb,'markersize',2); ylabel('Amplitude');
title('Interpolated by I = 2','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);
```

```
% (c) Interpolation by I = 4
I = 4; y = interp(x,I); subplot(2,2,3);
Hc = stem(m,y(m+k1*I+1),'r','filled');  axis([-1,33,-1.1,1.1]);
set(Hc,'markersize',2); ylabel('Amplitude');
title('Interpolated by I = 4','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);
xlabel('n');

% (d) Interpolation by I = 8
I = 8; y = interp(x,I); subplot(2,2,4);
Hd = stem(m,y(m+k1*I+1),'m','filled');  axis([-1,33,-1.1,1.1]);
set(Hd,'markersize',2); ylabel('Amplitude');
title('Interpolated by I = 8','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);
xlabel('n');
```

From Figure 9.11, we observe that the interpolated sequences for all three values of I are appropriate and represent the original sinusoidal signal $x(n)$ at higher sampling rates. In the case of $I = 8$, the resulting sequence does not appear to be perfectly sinusoidal in shape. This may be due the fact the lowpass filter is not close to an ideal filter. □

□ **EXAMPLE 9.5** Examine the frequency response of the lowpass filter used in the interpolation of the signal in Example 10.4.

Solution The second optional argument in the `interp` function provides the impulse response from which we can compute the frequency response, as shown in the following MATLAB script.

```
n = 0:256; x = cos(pi*n); w = [0:100]*pi/100;
Hf1 = figure('units','inches','position',[1,1,6,4],...
    'paperunits','inches','paperposition',[0,0,6,4]);

% (a) Interpolation by I = 2, L = 5;
I = 2; [y,h] = interp(x,I); H = freqz(h,1,w); H = abs(H);
subplot(2,2,1);  plot(w/pi,H,'g'); axis([0,1,0,I+0.1]); ylabel('Magnitude');
title('I = 2, L = 5','fontsize',TF);
set(gca,'xtick',[0,0.5,1]); set(gca,'ytick',[0:1:I]);

% (b) Interpolation by I = 4, L = 5;
I = 4; [y,h] = interp(x,I); H = freqz(h,1,w); H = abs(H);
subplot(2,2,2);  plot(w/pi,H,'g'); axis([0,1,0,I+0.2]); ylabel('Magnitude');
title('I = 4, L = 5','fontsize',TF);
set(gca,'xtick',[0,0.25,1]); set(gca,'ytick',[0:1:I]);
```

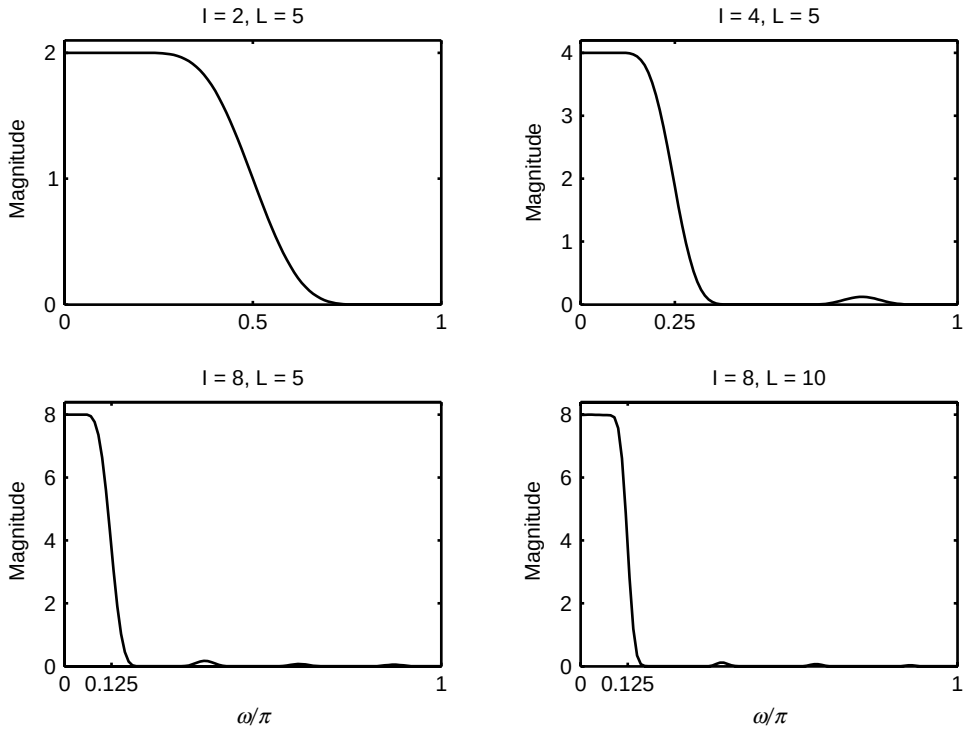


FIGURE 9.13 *Filter frequency responses in Example 9.5*

```
% (c) Interpolation by I = 8, L = 5;
I = 8; [y,h] = interp(x,I); H = freqz(h,1,w); H = abs(H);
subplot(2,2,3);  plot(w/pi,H,'g'); axis([0,1,0,I+0.4]); ylabel('Magnitude');
title('I = 8, L = 5','fontsize',TF); xlabel('\omega/\pi','fontsize',10)
set(gca,'xtick',[0,0.125,1]); set(gca,'ytick',[0:2:I]);

% (d) Interpolation by I = 8, L = 10;
I = 8; [y,h] = interp(x,I,10); H = freqz(h,1,w); H = abs(H);
subplot(2,2,4);  plot(w/pi,H,'g'); axis([0,1,0,I+0.4]); ylabel('Magnitude');
title('I = 8, L = 10','fontsize',TF); xlabel('\omega/\pi','fontsize',10)
set(gca,'xtick',[0,0.125,1]); set(gca,'ytick',[0:2:I]);
```

The frequency response plots are shown in Figure 9.13. The first three plots are for `L = 5` and, as expected, the filters are all lowpass with passband edges approximately around π/I frequencies and the gain of I. Also note that the filters do not have sharp transitions and thus are not good approximations to the ideal filter. The last plot shows the response for `L = 10`, which indicates a more sharp transition, which is to be expected. Any value beyond `L = 10` results in an unstable filter design and hence should be avoided. □

9.4 SAMPLING RATE CONVERSION BY A RATIONAL FACTOR I/D

Having discussed the special cases of decimation (downsampling by a factor D) and interpolation (upsampling by a factor I), we now consider the general case of sampling rate conversion by a rational factor I/D. Basically, we can achieve this sampling rate conversion by first performing interpolation by the factor I and then decimating the output of the interpolator by the factor D. In other words, a sampling rate conversion by the rational factor I/D is accomplished by cascading an interpolator with a decimator, as illustrated in Figure 9.14.

We emphasize that the importance of performing the interpolation first and the decimation second is to preserve the desired spectral characteristics of $x(n)$. Furthermore, with the cascade configuration illustrated in Figure 9.14, the two filters with impulse response $\{h_{\mathrm{u}}(k)\}$ and $\{h_{\mathrm{d}}(k)\}$ are operated at the same rate, namely IF_x, and hence can be combined into a single lowpass filter with impulse response $h(k)$, as illustrated in Figure 9.15. The frequency response $H(\omega_v)$ of the combined filter must incorporate the filtering operations for both interpolation and decimation, and hence it should ideally possess the frequency-response characteristic

$$H(\omega_v) = \begin{cases} I, \ 0 \le |\omega_v| \le \min(\pi/D, \pi/I) \\ 0, \ \text{otherwise} \end{cases} \tag{9.36}$$

where $\omega_v = 2\pi F/F_v = 2\pi F/IF_x = \omega_x/I$.

Explanation of (9.36) Note that $V(\omega_v)$ and hence $W(\omega_v)$ in Figure 9.15 are periodic with period $2\pi/I$. Thus, if

- $D < I$, then filter $H(\omega_v)$ allows a full period through and there is no net lowpass filtering.
- $D > I$, then filter must first truncate the fundamental period of $W(\omega_v)$ to avoid aliasing error in the $(D\downarrow 1)$ decimation stage to follow.

Putting these two observations together, we can state that when $D/I < 1$, we have net interpolation and no smoothing is required by

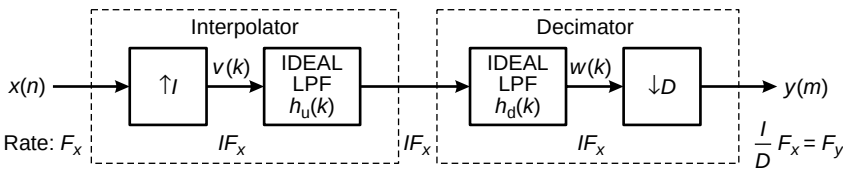


FIGURE 9.14 *Cascade of interpolator and decimator for sampling rate conversion by a factor I/D*

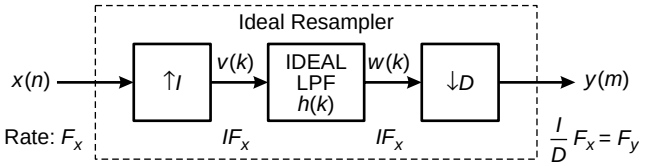


FIGURE 9.15 *Method for sampling rate conversion by a factor I/D*

$H(\omega_v)$ other than to extract the fundamental period of $W(\omega_v)$. In this respect, $H(\omega_v)$ acts as a lowpass filter as in the ideal interpolator. On the other hand, if $D/I > 1$, then we have net decimation. Hence it is necessary to first truncate even the fundamental period of $W(\omega_v)$ to get the frequency band down to $[-\pi/D, \pi/D]$ and to avoid aliasing in the decimation that follows. In this respect, $H(\omega_v)$ acts as a smoothing filter in the ideal decimator. When D or I is equal to 1, the general decimator/interpolator in Figure 9.15 along with (9.36) reduces to the ideal interpolator or decimator as special case, respectively.

In the time domain, the output of the upsampler is the sequence

$$v(k) = \begin{cases} x(k/I), & k = 0, \pm I, \pm 2I, \ldots \\ 0, & \text{otherwise} \end{cases} \tag{9.37}$$

and the output of the linear time-invariant filter is

$$w(k) = \sum_{\ell=-\infty}^{\infty} h(k-\ell)v(\ell) = \sum_{\ell=-\infty}^{\infty} h(k-\ell I)x(\ell) \tag{9.38}$$

Finally, the output of the sampling rate converter is the sequence $\{y(m)\}$, which is obtained by downsampling the sequence $\{w(k)\}$ by a factor of D. Thus

$$y(m) = w(mD) = \sum_{\ell=-\infty}^{\infty} h(mD-\ell I)x(\ell) \tag{9.39}$$

It is illuminating to express (9.39) in a different form by making a change in variable. Let

$$\ell = \left\lfloor \frac{mD}{I} \right\rfloor - n \tag{9.40}$$

where the notation $\lfloor r \rfloor$ denotes the largest integer contained in r. With this change in variable, (9.39) becomes

$$y(m) = \sum_{n=-\infty}^{\infty} h\left(mD - \left\lfloor \frac{mD}{I} \right\rfloor I + nI\right) x\left(\left\lfloor \frac{mD}{I} \right\rfloor - n\right) \tag{9.41}$$

We note that

$$mD - \left\lfloor \frac{mD}{I} \right\rfloor I = (mD) \text{ modulo } I = ((mD))_I$$

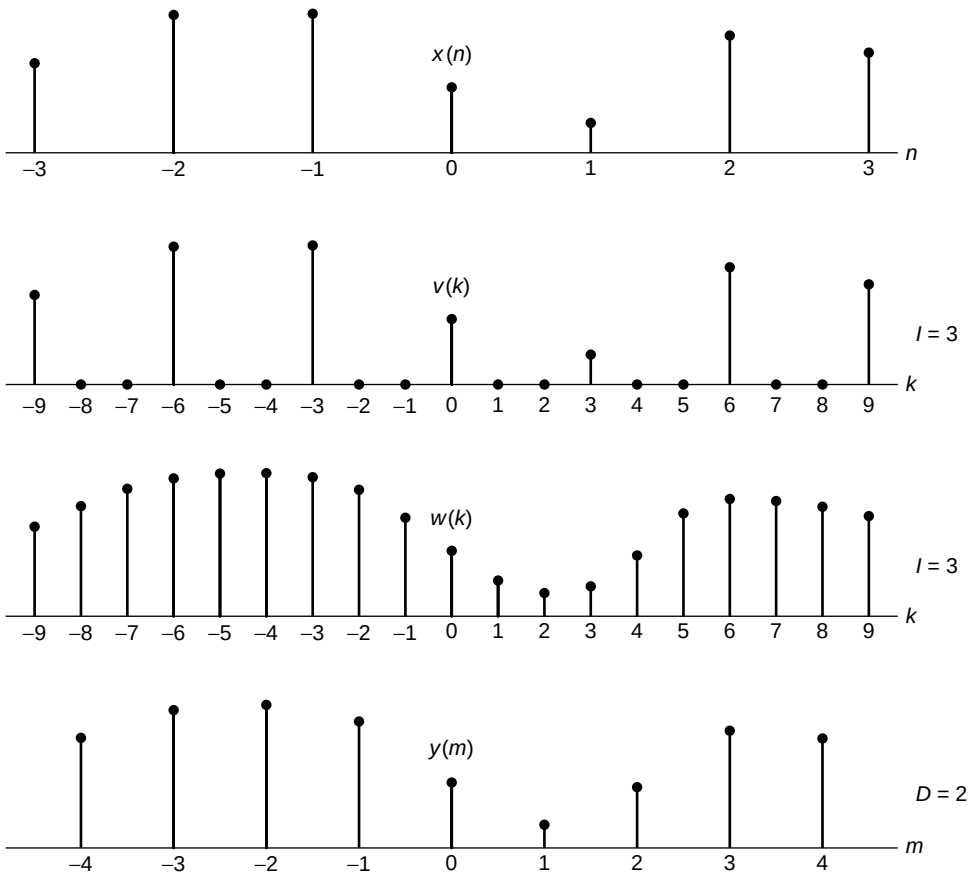


FIGURE 9.16 *Examples of signals $x(n)$, $v(k)$, $w(k)$, and $y(m)$ in the sampling rate converter of Figure 9.15 for $I = 3$ and $D = 2$*

Consequently, (9.41) can be expressed as

$$y(m) = \sum_{n=-\infty}^{\infty} h[nI + ((mD))_I]\, x\left(\left\lfloor \frac{mD}{I} \right\rfloor - n\right) \qquad \textbf{(9.42)}$$

These operations are shown in Figure 9.16 for $I = 3$ and $D = 2$.

It is apparent from (9.41) and Figure 9.16 that the output $y(m)$ is obtained by passing the input sequence $x(n)$ through a time-variant filter with impulse response

$$g(n, m) = h[nI + ((mD))_I] \quad -\infty < m, n < \infty \qquad \textbf{(9.43)}$$

where $h(k)$ is the impulse response of the time-invariant lowpass filter operating at the sampling rate IF_x. We further observe that for any integer k,

$$\begin{aligned} g(n, m + kI) &= h[nI + ((mD + kDI))_I] = h[nI + ((mD))_I] \\ &= g(n, m) \end{aligned} \tag{9.44}$$

Hence $g(n, m)$ is periodic in the variable m with period I.

Regarding the computational complexity of the lowpass filter in the general resampler, we note that it has a nonzero input only every I samples and the output is required only every D samples. If we use an FIR implementation for this lowpass filter, we need only compute its output one out of every D samples. However, if we instead use IIR implementation, we would generally have to compute intermediate outputs also because of the recursive nature of the filter. However, both types of filter benefit from the computational savings due to their sparse input.

The frequency-domain representation of the resampled signal $y(m)$ The frequency-domain relationships can be obtained by combining the results of the interpolation and decimation process. Thus, the spectrum at the output of the linear filter with impulse response $h(k)$ is

$$\begin{aligned} V(\omega_v) &= H(\omega_v)X(\omega_v I) \\ &= \begin{cases} IX(\omega_v I), & 0 \le |\omega_v| \le \min(\pi/D, \pi/I) \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{9.45}$$

The spectrum of the output sequence $y(m)$, obtained by decimating the sequence $v(n)$ by a factor of D, is

$$Y(\omega_y) = \frac{1}{D} \sum_{k=0}^{D-1} V\left(\frac{\omega_y - 2\pi k}{D}\right) \tag{9.46}$$

where $\omega_y = D\omega_v$. Since the linear filter prevents aliasing as implied by (9.45), the spectrum of the output sequence given by (9.46) reduces to

$$Y(\omega_y) = \begin{cases} \dfrac{I}{D} X\left(\dfrac{\omega_y}{D}\right), & 0 \le |\omega_y| \le \min\left(\pi, \frac{\pi D}{I}\right) \\ 0, & \text{otherwise} \end{cases} \tag{9.47}$$

MATLAB Implementation MATLAB provides the function `[y,h] = resample(x,I,D)` that resamples the signal in array `x` at `I/D` times the original sampling rate. The resulting resampled array `y` is `I/D` times longer (or the ceiling of it if the ratio is not an integer)—i.e., `length(y) = ceil(I/D)*length(x)`. The function approximates the anti-aliasing (lowpass) filter given in (9.36) by an FIR filter, `h`, designed (internally) using the Kaiser window. It also compensates for the filter's delay.

The length of the FIR filter `h` that `resample` uses is proportional to the fourth (optional) parameter `L` that has the default value of 10. For `L = 0`, `resample` performs a nearest-neighbor interpolation. The fifth optional parameter `beta` (default value 5) can be used to specify the Kaiser window stopband attenuation parameter β. The filter characteristics can be studied using the impulse response `h`.

□ **EXAMPLE 9.6** Consider the sequence $x(n) = \cos(0.125\pi n)$ discussed in Example 9.2. Change its sampling rate by 3/2, 3/4, and 5/8.

Solution The following MATLAB script shows the details.

```
n = 0:2048; k1 = 256; k2 = k1+32; m = 0:(k2-k1);
Hf1 = figure('units','inches','position',[1,1,6,4],...
    'paperunits','inches','paperposition',[0,0,6,4]);

% (a) Original signal
x = cos(0.125*pi*n); subplot(2,2,1);
Ha = stem(m,x(m+k1+1),'g','filled'); axis([-1,33,-1.1,1.1]);
set(Ha,'markersize',2); ylabel('Amplitude');
title('Original Sequence x(n)','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);

% (b) Sample rate Conversion by 3/2: I= 3, D = 2
I = 3; D = 2; y = resample(x,I,D); subplot(2,2,2);
Hb = stem(m,y(m+k1*I/D+1),'c','filled');  axis([-1,33,-1.1,1.1]);
set(Hb,'markersize',2); ylabel('Amplitude');
title('Sample Rate I/D: I = 3, D = 2','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);

% (c) Sample rate Conversion by 3/4: I= 3, D = 4
I = 3; D = 4; y = resample(x,I,D); subplot(2,2,3);
Hc = stem(m,y(m+k1*I/D+1),'r','filled');  axis([-1,33,-1.1,1.1]);
set(Hc,'markersize',2); ylabel('Amplitude');
title('Sample Rate I/D: I = 3, D = 4','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);
xlabel('n', 'fontsize',LF);

% (d) Sample rate Conversion by 5/8: I= 5, D = 8
I = 5; D = 8; y = resample(x,I,D); subplot(2,2,4);
Hd = stem(m,y(m+k1*I/D+1),'m','filled');  axis([-1,33,-1.1,1.1]);
set(Hd,'markersize',2); ylabel('Amplitude');
title('Sample Rate I/D: I = 5, D = 8','fontsize',TF);
set(gca,'xtick',[0,16,32]); set(gca,'ytick',[-1,0,1]);
xlabel('n', 'fontsize',LF);
```

The resulting plots are shown in Figure 9.17. The original $x(n)$ signal has 16 samples in one period of the cosine waveform. Since the first sampling rate

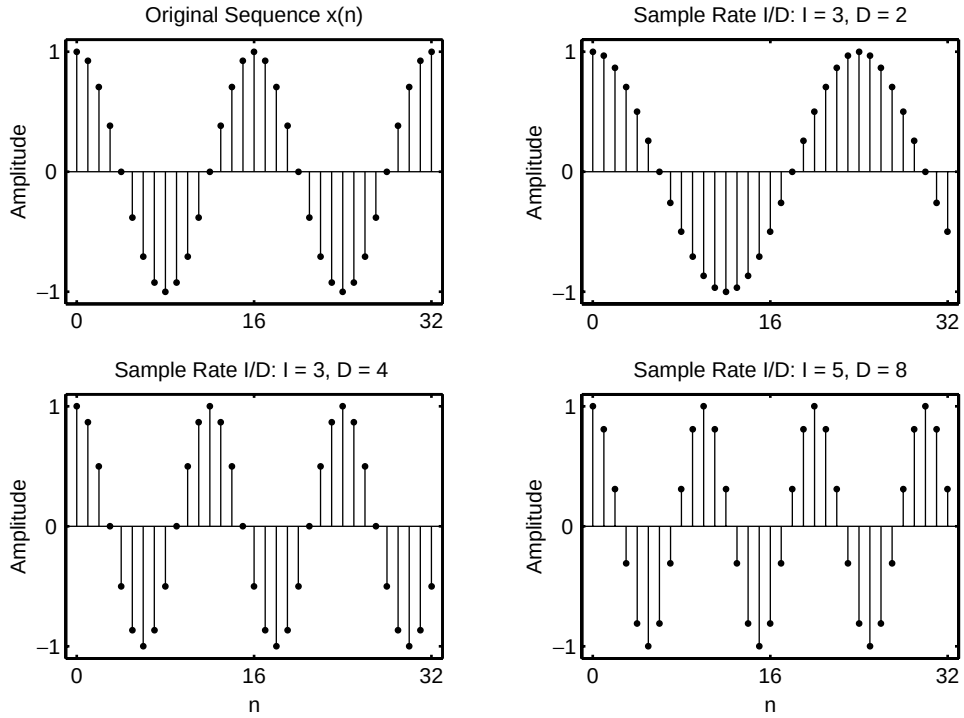


FIGURE 9.17 *Original and resampled signals in Example 9.6*

conversion by 3/2 is greater than one, the overall effect is to interpolate $x(n)$. The resulting signal has $16 \times 3/2 = 24$ samples in one period. The other two sampling rate conversion factors are less than one; thus, overall effect is to decimate $x(n)$. The resulting signals have $16 \times 3/4 = 12$ and $16 \times 5/8 = 10$ samples per period, respectively. □

9.5 FIR FILTER DESIGNS FOR SAMPLING RATE CONVERSION

In practical implementation of sampling rate converters we must replace the ideal lowpass filters of equations (9.20), (9.30), and (9.36) by a practical finite-order filter. The lowpass filter can be designed to have linear phase, a specified passband ripple, and stopband attenuation. Any of the standard, well-known FIR filter design techniques (e.g., window method, frequency sampling method) can be used to carry out this design. We consider linear-phase FIR filters for this purpose because of their ease of design and because they fit very nicely into a decimator stage where

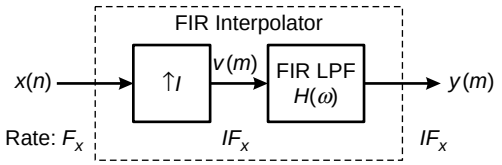


FIGURE 9.18 *An FIR integer interpolator*

only one of D outputs is required [see the discussion following (9.44) on page 496]. We will first discuss integer interpolators, followed by integer decimators and then the rational resamplers. The main emphasis will be on the specifications of these FIR lowpass filters, since the design problem has already been considered in Chapter 7.

9.5.1 FIR INTEGER INTERPOLATION

Replacing the ideal filter of the system given on page 489 with an FIR filter, we obtain the system shown in Figure 9.18. The relevant equation that relates the Fourier transforms $V(\omega)$ and $X(\omega)$ is (9.28), repeated here for convenience.

$$V(\omega) = X(\omega I) \tag{9.48}$$

Considering the frequency compression by I and the required amplitude scale factor of I, the ideal lowpass filter was determined in (9.30) and (9.33) to be

$$H_{\mathrm{I}}(\omega) = \begin{cases} I, & |\omega| < \pi/I; \\ 0, & \text{otherwise.} \end{cases} \tag{9.49}$$

MATLAB Implementation To design a linear-phase FIR filter for use in interpolation (and as we shall see later for decimation) operation, MATLAB provides the function `h = intfilt(I,L,alpha)`. When used on a sequence interspersed with `I-1` consecutive zeros between every `I` samples, the function performs ideal bandlimited interpolation using the nearest `2*L` nonzero samples. It assumes that the bandwidth of the signal $x(n)$ is `alpha` times π radians/sample—i.e., `alpha=1` means the full signal bandwidth. The length of the filter impulse response array `h` is `2*I*L-1`. The designed filter is identical to that used by the `interp` function. Therefore, the parameter `L` should be chosen carefully to avoid numerical instability. It should be a smaller value for higher `I` value but no more than ten.

□ **EXAMPLE 9.7** Design a linear-phase FIR interpolation filter to interpolate a signal by a factor of 4, using the bandlimited method.

Solution We will explore the `intfilt` function for the design using `L = 5` and study the effect of `alpha` on the filter design. The following MATLAB script provides the detail.

```
I = 4; L = 5;
Hf1 = figure('units','inches','position',[1,1,6,4],...
    'paperunits','inches','paperposition',[0,0,6,4]);

% (a) Full signal bandwidth: alpha = 1
alpha = 1; h = intfilt(I,L,alpha);
[Hr,w,a,L] = Hr_Type1(h); Hr_min = min(Hr); w_min = find(Hr == Hr_min);
H = abs(freqz(h,1,w)); Hdb = 20*log10(H/max(H)); min_attn = Hdb(w_min);
subplot(2,2,1); plot(ww/pi,Hr,'g','linewidth',1.0); axis([0,1,-1,5]);
set(gca,'xtick',[0,1/I,1],'ytick',[0,I]); grid; ylabel('Amplitude');
title('Amplitude Response: alpha = 1','fontsize',TF)

subplot(2,2,3); plot(w/pi,Hdb,'m','linewidth',1.0); axis([0,1,-50,10]);
set(gca,'xtick',[0,1/I,1],'ytick',[-50,round(min_attn),0]); grid
ylabel('Decibels'); xlabel('\omega/\pi', 'fontsize',10);
title('Log-mag Response: alpha = 1','fontsize',TF)

% (b) Partial signal bandwidth: alpha = 0.75
alpha = 0.75; h = intfilt(I,L,alpha);
[Hr,w,a,L] = Hr_Type1(h); Hr_min = max(Hr(end/2:end)); w_min = find(Hr == Hr_min);
H = abs(freqz(h,1,w)); Hdb = 20*log10(H/max(H)); min_attn = Hdb(w_min);
subplot(2,2,2); plot(ww/pi,Hr,'g','linewidth',1.0); axis([0,1,-1,5]);
set(gca,'xtick',[0,1/I,1],'ytick',[0,I]); grid; ylabel('Amplitude');
title('Amplitude Response: alpha = 0.75','fontsize',TF)

subplot(2,2,4); plot(w/pi,Hdb,'m','linewidth',1.0); axis([0,1,-50,10]);
set(gca,'xtick',[0,1/I,1],'ytick',[-50,round(min_attn),0]); grid
ylabel('Decibels'); xlabel('\omega/\pi', 'fontsize',10);
title('Log-mag Response: alpha = 0.75','fontsize',TF)
```

The plots are shown in Figure 9.19. For the full bandwidth case of `alpha = 1`, the filter has more ripple in both the passband and the stopband with the minimum stopband attenuation of 22 DB. This is because the filter transition band is very narrow. For `alpha = 0.75`, the filter specifications are more lenient, and hence its response is well behaved with minimum stopband attenuation of 40 DB. Note that we do not have complete control over other design parameters. These issues are discussed in more detail further along in this section. □

In the following example, we design a linear-phase equiripple FIR interpolation filter using the Parks-McClellen algorithm.

□ **EXAMPLE 9.8** Design an interpolator that increases the input sampling rate by a factor of $I = 5$. Use the `firpm` algorithm to determine the coefficients of the FIR filter

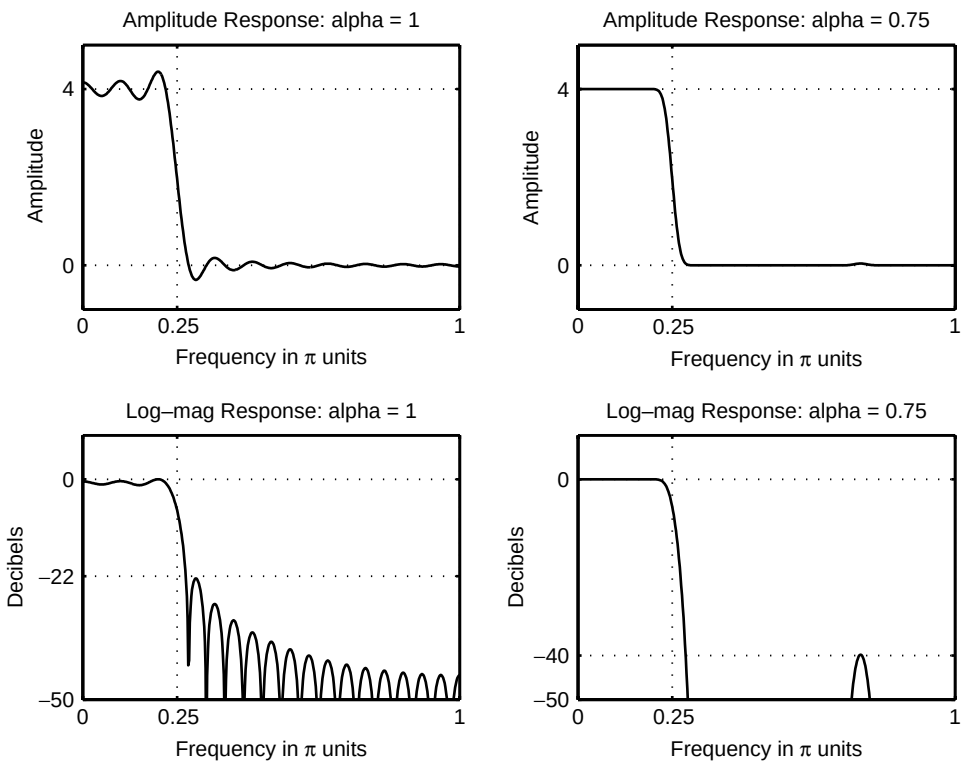


FIGURE 9.19 *FIR interpolation filter design plots for* `I = 4` *and* `L = 5`

that has 0.1 dB ripple in the passband and is down by at least 30 dB in the stopband. Choose reasonable values for band-edge frequencies.

Solution

The passband cutoff frequency should be $\omega_p = \pi/I = 0.2\pi$. To get a reasonable value for the filter length we choose the transition width of 0.12π, which gives stopband cutoff frequency of $\omega_s = 0.32\pi$. Note that the nominal gain of the filter in the passband should be equal to $I = 5$, which means that the ripple values computed using the decibel values are scaled by 5. A filter of length $M = 31$ achieves the design specifications given above. The details are given in the following MATLAB script.

```
I = 5; Rp = 0.1; As = 30; wp = pi/I; ws = wp+pi*0.12;
[delta1,delta2] = db2delta(Rp,As); weights = [delta2/delta1,1];
F = [0,wp,ws,pi]/pi; A = [I,I,0,0];
h = firpm(30,F,A,weights); n = [0:length(h)-1];
[Hr,w,a,L] = Hr_Type1(h); Hr_min = min(Hr); w_min = find(Hr == Hr_min);
H = abs(freqz(h,1,w)); Hdb = 20*log10(H/max(H)); min_attn = Hdb(w_min);
```

```
Hf1 = figure('units','inches','position',[1,1,6,4],...
    'paperunits','inches','paperposition',[0,0,6,4]);
subplot(2,2,1); Hs1 = stem(n,h,'filled'); set(Hs1,'markersize',2);
axis([-1,length(n),-0.5,1.5]); ylabel('Amplitude'); xlabel('n','vertical','bottom');
Title('Impulse Response','fontsize',TF);
subplot(2,2,3); plot(ww/pi,Hr,'m','linewidth',1.0); axis([0,1,-1,6]);
set(gca,'xtick',[0,wp/pi,ws/pi,1],'ytick',[0,I]); grid; ylabel('Amplitude');
title('Amplitude Response','fontsize',TF); xlabel('Frequency in \pi units');

subplot(2,2,2); plot(w/pi,Hdb,'m','linewidth',1.0); axis([0,1,-50,10]);
set(gca,'xtick',[0,wp/pi,ws/pi,1],'ytick',[-50,round(min_attn),0]); grid
ylabel('Decibels');
title('Log-magnitude Response','fontsize',TF);

subplot(2,2,4);
lw = length(w)-1; PB = [0:floor(wp/pi*lw)]; HrPB = Hr(PB+1)-I;
SB = [ceil(ws/pi*lw):lw]; HrSB = Hr(SB+1);
[AX,H1,H2] = plotyy(PB/lw,HrPB,SB/lw,HrSB);

delta1 = round(delta1*I*100)/100; delta2 = round(delta2*I*100)/100;
set(AX(1),'xtick',[0,wp/pi,ws/pi,1],'ytick',[-delta1,0,delta1],'Ycolor','g');
set(AX(2),'xtick',[0,wp/pi,ws/pi,1],'ytick',[-delta2,0,delta2],'Ycolor','r');
set(H1,'color','g','linewidth',1); set(H2,'color','r','linewidth',1);
title('Scaled Ripples','fontsize',TF); xlabel('Frequency in \pi units');
```

The responses of the designed FIR filter are given in Figure 9.20. Even though this filter passes the original signal, it is possible that some of the neighboring spectral energy may also leak through if the signal is of full bandwidth of π radians. Hence we need better design specifications, which are discussed further along in this section. □

MATLAB Implementation To use the FIR filter for interpolation purposes (such as the one designed in Example 9.8), MATLAB has provided a general function, `upfirdn`, that can be used for interpolation and decimation as well as for resampling purposes. Unlike other functions discussed in this chapter, `upfirdn` incorporates the user-defined FIR filter (which need not be linear phase) in the operation. When invoked as `y = upfirdn(x,h,I)`, the function upsamples the input data in the array `x` by a factor of the integer `I` and then filters the upsampled signal data with the impulse response sequence given in the array `h` to produce the output array `y`, thus implementing the system in Figure 9.18.

□ **EXAMPLE 9.9** Let $x(n) = \cos(0.5\pi n)$. Increase the input sampling rate by a factor of $I = 5$, using the filter designed in Example 9.8.

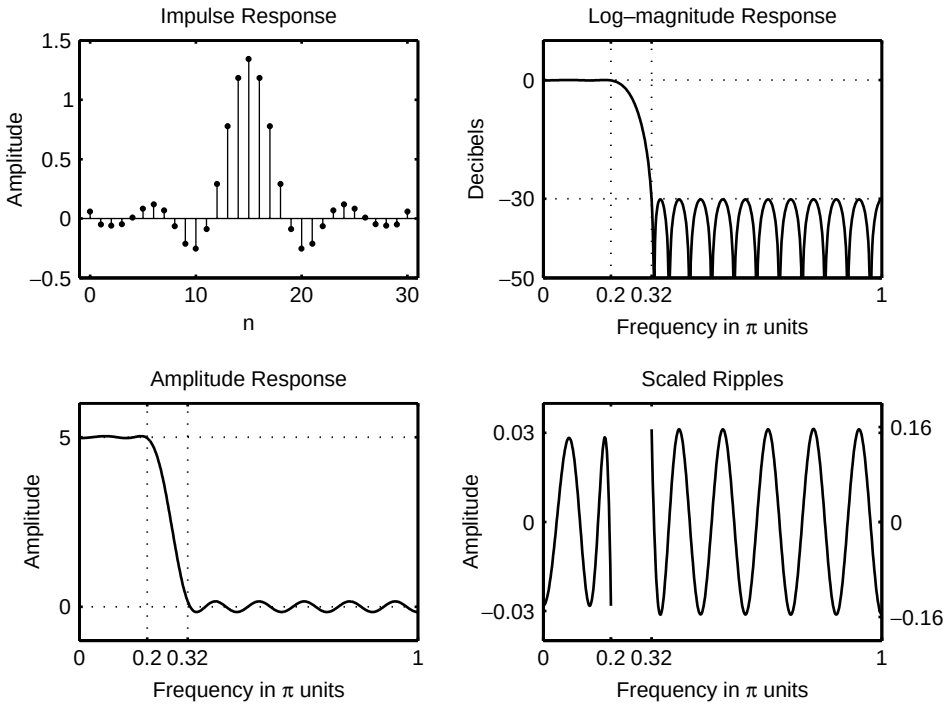


FIGURE 9.20 *Responses of the FIR interpolation filter in Example 9.8*

Solution The steps are given in the following MATLAB script.

```
% Given Parameters:
I = 5; Rp = 0.1; As = 30; wp = pi/I; ws = 0.32*pi;
[delta1,delta2] = db2delta(Rp,As); weights = [delta2/delta1,1];
n = [0:50]; x = cos(0.5*pi*n);
n1 = n(1:17); x1 = x(17:33); % for plotting purposes

% Input signal
Hf1 = figure('units','inches','position',[1,1,6,4],...
    'paperunits','inches','paperposition',[0,0,6,4]);
subplot(2,2,1); Hs1 = stem(n1,x1,'filled'); set(Hs1,'markersize',2,'color','g');
set(gca,'xtick',[0:4:16],'ytick',[-1,0,1]);
axis([-1,17,-1.2,1.2]); ylabel('Amplitude'); xlabel('n','vertical','middle');
Title('Input Signal x(n)','fontsize',TF);

% Interpolation with Filter Design: Length M = 31
M = 31; F = [0,wp,ws,pi]/pi; A = [I,I,0,0];
h = firpm(M-1,F,A,weights); y = upfirdn(x,h,I);
delay = (M-1)/2; % Delay imparted by the filter
m = delay+1:1:50*I+delay+1; y = y(m); m = 1:81; y = y(81:161); % for plotting
```

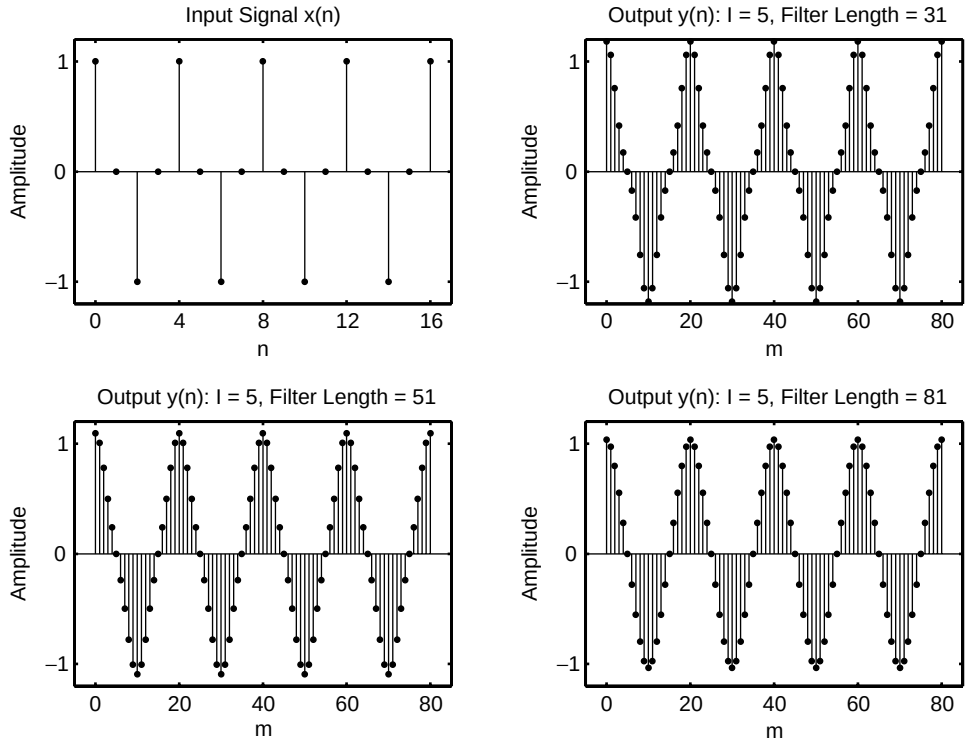


FIGURE 9.21 *Signal plots in Example 9.9*

```
subplot(2,2,2); Hs2 = stem(m,y,'filled'); set(Hs2,'markersize',2,'color','m');
axis([-5,85,-1.2,1.2]); set(gca,'xtick',[0:4:16]*I,'ytick',[-1,0,1]);
title('Output y(n): Filter length=31','fontsize',TF);
xlabel('n','vertical','middle');  ylabel('Amplitude');
```

The signal stem plots are shown in Figure 9.21. The upper left-hand plot shows a segment of the input signal $x(n)$, and the upper right-hand plot shows the interpolated signal $y(n)$ using the filter of length 31. The plot is corrected for filter delay and the effect of its transient response. It is somewhat surprising that the interpolated signal is not what it should be. The signal peak is more than one, and the shape is distorted. A careful observation of the filter response plot in Figure 9.20 reveals that the broad transition width and a smaller attenuation has allowed some of the spectral energy to leak, creating a distortion.

To investigate this further, we designed filters with larger orders of 51 and 81, as detailed in the following MATLAB script.

```
% Interpolation with Filter Design: Length M = 51
M = 51; F = [0,wp,ws,pi]/pi; A = [I,I,0,0];
h = firpm(M-1,F,A,weights); y = upfirdn(x,h,I);
```

```
delay = (M-1)/2; % Delay imparted by the filter
m = delay+1:1:50*I+delay+1; y = y(m); m = 1:81; y = y(81:161);
subplot(2,2,3); Hs3 = stem(m,y,'filled'); set(Hs3,'markersize',2,'color','m');
axis([-5,85,-1.2,1.2]); set(gca,'xtick',[0:4:16]*I,'ytick',[-1,0,1]);
title('Output y(n): Filter length=51','fontsize',TF);
xlabel('n','vertical','middle');  ylabel('Amplitude');

% Interpolation with Filter Design: Length M = 81
M = 81; F = [0,wp,ws,pi]/pi; A = [I,I,0,0];
h = firpm(M-1,F,A,weights); y = upfirdn(x,h,I);
delay = (M-1)/2; % Delay imparted by the filter
m = delay+1:1:50*I+delay+1; y = y(m); m = 1:81; y = y(81:161);
subplot(2,2,4); Hs3 = stem(m,y,'filled'); set(Hs3,'markersize',2,'color','m');
axis([-5,85,-1.2,1.2]); set(gca,'xtick',[0:4:16]*I,'ytick',[-1,0,1]);
title('Output y(n): Filter length=81','fontsize',TF);
xlabel('n','vertical','middle');  ylabel('Amplitude');
```

The resulting signals are also shown in lower plots in Figure 9.21. Clearly, for large orders, the filter has better lowpass characteristics. The signal peak value approaches 1, and its shape approaches the cosine waveform. Thus, a good filter design is critical even in a simple signal case. □

9.5.2 DESIGN SPECIFICATIONS

When we replace $H_{\rm I}(\omega)$ by a finite-order FIR filter $H(\omega)$, we must allow for a transition band; thus, the filter cannot have a passband edge up to π/I. Towards this, we define

- $\omega_{x,\rm p}$ as the highest frequency of the signal $x(n)$ that we want to preserve
- $\omega_{x,\rm s}$ as the full signal bandwidth of $x(n)$,—i.e., there is no energy in $x(n)$ above the frequency $\omega_{x,\rm s}$.

Thus, we have $0 < \omega_{x,\rm p} < \omega_{x,\rm s} < \pi$. Note that the parameters $\omega_{x,\rm p}$ and $\omega_{x,\rm s}$, as defined are *signal parameters*, not filter parameters; they are shown in Figure 9.22a. The filter parameters will be defined based on $\omega_{x,\rm p}$ and $\omega_{x,\rm s}$.

From equation (9.48), these signal parameter frequencies for $v(m)$ become $\omega_{x,p}/I$ and $\omega_{x,s}/I$, respectively, because the frequency scale is compressed by the factor I. This is shown in Figure 9.22b. A linear-phase FIR filter can now be designed to pass frequencies up to $\omega_{x,p}/I$ and to suppress frequencies starting at $(2\pi - \omega_{x,\rm s})/I$. Let

$$\omega_{\rm p} = \left(\frac{\omega_{x,\rm p}}{I}\right) \quad \text{and} \quad \omega_{\rm s} = \left(\frac{2\pi - \omega_{x,\rm s}}{I}\right) \tag{9.50}$$

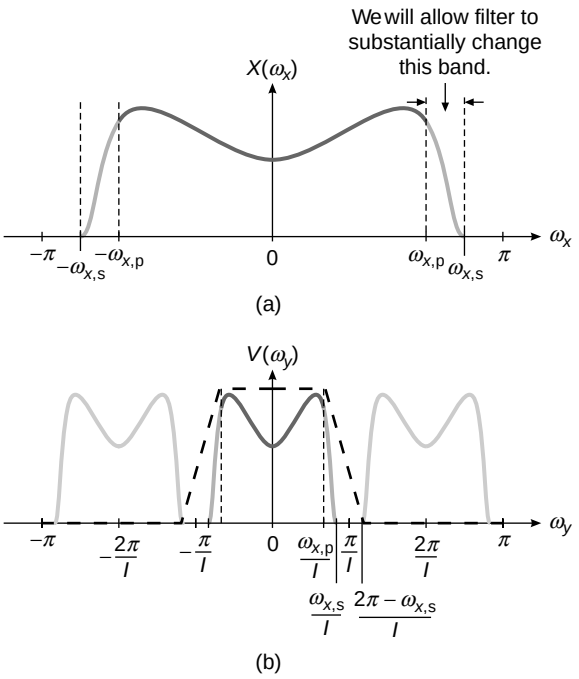


FIGURE 9.22 *Frequency parameters: (a) signal, (b) filter*

be the passband and stopband edge frequencies, respectively, of the lowpass linear-phase FIR filter given by

$$H(\omega) = H_{\mathrm{r}}(\omega)\mathrm{e}^{\jmath\theta(\omega)} \tag{9.51}$$

where $H_{\mathrm{r}}(\omega)$ is the real-valued amplitude response and $\theta(\omega)$ is the unwrapped phase response. Then we have the following filter design specifications:

$$\boxed{\begin{aligned} &\frac{1}{I}H_{\mathrm{r}}(\omega) \leq 1 \pm \delta_1 \text{ for } |\omega| \in [0, \omega_{\mathrm{p}}] \\ &\frac{1}{I}H_{\mathrm{r}}(\omega) \leq \pm\delta_2 \quad \text{for } |\omega| \in [\omega_{\mathrm{s}}, \pi] \end{aligned}} \tag{9.52}$$

where ω_{p} and ω_{s} are as given in (9.50) and δ_1 and δ_2 are the passband and stopband ripple parameters, respectively, of the lowpass FIR filter.

Comment: Instead of beginning the stopband at π/I, we were able to shift it to $(2\pi - \omega_{\mathrm{s}})/I$. If $\omega_{x,\mathrm{s}} \ll \pi$, then this will be an important consideration to lower filter order. However, in the worst-case scenario of

$\omega_{x,\mathrm{s}} = \pi$, the stopband will begin at $\frac{\pi}{I}$, which is the same as in the ideal lowpass filter of (9.49). Almost always $\omega_{x,\mathrm{s}} < \pi$, and we can then choose $\omega_{x,\mathrm{p}}$ as close to $\omega_{x,\mathrm{s}}$ as we want. However, this will reduce the size of the transition band, which means a higher filter order.

□ **EXAMPLE 9.10** Design a better FIR lowpass filter for sampling rate increase by a factor of $I = 5$ for the signal in Example 9.9.

Solution Since $x(n) = \cos(0.5\pi n)$, the signal bandwidth and bandwidth to be preserved are the same—i.e., $\omega_{x,\mathrm{p}} = \omega_{x,\mathrm{s}} = 0.5\pi$. Thus, from (9.50), $\omega_{\mathrm{p}} = 0.5\pi/5 = 0.1\pi$ and $\omega_{\mathrm{s}} = (2\pi - 0.5\pi)/5 = 0.3\pi$. We will design the filter for $R_p = 0.01$ and $A_s = 50$ dB. The resulting filter order is 32, which is 2 higher than the one in Example 9.9 but with much superior attenuation. The details are given below.

```
% Given Parameters:
n = [0:50]; wxp = 0.5*pi; x = cos(wxp*n);
n1 = n(1:9); x1 = x(9:17); % for plotting purposes
I = 5; I = 5; Rp = 0.01; As = 50; wp = wxp/I; ws = (2*pi-wxp)/I;
[delta1,delta2] = db2delta(Rp,As); weights = [delta2/delta1,1];
[N,Fo,Ao,weights] = firpmord([wp,ws]/pi,[1,0],[delta1,delta2],2);N = N+2;

% Input signal
Hf1 = figure('units','inches','position',[1,1,6,4],...
    'paperunits','inches','paperposition',[0,0,6,4]);
subplot(2,2,1); Hs1 = stem(n1,x1,'filled'); set(Hs1,'markersize',2,'color','g');
set(gca,'xtick',[0:4:16],'ytick',[-1,0,1]);
axis([-1,9,-1.2,1.2]); ylabel('Amplitude'); xlabel('n','vertical','middle');
Title('Input Signal x(n)','fontsize',TF);

% Interpolation with Filter Design: Length M = 31
h = firpm(N,Fo,I*Ao,weights); y = upfirdn(x,h,I);
delay = (N)/2; % Delay imparted by the filter
m = delay+1:1:50*I+delay+1; y = y(m); m = 0:40; y = y(81:121);
subplot(2,2,3); Hs2 = stem(m,y,'filled'); set(Hs2,'markersize',2,'color','m');
axis([-5,45,-1.2,1.2]); set(gca,'xtick',[0:4:16]*I,'ytick',[-1,0,1]);
title('Output Signal y(n): I=5','fontsize',TF);
xlabel('m','vertical','middle');  ylabel('Amplitude');

% Filter Design Plots
[Hr,w,a,L] = Hr_Type1(h); Hr_min = min(Hr); w_min = find(Hr == Hr_min);
H = abs(freqz(h,1,w)); Hdb = 20*log10(H/max(H)); min_attn = Hdb(w_min);
subplot(2,2,2); plot(ww/pi,Hr,'m','linewidth',1.0); axis([0,1,-1,6]);
set(gca,'xtick',[0,wp/pi,ws/pi,1],'ytick',[0,I]); grid; ylabel('Amplitude');
title('Amplitude Response','fontsize',TF);
xlabel('Frequency in \pi units','vertical','middle');
subplot(2,2,4); plot(w/pi,Hdb,'m','linewidth',1.0); axis([0,1,-60,10]);
```

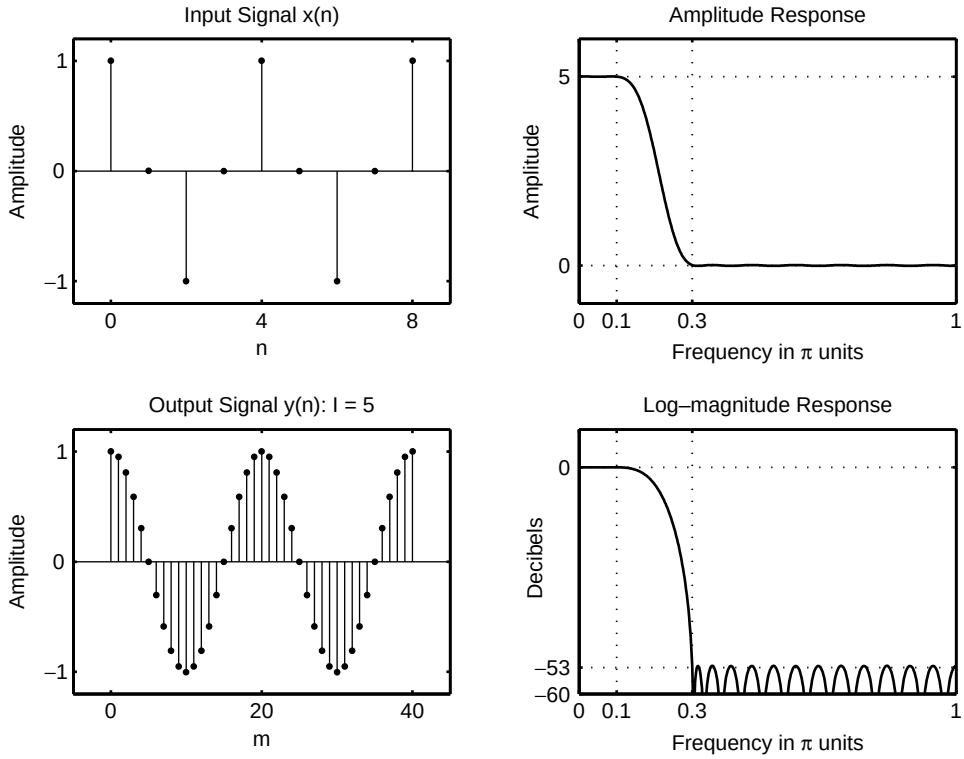


FIGURE 9.23 *Signal plots and filter design plots in Example 9.10*

```
set(gca,'xtick',[0,wp/pi,ws/pi,1],'ytick',[-60,round(min_attn),0]); grid
ylabel('Decibels'); xlabel('Frequency in \pi units','vertical','middle');
title('Log-magnitude Response','fontsize',TF);
```

The signal stem plots and filter design plots are shown in Figure 9.23. The designed filter has a minimum stopband attenuation of 53 dB, and the resulting interpolation is accurate even with the filter order of 32. □

9.5.3 FIR INTEGER DECIMATION

Consider the system in Figure 9.5 on page 481 in which the ideal lowpass filter is replaced by an FIR filter $H(\omega)$, which then results in the system shown in Figure 9.24. The relationship between $Y(\omega_y)$ and $X(\omega)$ is given by (9.24), which is repeated here for convenience

$$Y(\omega_y) = \frac{1}{D}\sum_{k=0}^{D-1} H\left(\omega - \frac{2\pi k}{D}\right) X\left(\omega - \frac{2\pi k}{D}\right); \quad \omega = \frac{\omega_y}{D} \qquad \textbf{(9.53)}$$

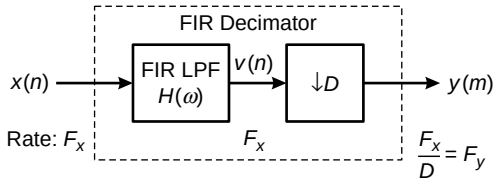


FIGURE 9.24 *An FIR integer decimator*

which is nothing but the aliased sum of the $H(\omega)X(\omega)$. Thus, the condition necessary to avoid aliasing is

$$H(\omega)X(\omega) = 0 \quad \text{for} \quad \frac{\pi}{D} \leq |\omega| \leq \pi \tag{9.54}$$

Then,

$$Y(\omega_y) = \frac{1}{D}X(\omega)H(\omega) \tag{9.55}$$

as in (9.25), where the ideal filtering was accomplished with $H_{\mathrm{D}}(\omega)$ as given in (9.20).

□ **EXAMPLE 9.11** Design a decimator that downsamples an input signal $x(n)$ by a factor $D = 2$. Use the `firpm` algorithm to determine the coefficients of the FIR filter that has a 0.1 dB ripple in the passband and is down by at least 30 dB in the stopband. Choose reasonable values for band-edge frequencies.

Solution The passband cutoff frequency should be $\omega_{\mathrm{p}} = \pi/D = 0.5\pi$. To get a reasonable value for the filter length we choose the transition width of 0.1π, which gives stopband a cutoff frequency of $\omega_{\mathrm{s}} = 0.3\pi$. A filter of length $M = 37$ achieves the preceding design specifications. The details are given in the following MATLAB script.

```
% Filter Design
D = 2; Rp = 0.1; As = 30; wp = pi/D; ws = wp+0.1*pi;
[delta1,delta2] = db2delta(Rp,As);
[N,F,A,weights] = firpmord([wp,ws]/pi,[1,0],[delta1,delta2],2);
h = firpm(N,F,A,weights); n = [0:length(h)-1];
[Hr,w,a,L] = Hr_Type1(h); Hr_min = min(Hr); w_min = find(Hr == Hr_min);
H = abs(freqz(h,1,w)); Hdb = 20*log10(H/max(H)); min_attn = Hdb(w_min);

Hf1 = figure('units','inches','position',[1,1,6,4],...
    'paperunits','inches','paperposition',[0,0,6,4]);
subplot(2,2,1); Hs1 = stem(n,h,'filled'); set(Hs1,'markersize',2);
axis([-1,length(n),-0.15,0.6]); ylabel('Amplitude','vertical','cap');
xlabel('n','vertical','bottom');set(gca,'xtick',[n(1),n(end)],'ytick',[0,0.5]);
Title('Impulse Response','fontsize',TF,'vertical','baseline');

subplot(2,2,3); plot(w/pi,Hr,'m','linewidth',1.0); axis([0,1,-0.1,1.1]);
set(gca,'xtick',[0,wp/pi,ws/pi,1],'ytick',[0,1]); grid;
```

```
ylabel('Amplitude','vertical','cap');
title('Amplitude Response','fontsize',TF,'vertical','baseline');
xlabel('Frequency in \pi units','vertical','middle');

subplot(2,2,2); plot(w/pi,Hdb,'m','linewidth',1.0); axis([0,1,-50,10]);
set(gca,'xtick',[0,wp/pi,ws/pi,1],'ytick',[-50,round(min_attn),0]); grid
ylabel('Decibels','vertical','cap');
xlabel('Frequency in \pi units','vertical','middle');
title('Log-magnitude Response','fontsize',TF,'vertical','baseline');

subplot(2,2,4);
lw = length(w)-1; PB = [0:floor(wp/pi*lw)]; HrPB = Hr(PB+1)-1;
SB = [ceil(ws/pi*lw):lw]; HrSB = Hr(SB+1);
[AX,H1,H2] = plotyy(PB/lw,HrPB,SB/lw,HrSB);
delta1 = round(delta1*1000)/1000; delta2 = round(delta2*100)/100;
set(AX(1),'xtick',[0,wp/pi,ws/pi,1],'ytick',[-delta1,0,delta1],'Ycolor','g');
set(AX(2),'xtick',[0,wp/pi,ws/pi,1],'ytick',[-delta2,0,delta2],'Ycolor','r');
set(H1,'color','g','linewidth',1); set(H2,'color','r','linewidth',1);
title('Unweighted Ripples','fontsize',TF,'vertical','baseline');
ylabel('Amplitude','vertical','cap')
xlabel('Frequency in \pi units','vertical','middle');
```

The responses of the designed FIR filter are given in Figure 9.25. This filter passes the signal spectrum over the passband $[0, \pi/2]$ without any distortion. However, since the transition width is not very narrow, it is possible that some of the signal over the transition band may alias into the band of interest. Also the 30 db attenuation may allow a small fraction of the signal spectrum from the stopband into the passband after downsampling. Therefore, we need a better approach for filter specifications, as discussed further along in this section. □

MATLAB Implementation As discussed, the `upfirdn` function can also be used for implementing the user-designed FIR filter in the decimation operation. When invoked as `y = upfirdn(x,h,1,D)`, the function filters the signal data in the array `x` with the impulse response given in the array `h` and then downsamples the filtered data by the integer factor `D` to produce the output array `y`, thus implementing the system in Figure 9.24.

□ **EXAMPLE 9.12** Using the filter designed in Example 9.11 decimate sinusoidal signals $x_1(n) = \cos(\pi n/8)$ and $x_2(n) = \cos(\pi n/2)$ with frequencies within the passband of the filter. Verify the performance of the FIR filter and the results of the decimation.

Solution The following MATLAB script provides the details.

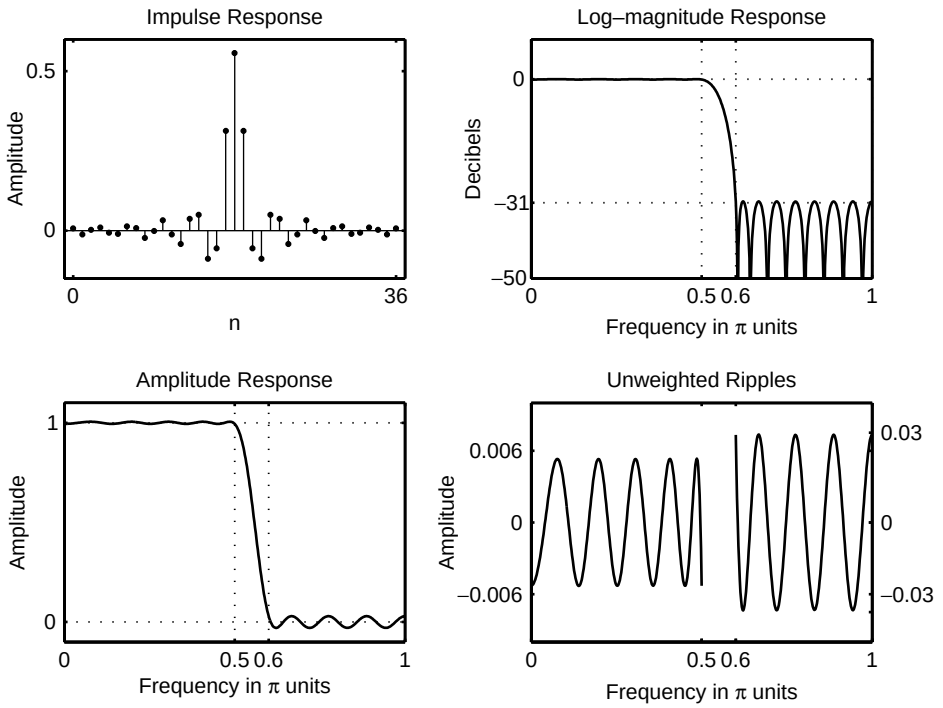


FIGURE 9.25 *Responses of the FIR decimation filter in Example 9.11*

```
% Given Parameters:
D = 2; Rp = 0.1; As = 30; wp = pi/D; ws = wp+0.1*pi;
% Filter Design
[delta1,delta2] = db2delta(Rp,As);
[N,F,A,weights] = firpmord([wp,ws]/pi,[1,0],[delta1,delta2],2);
h = firpm(N,F,A,weights); delay = N/2; % Delay imparted by the filter

Hf1 = figure('units','inches','position',[1,1,6,4],...
    'paperunits','inches','paperposition',[0,0,6,4]);

% Input signal x1(n) = cos(2*pi*n/16)
n = [0:256]; x = cos(pi*n/8);
n1 = n(1:33); x1 = x(33:65); % for plotting purposes
subplot(2,2,1); Hs1 = stem(n1,x1,'filled'); set(Hs1,'markersize',2,'color','g');
set(gca,'xtick',[0:8:32],'ytick',[-1,0,1]);
axis([-2,34,-1.2,1.2]); ylabel('Amplitude'); xlabel('n','vertical','middle');
Title('Input Signal: x1(n) = cos(\pin/8)','fontsize',TF,'vertical','baseline');

% Decimation of x1(n): D = 2
y = upfirdn(x,h,1,D);
```

```
m = delay+1:1:128/D+delay+1; y = y(m); m = 0:16; y = y(16:32);
subplot(2,2,3); Hs2 = stem(m,y,'filled'); set(Hs2,'markersize',2,'color','m');
axis([-1,17,-1.2,1.2]); set(gca,'xtick',[0:8:32]/D,'ytick',[-1,0,1]);
title('Output signal: y1(n): D=2','fontsize',TF,'vertical','baseline');
xlabel('m','vertical','middle');  ylabel('Amplitude');


% Input signal x2(n) = cos(8*pi*n/16)
n = [0:256]; x = cos(8*pi*n/(16));
n1 = n(1:33); x1 = x(33:65); % for plotting purposes
subplot(2,2,2); Hs1 = stem(n1,x1,'filled'); set(Hs1,'markersize',2,'color','g');
set(gca,'xtick',[0:8:32],'ytick',[-1,0,1]);
axis([-2,34,-1.2,1.2]); ylabel('Amplitude'); xlabel('n','vertical','middle');
Title('Input Signal: x2(n) = cos(\pi/2)','fontsize',TF,'vertical','baseline');

% Decimation of x2(n): D = 2
y = upfirdn(x,[h],1,D); %y = downsample(conv(x,h),2);
m = delay+1:1:128/D+delay+1; y = y(m); m = 0:16; y = y(16:32);
subplot(2,2,4); Hs2 = stem(m,y,'filled'); set(Hs2,'markersize',2,'color','m');
axis([-1,17,-1.2,1.2]); set(gca,'xtick',[0:8:32]/D,'ytick',[-1,0,1]);
title('Output signal: y2(n): D=2','fontsize',TF,'vertical','baseline');
xlabel('m','vertical','middle');  ylabel('Amplitude');
```

The signal stem plots are shown in Figure 9.26. The leftside plots show the signal $x_1(n)$ and the corresponding decimated signal $y_1(n)$, and the rightside plots show the same for $x_2(n)$ and $y_2(n)$. In both cases the decimation appears to be correct. If we had chosen any frequency above $\pi/2$, then the filter would have attenuated or eliminated the signal. □

9.5.4 DESIGN SPECIFICATIONS

When we replace the ideal lowpass filter $H_D(\omega)$ by a finite-order FIR filter $H(\omega)$, we must allow for a transition band. Again we define

- $\omega_{x,p}$ as the signal bandwidth to be preserved
- $\omega_{x,s}$ as the frequency above which aliasing error is tolerated

Then we have $0 < \omega_{x,p} \leq \omega_{x,s} \leq \pi/D$. If we choose $\omega_{x,s} = \pi/D$, then the decimator will give no aliasing error. If we choose $\omega_{x,s} = \omega_{x,p}$, then the band above the signal band will contain aliasing errors. With these definitions and observations we can now specify the desired filter specifications. The filter must pass frequencies up to $\omega_{x,p}$, and its stopband must begin at $\left(\frac{2\pi}{D} - \omega_{x,s}\right)$ and continue up to π. Then, none of the $k \neq 0$ terms in (9.53)—i.e., the "aliases," will cause appreciable distortion in the band

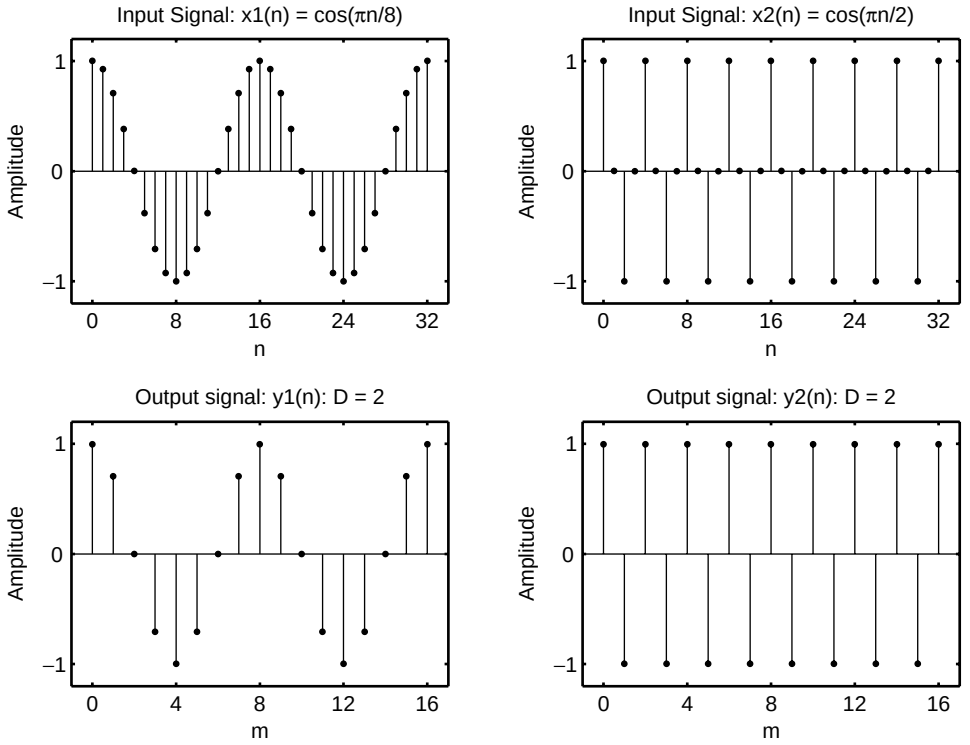


FIGURE 9.26 *Signal plots in Example 9.12*

up to $\omega_{x,s}$. Let

$$\omega_{\rm p} = \omega_{x,{\rm p}} \quad \text{and} \quad \omega_{\rm s} = \left(\frac{2\pi}{D} - \omega_{x,{\rm s}}\right) \tag{9.56}$$

be the passband and stopband edge frequencies, respectively, of the lowpass linear-phase FIR filter given in (9.51). Then we have the following filter design specifications:

$$\boxed{\begin{aligned} H_{\rm r}(\omega) &\le 1 \pm \delta_1 \ \text{ for } |\omega| \in [0, \omega_{\rm p}] \\ H_{\rm r}(\omega) &\le \pm\delta_2 \quad \text{ for } |\omega| \in [\omega_{\rm s}, \pi] \end{aligned}} \tag{9.57}$$

where $\omega_{\rm p}$ and $\omega_{\rm s}$ are as given in (9.56) and δ_1 and δ_2 are the passband and stopband ripple parameters of the lowpass FIR filter, respectively. Note that it does not matter what the spectrum $X(\omega)$ is. We simply require that the product $X(\omega)H(\omega)$ be very small beginning at $\omega_| = 2\pi/D - \omega_{x,{\rm s}}$ so that $k \neq 0$ terms in (9.53) do not provide significant contribution in the band $[-\omega_{x,{\rm s}}, \omega_{x,{\rm s}}]$, which is required to be free of aliasing.

Significance of δ_1 and δ_2 The filter ripple parameters δ_1 and δ_2 have the following significance, which must be taken into consideration while specifying their values:

- The passband ripple δ_1 measures the ripple in the passband and hence controls the distortion in the signal bandwidth $\omega_{\rm p}$.
- The stopband ripple δ_2 controls the amount of aliased energy (also called leakage) that gets into the band up to $\omega_{x,{\rm s}}$.

There are $(D-1)$ contributions due to $k \neq 0$ terms in (9.53). These are expected to add incoherently (i.e., have peaks at different locations), so the overall peak error should be about δ_2. The actual error depends on how $X(\omega)$ varies over the rest of the band $|\omega| > \omega_{x,{\rm p}}$. Clearly, the filter stopband ripple δ_2 controls the aliasing error in the signal passband. Therefore, *both δ_1 and δ_2 affect the decimated signal in its passband.*

Comment: Comparing the FIR decimator filter specifications (9.57) to those for the FIR interpolator in (9.52), we see a high degree of similarity. In fact, a filter designed to decimate by factor D can also be used to interpolate by the factor $I = D$, as we see from the following example. This means that the function `intfilt` can also be used to design FIR filters for decimation.

□ **EXAMPLE 9.13** To design a decimate by D stage we need values for $\omega_{x,{\rm p}}$ and $\omega_{x,{\rm s}}$ (remember that these are signal parameters). Assume $\omega_{x,{\rm p}} = \pi/(2D)$, which satisfies the constraint $\omega_{x,{\rm p}} \leq \pi/D$ and is exactly half the decimated bandwidth. Let $\omega_{x,{\rm s}} = \omega_{x,{\rm p}}$. Then the FIR lowpass filter must pass frequencies up to $\omega_{\rm p} = \pi/(2D)$ and stop frequencies above $\omega_{\rm s} = 2\pi/D - \pi/(2D) = 3\pi/(2D)$.

Now consider the corresponding interpolation problem. We want to interpolate by I. We again choose $\omega_{x,{\rm s}} = \omega_{x,{\rm p}}$, but now the range is $\omega_{x,{\rm p}} < \pi$. If we take exactly half this band, we get $\omega_{x,{\rm p}} = \pi/2$. Then according to the specifications (9.52) for the interpolation, we want the filter to pass frequencies up to $\pi/2I$ and to stop above $3\pi/2I$. Thus, for $I = D$, we have the same filter specifications, so the same filter could serve both the decimation and interpolation problems. □

□ **EXAMPLE 9.14** Design a decimation FIR filter for the signal $x_1(n)$ in Example 9.12 that has a better stopband attenuation of $A_s = 50$ dB and a lower filter order.

Solution The signal bandwidth is $\omega_{x,{\rm p}} = \pi/8$, and we will choose $\omega_{x,{\rm s}} = \pi/D = \pi/2$. Then $\omega_{\rm p} = \pi/8$ and $\omega_{\rm s} = (2\pi/D) - \omega_{x,{\rm s}} = \pi/2$. With these parameters the optimum FIR filter length is 13, which is much lower than the previous one of 37 with a higher attenuation.

MATLAB script:

```
% Given Parameters:
D = 2; Rp = 0.1; As = 50; wxp = pi/8; wxs = pi/D; wp = wxp; ws = (2*pi/D)-wxs;
```

```
% Filter Design
[delta1,delta2] = db2delta(Rp,As);
[N,F,A,weights] = firpmord([wp,ws]/pi,[1,0],[delta1,delta2],2); N = ceil(N/2)*2;
h = firpm(N,F,A,weights); delay = N/2; % Delay imparted by the filter

Hf1 = figure('units','inches','position',[1,1,6,4],...
    'paperunits','inches','paperposition',[0,0,6,4]);

% Input signal x(n) = cos(2*pi*n/16)
n = [0:256]; x = cos(pi*n/8);
n1 = n(1:33); x1 = x(33:65); % for plotting purposes
subplot(2,2,1); Hs1 = stem(n1,x1,'filled'); set(Hs1,'markersize',2,'color','g');
set(gca,'xtick',[0:8:32],'ytick',[-1,0,1]);
axis([-2,34,-1.2,1.2]); ylabel('Amplitude','vertical','cap');
xlabel('n','vertical','middle');
Title('Input Signal: x(n) = cos(\pin/8)','fontsize',TF,'vertical','baseline');

% Decimation of x(n): D = 2
y = upfirdn(x,h,1,D);
m = delay+1:1:128/D+delay+1; y1 = y(m); m = 0:16; y1 = y1(14:30);
subplot(2,2,3); Hs2 = stem(m,y1,'filled'); set(Hs2,'markersize',2,'color','m');
axis([-1,17,-1.2,1.2]); set(gca,'xtick',[0:8:32]/D,'ytick',[-1,0,1]);
title('Output signal y(n): D=2','fontsize',TF,'vertical','baseline');
xlabel('m','vertical','middle');  ylabel('Amplitude','vertical','cap');

% Filter Design Plots
[Hr,w,a,L] = Hr_Type1(h); Hr_min = min(Hr); w_min = find(Hr == Hr_min);
H = abs(freqz(h,1,w)); Hdb = 20*log10(H/max(H)); min_attn = Hdb(w_min);
subplot(2,2,2); plot(w/pi,Hr,'m','linewidth',1.0); axis([0,1,-0.1,1.1]);
set(gca,'xtick',[0,wp/pi,ws/pi,1],'ytick',[0,1]); grid;
ylabel('Amplitude','vertical','cap');
title('Amplitude Response','fontsize',TF,'vertical','baseline');
xlabel('Frequency in \pi units','vertical','middle');
subplot(2,2,4); plot(w/pi,Hdb,'m','linewidth',1.0); axis([0,1,-60,10]);
set(gca,'xtick',[0,wp/pi,ws/pi,1],'ytick',[-60,round(min_attn),0]); grid
ylabel('Decibels','vertical','cap');
xlabel('Frequency in \pi units','vertical','middle');
title('Log-magnitude Response','fontsize',TF,'vertical','baseline');
```

The signal stem plots and the filter responses are shown in Figure 9.27. The designed filter achieves an attenuation of 51 dB, and the decimated signal is correct. □

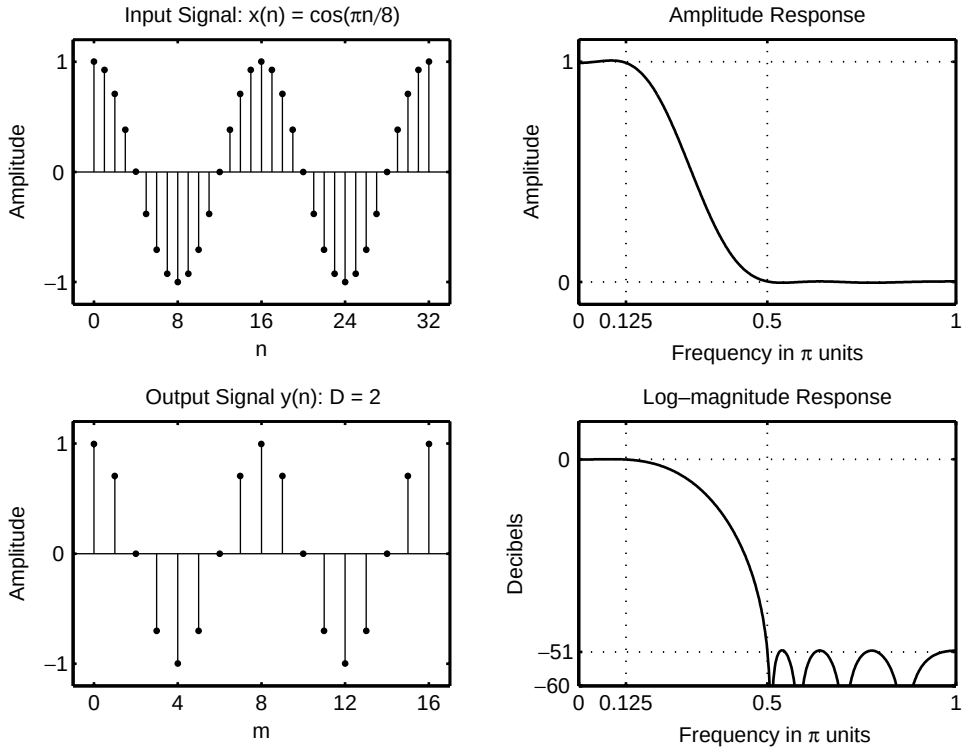


FIGURE 9.27 *Signal plots and filter design plots in Example 9.14*

9.5.5 FIR RATIONAL-FACTOR RATE CONVERSION

Replacing the ideal filter of the system given on page 494 with an FIR filter $H(\omega)$, we obtain the system shown in Figure 9.28. In this case the relevant ideal lowpass filter is given by (9.36), which is repeated here for convenience.

$$H(\omega) = \begin{cases} I, \ 0 \le |\omega| \le \min(\pi/D, \pi/I) \\ 0, \ \text{otherwise} \end{cases} \tag{9.58}$$

For the signal $x(n)$ we define

- $\omega_{x,\mathrm{p}}$ as the signal bandwidth that should be preserved
- ω_{x,s_1} as the overall signal bandwidth
- ω_{x,s_2} as the signal bandwidth that is required to be free of aliasing error after resampling

Then we have

$$0 < \omega_{x,\mathrm{p}} \le \omega_{x,\mathrm{s}_2} \le \frac{I\pi}{D} \quad \text{and} \quad \omega_{x,\mathrm{s}_1} \le \pi \tag{9.59}$$

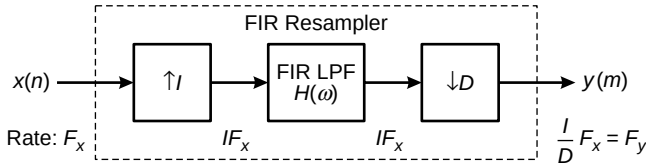


FIGURE 9.28 *An FIR rational-factor resampler*

Now for the interpolation part, the lowpass filter must pass frequencies up to $\omega_{x,\mathrm{p}}/I$ and attenuate frequencies starting at $(2\pi/I - \omega_{x,\mathrm{s}_1}/I)$. The decimation part of the filter must again pass frequencies up to $\omega_{x,\mathrm{p}}/I$ but attenuate frequencies above $(2\pi/D - \omega_{x,\mathrm{s}_2}/I)$. Therefore, the stopband must start at the lower of these two values. Defining filter cutoff frequencies as

$$\omega_{\mathrm{p}} = \left(\frac{\omega_{x,\mathrm{p}}}{I}\right) \quad \text{and} \quad \omega_{\mathrm{s}} = \min\left[\frac{2\pi}{I} - \frac{\omega_{x,\mathrm{s}_1}}{I}, \frac{2\pi}{D} - \frac{\omega_{x,\mathrm{s}_2}}{I}\right] \tag{9.60}$$

and the corresponding ripple parameters as δ_1 and δ_2, we have the following filter specifications:

$$\boxed{\begin{aligned} &\frac{1}{I}H_{\mathrm{r}}(\omega) \le 1 \pm \delta_1 \ \text{ for } |\omega| \in [0, \omega_{\mathrm{p}}] \\ &\frac{1}{I}H_{\mathrm{r}}(\omega) \le \pm\delta_2 \quad \text{ for } |\omega| \in [\omega_{\mathrm{s}}, \pi] \end{aligned}} \tag{9.61}$$

where $H_{\mathrm{r}}(\omega)$ is the amplitude response. Note that if we set $\omega_{x,\mathrm{s}_1} = \pi$ and $\omega_{x,\mathrm{s}_2} = I\pi/D$, which are their maximum values, then we get the ideal cutoff frequency $\max[\pi/I, \pi/D]$, as given before in (9.36).

MATLAB Implementation Clearly, the `upfirdn` function implements all the necessary operations needed in the rational sampling rate conversion system shown in Figure 9.28. When invoked as `y = upfirdn(x,h, I,D)`, it performs a cascade of three operations: upsampling the input data array `x` by a factor of the integer I, FIR filtering the upsampled signal data with the impulse response sequence given in the array `h`, and finally downsampling the result by a factor of the integer D. Using a well designed filter, we have a complete control over the sampling rate conversion operation.

□ **EXAMPLE 9.15** Design a sampling rate converter that increases the sampling rate by a factor of 2.5. Use the `firpm` algorithm to determine the coefficients of the FIR filter that has 0.1 dB ripple in the passband and is down by at least 30 dB in the stopband.

Solution The FIR filter that meets the specifications of this problem is exactly the same as the filter designed in Example 9.8. Its bandwidth is $\pi/5$. □

□ **EXAMPLE 9.16** A signal $x(n)$ has a total bandwidth of 0.9π. It is resampled by a factor of $4/3$ to obtain $y(m)$. We want to preserve the frequency band up to 0.8π and require that the band up to 0.7π be free of aliasing. Using the Parks-McClellan algorithm, determine the coefficients of the FIR filter that has 0.1 dB ripple in the passband and 40 dB attenuation in the stopband.

Solution The overall signal bandwidth is $\omega_{x,\mathrm{s}_1} = 0.9\pi$, the bandwidth to be preserved is $\omega_{x,\mathrm{p}} = 0.8\pi$, and the bandwidth above which aliasing is tolerated is $\omega_{x,\mathrm{s}_2} = 0.7\pi$. From (9.60) and using $I = 4$ and $D = 3$, the FIR filter design parameters are $\omega_p = 0.2\pi$ and $\omega_s = 0.275\pi$. With these parameters, along with the passband ripple of 0.1 dB and stopband attenuation of 40 dB, the optimum FIR filter length is 58. The details and computation of design plots follow.

```
% Given Parameters:
I = 4; D = 3; Rp = 0.1; As = 40;
wxp = 0.8*pi; wxs1 = 0.9*pi; wxs2 = 0.7*pi;
% Computed Filter Parameters
wp = wxp/I; ws = min((2*pi/I-wxs1/I),(2*pi/D-wxs2/I));
% Filter Design
[delta1,delta2] = db2delta(Rp,As);
[N,F,A,weights] = firpmord([wp,ws]/pi,[1,0],[delta1,delta2],2);
N = ceil(N/2)*2+1; h = firpm(N,F,I*A,weights);
Hf1 = figure('units','inches','position',[1,1,6,3],...
    'paperunits','inches','paperposition',[0,0,6,3]);
% Filter Design Plots
[Hr,w,a,L] = Ampl_res(h); Hr_min = min(Hr); w_min = find(Hr == Hr_min);
H = abs(freqz(h,1,w)); Hdb = 20*log10(H/max(H)); min_attn = Hdb(w_min);
subplot(2,1,1); plot(w/pi,Hr,'m','linewidth',1.0); axis([0,1,-0.1,I+0.1]);
set(gca,'xtick',[0,wp/pi,ws/pi,1],'ytick',[0,I]); grid;
ylabel('Amplitude','vertical','cap');
title('Amplitude Response','fontsize',TF,'vertical','baseline');
xlabel('Frequency in \pi units','vertical','middle');
subplot(2,1,2); plot(w/pi,Hdb,'m','linewidth',1.0); axis([0,1,-60,10]);
set(gca,'xtick',[0,wp/pi,ws/pi,1],'ytick',[-60,round(min_attn),0]); grid
ylabel('Decibels','vertical','cap');
xlabel('Frequency in \pi units','vertical','middle');
title('Log-magnitude Response','fontsize',TF,'vertical','baseline');
```

The filter responses are shown in Figure 9.29, which shows that the designed filter achieves the attenuation of 40 db. □

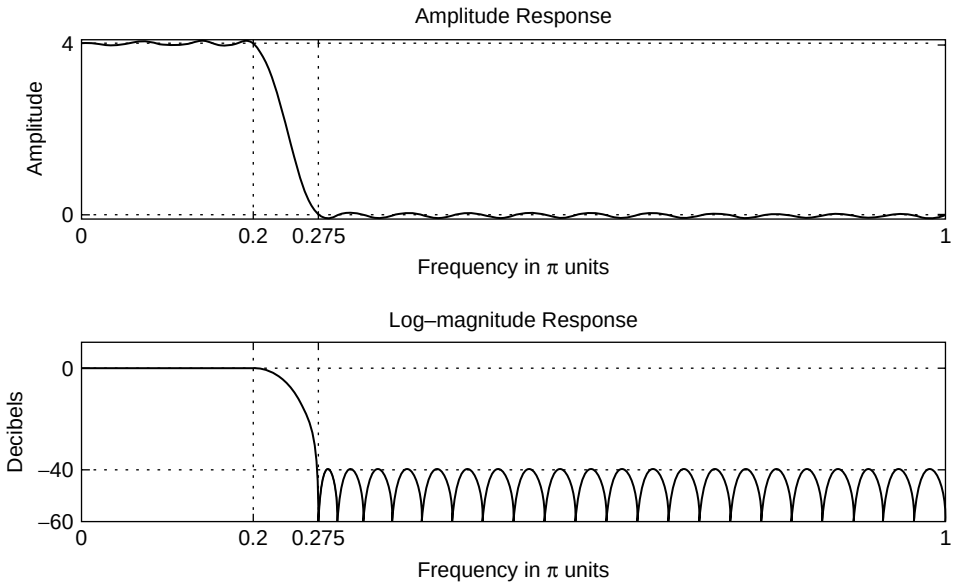


FIGURE 9.29 *The filter design plots in Example 9.16*

9.5.6 FIR FILTERS WITH MULTIPLE STOPBANDS

We now discuss the use of multiple stopbands in the design of FIR integer interpolators when the low sampling rate is more than two times that required. Let us refer back to the Figure 9.22b on page 506, which illustrates a typical spectrum $V(\omega)$ in integer interpolators. We could use a lowpass filter with multiple stopbands of bandwidth ω_s/I centered at $2\pi k/I$ for $k \neq 0$. For $I = 4$, such a spectrum is shown in Figure 9.30(a), and the corresponding filter specifications are shown in Figure 9.30b.

Clearly, these filter specifications differ from those given in (9.52) on page 506 in that the stopband is no longer one contiguous interval. Now if $\omega_s < \pi/2$, then there is a practical advantage to using this multiband design because it results in a lower order filter [2]. For $\pi \geq \omega_s > \pi/2$, the single-band lowpass filter specification (9.52) is easier and works as well.

Similar advantages can be obtained for FIR integer decimators. We again find that we can substitute a multiple stopband lowpass filter for the single stopband design given in (9.57). With reference to the signal specifications on page 513, we note that only part of the bands $[\pi/D, 3\pi/D]$, $[3\pi/D, 5\pi/D]$, ... etc. will get aliased into $[-\omega_s, +\omega_s]$. Therefore, the multiple stopbands are given by $[(2\pi/D) - \omega_s, (2\pi/D) + \omega_s]$, $[(4\pi/D) - \omega_s, (4\pi/D) + \omega_s]$, etc., centered at $2\pi k/D$, $k \neq 0$. Once again there are practical advantages when $\omega_s < \pi/2M$.

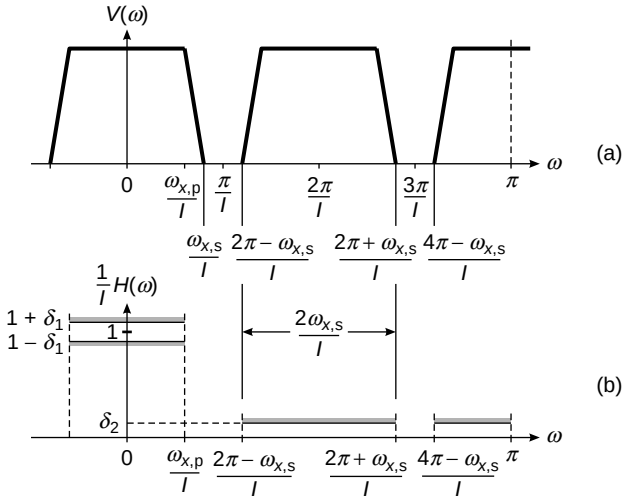


FIGURE 9.30 *Multiple stopband design: (a) signal spectrum, (b) filter specifications*

9.6 FIR FILTER STRUCTURES FOR SAMPLING RATE CONVERSION

As indicated in the discussion in section 9.4, sampling rate conversion by a factor I/D can be achieved by first increasing the sampling rate by I, accomplished by inserting $I-1$ zeros between successive values of the input signal $x(n)$, followed by linear filtering of the resulting sequence to eliminate the unwanted images of $X(\omega)$, and finally by downsampling the filtered signal by the factor D. In this section we consider the design and implementation of the linear filter. We begin with the simplest structure, which is the direct-form FIR filter structure, and develop its computationally efficient implementation. We then consider another computationally efficient structure called the *polyphase* structure, which is used in the implementation of the MATLAB functions `resample` and `upfirdn`. Finally, we close this section by discussing the time-variant filter structures for the general case of sampling rate conversion.

9.6.1 DIRECT-FORM FIR FILTER STRUCTURES

In principle, the simplest realization of the filter is the direct-form FIR structure with system function

$$H(z) = \sum_{k=0}^{M-1} h(k) z^{-k} \tag{9.62}$$

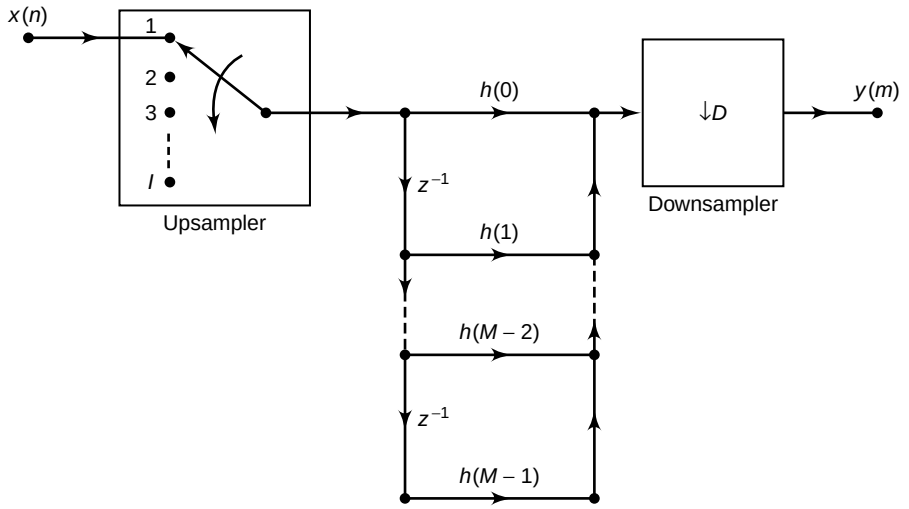


FIGURE 9.31 *Direct-form realization of FIR filter in sampling rate conversion by a factor I/D*

where $h(k)$ is the unit sample response of the FIR filter. After designing the filter as discussed in the previous section, we will have the filter parameters $h(k)$, which allow us to implement the FIR filter directly, as shown in Figure 9.31.

Although the direct-form FIR filter realization illustrated in Figure 9.31 is simple, it is also very inefficient. The inefficiency results from the fact that the upsampling process introduces $I-1$ zeros between successive points of the input signal. If I is large, most of the signal components in the FIR filter are zero. Consequently, most of the multiplications and additions result in zeros. Furthermore, the downsampling process at the output of the filter implies that only one out of every D output samples is required at the output of the filter. Consequently, only one out of every D possible values at the output of the filter should be computed.

To develop a more efficient filter structure, let us begin with a decimator that reduces the sampling rate by an integer factor D. From our previous discussion, the decimator is obtained by passing the input sequence $x(n)$ through an FIR filter and then downsampling the filter output by a factor D, as illustrated in Figure 9.32a. In this configuration, the filter is operating at the high sampling rate F_x, while only one out of every D output samples is actually needed. The logical solution to this inefficiency problem is to embed the downsampling operation within the filter, as illustrated in the filter realization given in Figure 9.32b. In this filter structure, all the multiplications and additions are performed at the

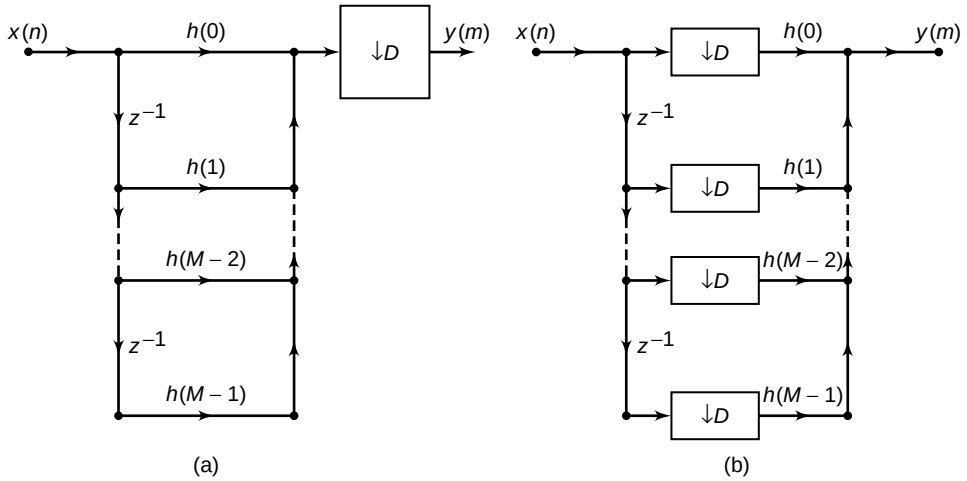


FIGURE 9.32 *Decimation by a factor D: (a) standard realization, (b) efficient realization*

lower sampling rate F_x/D. Thus, we have achieved the desired efficiency. Additional reduction in computation can be achieved by exploiting the symmetry characteristics of $\{h(k)\}$. Figure 9.33 illustrates an efficient realization of the decimator in which the FIR filter has linear phase and hence $\{h(k)\}$ is symmetric.

Next, let us consider the efficient implementation of an interpolator, which is realized by first inserting $I - 1$ zeros between samples of $x(n)$ and then filtering the resulting sequence. The direct-form realization is illustrated in Figure 9.34. The major problem with this structure is that the filter computations are performed at the high sampling rate of IF_x. The desired simplification is achieved by first using the transposed form of the FIR filter, as illustrated in Figure 9.35a, and then embedding the upsampler within the filter, as shown in Figure 9.35b. Thus, all the filter multiplications are performed at the low rate F_x, while the upsampling process introduces $I-1$ zeros in each of the filter branches of the structure shown in Figure 9.35b. The reader can easily verify that the two filter structures in Figure 9.35 are equivalent.

It is interesting to note that the structure of the interpolator, shown in Figure 9.35b, can be obtained by transposing the structure of the decimator shown in Figure 9.32. We observe that the transpose of a decimator is an interpolator, and vice versa. These relationships are illustrated in Figure 9.36, where part b is obtained by transposing part a and part d is obtained by transposing part c. Consequently, a decimator is the dual of an interpolator, and vice versa. From these relationships, it follows that there is an interpolator whose structure is the dual of the decimator shown in Figure 9.33, which exploits the symmetry in $h(n)$.

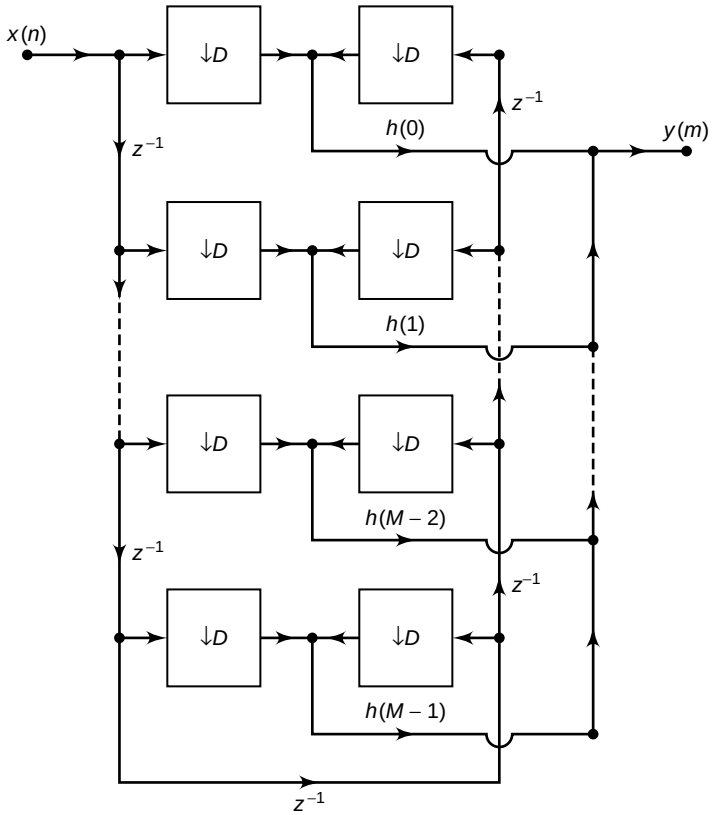


FIGURE 9.33 *Efficient realization of a decimator that exploits the symmetry in the FIR filter*

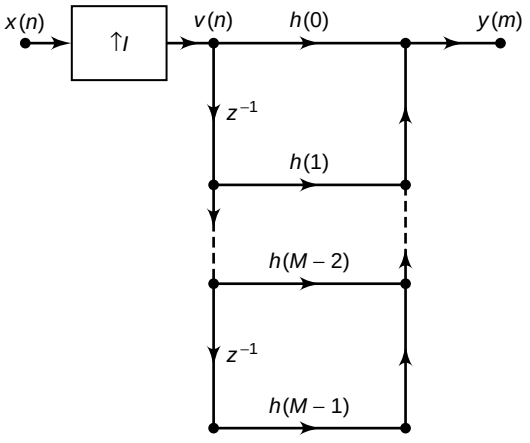


FIGURE 9.34 *Direct-form realization of FIR filter in interpolation by a factor I*

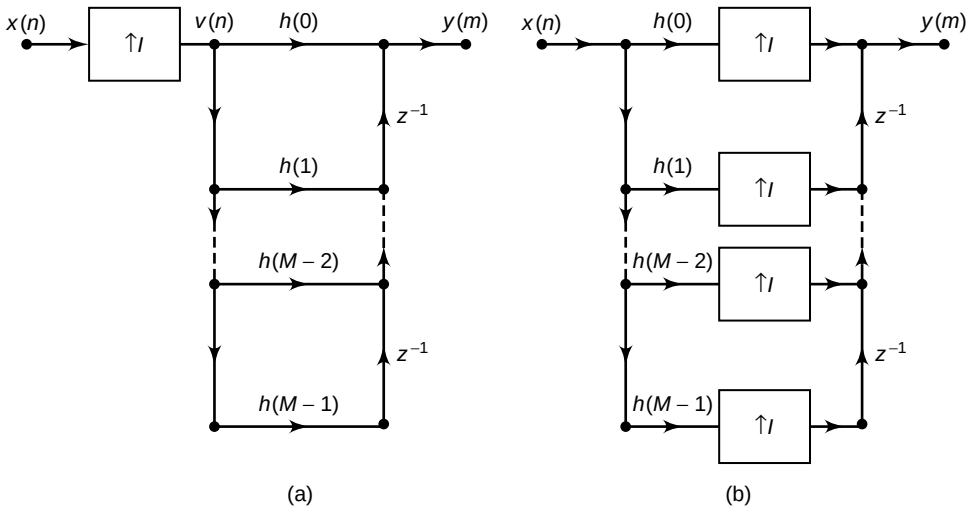


FIGURE 9.35 *Efficient realization of an interpolator*

9.6.2 POLYPHASE FILTER STRUCTURE

The computational efficiency of the filter structure shown in Figure 9.35 can also be achieved by reducing the large FIR filter of length M into a set of smaller filters of length $K = M/I$, where M is selected to be a multiple of I. To demonstrate this point, let us consider the interpolator given in Figure 9.34. Since the upsampling process inserts $I - 1$ zeros between successive values of $x(n)$, only K out of the M input values stored in the FIR filter at any one time are nonzero. At one time-instant, these nonzero values coincide and are multiplied by the filter coefficients $h(0), h(I), h(2I), \ldots, h(M - I)$. In the following time instant, the nonzero values of the input sequence coincide and are multiplied by the filter coefficients $h(1), h(I + 1), h(2I + 1)$, and so on. This observation leads us

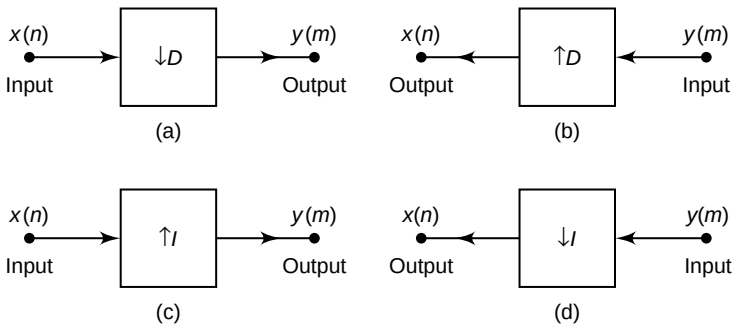


FIGURE 9.36 *Duality relationships obtained through transpositions*

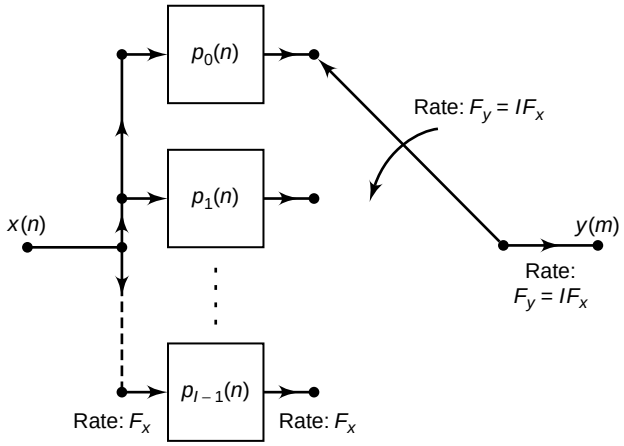


FIGURE 9.37 *Interpolation by use of polyphase filters*

to define a set of smaller filters, called polyphase filters, with unit sample responses

$$p_k(n) = h(k + nI); \quad k = 0, 1, \ldots, I-1, \quad n = 0, 1, \ldots, K-1 \qquad \textbf{(9.63)}$$

where $K = M/I$ is an integer.

From this discussion it follows that the set of I polyphase filters can be arranged as a parallel realization, and the output of each filter can be selected by a commutator, as illustrated in Figure 9.37. The rotation of the commutator is in the counterclockwise direction, beginning with the point at $m = 0$. Thus, the polyphase filters perform the computations at the low sampling rate F_x, and the rate conversion results from the fact that I output samples are generated, one from each of the filters, for each input sample.

The decomposition of $\{h(k)\}$ into the set of I subfilters with impulse response $p_k(n), k = 0, 1, \ldots, I-1$ is consistent with our previous observation that the input signal was being filtered by a periodically time-variant linear filter with impulse response

$$g(n, m) = h(nI + (mD)_I) \qquad \textbf{(9.64)}$$

where $D = 1$ in the case of the interpolator. We noted previously that $g(n, m)$ varies periodically with period I. Consequently, a different set of coefficients is used to generate the set of I output samples $y(m), m = 0, 1, \ldots, I-1$.

Additional insight can be gained about the characteristics of the set of polyphase subfilters by noting that $p_k(n)$ is obtained from $h(n)$ by decimation with a factor I. Consequently, if the original filter frequency

response $H(\omega)$ is flat over the range $0 \leq |\omega| \leq \omega/I$, each of the polyphase subfilters possesses a relatively flat response over the range $0 \leq |\omega| \leq \pi$ (i.e., the polyphase subfilters are basically allpass filters and differ primarily in their phase characteristics). This explains the reason for using the term *polyphase* in describing these filters.

The polyphase filter can also be viewed as a set of I subfilters connected to a common delay line. Ideally, the kth subfilter will generate a forward time shift of $(k/I)T_x$, for $k = 0, 1, 2, \ldots, I-1$, relative to the zeroth subfilter. Therefore, if the 0th filter generates zero delay, the frequency response of the kth subfilter is

$$p_k(\omega) = e^{j\omega k/I}$$

A time shift of an integer number of input sampling intervals (e.g., kT_x) can be generated by shifting the input data in the delay line by I samples and using the same subfilters. By combining these two methods, we can generate an output that is shifted forward by an amount $(k + i/I)T_x$ relative to the previous output.

By transposing the interpolator structure in Figure 9.37, we obtain a commutator structure for a decimator based on the parallel bank of polyphase filters, as illustrated in Figure 9.38. The unit sample responses of the polyphase filters are now defined as

$$p_k(n) = h(k + nD); \quad k = 0, 1, \ldots, D-1, \quad n = 0, 1, \ldots, K-1 \quad \textbf{(9.65)}$$

where $K = M/D$ is an integer when M is selected to be a multiple of D. The commutator rotates in a counterclockwise direction, starting with the filter $p_0(n)$ at $m = 0$.

Although the two commutator structures for the interpolator and the decimator just described rotate in a counterclockwise direction, it is also possible to derive an equivalent pair of commutator structures having a clockwise rotation. In this alternative formulation, the sets of polyphase filters are defined to have impulse responses

$$p_k(n) = h(nI - k), \quad k = 0, 1, \ldots, I-1 \quad \textbf{(9.66)}$$

and

$$p_k(n) = h(nD - k), \quad k = 0, 1, \ldots, D-1 \quad \textbf{(9.67)}$$

for the interpolator and decimator, respectively,

□ **EXAMPLE 9.17** For the decimation filter designed in Example 9.11, determine the polyphase filter coefficients $\{p_k(n)\}$ in terms of the FIR filter coefficients $\{h(n)\}$

Solution The polyphase filters obtained from $h(n)$ have impulse responses

$$p_k(n) = h(2n + k) \quad k = 0, 1; \quad n = 0, 1, \ldots, 14$$

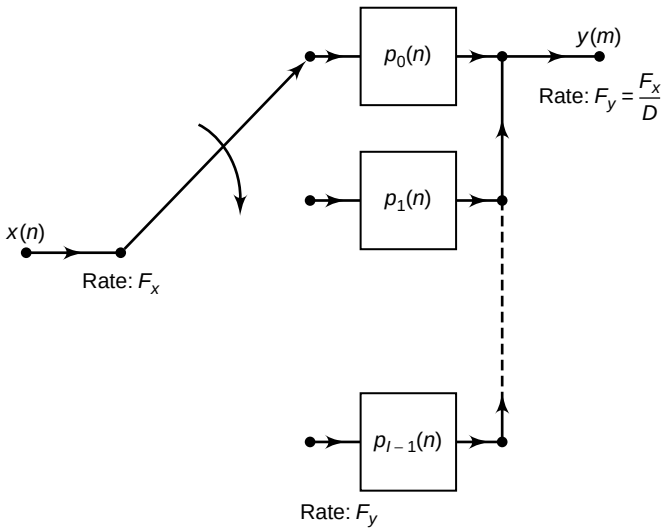


FIGURE 9.38 *Decimation by use of polyphase filters*

Note that $p_0(n) = h(2n)$ and $p_1(n) = h(2n + 1)$. Hence one filter consists of the even-numbered samples of $h(n)$, and the other filter consists of the odd-numbered samples of $h(n)$. □

□ **EXAMPLE 9.18** For the interpolation filter designed in Example 9.8, determine the polyphase filter coefficients $\{p_k(n)\}$ in terms of the filter coefficients $\{h(n)\}$.

Solution The polyphase filters obtained from $h(n)$ have impulse responses

$$p_k(n) = h(5n + k) \quad k = 0, 1, 2, 3, 4$$

Consequently, each filter has length 6. □

9.6.3 TIME-VARIANT FILTER STRUCTURES

Having described the filter implementation for a decimator and an interpolator, let us now consider the general problem of sampling rate conversion by the factor I/D. In the general case of sampling rate conversion by a factor I/D, the filtering can be accomplished by means of the linear time-variant filter described by the response function

$$g(n, m) = h[nI - ((mD))_I] \tag{9.68}$$

where $h(n)$ is the impulse response of the low-pass FIR filter, which ideally, has the frequency response specified by (9.36). For convenience

we select the length of the FIR filter $\{h(n)\}$ to a multiple of I (i.e., $M = KI$). As a consequence, the set of coefficients $\{g(n, m)\}$ for each $m = 0, 1, 2, \ldots, I-1$, contains K elements. Since $g(n, m)$ is also periodic with period I, as demonstrated in (9.44), it follows that the output $y(m)$ can be expressed as

$$y(m) = \sum_{n=0}^{K-1} g\left(n, m - \left\lfloor \frac{m}{I} \right\rfloor I\right) x\left(\left\lfloor \frac{mD}{I} \right\rfloor - n\right) \qquad \textbf{(9.69)}$$

Conceptually, we can think of performing the computations specified by (9.69) by processing blocks of data of length K by a set of K filter coefficients $g(n, m - \lfloor m/I \rfloor I), n = 0, 1, \ldots, K-1$. There are I such sets of coefficients, one set for each block of I output points of $y(m)$. For each block of I output points, there is a corresponding block of D input points of $x(n)$ that enter in the computation.

The block processing algorithm for computing (9.69) can be visualized as illustrated in Figure 9.39. A block of D input samples is buffered and shifted into a second buffer of length K, one sample at a time. The shifting from the input buffer to the second buffer occurs at a rate of one sample each time the quantity $\lfloor mD/I \rfloor$ increases by one. For each output sample $y(l)$, the samples from the second buffer are multiplied by the corresponding set of filter coefficients $g(n, l)$ for $n = 0, 1, \ldots, K-1$, and the K products are accumulated to give $y(l)$, for $l = 0, 1, \ldots, I-1$.

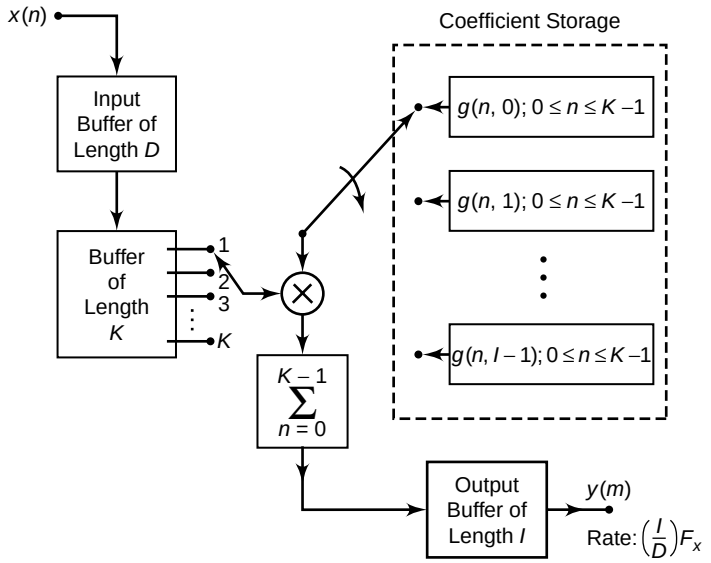


FIGURE 9.39 *Efficient implementation of sampling rate conversion by block processing*

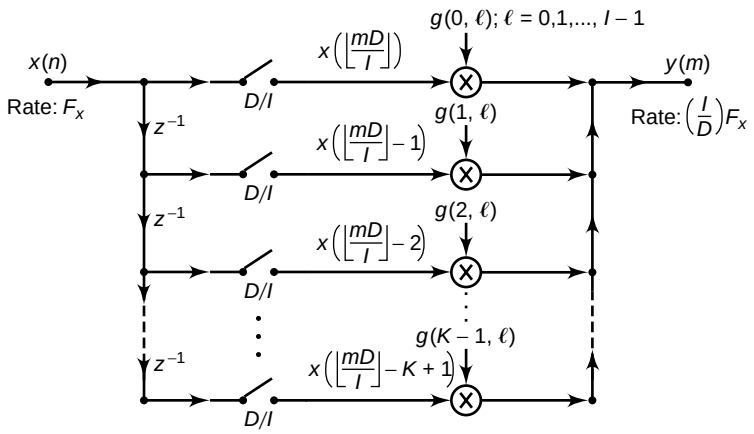


FIGURE 9.40 *Efficient realization of sampling rate conversion by a factor I/D*

Thus this computation produces I outputs. It is then repeated for a new set of D input samples, and so on.

An alternative method for computing the output of the sampling rate converter, specified by (9.69), is by means of an FIR filter structure with periodically varying filter coefficients. Such a structure is illustrated in Figure 9.40. The input samples $x(n)$ are passed into a shift register that operates at the sampling rate F_x and is of length $K = M/I$, where M is the length of the time-invariant FIR filter specified by the frequency response given by (9.36). Each stage of the register is connected to a hold-and-sample device that serves to couple the input sampling rate F_x to the output sampling rate $F_y = (I/D)F_x$. The sample at the input to each hold-and-sample device is held until the next input sample arrives and then is discarded. The output samples on the hold-and-sample device are taken at times $mD/I, m = 0, 1, 2, \ldots$. When both the input and output sampling times coincide (i.e., when mD/I is an integer), the input to the hold-and-sample is changed first; then the output samples the new input. The K outputs from the K hold-and-sample devices are multiplied by the periodically time-varying coefficients $g(n, m - \lfloor m/I \rfloor I)$, for $n = 0$, $1, \ldots, K-1$, and the resulting products are summed to yield $y(m)$. The computations at the output of the hold-and-sample devices are repeated at the output sampling rate of $F_y = (I/D)F_x$.

Finally, rate conversion by a rational factor I/D can also be performed by use of a polyphase filter having I subfilters. If we assume that the mth sample $y(m)$ is computed by taking the output of the i_mth subfilter with input data $x(n), x(n-1), \ldots, x(n-K+1)$, in the delay line, the next sample $y(m+1)$ is taken from the (i_{m+1})st subfilter after shifting l_{m+1} new samples in the delay lines where $i_{m+1} = (i_m + D)_{\text{mod } I}$ and l_{m+1} is

the integer part of $(i_m + D)/I$. The integer i_{m+1} should be saved to be used in determining the subfilter from which the next sample is taken.

□ **EXAMPLE 9.19** For the sampling rate converter designed in Example 9.15, specify the set of time-varying coefficients $\{g(n, m)\}$ used in the realization of the converter based on the structure given in Figure 9.19. Also, specify the corresponding implementation based in polyphase filters.

Solution The coefficients of the filter are given by (9.43)

$$g(n, m) = h(nI + (mD)_I) = h\left(nI + mD - \left\lfloor \frac{D}{I} m \right\rfloor I\right)$$

By substituting $I = 5$ and $D = 2$, we obtain

$$g(n, m) = h\left(5n + 2m - 5\left\lfloor \frac{2m}{5} \right\rfloor\right)$$

By evaluating $g(n, m)$ for $n = 0, 1, \ldots, 5$ and $m = 0, 1, \ldots, 4$ we obtain the following coefficients for the time-variant filter:

$$\begin{aligned}
g(0, m) &= \{h(0) \quad h(2) \quad h(4) \quad h(1) \quad h(3)\} \\
g(1, m) &= \{h(5) \quad h(7) \quad h(9) \quad h(6) \quad h(8)\} \\
g(2, m) &= \{h(10) \quad h(12) \quad h(14) \quad h(11) \quad h(13)\} \\
g(3, m) &= \{h(15) \quad h(17) \quad h(19) \quad h(16) \quad h(18)\} \\
g(4, m) &= \{h(20) \quad h(22) \quad h(24) \quad h(21) \quad h(23)\} \\
g(5, m) &= \{h(25) \quad h(27) \quad h(29) \quad h(26) \quad h(28)\}
\end{aligned}$$

A polyphase filter implementation would employ five subfilters, each of length six. To decimate the output of the polyphase filters by a factor of $D = 2$ simply means that we take every other output from the polyphase filters. Thus, the first output $y(0)$ is taken from $p_0(n)$, the second output $y(1)$ is taken from $p_2(n)$, the third output is taken from $p_4(n)$, the fourth output is taken from $p_1(n)$, the fifth output is taken from $p_3(n)$, and so on. □

9.7 PROBLEMS

P9.1 Consider the upsampler with input $x(n)$ and output $v(m)$ given in (9.26). Show that the upsampler is a linear but time-varying system.

P9.2 Let $x(n) = 0.9^n u(n)$. The signal is applied to a downsampler that reduces the rate by a factor of 2 to obtain the signal $y(m)$.

1. Determine and plot the spectrum $X(\omega)$.
2. Determine and plot the spectrum $Y(\omega)$.
3. Show that the spectrum in part (2) is simply the DTFT of $x(2n)$.

P9.3 Consider a signal with spectrum

$$X(\omega) = \begin{cases} \text{nonzero}, & |\omega| \le \omega_0; \\ 0, & \omega_0 < |\omega| \le \pi. \end{cases}$$

1. Show that the signal $x(n)$ can be recovered from its samples $x(mD)$ if the sampling frequency $\omega_s \triangleq 2\pi/D \ge 2\omega_0$.
2. Sketch the spectra of $x(n)$ and $x(mD)$ for $D = 4$.
3. Show that $x(n)$ can reconstructed from the bandlimited interpolation

$$x(n) = \sum_{k=-\infty}^{\infty} x(kD)\ \text{sinc}[f_c(n - kD)]; \quad \omega_0 < 2\pi f_c < \omega_s - \omega_0, f_c = \frac{1}{D}$$

P9.4 Using the function `downsample`, study the operation of factor-of-4 downsampling on the following sequences. Use the `stem` function to plot the original and the downsampled sequences. Experiment using the default offset value of zero and the offset value equal to 2. Comment on any differences.

1. $x_1(n) = \cos(0.15\pi n),\ 0 \le n \le 100$
2. $x_2(n) = \sin(0.1\pi n) + \sin(0.4\pi n),\ 0 \le n \le 100$
3. $x_3(n) = 1 - \cos(0.25\pi n),\ 0 \le n \le 100$
4. $x_4(n) = 0.1\,n,\ 0 \le n \le 100$
5. $x_5(n) = \{0, 1, 2, 3, 4, 5, 4, 3, 2, 1\}_{\text{PERIODIC}},\ 0 \le n \le 100$

P9.5 Repeat Problem P9.4 using the factor-of-5 downsampler.

P9.6 Using the `fir2` function, generate a 101-length sequence $x(n)$ whose frequency-domain sampled values are 0.5 at $\omega = 0$, 1 at $\omega = 0.1\pi$, 1 at $\omega = 0.2$, 0 at $\omega = 0.22\pi$, and 0 at $\omega = \pi$.

1. Compute and plot the DTFT magnitude of $x(n)$.
2. Downsample $x(n)$ by a factor of 2, and plot the DTFT of the resulting sequence.
3. Downsample $x(n)$ by a factor of 4, and plot the DTFT of the resulting sequence.
4. Downsample $x(n)$ by a factor of 5, and plot the DTFT of the resulting sequence.
5. Comment on your results.

P9.7 Using the function `decimate`, study the operation of factor-of-4 decimation on the following sequences. Use the `stem` function to plot the original and the decimated sequences. Experiment, using both the default IIR and FIR decimation filters. Comment on any differences.

1. $x_1(n) = \sin(0.15\pi n),\ 0 \le n \le 100$
2. $x_2(n) = \cos(0.1\pi n) + \cos(0.4\pi n),\ 0 \le n \le 100$
3. $x_3(n) = 1 - \cos(0.25\pi n),\ 0 \le n \le 100$
4. $x_4(n) = 0.1\,n,\ 0 \le n \le 100$
5. $x_5(n) = \{0, 1, 2, 3, 4, 5, 4, 3, 2, 1\}_{\text{PERIODIC}},\ 0 \le n \le 100$

P9.8 Repeat Problem P9.7 using the 4th-order IIR filter and the 15th-order FIR decimation filters. Comment on any performance differences.

P9.9 Repeat Problem P9.7 using the factor-of-5 decimation. Comment on any differences.

P9.10 Repeat Problem P9.9 using the the 4th-order IIR filter and the 15th-order FIR decimation filters. Comment on any differences.

P9.11 Using the `fir2` function, generate a 101-length sequence $x(n)$ whose frequency-domain sampled values are 0.5 at $\omega = 0$, 1 at $\omega = 0.1\pi$, 1 at $\omega = 0.2$, 0 at $\omega = 0.22\pi$, and 0 at $\omega = \pi$.

1. Compute and plot the DTFT of $x(n)$.
2. Decimate $x(n)$ by a factor of 2, and plot the DTFT of the resulting sequence.
3. Decimate $x(n)$ by a factor of 4, and plot the DTFT of the resulting sequence.
4. Decimate $x(n)$ by a factor of 5, and plot the DTFT of the resulting sequence.
5. Comment on your results.

P9.12 Using the function `upsample`, study the operation of factor-of-4 upsampling on the following sequences. Use the `stem` function to plot the original and the upsampled sequences. Experiment using the default offset value of zero and the offset value equal to 2.

1. $x_1(n) = \sin(0.6\pi n),\ 0 \le n \le 100$
2. $x_2(n) = \sin(0.8\pi n) + \cos(0.5\pi n),\ 0 \le n \le 100$
3. $x_3(n) = 1 + \cos(\pi n),\ 0 \le n \le 100$
4. $x_4(n) = 0.2\,n,\ 0 \le n \le 100$
5. $x_5(n) = \{1, 1, 1, 1, 0, 0, 0, 0, 0, 0\}_{\text{PERIODIC}},\ 0 \le n \le 100$

P9.13 Using the `fir2` function, generate a 91-length sequence $x(n)$ whose frequency-domain sampled values are 0 at $\omega = 0$, 0.5 at $\omega = 0.1\pi$, 1 at $\omega = 0.2$, 1 at $\omega = 0.7\pi$, 0.5 at $\omega = 0.75\pi$, 0 at $\omega = 0.8\pi$, and 0 at $\omega = \pi$.

1. Compute and plot the DTFT magnitude of $x(n)$.
2. Upsample $x(n)$ by a factor of 2, and plot the DTFT magnitude of the resulting sequence.
3. Upsample $x(n)$ by a factor of 3, and plot the DTFT magnitude of the resulting sequence.
4. Upsample $x(n)$ by a factor of 4, and plot the DTFT magnitude of the resulting sequence.
5. Comment on your results.

P9.14 Using the function `interp`, study the operation of factor-of-4 interpolation on the sequences of Problem P9.12. Use the `stem` function to plot the original and the interpolated sequences. Experiment, using the filter length parameter values equal to 3 and 5. Comment on any differences in performance of the interpolation.

P9.15 Provide the frequency response plots of the lowpass filters used in the interpolators of Problem P9.14.

P9.16 Repeat Problem P9.14, using the factor-of-3 interpolation.

P9.17 Provide the frequency response plots of the lowpass filters used in the interpolators of Problem P9.16.

P9.18 Repeat Problem P9.14, using the factor-of-5 interpolation.

P9.19 Provide the frequency response plots of the lowpass filters used in the interpolators of Problem P9.18.

P9.20 Using the `fir2` function generate a 91-length sequence $x(n)$ whose frequency-domain sampled values are 0 at $\omega = 0$, 0.5 at $\omega = 0.1\pi$, 1 at $\omega = 0.2$, 1 at $\omega = 0.7\pi$, 0.5 at $\omega = 0.75\pi$, 0 at $\omega = 0.8\pi$, and 0 at $\omega = \pi$.

1. Compute and plot the DTFT of $x(n)$.
2. Interpolate $x(n)$ by a factor of 2, and plot the DTFT of the resulting sequence.
3. Interpolate $x(n)$ by a factor of 3, and plot the DTFT of the resulting sequence.
4. Interpolate $x(n)$ by a factor of 4 and plot the DTFT of the resulting sequence.
5. Comment on your results.

P9.21 Consider two sequences $x_1(n)$ and $x_2(n)$, which appear to be related.

$$x_1(n) = \max(10 - |n|, 0) \quad \text{and} \quad x_2(n) = \min(|n|, 10)$$

Use the `resample` function with default parameters.

1. Resample the sequence $x_1(n)$ at 3/2 times the original rate to obtain $y_1(m)$, and provide the `stem` plots of both sequences.
2. Resample the sequence $x_2(n)$ at 3/2 times the original rate to obtain $y_2(m)$, and provide the `stem` plots of both sequences.
3. Explain why the resampled plot of $y_2(n)$ has inaccuracies near the boundaries that $y_1(n)$ does not have.
4. Plot the frequency response of the filter used in the resampling operation.

P9.22 Let $x(n) = \cos(0.1\pi n) + 0.5\sin(0.2\pi n) + 0.25\cos(0.4\pi n)$. Use the `resample` function with default parameters.

1. Resample the sequence $x(n)$ at 4/5 times the original rate to obtain $y_1(m)$, and provide the `stem` plots of both sequences.
2. Resample the sequence $x(n)$ at 5/4 times the original rate to obtain $y_2(m)$, and provide the `stem` plots of both sequences.
3. Resample the sequence $x(n)$ at 2/3 times the original rate to obtain $y_3(m)$, and provide the `stem` plots of both sequences.
4. Explain which of the three output sequences retain the "shape" of the original sequence $x(n)$.

P9.23 Let $x(n) = \{0, 0, 0, 1, 1, 1, 1, 0, 0, 0\}_{\text{PERIODIC}}$ be a periodic sequence with period 10. Use the `resample` function for the following parts to resample the sequence $x(n)$ at 3/5 times the original rate. Consider the length of the input sequence to be 80.

1. Use the filter length parameter `L` equal to zero to obtain $y_1(m)$ and provide the `stem` plots of $x(n)$ and $y_1(m)$ sequences.
2. Use the default value of the filter length parameter `L` to obtain $y_2(m)$ and provide the `stem` plots of $x(n)$ and $y_2(m)$ sequences.
3. Use the filter length parameter `L` equal to 15 to obtain $y_3(m)$ and provide the `stem` plots of $x(n)$ and $y_3(m)$ sequences.

P9.24 Using the `fir2` function, generate a 101-length sequence $x(n)$ whose frequency-domain sampled values are 0 at $\omega = 0$, 0.5 at $\omega = 0.1\pi$, 1 at $\omega = 0.2\pi$, 1 at $\omega = 0.5\pi$, 0.5 at $\omega = 0.55\pi$, 0 at $\omega = 0.6\pi$, and 0 at $\omega = \pi$.

1. Compute and plot the DTFT of $x(n)$.
2. Resample $x(n)$ by a factor of 4/3, and plot the DTFT of the resulting sequence.
3. Resample $x(n)$ by a factor of 3/4, and plot the DTFT of the resulting sequence.
4. Resample $x(n)$ by a factor of 4/5, and plot the DTFT of the resulting sequence.
5. Comment on your results.

P9.25 We want to design a linear-phase FIR filter to increase the input sampling rate by a factor of 3 using the `intfilt` function.

1. Assuming full bandwidth of the signal to be interpolated, determine the impulse response of the required FIR filter. Plot its amplitude response and the log-magnitude response in dB. Experiment with the length parameter `L` to obtain a reasonable stopband attenuation.
2. Assuming that bandwidth of the signal to be interpolated is $\pi/2$, determine the impulse response of the required FIR filter. Plot its amplitude response and the log-magnitude response in decibels. Again experiment with the length parameter `L` to obtain a reasonable stopband attenuation.

P9.26 We want to design a linear-phase FIR filter to increase the input sampling rate by a factor of 5 using the `intfilt` function.

1. Assuming full bandwidth of the signal to be interpolated, determine the impulse response of the required FIR filter. Plot its amplitude response and the log-magnitude response in decibels. Experiment with the length parameter `L` to obtain a reasonable stopband attenuation.
2. Assuming that bandwidth of the signal to be interpolated is $4\pi/5$, determine the impulse response of the required FIR filter. Plot its amplitude response and the log-magnitude response in decibels. Again experiment with the length parameter `L` to obtain a reasonable stopband attenuation.

P9.27 Using the Parks-McClellan algorithm, design an interpolator that increases the input sampling rate by a factor of $I = 2$.

1. Determine the coefficients of the FIR filter that has 0.5 dB ripple in the passband and 50 dB attenuation in the stopband. Choose reasonable values for the band-edge frequencies.
2. Provide plots of the impulse and the log-magnitude responses.
3. Determine the corresponding polyphase structure for implementing the filter.
4. Let $x(n) = \cos(0.4\pi n)$. Generate 100 samples of $x(n)$, and process it using this filter to interpolate by $I = 2$ to obtain $y(m)$. Provide the `stem` plots of the both sequences.

P9.28 Using the Parks-McClellan algorithm, design an interpolator that increases the input sampling rate by a factor of $I = 3$.

1. Determine the coefficients of the FIR filter that has 0.1 dB ripple in the passband and 40 dB attenuation in the stopband. Choose reasonable values for the band-edge frequencies.
2. Provide plots of the impulse and the log-magnitude responses.
3. Determine the corresponding polyphase structure for implementing the filter.
4. Let $x(n) = \cos(0.3\pi n)$. Generate 100 samples of $x(n)$ and process it using this filter to interpolate by $I = 3$ to obtain $y(m)$. Provide the `stem` plots of both sequences.

P9.29 A signal $x(n)$ is to be interpolated by a factor of 3. It has a bandwidth of 0.4π, but we want to preserve frequency band up to 0.3π in the interpolated signal. Using the Parks-McClellan algorithm, we want to design such an interpolator.

1. Determine the coefficients of the FIR filter that has 0.1 dB ripple in the passband and 40 dB attenuation in the stopband.
2. Provide plots of the impulse and the log-magnitude responses.

3. Let $x(n) = \cos(0.3\pi n) + 0.5\sin(0.4\pi n)$. Generate 100 samples of $x(n)$, and process it using this filter to interpolate by $I = 3$ to obtain $y(m)$. Provide the `stem` plots of both sequences.

P9.30 A signal $x(n)$ is to be interpolated by a factor of 4. It has a bandwidth of 0.7π, but we want to preserve frequency band up to 0.6π in the interpolated signal. Using the Parks-McClellan algorithm, we want to design such an interpolator.

1. Determine the coefficients of the FIR filter that has 0.5 dB ripple in the passband and 50 dB attenuation in the stopband.
2. Provide plots of the impulse and the log-magnitude responses.
3. Let $x(n) = \sin(0.5\pi n) + \cos(0.7\pi n)$. Generate 100 samples of $x(n)$ and process it using this filter to interpolate by $I = 4$ to obtain $y(m)$. Provide the `stem` plots of both sequences.

P9.31 Using the Parks-McClellan algorithm, design a decimator that downsamples an input signal $x(n)$ by a factor of $D = 5$.

1. Determine the coefficients of the FIR filter that has 0.1 dB ripple in the passband and 30 dB attenuation in the stopband. Choose reasonable values for the band-edge frequencies.
2. Provide plots of the impulse and the log-magnitude responses.
3. Determine the corresponding polyphase structure for implementing the filter.
4. Using the `fir2` function, generate a 131-length sequence $x(n)$ whose frequency-domain sampled values are 1 at $\omega = 0$, 0.9 at $\omega = 0.1\pi$, 1 at $\omega = 0.2\pi$, 1 at $\omega = 0.5\pi$, 0.5 at $\omega = 0.55\pi$, 0 at $\omega = 0.6\pi$, and 0 at $\omega = \pi$. Process $x(n)$ using this filter to decimate it by a factor of 5 to obtain $y(m)$. Provide the spectral plots of both sequences.

P9.32 Using the Parks-McClellan algorithm, design a decimator that downsamples an input signal $x(n)$ by a factor of $D = 3$.

1. Determine the coefficients of the FIR filter that has 0.5 dB ripple in the passband and 30 dB attenuation in the stopband. Choose reasonable values for the band-edge frequencies.
2. Provide plots of the impulse and the log-magnitude responses.
3. Let $x_1(n) = \sin(0.2\pi n) + 0.2\cos(0.5\pi n)$. Generate 500 samples of $x_1(n)$, and process it using this to decimate by $D = 3$ to obtain $y_1(m)$. Provide the `stem` plots of both sequences.
4. Using the `fir2` function, generate a 131-length sequence $x_2(n)$ whose frequency-domain sampled values are 1 at $\omega = 0$, 0.8 at $\omega = 0.15\pi$, 1 at $\omega = 0.3\pi$, 1 at $\omega = 0.4\pi$, 0.5 at $\omega = 0.45\pi$, 0 at $\omega = 0.5\pi$, and 0 at $\omega = \pi$. Process $x_2(n)$, using this filter to decimate it by a factor of 3 to obtain $y_2(m)$. Provide the spectral plots of both sequences.

P9.33 A signal $x(n)$ is to be decimated by a factor of $D = 2$. It has a bandwidth of 0.4π, and we will tolerate aliasing this frequency 0.45π in the decimated signal. Using the Parks-McClellan algorithm, we want to design such a decimator.

1. Determine the coefficients of the FIR filter that has 0.1 dB ripple in the passband and 45 dB attenuation in the stopband.
2. Provide plots of the impulse and the log-magnitude responses.
3. Let $x_1(n) = \cos(0.4\pi n) + 2\sin(0.45\pi n)$. Generate 200 samples of $x_1(n)$, and process it using this filter to decimate by $D = 2$ to obtain $y_1(m)$. Provide the `stem` plots of both sequences.

4. Using the `fir2` function, generate a 151-length sequence $x_2(n)$ whose frequency-domain sampled values are 1 at $\omega = 0$, 0.9 at $\omega = 0.2\pi$, 1 at $\omega = 0.4\pi$, 0.5 at $\omega = 0.45\pi$, 0 at $\omega = 0.5\pi$, and 0 at $\omega = \pi$. Process $x_2(n)$, using this filter to decimate it by a factor of 2 to obtain $y_2(m)$. Provide the spectral plots of both sequences.

P9.34 A signal $x(n)$ is to be decimated by a factor of $D = 3$. It has a bandwidth of 0.25π, and we will tolerate aliasing this frequency 0.3π in the decimated signal. Using the Parks-McClellan algorithm, we want to design such a decimator.

1. Determine the coefficients of the FIR filter that has 0.1 dB ripple in the passband and 40 dB attenuation in the stopband.
2. Provide plots of the impulse and the log-magnitude responses.
3. Let $x_1(n) = \cos(0.2\pi n) + 2\sin(0.3\pi n)$. Generate 300 samples of $x_1(n)$, and process it using this filter to decimate by $D = 3$ to obtain $y_1(m)$. Provide the `stem` plots of both sequences.
4. Using the `fir2` function, generate a 151-length sequence $x_2(n)$ whose frequency-domain sampled values are 1 at $\omega = 0$, 1 at $\omega = 0.1\pi$, 1 at $\omega = 0.25\pi$, 0.5 at $\omega = 0.3\pi$, 0 at $\omega = 0.35\pi$, and 0 at $\omega = \pi$. Process $x_2(n)$, using this filter to decimate it by a factor of 3 to obtain $y_2(m)$. Provide the spectral plots of both sequences.

P9.35 Design a sampling rate converter that reduces the sampling rate by a factor of 2/5.

1. Using the Parks-McClellan algorithm, determine the coefficients of the FIR filter that has 0.1 dB ripple in the passband and 30 dB attenuation in the stopband. Choose reasonable values for the band-edge frequencies.
2. Provide plots of the impulse and the log-magnitude responses.
3. Specify the sets of the time-varying coefficients $g(m, n)$ and the corresponding coefficients in the polyphase filter realization.
4. Let $x(n) = \sin(0.35\pi n) + 2\cos(0.45\pi n)$. Generate 500 samples of $x(n)$ and process it using this filter to resample by 2/5 to obtain $y(m)$. Provide the `stem` plots of both sequences.

P9.36 Design a sampling rate converter that increases the sampling rate by a factor of 7/4.

1. Using the Parks-McClellan algorithm, determine the coefficients of the FIR filter that has 0.1 dB ripple in the passband and 40 dB attenuation in the stopband. Choose reasonable values for the band-edge frequencies.
2. Provide plots of the impulse and the log-magnitude responses.
3. Specify the sets of the time-varying coefficients $g(m, n)$ and the corresponding coefficients in the polyphase filter realization.
4. Let $x(n) = 2\sin(0.35\pi n) + \cos(0.95\pi n)$. Generate 500 samples of $x(n)$ and process it, using this filter to resample by 7/4 to obtain $y(m)$. Provide the `stem` plots of both sequences.

P9.37 A signal $x(n)$ is to be resampled by a factor of 3/2. It has a total bandwidth of 0.8π, but we want to preserve frequencies only up to 0.6π and require that the band up to 0.75π be free of aliasing in the resampled signal.

1. Using the Parks-McClellan algorithm, determine the coefficients of the FIR filter that has 0.5 dB ripple in the passband and 50 dB attenuation in the stopband.
2. Provide plots of the impulse and the log-magnitude responses.

3. Using the `fir2` function, generate a 101-length sequence $x(n)$ whose frequency-domain sampled values are 0.7 at $\omega = 0$, 1 at $\omega = 0.3\pi$, 1 at $\omega = 0.7\pi$, 0.5 at $\omega = 0.75\pi$, 0 at $\omega = 0.8\pi$, and 0 at $\omega = \pi$. Process $x(n)$ using this filter to resample it by 3/2 to obtain $y(m)$. Provide the spectral plots of both sequences.

P9.38 A signal $x(n)$ is to be resampled by a factor of 4/5. It has a total bandwidth of 0.8π, but we want to preserve frequencies only up to 0.5π and require that the band up to 0.75π be free of aliasing in the resampled signal.

1. Using the Parks-McClellan algorithm, determine the coefficients of the FIR filter that has 0.1 dB ripple in the passband and 40 dB attenuation in the stopband.
2. Provide plots of the impulse and the log-magnitude responses.
3. Using the `fir2` function, generate a 101-length sequence $x(n)$ whose frequency-domain sampled values are 0.7 at $\omega = 0$, 1 at $\omega = 0.3\pi$, 1 at $\omega = 0.7\pi$, 0.5 at $\omega = 0.75\pi$, 0 at $\omega = 0.8\pi$, and 0 at $\omega = \pi$. Process $x(n)$, using this filter to resample it by 4/5 to obtain $y(m)$. Provide the spectral plots of both sequences.

P9.39 A signal $x(n)$ is to be resampled by a factor of 5/2. It has a total bandwidth of 0.8π, but we want to preserve frequencies only up to 0.7π and require that the band up to 0.75π be free of aliasing in the resampled signal.

1. Using the Parks-McClellan algorithm, determine the coefficients of the FIR filter that has 0.5 dB ripple in the passband and 50 dB attenuation in the stopband.
2. Provide plots of the impulse and the log-magnitude responses.
3. Using the `fir2` function, generate a 101-length sequence $x(n)$ whose frequency-domain sampled values are 0.7 at $\omega = 0$, 1 at $\omega = 0.3\pi$, 1 at $\omega = 0.7\pi$, 0.5 at $\omega = 0.75\pi$, 0 at $\omega = 0.8\pi$, and 0 at $\omega = \pi$. Process $x(n)$ using this filter to resample it by a 5/2 to obtain $y(m)$. Provide the spectral plots of both sequences.

P9.40 A signal $x(n)$ is to be resampled by a factor of 3/8. It has a total bandwidth of 0.5π, but we want to preserve frequencies only up to 0.3π and require that the band up to 0.35π be free of aliasing in the resampled signal.

1. Using the Parks-McClellan algorithm, determine the coefficients of the FIR filter that has 0.1 dB ripple in the passband and 40 dB attenuation in the stopband.
2. Provide plots of the impulse and the log-magnitude responses.
3. Using the `fir2` function, generate a 101-length sequence $x(n)$ whose frequency-domain sampled values are 1 at $\omega = 0$, 1 at $\omega = 0.25\pi$, 1 at $\omega = 0.5\pi$, 0.5 at $\omega = 0.55\pi$, 0 at $\omega = 0.6\pi$, and 0 at $\omega = \pi$. Process $x(n)$ using this filter to resample it by 3/8 to obtain $y(m)$. Provide the spectral plots of both sequences.

CHAPTER 10

Round-off Effects in Digital Filters

In the latter part of Chapter 6 we discussed the finite-precision number representations for the purpose of implementing filtering operations on digital hardware. In particular, we focused on the process of number quantization, the resulting error characterizations, and the effects of filter coefficient quantization on filter specifications and responses. In this chapter, we further extend the effects of finite-precision numerical effects to the filtering aspects in signal processing.

We begin by discussing analog-to-digital (A/D) conversion noise using the number representations and quantization error characteristics developed in Chapter 6. We then analyze the multiplication and addition quantization (collectively known as arithmetic round-off error) models. The effects of these errors on filter output are discussed as two topics: correlated errors called *limit cycles* and uncorrelated *round-off noise*.

10.1 ANALYSIS OF A/D QUANTIZATION NOISE

From the quantizer characteristics obtained in Chapter 6, it is obvious that the quantized value $\mathcal{Q}[x]$ is a nonlinear operation on the value x. Hence the exact analysis of the finite word-length effects in digital filters is generally difficult and one has to consider less ideal analysis techniques that work well in practice.

One such technique is the statistical modeling technique. It converts the nonlinear analysis problem into a linear one and allows us to examine

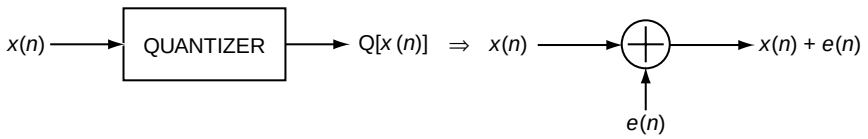


FIGURE 10.1 *Statistical model of a quantizer*

output-error characteristics. In this technique, we assume that the quantized value $\mathcal{Q}[x]$ is a sum of the exact value x and the quantization error e, which is assumed to be a random variable. When $x(n)$ is applied as an input sequence to the quantizer, the error $e(n)$ is assumed to be a random sequence. We then develop a statistical model for this random sequence to analyze its effects through a digital filter.

For the purpose of analysis, we assume that the quantizer employs fixed-point two's-complement number format representation. Using the results given previously, we can extend this analysis to other formats as well.

10.1.1 STATISTICAL MODEL

We model the quantizer block on the input as a signal-plus-noise operation—that is, from (6.46)

$$\mathcal{Q}[x(n)] = x(n) + e(n) \tag{10.1}$$

where $e(n)$ is a random sequence that describes the quantization error sequence and is termed the *quantization noise*. This is shown in Figure 10.1.

Model assumptions For the model in (10.1) to be mathematically convenient and hence practically useful, we have to assume *reasonable* statistical properties for the sequences involved. That these assumptions are practically reasonable can be ascertained using simple MATLAB examples, as we shall see. We assume that the error sequence, $e(n)$ has the following characteristics:[1]

1. The sequence $e(n)$ is a sample sequence from a stationary random process $\{e(n)\}$.
2. This random process $\{e(n)\}$ is *uncorrelated* with sequence $x(n)$.
3. The process $\{e(n)\}$ is an independent process (i.e., the samples are independent of each other).
4. The probability density function (pdf), $f_{\mathrm{E}}(e)$, of sample $e(n)$ for each n is uniformly distributed over the interval of width $\Delta = 2^{-B}$, which is the quantizer resolution.

[1]We assume that the reader is familiar with the topic of random variables and processes and the terminology associated with it.

These assumptions are reasonable in practice if the sequence $x(n)$ is sufficiently random to traverse many quantization steps in going from time n to $n+1$.

10.1.2 ANALYSIS USING MATLAB

To investigate the statistical properties of the error samples, we will have to generate a large number of these samples and plot their distribution using a histogram (or a probability bar graph). Furthermore, we have to design the sequence $x(n)$ so that its samples do not repeat; otherwise, the error samples will also repeat, which will result in an inaccurate analysis. This can be guaranteed either by choosing a well-defined aperiodic sequence or a random sequence.

We will quantize $x(n)$ using B-bit *rounding* operation. A similar implementation can be developed for the truncation operation. Since all three error characteristics are exactly the same under the rounding operation, we will choose the sign-magnitude format for ease in implementation. After quantization, the resulting error samples $e(n)$ are uniformly distributed over the $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$ interval. Let $e_1(n)$ be the normalized error given by

$$e_1(n) \triangleq \frac{e(n)}{\Delta} = e(n)\,2^B \quad \Rightarrow \quad e_1(n) \in [-1/2, 1/2] \tag{10.2}$$

Then $e_1(n)$ is uniform over the interval $[-\frac{1}{2}, +\frac{1}{2}]$, as shown in Figure 10.2a. Thus the histogram interval will be uniform across all B-bit values, which will make its computation and plotting easier. This interval will be divided into 128 bins for the purpose of plotting.

To determine the sample independence we consider the histogram of the sequence

$$e_2(n) \triangleq \frac{e_1(n) + e_1(n-1)}{2} \tag{10.3}$$

which is the average of two consecutive normalized error samples. If $e_1(n)$ is uniformly distributed between $[-1/2, 1/2]$, then, for sample independence, $e_2(n)$ must have a triangle-shaped distribution between $[-1/2, 1/2]$, as shown in Figure 10.2b. We will again generate a 128-bin histogram for $e_2(n)$. These steps are implemented in the following MATLAB function.

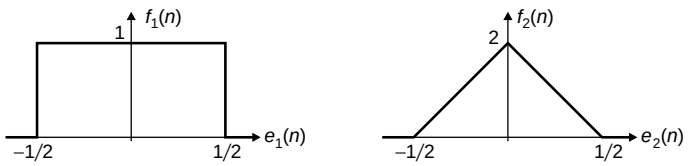


FIGURE 10.2 *Probability distributions of the normalized errors $e_1(n)$ and $e_2(n)$*

```
function [H1,H2,Q, estat] = StatModelR(xn,B,N);
% Statistical Model (Rounding) for A/D Quantization error and its Distribution
% -------------  ------------------------------------------------------------
% [H1,H2,Q] = StatModelR(xn,B,N);
%   OUT: H1 = Normalized histogram of e1
%        H2 = Normalized histogram of e2
%         Q = Normalized histogram bins
%     estat = row vector: [[e1avg,e1std,e2avg,e2std]
%    IN:  B = bits to quantize
%         N = number of samples of x(n)
%        xn = samples of the sequence
% Plot variables
  bM = 7; DbM = 2^bM;                        % bin parameter
   M = round((DbM)/2);                       % Half number of bins
bins = [-M+0.5:1:M-0.5];                     % Bin values from -M to M
   Q = bins/(DbM);                           % Normalized bins
% Quantization error analysis
    xq = (round(xn*(2^B)))/(2^B);            % Quantized to B bits
    e1 = xq-xn; clear xn xq;                 % Quantization error
    e2 = 0.5*(e1(1:N-1)+e1(2:N));            % Average of two adj errors
 e1avg = mean(e1); e1std = std(e1);          % Mean & std dev of the error e1
 e2avg = mean(e2); e2std = std(e2);          % Mean & std dev of the error e2
 estat = [e1avg,e1std,e2avg,e2std];
% Probability distribution of e1
e1 = floor(e1*(2^(B+bM)));                   % Normalized e1 (int between -M & M)
e1 = sort([e1,-M-1:1:M]);                    %
H1 = diff(find(diff(e1)))-1; clear e1;       % Error histogram
if length(H1) == DbM+1
   H1(DbM) = H1(DbM)+H1(DbM+1);
   H1 = H1(1:DbM);
 end
H1 = H1/N;                                   % Normalized histogram
% Probability distribution of e2
e2 = floor(e2*(2^(B+bM)));                   % Normalized e2 (int between -M & M)
e2 = sort([e2,-M-1:1:M]);                    %
H2 = diff(find(diff(e2)))-1; clear e2;       % Error histogram
if length(H2) == DbM+1
   H2(DbM) = H2(DbM)+H2(DbM+1);
   H2 = H2(1:DbM);
 end
H2 = H2/N;                                   % Normalized histogram
```

To validate the model assumptions, we consider the following two examples. In the first example an aperiodic sinusoidal sequence is quantized to B bits, and in the second example a random sequence is quantized to B bits. The resulting quantization errors are analyzed for their distribution properties and for their sample independence for various values of B.

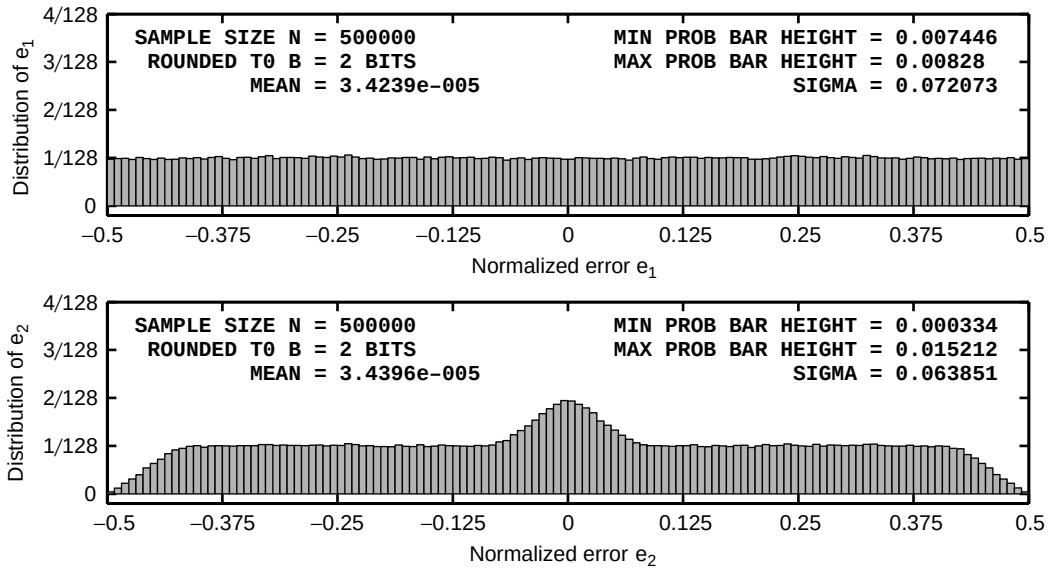


FIGURE 10.3 *A/D quantization error distribution for the sinusoidal signal in Example 10.1, $B = 2$ bits*

Through these examples we hope to learn how small error e must be (or equivalently, how large B must be) for the above assumptions to be valid.

□ **EXAMPLE 10.1** Let $x(n) = \frac{1}{3}\{\sin(n/11) + \sin(n/31) + \cos(n/67)\}$. This sequence is not periodic, and hence its samples never repeat using infinite-precision representation. However, since the sequence is of sinusoidal nature, its continuous envelope is periodic and the samples are continuously distributed over the fundamental period of this envelope. Determine the error distributions for B = 2 and 6 bits.

Solution To minimize statistical variations, the sample size must be large. We choose 500,000 samples. The following MATLAB script computes the distributions for $B = 2$ bits.

```
clear; close all;
% Example parameters
B = 2; N = 500000; n = [1:N];
xn = (1/3)*(sin(n/11)+sin(n/31)+cos(n/67)); clear n;
% Quantization error analysis
[H1,H2,Q, estat]] = StatModelR(xn,B,N);    % Compute histograms
H1max = max(H1); H1min = min(H1);  % Max and Min of H1
H2max = max(H2); H2min = min(H2);  % Max and Min of H2
```

The plots of the resulting histogram are shown in Figure 10.3. Clearly, even though the error samples appear to be uniformly distributed, the samples are not independent. The corresponding plots for $B = 6$ bits are shown in

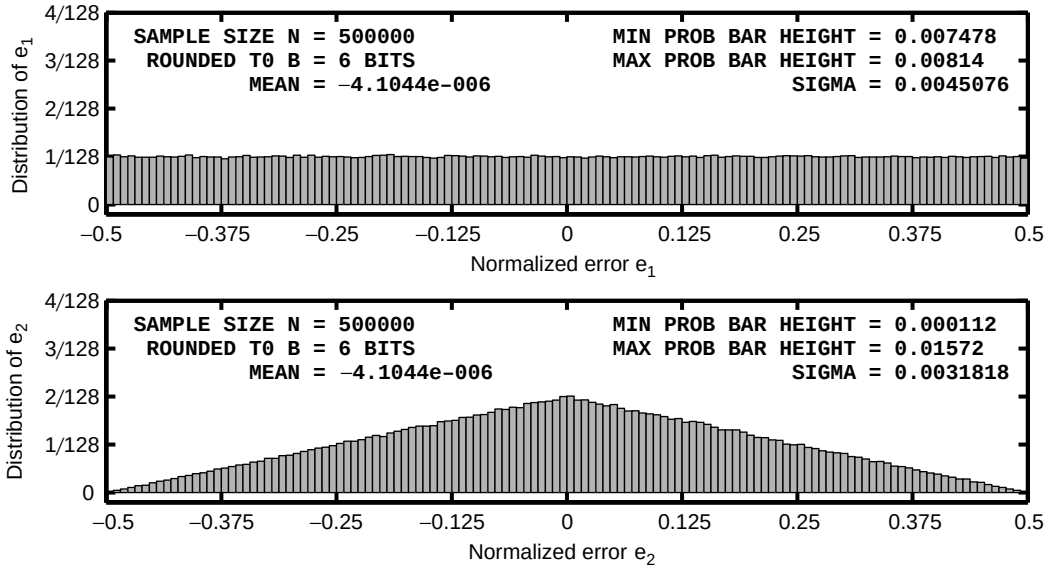


FIGURE 10.4 *Quantization error distribution for the sinusoidal signal in Example 10.1, $B = 6$ bits*

Figure 10.4, from which we observe that the quantization error sequence appears to satisfy the model assumptions for $B \geq 6$ bits. □

□ **EXAMPLE 10.2** Let $x(n)$ be an independent and identically distributed random sequence whose samples are uniformly distributed over the $[-1, 1]$ interval. Determine the error distributions for B = 2 and 6 bits.

Solution We again choose 500,000 samples to minimize any statistical variations. The following MATLAB fragment computes the distributions for $B = 2$ bits.

```
clear; close all;
% Example parameters
B = 2; N = 500000; xn = (2*rand(1,N)-1);
% Quantization error analysis
[H1,H2,Q, estat]] = StatModelR(xn,B,N);    % Compute histograms
H1max = max(H1); H1min = min(H1);  % Max and Min of H1
H2max = max(H2); H2min = min(H2);  % Max and Min of H2
```

The plots of the resulting histogram are shown in Figure 10.5. The corresponding plots for $B = 6$ bits are shown in Figure 10.6. From these plots we observe that even for $B = 2$ bits the quantization error samples are independent and uniformly distributed. □

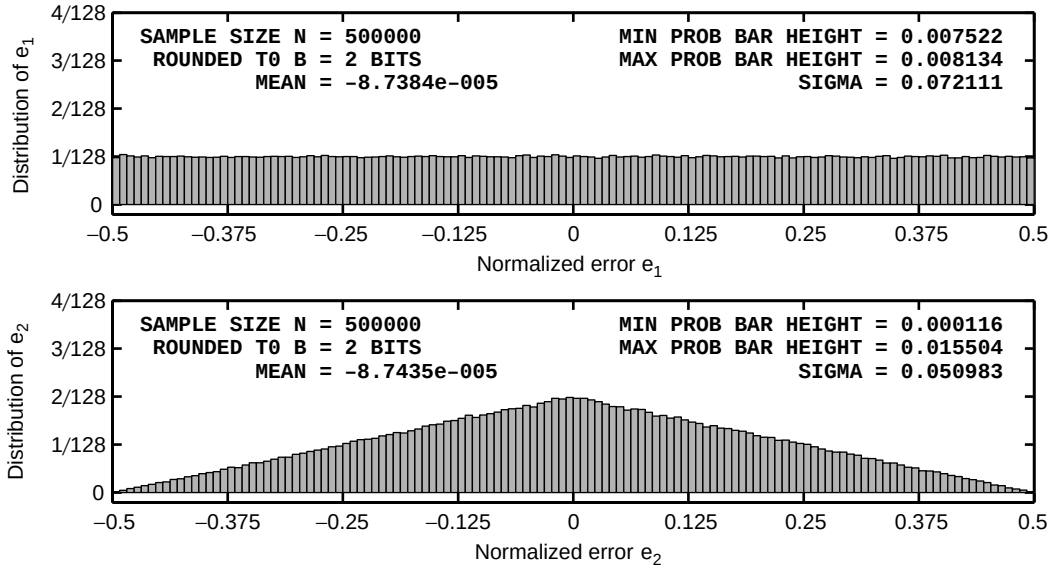


FIGURE 10.5 *A/D quantization error distribution for the random signal in Example 10.2, $B = 2$ bits*

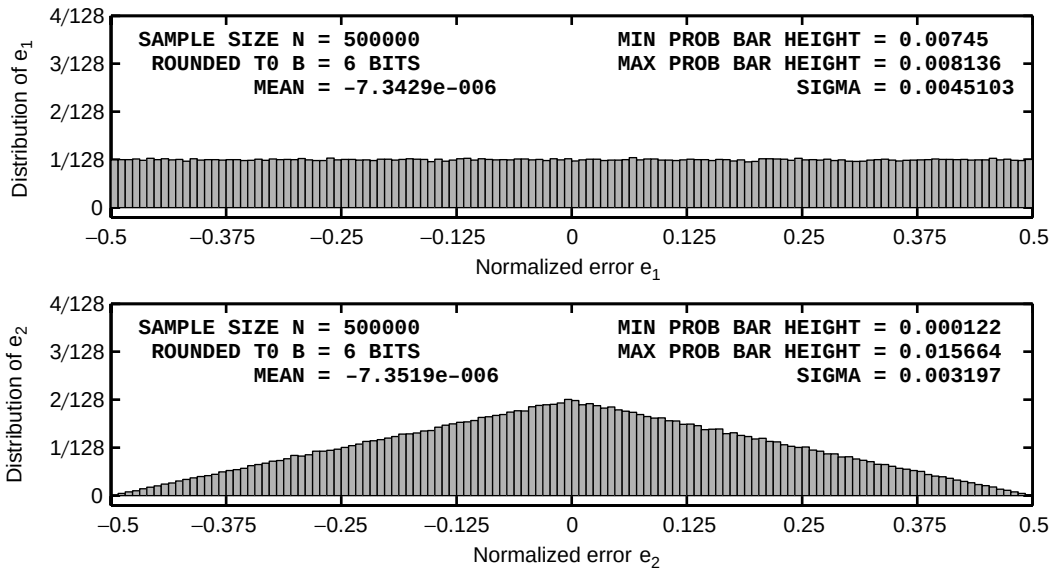


FIGURE 10.6 *Quantization error distribution for the random signal in Example 10.2, $B = 6$ bits*

Since practical signals processed using a DSP chip are typically random in nature (or can be modeled as such), we conclude from these two examples that the statistical model, with its stated assumptions, is a very good model.

10.1.3 STATISTICS OF A/D QUANTIZATION NOISE

We now develop a second-order statistical description of the error sequence $e(n)$ for both the truncation and rounding operations.

10.1.4 TRUNCATION

From (6.57), the pdf $f_{\mathrm{E_T}}(e)$ of $e_{\mathrm{T}}(n)$ is uniform over $[-\Delta, 0]$, as shown in Figure 10.7a. Then the average of $e_{\mathrm{T}}(n)$ is given by

$$m_{e_{\mathrm{T}}} \triangleq \mathrm{E}[e_{\mathrm{T}}(n)] = -\Delta/2 \tag{10.4}$$

and the variance is

$$\begin{aligned}\sigma^2_{e_{\mathrm{T}}} &\triangleq \mathrm{E}\left[\left(e_{\mathrm{T}}(n) - m_{e_{\mathrm{T}}}\right)^2\right] = \int_{-\Delta}^{0} (e - \Delta/2)^2 f_{\mathrm{E_T}}(e)\,\mathrm{d}e \\ &= \int_{-\Delta/2}^{\Delta/2} e^2 \left(\frac{1}{\Delta}\right) \mathrm{d}e = \frac{\Delta^2}{12}\end{aligned} \tag{10.5}$$

Using $\Delta = 2^{-B}$, we obtain

$$\sigma^2_{e_{\mathrm{T}}} = \frac{2^{-2B}}{12} \quad \text{or} \quad \sigma_{e_{\mathrm{T}}} = \frac{2^{-B}}{2\sqrt{3}} \tag{10.6}$$

Rounding From (6.59), the pdf $f_{E_{\mathrm{R}}}(e)$ of $e_{\mathrm{R}}(n)$ is uniform over $[-\Delta/2, \Delta/2]$, as shown in Figure 10.7b. Then the average of $e_{\mathrm{R}}(n)$ is given by

$$m_{e_{\mathrm{R}}} \triangleq [\mathrm{E}e_{\mathrm{R}}] = 0 \tag{10.7}$$

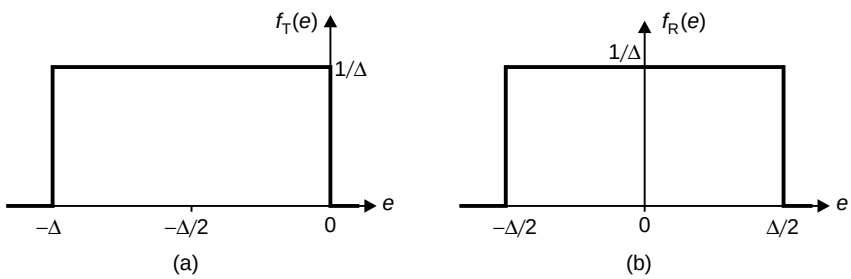


FIGURE 10.7 *Probability density functions: (a) truncation and (b) rounding*

and the variance is

$$\sigma_{e_{\mathrm{R}}}^2 \triangleq \mathrm{E}\left[(e_{\mathrm{R}}(n) - m_{e_{\mathrm{R}}})^2\right] = \int_{-\Delta/2}^{\Delta/2} e^2 f_{E_{\mathrm{R}}}(e)\,\mathrm{d}e = \int_{-\Delta/2}^{\Delta/2} e^2 \left(\frac{1}{\Delta}\right)\mathrm{d}e$$
$$= \frac{\Delta^2}{12} \tag{10.8}$$

Using (6.45), we obtain

$$\sigma_{e_{\mathrm{R}}}^2 = \frac{2^{-2B}}{12} \quad \text{or} \quad \sigma_{e_{\mathrm{R}}} = \frac{2^{-B}}{2\sqrt{3}} \tag{10.9}$$

Since the samples of the sequence $e_{\mathrm{R}}(n)$ are assumed to be independent of each other, the variance of $[e_{\mathrm{R}}(n) + e_{\mathrm{R}}(n-1)]/2$ is given by

$$\mathrm{var}\left[\frac{e_{\mathrm{R}}(n) + e_{\mathrm{R}}(n-1)}{2}\right] = \frac{1}{4}\left(\frac{2^{-2B}}{12} + \frac{2^{-2B}}{12}\right) = \frac{2^{-2B}}{24} = \frac{1}{2}\sigma_{e_{\mathrm{R}}}^2 \tag{10.10}$$

or the standard deviation is $\sigma_{e_{\mathrm{R}}}/\sqrt{2}$.

From the model assumptions and (10.6) or (10.9), the covariance of the error sequence (which is an independent sequence) is given by

$$\mathrm{E}[e(m)e(n)] \triangleq C_e(m-n) \triangleq C_e(\ell) = \frac{2^{-2B}}{12}\delta(\ell) \tag{10.11}$$

where $\ell \triangleq m - n$ is called the lag variable. Such an error sequence is also known as a white noise sequence.

10.1.5 MATLAB IMPLEMENTATION

In MATLAB, the sample mean and standard deviation are computed using the functions `mean` and `std`, respectively. The last argument of the function `StatModelR` is a vector containing sample means and standard deviations of unnormalized errors $e(n)$ and $[e(n) + e(n-1)]/2$. Thus, these values can be compared with the theoretical values obtained from the statistical model.

□ **EXAMPLE 10.3** The plots in Example 10.1 also indicate the sample means and standard deviations of the errors $e(n)$ and $[e(n) + e(n-1)]/2$. For $B = 2$, these computed values are shown in Figure 10.3. Since $e(n)$ is uniformly distributed over the interval $[-2^{-3}, 2^{-3}]$, its mean value is 0, and so is the mean of $[e(n)+e(n-1)]/2$. The computed values are 3.4239×10^{-5} and 3.4396×10^{-5}, respectively, which agree fairly well with the model. The standard deviation of $e(n)$, from (10.9), is 0.072169, while that from the top plot in Figure 10.3 is 0.072073, which again agrees closely with the model. The standard deviation of the average of the two consecutive samples, from (10.10), is 0.051031, and from the bottom plot in Figure 10.3 it is 0.063851, which clearly does not agree with the model. Hence the samples of $e(n)$ for $B = 2$ are not independent. This was confirmed by the bottom plot in Figure 10.3.

Similarly, for $B = 6$ computed statistical values are shown in Figure 10.4. The computed values of the two means are -4.1044×10^{-6}, which agree very well with the model. The standard deviation of $e(n)$, from (10.9), is 0.0045105, while that from the top plot in Figure 10.4 is 0.0045076, which again agrees closely with the model. The standard deviation of the average of the two consecutive samples, from (10.10), is 0.0031894, while from the bottom plot in Figure 10.4 it is 0.00318181, which clearly agrees with the model. Hence the samples of $e(n)$ for $B = 6$ are independent. This was also confirmed by the bottom plot in Figure 10.4. □

Similar calculations can be carried out for the signal in Example 10.2. The details are left to the reader.

10.1.6 A/D QUANTIZATION NOISE THROUGH DIGITAL FILTERS

Let a digital filter be described by the impulse response, $h(n)$, or the frequency response, $H(\mathrm{e}^{\jmath\omega})$. When a quantized input, $\mathcal{Q}[x(n)] = x(n)+e(n)$, is applied to this system, we can determine the effects of the error sequence $e(n)$ on the filter output as it propagates through the filter, assuming infinite-precision arithmetic implementation in the filter. We are generally interested in the mean and variance of this output-noise sequence, which we can obtain using linear system theory concepts. Details of these results can be found in many introductory texts on random processes, including [27].

Referring to Figure 10.8, let the output of the filter be $\hat{y}(n)$. Using LTI properties and the statistical independence between $x(n)$ and $e(n)$, the output $\hat{y}(n)$ can be expressed as the sum of two components. Let $y(n)$ be the (true) output due to $x(n)$ and $q(n)$ the response due to $e(n)$. Then we can show that $q(n)$ is also a random sequence with mean

$$m_q \stackrel{\triangle}{=} \mathrm{E}[q(n)] = m_e \sum_{-\infty}^{\infty} h(n) = m_e\, H(\mathrm{e}^{\jmath 0}) \tag{10.12}$$

where the term $H(\mathrm{e}^{\jmath 0})$ is termed the *DC gain* of the filter. For truncation, $m_{e_\mathrm{T}} = -\Delta/2$, which gives

$$m_{q_\mathrm{T}} = -\frac{\Delta}{2} H(e^{\jmath 0}) \tag{10.13}$$

For rounding, $m_{e_\mathrm{R}} = 0$ or

$$m_{q_\mathrm{R}} = 0 \tag{10.14}$$

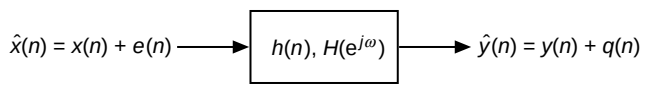


FIGURE 10.8 *Noise through digital filter*

We can also show that the variance of $q(n)$, for both the truncation or rounding, is given by

$$\sigma_q^2 = \sigma_e^2 \sum_{-\infty}^{\infty} |h(n)|^2 = \frac{\sigma_e^2}{2\pi} \int_{-\pi}^{\pi} |H(\mathrm{e}^{\jmath\omega})|^2 \, \mathrm{d}\omega \tag{10.15}$$

The *variance gain* from the input to the output (also known as the *normalized output variance*) is the ratio

$$\frac{\sigma_q^2}{\sigma_e^2} = \sum_{-\infty}^{\infty} |h(n)|^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} |H(\mathrm{e}^{\jmath\omega})|^2 \, \mathrm{d}\omega \tag{10.16}$$

For a real and stable filter, using the substitution $z = \mathrm{e}^{\jmath\omega}$, the integral in (10.16) can be further expressed as a complex contour integral

$$\int_{-\pi}^{\pi} |H(\mathrm{e}^{\jmath\omega})|^2 \, \mathrm{d}\omega = \frac{1}{2\pi\jmath} \oint_{\mathrm{UC}} H(z)H(z^{-1})z^{-1}\mathrm{d}z \tag{10.17}$$

where UC is the unit circle and can be computed using residues (or the inverse $\mathcal{Z}$-transform) as

$$\int_{-\pi}^{\pi} |H(\mathrm{e}^{\jmath\omega})|^2 \, \mathrm{d}\omega = \sum [\text{Residues of } H(z)H(z^{-1})z^{-1} \text{ inside UC}] \tag{10.18a}$$

$$= \mathcal{Z}^{-1}\left[H(z)H(z^{-1})\right]\big|_{n=0} \tag{10.18b}$$

10.1.7 MATLAB IMPLEMENTATION

Computation of the variance-gain for the A/D quantization noise can be carried out in MATLAB using (10.16) and (10.18). For FIR filters, we can perform exact calculations using the time-domain expression in (10.16). In the case of IIR filters, exact calculations can only be done using (10.18) in special cases, as we shall see (fortunately, this works for most practical filters). The approximate computations can always be done using the time-domain expression.

Let the FIR filter be given by the coefficients $\{b_k\}_0^{M-1}$. Then using the time-domain expression in (10.16), the variance-gain is given by

$$\frac{\sigma_q^2}{\sigma_e^2} = \sum_{k=0}^{M-1} |b_k|^2 \tag{10.19}$$

Let an IIR filter be given by the system function

$$H(z) = \frac{\sum_{\ell=0}^{N-1} b_\ell z^{-\ell}}{1 + \sum_{k=1}^{N-1} a_k z^{-k}} \tag{10.20}$$

with impulse response $h(n)$. If we assume that the filter is real, causal, and stable and has only simple poles, then using the partial fraction expansion, we can write

$$H(z) = R_0 + \sum_{k=1}^{N-1} \frac{R_k}{z - p_k} \tag{10.21}$$

where R_0 is the constant term and R_k's are the residues at the pole locations p_k. This expansion can be computed using the `residue` function. Note that both poles and the corresponding residues are either real-valued or occur in complex-conjugate pairs. Then using (10.18a), we can show that (see [17] and also Problem P10.3)

$$\frac{\sigma_q^2}{\sigma_e^2} = R_0^2 + \sum_{k=1}^{N-1}\sum_{\ell=1}^{N-1} \frac{R_k R_\ell^*}{1 - p_k p_\ell^*} \tag{10.22}$$

The variance-gain expression in (10.22) is applicable for most practical filters since rarely do they have multiple poles. The approximate value of the variance-gain for IIR filters is given by

$$\frac{\sigma_q^2}{\sigma_e^2} \simeq \sum_{k=0}^{K-1} |h(n)|^2, \quad K \gg 1 \tag{10.23}$$

where K is chosen so that the impulse response values (magnitudewise) are almost zero beyond K samples. The following MATLAB function, `VarGain`, computes variance-gain using (10.19) or (10.22).

```
function Gv = VarGain(b,a)
% Computation of variance-gain for the output noise process
% of digital filter described by b(z)/a(z)
%  Gv = VarGain(b,a)
a0 = a(1); a = a/a0; b = b/a0; M = length(b); N = length(a);
if N == 1                              % FIR Filter
    Gv = sum(b.*b);
    return
else                                   % IIR Filter
    [R,p,P] = residue(b,a);
    if length(P) > 1
        error('*** Variance Gain Not computable ***');
    elseif length(P) == 1
        Gv = P*P;
    else
        Gv = 0;
    end
    Rnum = R*R'; pden = 1-p*p';
    H = Rnum./pden; Gv = Gv + real(sum(H(:)));
end
```

It should be noted that the actual output noise variance is obtained by multiplying the A/D quantization noise variance by the variance-gain.

□ **EXAMPLE 10.4** Consider an 8-order IIR filter with poles at $p_k = r\,e^{j2\pi k/8}$, $k = 0, \ldots, 7$. If r is close to 1, then the filter has 4 narrowband peaks. Determine the variance-gain for this filter when $r = 0.9$ and $r = 0.99$.

Solution The following MATLAB script illustrates calculations for $r = 0.9$, which implements exact as well as approximate approaches.

```
% Filter Parameters
N = 8; r = 0.9; b = 1; pl = r*exp(j*2*pi*[0:N-1]/N); a = real(poly(pl));

% Variance-gain (Exact)
Vg = VarGain(b,a)
Vg =
   1.02896272593178
% Variance-Gain (approximate)
x = [1,zeros(1,10000)]; % Unit sample sequence
h = filter(b,a,x);      % Impulse response
VgCheck = sum(h.*h)
VgCheck =
   1.02896272593178
```

Clearly, both approaches give the same variance-gain, which for $r = 0.9$ is about 3% above unity. For $r = 0.99$ the calculations are:

```
% Filter Parameters
N = 8; r = 0.99; b = 1; pl = r*exp(j*2*pi*[0:N-1]/N); a = real(poly(pl));
% Variance-gain (Exact)
Vg = VarGain(b,a)
Vg =
   6.73209233071894
```

The variance-gain is more than 673%, which means that when poles are close to the unit circle, the filter output can be very noisy. □

10.2 ROUND-OFF EFFECTS IN IIR DIGITAL FILTERS

With our insight into the quantizer operation and its simpler statistical model, we are now ready to delve into the analysis of finite word-length effects in both IIR and FIR digital filters. We have already studied the effects of input signal quantization and filter coefficient quantization on filter behavior. We will now turn our attention to the effects of arithmetic operation quantization on filter output responses (in terms of signal-to-noise ratios). For this study we will consider both fixed-point and floating-point arithmetic. We first consider the effects on IIR filters since, due to feedback paths, the results are more complicated—yet more interesting—than those in FIR filters. The effects on FIR filters are studied in the next section.

We will restrict ourselves to the rounding operation of the quantizer due to its superior statistical qualities (no bias or average value). From (6.59), we know that, for the rounding operation, the quantizer error, e_{R}, has the same characteristics across all three number representation formats. Hence for MATLAB simulation purposes, we will consider the sign-magnitude format because it is easy to program and simulate for arithmetic operation. However, in practice, two's-complement format number representation has advantages over the others in terms of hardware implementation.

Digital filter implementation requires arithmetic operations of multiplication and addition. If two B-bit fractional numbers are multiplied, the result is a $2B$-bit fractional number that must be quantized to B bits. Similarly, if two B-bit fractional numbers are added, the sum could be more than one, which results in an *overflow*, which in itself is a nonlinear characteristic; or the sum must be corrected using a *saturation* strategy, which is also nonlinear. Thus, a finite word-length implementation of the filter is a highly nonlinear filter and must be analyzed carefully for any meaningful results.

In this section, we will consider two approaches to deal with errors due to finite word-length representation. The first type of error can occur when error samples become *correlated* with each other due to the nonlinearity of the quantizer. This is called *limit-cycle behavior*, and it can exist only in IIR filters. We will analyze this problem using the nonlinear quantizer model rather than the statistical model of the quantizer. In the second type of error, we assume that more nonlinear effects in the quantizer have been suppressed. Then, using the statistical model of the quantizer, we develop a quantization noise model for IIR filters that is more useful in predicting the finite word-length effects.

10.2.1 LIMIT CYCLES

Digital filters are linear systems, but when quantizers are incorporated in their implementation, they become nonlinear systems. For nonlinear systems it is possible to have an output sequence even when there is no input. Limit cycles is one such behavior that creates an oscillatory periodic output that is highly undesirable.

DEFINITION 1 ***Limit cycle***

A zero-input limit cycle is a nonzero periodic output sequence produced by nonlinear elements or quantizers in the feedback loop of a digital filter. □

There are two types of limit cycles. The *granular* limit cycles are due to nonlinearities in multiplication quantization and are of low amplitude. The *overflow* limit cycles are a result of overflow in addition and can have large amplitudes.

10.2.2 GRANULAR LIMIT CYCLES

This type of limit cycle can easily be demonstrated with a simple rounding quantizer following a multiplication. We illustrate with the following example.

□ **EXAMPLE 10.5** Consider a simple 1st-order IIR filter given by

$$y(n) = \alpha\, y(n-1) + x(n); \quad y(-1) = 0, \quad n \geq 0 \tag{10.24}$$

Let $\alpha = -\frac{1}{2}$; then this is a highpass filter, since its pole is near $z = -1$. Determine the output $y(n)$ when $x(n) = \frac{7}{8}\delta(n)$, assuming a 3-bit quantizer in the multiplier.

Solution After multiplication by α, we have to quantize the result. Let the output due to this quantization be $\hat{y}(n)$. Then the actual implementable digital filter is

$$\hat{y}(n) = \mathcal{Q}\left[-\frac{1}{2}\hat{y}(n-1)\right] + x(n); \quad \hat{y}(-1) = 0, \quad n \geq 0 \tag{10.25}$$

We assume that the input in (10.24) is quantized and that there is no overflow due to addition. Let $B = 3$ (that is, we have 3 fraction bits and 1 sign bit) and let $x(n) = \frac{7}{8}\delta(n)$. Now $\alpha = -\frac{1}{2}$ is represented by $1\blacktriangle 110$ in two's-complement format. Hence the output sequence is obtained as:

$$\begin{aligned}
\hat{y}(0) &= x(0) &&&&&&= +\frac{7}{8} &&: 0\blacktriangle 111\\
\hat{y}(1) &= \mathcal{Q}\left[\alpha\,\hat{y}(0)\right] &&= \mathcal{Q}\left[-\frac{1}{2}\left(+\frac{7}{8}\right)\right] &&= \mathcal{Q}\left[-\frac{7}{16}\right] &&= -\frac{1}{2} &&: 1\blacktriangle 100\\
\hat{y}(2) &= \mathcal{Q}\left[\alpha\,\hat{y}(1)\right] &&= \mathcal{Q}\left[-\frac{1}{2}\left(-\frac{1}{2}\right)\right] &&= \mathcal{Q}\left[+\frac{1}{4}\right] &&= +\frac{1}{4} &&: 0\blacktriangle 010\\
\hat{y}(3) &= \mathcal{Q}\left[\alpha\,\hat{y}(2)\right] &&= \mathcal{Q}\left[-\frac{1}{2}\left(+\frac{1}{4}\right)\right] &&= \mathcal{Q}\left[-\frac{1}{8}\right] &&= -\frac{1}{8} &&: 1\blacktriangle 111\\
\hat{y}(4) &= \mathcal{Q}\left[\alpha\,\hat{y}(3)\right] &&= \mathcal{Q}\left[-\frac{1}{2}\left(-\frac{1}{8}\right)\right] &&= \mathcal{Q}\left[+\frac{1}{16}\right] &&= +\frac{1}{8} &&: 0\blacktriangle 001\\
\hat{y}(5) &= \mathcal{Q}\left[\alpha\,\hat{y}(4)\right] &&= \mathcal{Q}\left[-\frac{1}{2}\left(+\frac{1}{8}\right)\right] &&= \mathcal{Q}\left[-\frac{1}{16}\right] &&= -\frac{1}{8} &&: 1\blacktriangle 111\\
\vdots\ \vdots &&&\vdots &&\vdots &&\vdots
\end{aligned} \tag{10.26}$$

Thus $\hat{y}(n) = \pm\frac{1}{8}$ for $n \geq 5$. The desired output $y(n)$ is

$$y(n) = \left\{\frac{7}{8}, -\frac{7}{16}, \frac{7}{32}, -\frac{7}{64}, \frac{7}{128}, \cdots, \rightarrow 0\right\} \tag{10.27}$$

Hence the error sequence is

$$e(n) = \hat{y}(n) - y(n) = \left\{0, -\frac{1}{16}, \frac{1}{32}, -\frac{1}{64}, \frac{9}{128}, \cdots, \rightarrow \pm\frac{1}{8}\right\} \tag{10.28}$$

This shows that the error $e(n)$ slowly builds up to $\pm\frac{1}{8}$. Hence the error is *asymptotically periodic* with period 2. □

From Example 10.5, it is clear that, in the steady state, the system has poles on the unit circle and hence the nonlinear system has effectively become a linear system [12]. This implies that, effectively, for the system in (10.24)

$$\mathcal{Q}\left[\alpha\hat{y}(n-1)\right] = \begin{cases} \hat{y}(n-1), & \alpha > 0, \\ -\hat{y}(n-1), & \alpha < 0. \end{cases} \tag{10.29}$$

Also due to the rounding operation, the quantization error is bounded by $\pm\Delta/2$ where $\Delta = 2^{-B}$ is the quantization step, or

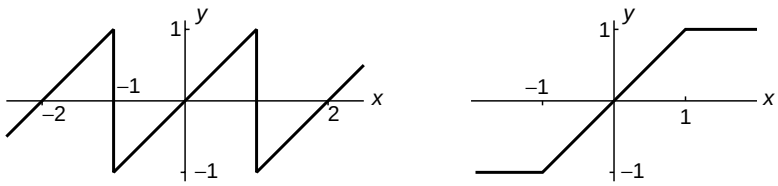
$$\left|\mathcal{Q}\left[\alpha\hat{y}(n-1)\right] - \alpha\hat{y}(n-1)\right| \leq \frac{\Delta}{2} \tag{10.30}$$

From (10.29) and (10.30), we conclude that

$$|\hat{y}(n-1)| \leq \frac{\Delta}{2(1-|\alpha|)} \tag{10.31}$$

which is the amplitude range of limit-cycle oscillations and is called a *dead band*. For the system in Example 10.5, $B = 3$ and $\alpha = -\frac{1}{2}$. Hence the dead-band range is $\pm\frac{1}{8}$, which agrees with (10.31). If the output $\hat{y}(n-1)$ gets trapped in this band when the input is zero, the filter exhibits the granular limit cycle. From (10.29), the period of the oscillation is either 1 or 2.

Analysis using MATLAB In our previous MATLAB simulations, we did not worry about the quantization in multiplication or addition operations because the emphasis was on either signal quantization or on filter coefficient quantization. The important operation that we have to consider is the arithmetic overflow characteristics. We assume that the represented numbers are in fractional two's-complement format. Then in practice, two overflow characteristics are used: a two's-complement overflow, which is a modulo (periodic) function, and a saturation, which is a limiting function. These characteristics are shown in Figure 10.9.



(a) Two's-complement Overflow (b) Saturation

FIGURE 10.9 *Overflow characteristics used in* `Qfix`

To simulate these two effects, we provide the function `y = Qfix(x,B, 'Qmode','Omode')`. This function performs a fixed-point two's-complement format quantization using `(B+1)`-bit representation so that the resulting number `y` is between $-1 \leq y < 1$. The quantization mode, `Qmode`, is either a rounding or a truncation operation. The overflow characteristic is provided in `Omode`. Using this function, we can study both types of limit cycles.

```
function [y] = QFix(x,B,Qmode,Omode)
% Fixed-point Arithmetic using (B+1)-bit Representation
% ------------------------------------------------------
%   [y] = QFix(x,B,Qmode,Omode)
%      y: Decimal equivalent of quantized x with values in [-1,1)
%      x: a real number array
%      B: Number of fractional bits
%  Qmode: Quantizer mode
%         'round': two's-complement rounding characteristics
%         'trunc': Two's complement truncation characteristics
%  Omode: Overflow mode
%         'satur': Saturation limiter
%         'twosc': Two's-complement overflow
% Quantization operation
if strcmp(lower(Qmode), 'round');
    y = round(x.*(2^B));
elseif strcmp(lower(Qmode), 'trunc');
    y = floor(x.*(2^B));
else
    error('Use Qmode = "round" or "trunc"');
end;
y = y*(2^(-B)); % (B+1)-bit representation
% Overflow operation
if strcmp(lower(Omode), 'satur');
    y = min(y,1-2^(-B)); y = max(-1,y); % Saturation
elseif strcmp(lower(Omode), 'twosc');
    y = 2*(mod(y/2-0.5,1)-0.5);         % Overflow
else error('Use Omode = "satur" or "twosc"');
end;
```

□ **EXAMPLE 10.6** In this example simulate the results for the system given in Example 10.5 using the `Qfix` function with $B = 3$ bits. In addition, also examine limit-cycle behavior for the truncation operation in the multiplier and for the case when the system is a lowpass filter with coefficient $\alpha = 0.5$.

Solution The MATLAB scripts:

```
% Highpass filter, rounding operation in multiplier
a = -0.5; yn1 = 0; m = 0:10; y = [yn1, zeros(1,length(m))];
x = 0.875*impseq(m(1),m(1)-1,m(end));
for n = m+2
   yn1 = y(n-1);
   y(n) = QFix(a*yn1,3,'round','satur') + x(n);
end

subplot('position',[0.08,0.2,0.24,0.6]);
plot([-1,20],[0,0],'w'); axis([-1,10,-1,1]); hold on;
Hs_1 = stem([-1,m],y,'filled');set(Hs_1,'markersize',3,'color',[0,1,0]);
set(gca,'ytick',[-1:0.25:1],'fontsize',6); ylabel('Amplitude','fontsize',8);
title('\alpha = -0.5, Rounding','fontsize',10);
xlabel('Sample index n','fontsize',8);
% Lowpass filter, rounding operation in multiplier
a = 0.5; yn1 = 0; m = 0:10; y = [yn1, zeros(1,length(m))];
x = 0.875*impseq(m(1),m(1)-1,m(end));
for n = m+2
   yn1 = y(n-1);
   y(n) = QFix(a*yn1,3,'round','satur') + x(n);
end

subplot('position',[0.42,0.2,0.24,0.6]);
plot([-1,20],[0,0],'w'); axis([-1,10,-1,1]); hold on;
Hs_1 = stem([-1,m],y,'filled');set(Hs_1,'markersize',3,'color',[0,1,0]);
set(gca,'ytick',[-1:0.25:1],'fontsize',6); ylabel('Amplitude','fontsize',8);
title('\alpha = 0.5, Rounding','fontsize',10);
xlabel('Sample index n','fontsize',8);

% Highpass filter, Truncation operation in multiplier
a = -0.5; yn1 = 0; m = 0:10; y = [yn1, zeros(1,length(m))];
x = 0.875*impseq(m(1),m(1)-1,m(end));
for n = m+2
   yn1 = y(n-1);
   y(n) = QFix(a*yn1,3,'trunc','satur') + x(n);
end

subplot('position',[0.76,0.2,0.24,0.6]);
plot([-1,20],[0,0],'w'); axis([-1,10,-1,1]); hold on;
```

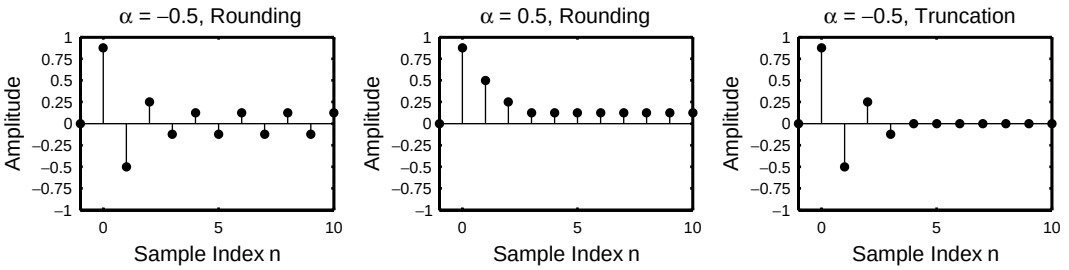


FIGURE 10.10 *Granular limit cycles in Example 10.6*

```
Hs_1 = stem([-1,m],y,'filled');set(Hs_1,'markersize',3,'color',[0,1,0]);
set(gca,'ytick',[-1:0.25:1],'fontsize',6); ylabel('Amplitude','fontsize',8);
title('\alpha = -0.5, Truncation','fontsize',10);
xlabel('Sample index n','fontsize',8);
```

The resulting plots are shown in Figure 10.10. The output signal in the left plot agrees with that in Example 10.5 and has an asymptotic period of two samples. The middle plot for $\alpha = 0.5$ (lowpass filter) shows that the limit cycle has a period of one sample with amplitude of $\frac{1}{8}$. Finally, the right plot shows that the limit cycles vanish for the truncation operation. This behavior for the truncation operation is also exhibited for lowpass filters. □

In the case of 2nd-order and higher-order digital filters, granular limit cycles not only exist but also are of various types. These cycles in 2nd-order filters can be analyzed, and dead-band as well as frequency of oscillations can be estimated. For example, if the recursive all-pole filter is implemented with rounding quantizers in the multipliers as

$$\hat{y}(n) = \mathcal{Q}[a_1\hat{y}(n-1)] + \mathcal{Q}[a_2\hat{y}(n-2)] + x(n) \tag{10.32}$$

where $\hat{y}(n)$ is the quantized output, then using the analysis similar to that of the 1-order case, the dead-band region is given by

$$\hat{y}(n-2) \leq \frac{\Delta}{2(1-|a_2|)} \tag{10.33}$$

with a_1 determining the frequency of oscillations. For more details see Proakis, and Manolakis [23]. We provide the following example to illustrate granular limit cycles in 2nd-order filters using 3-bit quantizers.

□ **EXAMPLE 10.7** Consider the 2nd-order recursive filter

$$y(n) = 0.875y(n-1) - 0.75y(n-2) + x(n) \tag{10.34}$$

with zero initial conditions. This filter has two complex-conjugate poles and hence is a bandpass filter. Let the input be $x(n) = 0.375\delta(n)$. Analyze the limit cycle behavior using a 3-bit quantizer.

Solution

In the filter implementation the coefficient products are quantized, which results in

$$\hat{y}(n) = \mathcal{Q}[0.875\hat{y}(n-1)] - \mathcal{Q}[0.75\hat{y}(n-2)] + x(n) \qquad \textbf{(10.35)}$$

where $\hat{y}(n)$ is the quantized output. We simulate (10.35) in MATLAB using both the rounding and truncation operations.

```
% Bandpass filter
a1 = 0.875; a2 = -0.75;

% Rounding operation in multipliers
yn1 = 0; yn2 = 0;
m = 0:20; y = [yn2,yn1,zeros(1,length(m))];
x = 0.375*impseq(m(1),m(1)-2,m(end));
for n = m+3
   yn1 = y(n-1); yn2 = y(n-2);
   y(n) = QFix(a1*yn1,3,'round','satur')+QFix(a2*yn2,3,'round','satur')+x(n);
end

subplot('position',[0.1,0.2,0.39,0.6]);
plot([-1,20],[0,0],'w'); axis([-1,20,-0.5,0.5]); hold on;
Hs_1 = stem([-2,-1,m],y,'filled'); set(Hs_1,'markersize',3,'color',[0,1,0]);
set(gca,'ytick',[-0.5:0.25:0.5],'fontsize',6); ylabel('Amplitude','fontsize',8);
title('Rounding Operation','fontsize',10);
xlabel('Sample index n','fontsize',8);

% Truncation operation in multipliers
yn1 = 0; yn2 = 0;
m = 0:20; y = [yn2,yn1,zeros(1,length(m))];
x = 0.375*impseq(m(1),m(1)-2,m(end));
for n = m+3
   yn1 = y(n-1); yn2 = y(n-2);
   y(n) = QFix(a1*yn1,3,'trunc','satur')+QFix(a2*yn2,3,'trunc','satur')+x(n);
end

subplot('position',[0.59,0.2,0.39,0.6]);
plot([-1,20],[0,0],'w'); axis([-1,20,-0.5,0.5]); hold on;
Hs_1 = stem([-2,-1,m],y,'filled'); set(Hs_1,'markersize',3,'color',[0,1,0]);
set(gca,'ytick',[-0.5:0.25:0.5],'fontsize',6); ylabel('Amplitude','fontsize',8);
title('Truncation Operation','fontsize',10);
xlabel('Sample index n','fontsize',8);
```

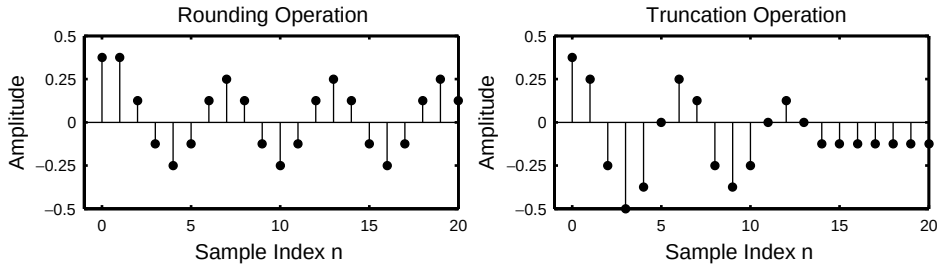


FIGURE 10.11 *Granular limit cycles in Example 10.7*

The resulting plots are shown in Figure 10.11. The round-off limit cycles have a period of six samples and amplitude of 0.25, which agrees with (10.33). Unlike in the case of 1st-order filters, the limit cycles for the 2nd-order exist even when truncation is used in the quantizer. □

10.2.3 OVERFLOW LIMIT CYCLES

This type of limit cycle is also a zero-input behavior that gives an oscillatory output. It is due to overflow in the addition even if we ignore multiplication or product quantization in the filter implementation. This is a more serious limit cycle because the oscillations can cover the entire dynamic range of the quantizer. It can be avoided in practice by using the saturation characteristics instead of overflow in the quantizer. In the following example, we simulate both granular and overflow limit cycles in a second-order filter, in addition to infinite precision implementation.

□ **EXAMPLE 10.8** To obtain overflow in addition we will consider the second-order filter with large coefficient values and initial conditions (magnitudewise) excited by a zero input:

$$y(n) = 0.875y(n-1) - 0.875y(n-1); \quad y(-1) = -0.875,\ y(-2) = 0.875 \tag{10.36}$$

The overflow in the addition is obtained by placing the quantizer after the additions as

$$\hat{y}(n) = \mathcal{Q}[0.875\hat{y}(n-1) - 0.875\hat{y}(n-1)]; \quad \hat{y}(-1) = -0.875,\ \hat{y}(-2) = 0.875 \tag{10.37}$$

where $\hat{y}(n)$ is the quantized output. We first simulate the infinite-precision operation of (10.36) and compare its output with the granular limit-cycle implementation in (10.35) and with the overflow limit-cycle in (10.37). We use the

rounding operation. The details are in the MATLAB script:

```
M = 100; B = 3; A = 1-2^(-B);
a1 = A; a2 = -A; yn1 = -A; yn2 = A;
m = 0:M; y = [yn2,yn1,zeros(1,length(m))];

% Infinite precision
for n = m+3
   yn1 = y(n-1); yn2 = y(n-2);
   y(n) = a1*yn1 + a2*yn2;
end
subplot('position',[0.08,0.2,0.24,0.6]);
plot([-1,100],[0,0],'w'); axis([-1,80,-1,1]); hold on;
Hs_1 = stem([-2,-1,m],y,'filled');set(Hs_1,'markersize',3,'color',[0,1,0]);
set(gca,'ytick',[-1:0.25:1],'fontsize',6); ylabel('Amplitude','fontsize',8);
title('No Limit Cycles','fontsize',10);
xlabel('Sample index n','fontsize',8);

% Granular limit cycle
for n = m+3
   yn1 = y(n-1); yn2 = y(n-2);
   y(n) = QFix(a1*yn1,B,'round','satur')+QFix(a2*yn2,B,'round','satur');
   y(n) = QFix(y(n),B,'round','satur');
end
subplot('position',[0.42,0.2,0.24,0.6]);
plot([-1,100],[0,0],'w'); axis([-1,80,-1,1]); hold on;
Hs_1 = stem([-2,-1,m],y,'filled');set(Hs_1,'markersize',3,'color',[0,1,0]);
set(gca,'ytick',[-1:0.25:1],'fontsize',6); ylabel('Amplitude','fontsize',8);
title('Granular Limit Cycles','fontsize',10);
xlabel('Sample index n','fontsize',8);

% Overflow limit cycle
for n = m+3
   yn1 = y(n-1); yn2 = y(n-2);
   y(n) = a1*yn1 + a2*yn2;
   y(n) = QFix(y(n),B,'round','twosc');
end
subplot('position',[0.76,0.2,0.23,0.6]);
plot([-1,100],[0,0],'w'); axis([-1,80,-1,1]); hold on;
Hs_1 = stem([-2,-1,m],y,'filled');set(Hs_1,'markersize',3,'color',[0,1,0]);
set(gca,'ytick',[-1:0.25:1],'fontsize',6); ylabel('Amplitude','fontsize',8);
title('Overflow Limit Cycles','fontsize',10);
xlabel('Sample index n','fontsize',8);
```

The resulting plots are shown in Figure 10.12. As expected, the infinite-precision implementation has no limit cycles. The granular limit cycles are of smaller amplitudes. Clearly, the overflow limit cycles have large amplitudes spanning the -1 to 1 range of the quantizers. □

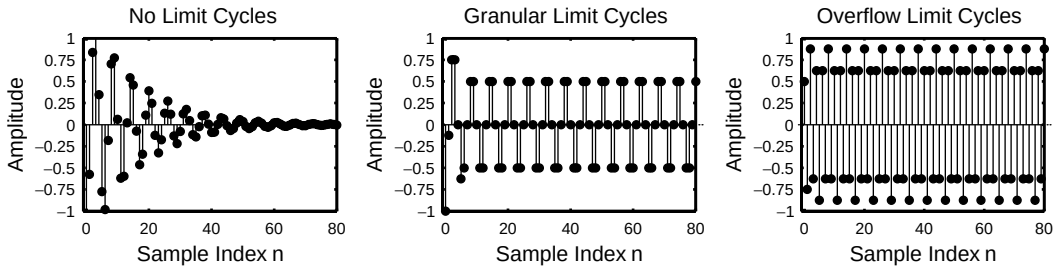


FIGURE 10.12 *Comparison of limit cycles in Example 10.8*

As shown in these examples, the limit-cycle behaviors of many different filters can be studied for different quantizer characteristics using the MATLAB function `QFix`.

10.2.4 MULTIPLICATION QUANTIZATION ERROR

A multiplier element in the filter implementation can introduce additional quantization errors since multiplication of two B-bit fractional numbers results in a $2B$-bit fraction and must be quantized to a B-bit fraction. Consider a multiplier in fixed-point arithmetic with $B = 8$. The number $\frac{1}{\sqrt{3}}$ is represented as 0.578125 in decimal. The square of 0.578125 rounded to 8 bits is 0.3359375 (which should not be confused with 1/3 rounded to 8 bits, which is 0.33203125). The additional error in the squaring operation is

$$0.3359375 - (0.578125)^2 = 0.001708984375$$

This additional error is termed as the *multiplication quantization error*. Its statistically equivalent model is similar to that of the A/D quantization error model, as shown in Figure 10.13.

Statistical model Consider the B-bit quantizer block following the multiplier element shown in Figure 10.13a. The sequence $x(n)$ and the constant c are quantized to B fractional bits prior to multiplication (as would be the case in a typical implementation). The multiplied sequence $\{c\,x(n)\}$ is quantized to obtain $y(n)$. We want to replace the quantizer by a simpler linear system model shown in Figure 10.13b, in which $y(n) = c\,x(n)+e(n)$,

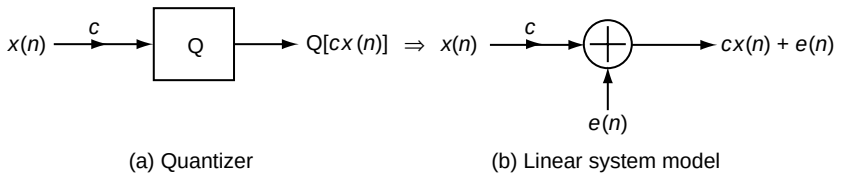


FIGURE 10.13 *Linear system model for multiplication quantization error*

where $e(n)$ is a multiplication quantization error. For analysis purposes we assume that the conditions on $e(n)$ are similar to those for the A/D quantization error:

1. The random signal $e(n)$ is *uncorrelated* with the sequence $x(n)$ for rounding operation (or two's-complement truncation operation) in the quantizer.
2. The signal $e(n)$ is an independent process (i.e., the samples are independent of each other).
3. The probability density function (pdf) $f_E(e)$ of $e(n)$ for each n is uniformly distributed over the interval of width $\Delta = 2^{-B}$, which is the quantizer resolution.

We will emphasize the rounding operation for the rest of this section. Based on the above model assumptions, the results given in (10.7), (10.9), and (10.10) are also applicable for the multiplication quantization error $e(n)$.

We offer the following two MATLAB examples to illustrate this model. A more thorough investigation of this error can be found in Rabiner and Tukey [25].

□ **EXAMPLE 10.9** Consider the sequence given in Example 10.1, which is repeated here.

$$x(n) = \frac{1}{3}\left[\sin(n/11) + \sin(n/31) + \cos(n/67)\right]$$

This signal is multiplied by $c = 1/\sqrt{2}$, quantized to B bits and the resulting multiplication is quantized to B bits with rounding. Using the `StatModelR` function and 500,000 samples, compute and analyze normalized errors $e_1(n)$ and $e_2(n)$, defined in (10.2) and (10.3), respectively.

Solution The following MATLAB script computes error distribution, for $B = 6$ bits.

```
clear; close all;
% Example parameters
B = 6; N = 500000; n = [1:N]; bM = 7;
xn = (1/3)*(sin(n/11)+sin(n/31)+cos(n/67)); clear n;
c = 1/sqrt(2);
% Signal and Coefficient Quantization
xq = (round(xn*(2^B)))/(2^B); c = (round(c*(2^B)))/(2^B);
cxq = c*xq;                                % Multiplication of constant and signal
% Quantization error analysis
[H1,H2,Q, estat] = StatModelR(cxq,B,N);
H1max = max(H1); H1min = min(H1);          % Max and Min of H1
H2max = max(H2); H2min = min(H2);          % Max and Min of H2
```

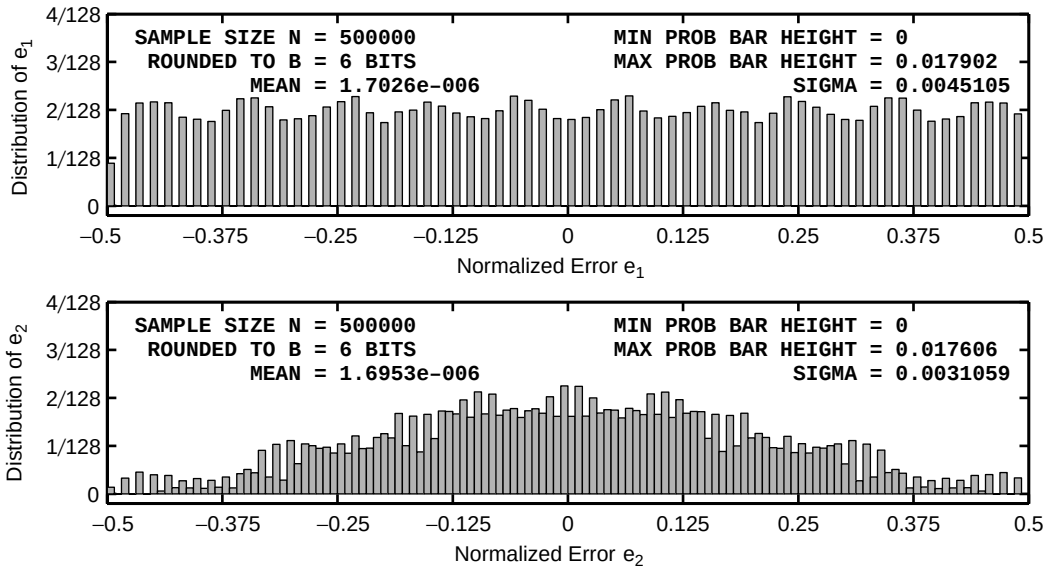


FIGURE 10.14 *Multiplication quantization error distribution for the sinusoidal signal in Example 10.9, $B = 6$ bits*

The plots of the resulting histogram are shown in Figure 10.14. For the sinusoidal signal, when $B = 6$ bits, the error samples are not uniformly distributed and the samples are not independent. The means of $e(n)$ and $[e(n)+e(n-1)]/2$ are small. Their standard deviations are 0.0045105 and 0.0031059, which do not agree with (10.10). The corresponding plots for $B = 12$ bits are shown in Figure 10.15 from which we observe that the quantization error sequence appears to satisfy the model assumptions for $B \geq 12$ bits. The means of $e(n)$ and $[e(n) + e(n-1)]/2$ are very small, and their standard deviations agree closely with (10.10). □

□ **EXAMPLE 10.10** Let $x(n)$ be an independent and identically distributed random sequence whose samples are uniformly distributed over the $[-1, 1]$ interval. Using 500,000 samples to minimize any statistical variations, analyze normalized errors.

Solution The following MATLAB script computes the distributions for $B = 6$ bits.

```
clear; close all;
% Example parameters
B = 6; N = 500000; xn = (2*rand(1,N)-1); bM = 7;  c = 1/sqrt(2);
% Signal and Coefficient Quantization
xq = (round(xn*(2^B)))/(2^B); c = (round(c*(2^B)))/(2^B);
cxq = c*xq;                                % Multiplication of constant and signal
```

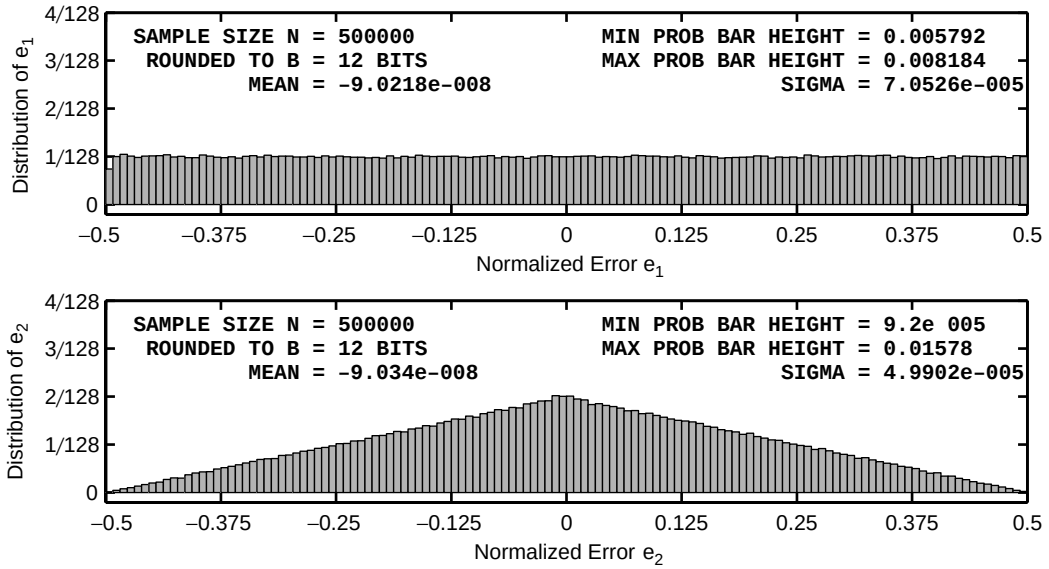


FIGURE 10.15 *Multiplication quantization error distribution for the sinusoidal signal in Example 10.9, $B = 12$ bits*

```
% Quantization error analysis
[H1,H2,Q, estat] = StatModelR(cxq,B,N);
H1max = max(H1); H1min = min(H1);          % Max and Min of H1
H2max = max(H2); H2min = min(H2);          % Max and Min of H2
```

The plots of the resulting histogram are shown in Figure 10.16. Even for $B = 6$ bits, the error samples appear to be uniformly distributed (albeit in discrete fashion) and are independent of each other. The corresponding plots for $B = 12$ bits are shown in Figure 10.17. It is clear for $B = 12$ bits that the quantization error samples are independent and uniformly distributed. Readers should verify the statistics of these errors given in (10.7), (10.9), and (10.10). □

From these two examples, we conclude that the statistical model for the multiplication quantization error, with its stated assumptions, is a very good model for random signals when the number of bits in the quantizer is large enough.

10.2.5 STATISTICAL ROUND-OFF NOISE—FIXED-POINT ARITHMETIC

In this and the next section, we will consider the round-off effects on IIR filters using the multiplication quantization error model developed in the

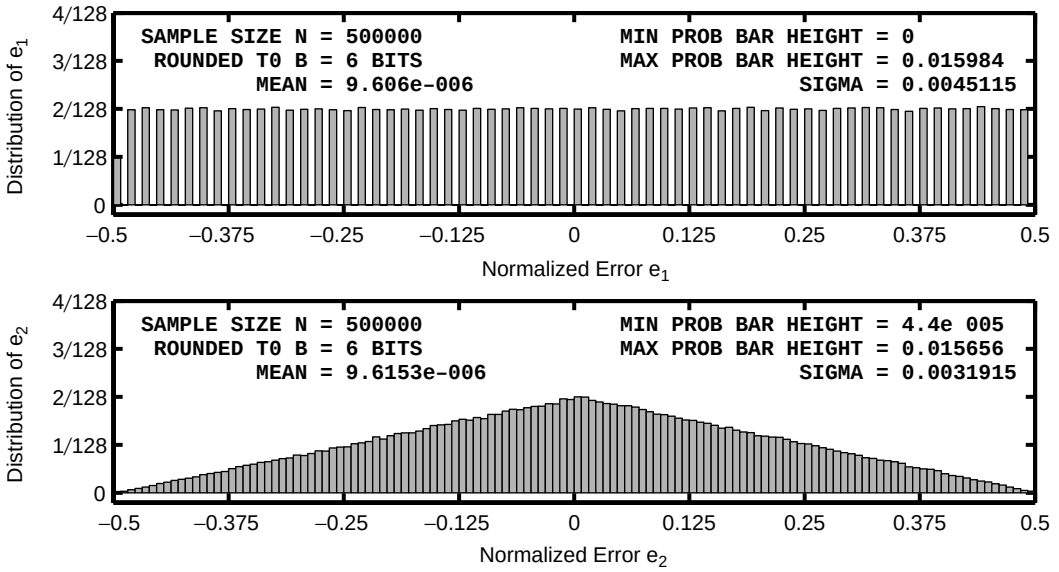


FIGURE 10.16 *Multiplication quantization error distribution for the random signal in Example 10.10, $B = 6$ bits*

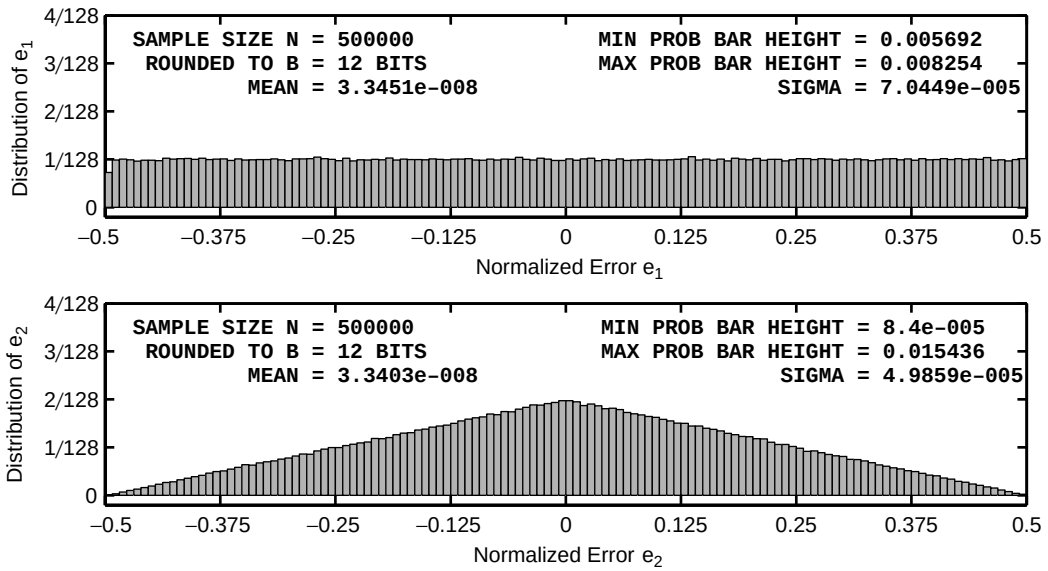


FIGURE 10.17 *Multiplication quantization error distribution for the random signal in Example 10.10, $B = 12$ bits*

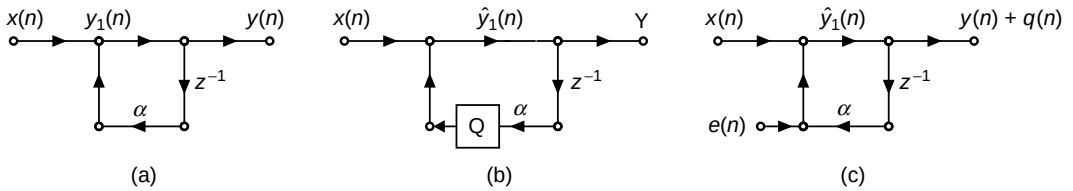


FIGURE 10.18 *First-order IIR filter: (a) structure, (b) structure with quantizer, (c) round-off noise model*

previous section. Since we emphasize the rounding operation, this model is also known as a round-off noise model. We will limit ourselves to the 1st- and 2nd-order filters since practical realizations involve 1st- or 2nd-order sections.

1st-order filter Consider the 1st-order filter shown in Figure 10.18a. When a quantizer $\mathcal{Q}[\cdot]$ is introduced after the multiplier, the resulting filter model is shown in Figure 10.18b, which is a nonlinear system. When $\mathcal{Q}[\cdot]$ is a quantizer based on the round-off characteristics, then its effect is to add a zero-mean, stationary white noise sequence $e(n)$ at the multiplier output as shown in Figure 10.18c.

Let $q(n)$ be the response due to $e(n)$ and let $h_e(n)$ be the noise impulse response (i.e., between $e(n)$ and $q(n)$). For the system in Figure 10.18c

$$h_e(n) = h(n) = \alpha^n u(n) \tag{10.38}$$

Using (10.12) and (10.7), the mean of $q(n)$ is

$$m_q = m_e \sum_{0}^{\infty} h_e(n) = 0 \tag{10.39}$$

Similarly, using (10.15), the variance of $q(n)$ is

$$\sigma_q^2 = \sigma_e^2 \left(\sum_{0}^{\infty} |h_e(n)|^2 \right) \tag{10.40}$$

Substituting $\sigma_e^2 = 2^{-2B}/12$ for rounding and $h_e(n)$ from (10.38), we obtain

$$\sigma_q^2 = \frac{2^{-2B}}{12} \left(\sum_{0}^{\infty} |\alpha^n|^2 \right) = \frac{2^{-2B}}{12} \sum_{0}^{\infty} \left(|\alpha|^2 \right)^n = \frac{2^{-2B}}{12\,(1 - |\alpha|^2)} \tag{10.41}$$

which is the output noise power due to rounding following the multiplication.

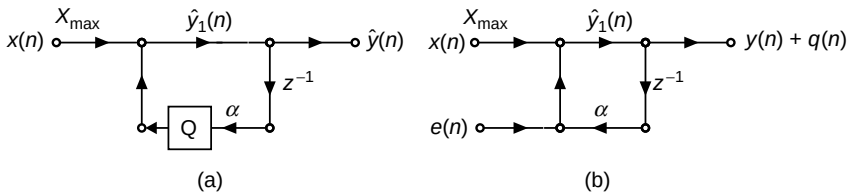


FIGURE 10.19 *Scaled first-order IIR filter: (a) structure with quantizer, (b) round-off noise model*

However, we also have to prevent a possible overflow following the adder. Let $y_1(n)$ be the signal at the output of the adder in Figure 10.18a, which in this case is equal to $y(n)$. Now the upper bound on $y_1(n)$ is

$$|y_1(n)| = |y(n)| = \left| \sum_0^\infty h(k)\, x(n-k) \right| \leq \sum_0^\infty |h(k)|\ |x(n-k)| \qquad \textbf{(10.42)}$$

Let the input sequence be bounded by $X_{\max}$ (i.e., $|x(n)| \leq X_{\max}$). Then

$$|y_1(n)| \leq X_{\max} \sum_0^\infty |h(k)| \qquad \textbf{(10.43)}$$

Since $y_1(n)$ is represented by B fraction bits, we have $|y_1(n)| \leq 1$. The condition (10.43) can be satisfied by requiring

$$X_{\max} = \frac{1}{\sum_0^\infty |h(k)|} = \frac{1}{1/\left(1-|\alpha|\right)} = 1 - |\alpha| \qquad \textbf{(10.44)}$$

Thus, to prevent overflow $x(n)$ must satisfy

$$-\left(1-|\alpha|\right) \leq x(n) \leq \left(1-|\alpha|\right) \qquad \textbf{(10.45)}$$

Thus, the input must be scaled before it is applied to the filter as shown in Figure 10.19.

Signal-to-noise ratio We will now compute the finite word-length effect in terms of the output signal-to-noise ratio (SNR). We assume that there is no overflow at the output by properly scaling $x(n)$. Let $x(n)$ be a stationary white sequence, uniformly distributed between $[-\left(1-|\alpha|\right), \left(1-|\alpha|\right)]$. Then

$$m_x = 0 \quad \text{and} \quad \sigma_x^2 = \frac{\left(1-|\alpha|\right)^2}{3} \qquad \textbf{(10.46)}$$

Therefore, $y(n)$ is also a stationary random sequence with mean $m_y = 0$ and

$$\sigma_y^2 = \sigma_x^2 \sum_0^\infty |h(n)|^2 = \frac{\left(1-|\alpha|\right)^2}{3} \frac{1}{1-|\alpha|^2} = \frac{\left(1-|\alpha|\right)^2}{3\left(1-|\alpha|^2\right)} \qquad \textbf{(10.47)}$$

Using (10.41) and (10.47), the output SNR is

$$\begin{aligned}\text{SNR} &\triangleq \frac{\sigma_y^2}{\sigma_q^2} = \frac{(1-|\alpha|)^2}{3\,(1-|\alpha|^2)}\,\frac{12\left(1-|\alpha|^2\right)}{2^{-2B}} \\ &= 4\left(2^{2B}\right)(1-|\alpha|)^2 = 2^{2(B+1)}\,(1-|\alpha|)^2\end{aligned} \tag{10.48}$$

or the SNR in dB is

$$\text{SNR}_{\text{dB}} \triangleq 10\log_{10}(\text{SNR}) = 6.02 + 6.02B + 20\log_{10}(1-|\alpha|) \tag{10.49}$$

Let $\delta = 1 - |\alpha|$, which is the distance of the pole from the unit circle. Then

$$\text{SNR}_{\text{dB}} = 6.02 + 6.02B + 20\log_{10}(\delta) \tag{10.50}$$

which is a very informative result. First, it shows that the SNR is directly proportional to B and increases by about 6 dB for each additional bit added to the word length. Second, the SNR is also directly proportional to the distance δ. The smaller the δ (or nearer the pole to the unit circle), the smaller is the SNR, which is a consequence of the filter characteristics. As an example, if $B = 6$ and $\delta = 0.05$, then SNR = 16.12 dB and if $B = 12$ and $\delta = 0.1$, then SNR = 58.26 dB.

10.2.6 ANALYSIS USING MATLAB

To analyze the properties of the round-off errors in IIR filters we will simulate them using the MATLAB function `QFix` with quantization mode `'round'` and overflow mode `'satur'`. If proper scaling to avoid overflow is performed, then only the multiplier output needs to be quantized at each n without worrying about the overflow. However, we will still saturate the final sum to avoid any unforeseen problems. In previous simulations, we could perform the quantization operations on vectors (i.e., perform parallel processing). Since IIR filters are recursive filters and since each error is fed back into the system, vector operation is generally not possible. Hence the filter output will be computed sequentially from the first to the last sample. For a large number of samples, this implementation will slow the execution speed in MATLAB since MATLAB is optimized for vector calculations. However, for newer fast processors, the execution time is within a few seconds. These simulation steps are detailed in the following example.

□ **EXAMPLE 10.11** Consider the model given in Figure 10.19b. We will simulate this model in MATLAB and investigate its output error characteristics. Let $a = 0.9$, which will be quantized to B bits. The input signal is uniformly distributed over the $[-1, +1]$ interval and is also quantized to B bits prior to filtering. The scaling factor $X_{\max}$ is computed from (10.44). Using $100,000$ signal samples and $B = 6$ bits, the following MATLAB script computes the true output $y(n)$, the quantized output $\hat{y}(n)$, the output error $q(n)$, and the output SNR.

```
close all; clc;

% Example Parameters
B = 6;                % # of fractional bits
N = 100000;           % # of samples
xn = (2*rand(1,N)-1); % Input sequence - Uniform Distribution
a = 0.9;              % Filter parameter
Xm = 1-abs(a);        % Scaling factor

% Local variables
  bM = 7; DbM = 2^bM;    % bin parameter
  BB = 2^B;              % useful factor in quantization
   M = round(DbM/2);     % Half number of bins
bins = [-M+0.5:1:M-0.5]; % Bin values from -M to M
   Q = bins/DbM;         % Normalized bins
 YTN = 2^(-bM);          % Ytick marks interval
 YLM = 4*YTN;            % Yaxis limit

% Quantize the input and the filter coefficients
  xn = QFix(Xm*xn,B,'round','satur'); % Scaled Input quant to B bits
   a = QFix(a,B,'round','satur');     % a quantized to B bits

% Filter output without multiplication quantization
  yn = filter(1,[1,-a],xn);  % output using filter routine

% Filter output with multiplication quantization
   yq = zeros(1,N);              % Initialize quantized output array
yq(1) = xn(1);                   % Calculation of the first sample yq(1)
for I = 2:N;
    A1Y = QFix(a*yq(I-1),B,'round','satur'); % Quantization of a*y(n-1)
  yq(I) = QFix(A1Y+xn(I),B,'round','satur'); % I-th sample yq(I)
end

% Output Error Analysis
   en = yn-yq;                           % Output error sequence
varyn = var(yn); varen = var(en);        % Signal and noise power
eemax = max(en); eemin = min(en);        % Maximum and minimum of the error
enmax = max(abs([eemax,eemin]));         % Absolute maximum range of the error
enavg = mean(en); enstd = std(en);       % Mean and std dev of the error
   en = round(en*(2^bM)/(2*enmax)+0.5); % Normalized en (integer between -M & M)
   en = sort([en,-M:1:(M+1)]);           %
    H = diff(find(diff(en)))-1;          % Error histogram
    H = H/N;                             % Normalized histogram
 Hmax = max(H); Hmin = min(H);           % Max and Min of the normalized histogram

% Output SNRs
SNR_C = 10*log10(varyn/varen); % Computed SNR
SNR_T = 6.02 + 6.02*B + 20*log10(Xm); % Theoretical SNR
```

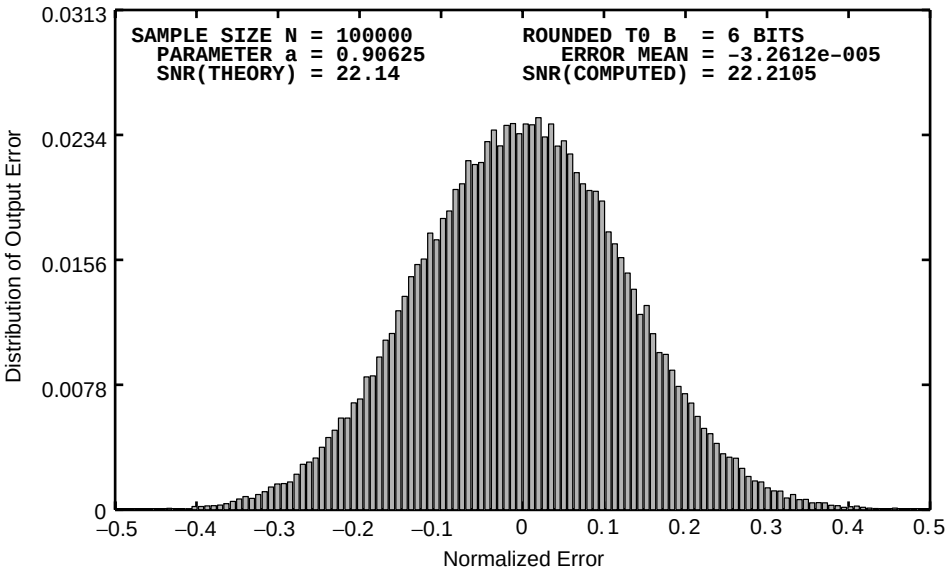


FIGURE 10.20 *Multiplication quantization effects in the first-order IIR filter in Example 10.11, $B = 6$ bits*

The part of the script not shown above also computes and plots the normalized histogram of the output error and prints the statistical values in the plot, as shown in Figure 10.20. The error appears to have a Gaussian distribution, which is to be expected. The exact value of the output SNR is 22.14 dB, which agrees with the computed value of 22.21 dB. Similar results done for $B = 12$ bits are shown in Figure 10.21. Again, the simulation results agree with the model results. □

2nd-order filter Similar analysis can be done for 2nd-order filters with poles near the unit circle. Let the two poles be at complex locations $re^{\jmath\theta}$ and $re^{-\jmath\theta}$. Then the system function of the filter is given by

$$H(z) = \frac{1}{(1 - re^{\jmath\theta}z^{-1})(1 - re^{-\jmath\theta}z^{-1})} = \frac{1}{1 - 2r\cos(\theta)\,z^{-1} + r^2z^{-2}} \tag{10.51}$$

with impulse response

$$h(n) = \frac{r^n \sin\{(n+1)\theta\}}{\sin(\theta)}u(n) \tag{10.52}$$

The difference equation from (10.51) is given by

$$y(n) = x(n) - a_1y(n-1) - a_2y(n-2); \quad a_1 = -2r\cos(\theta),\ a_2 = r^2 \tag{10.53}$$

which requires two multiplications and two additions, as shown in Figure 10.22a. Thus, there are two noise sources and two possible locations for overflow. The round-off noise model for quantization following

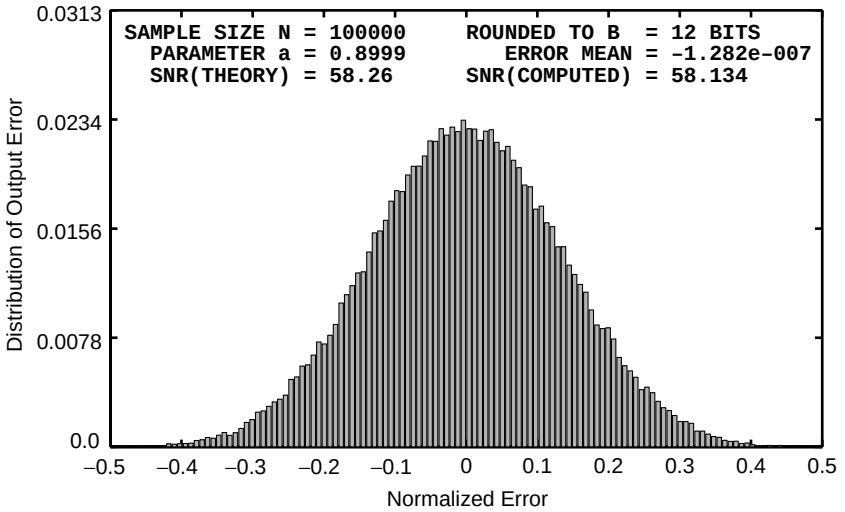


FIGURE 10.21 *Multiplication quantization effects in the 1st-order IIR filter in Example 10.11, $B = 12$ bits*

the two multipliers is shown in Figure 10.22b, where the responses $q_1(n)$ and $q_2(n)$ are due to noise sources $e_1(n)$ and $e_2(n)$, respectively. We can combine two noise sources into one. However, to avoid overflow we have to scale signals at the input of each adder, which can complicate this consolidation of sources.

In modern DSP chips, the intermediate results of multiply-add operations are stored in a multiply-accumulate or MAC unit that has a double precision register to accumulate sums. The final sum [which for Figure 10.22b is at the output of the top adder] is quantized to obtain $\hat{y}(n)$. This implementation not only reduces the total multiplication quantization noise but also makes the resulting analysis easier. Assuming this modern implementation, the resulting simplified model is shown in Figure 10.22c, where $e(n)$ is the single noise source that is uniformly distributed between $[-2^{-(B+1)}, 2^{-(B+1)}]$ and $q(n)$ is the response due to $e(n)$. Note that $e(n) \neq e_1(n)+e_2(n)$ and that $q(n) \neq q_1(n)+q_2(n)$. The only overflow that we have to worry about is at the output of the top adder, which can be controlled by scaling the input sequence $x(n)$ as shown in Figure 10.22d. Now the round-off noise analysis can be carried out in a fashion similar to that of the 1st-order filter. The details, however, are more involved due to the impulse response in (10.52).

Signal-to-noise ratio Referring to Figure 10.22d, the noise impulse response $h_e(n)$ is equal to $h(n)$. Hence the output round-off noise power

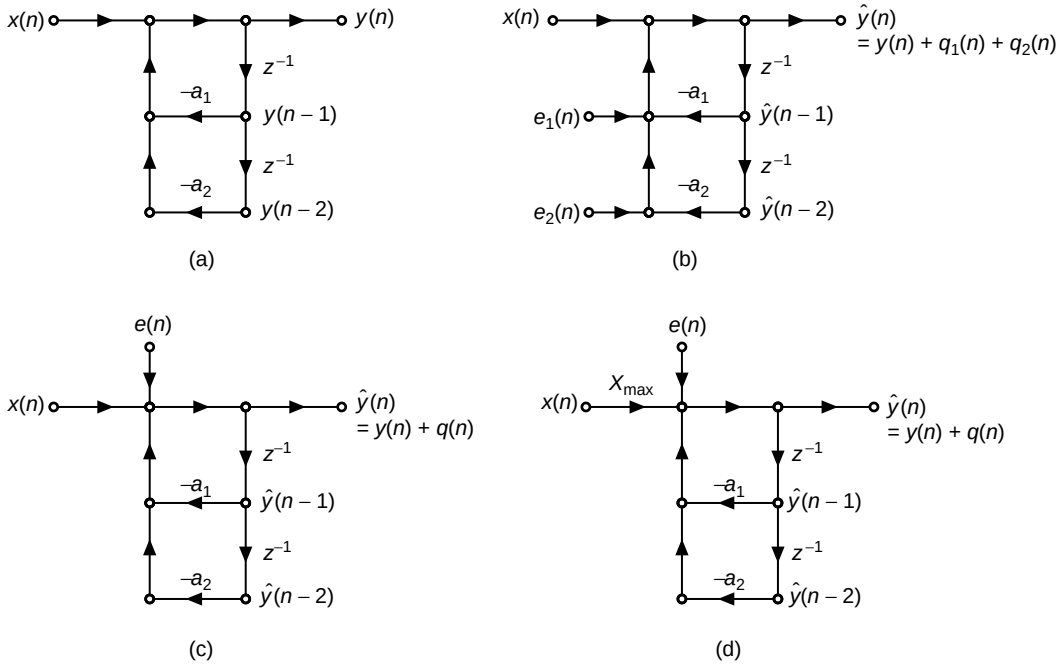


FIGURE 10.22 *2nd-order IIR filter: (a) structure, (b) round-off noise model, (c) simplified model, (d) scaled simplified model*

is given by

$$\sigma_q^2 = \sigma_e^2 \sum_{n=0}^{\infty} |h(n)|^2 = \frac{2^{-2B}}{12} \sum_{n=0}^{\infty} |h(n)|^2 \tag{10.54}$$

Since $x(n)$ is quantized, we have $|x(n)| \leq 1$. It is then scaled by $X_{\max}$ to avoid overflow in the adder. Hence the output signal power is given by

$$\sigma_y^2 = X_{\max}^2 \sigma_x^2 \sum_{n=0}^{\infty} |h(n)|^2 = \frac{X_{\max}^2}{3} \sum_{n=0}^{\infty} |h(n)|^2 \tag{10.55}$$

assuming that $x(n)$ is uniformly distributed over $[-1, +1]$. Hence the output SNR is given by

$$\text{SNR} = \frac{\sigma_y^2}{\sigma_q^2} = 4\left(2^{2B}\right) X_{\max}^2 = 2^{2(B+1)} X_{\max}^2 \tag{10.56}$$

or

$$\text{SNR}_{\text{dB}} = 6.02 + 6.02B + 20 \log_{10} X_{\max} \tag{10.57}$$

Following (10.43), (10.44), and (10.45), the scaling factor $X_{\max}$ is given by

$$X_{\max} = \frac{1}{\sum_{n=0}^{\infty} |h(n)|} \tag{10.58}$$

which is not easy to compute. However, lower and upper bounds on $X_{\max}$ are easy to obtain. From (10.52), the upper bound on the denominator of (10.58) is given by

$$\sum_{n=0}^{\infty} |h(n)| = \frac{1}{\sin\theta} \sum_{n=0}^{\infty} r^n |\sin[(n+1)\theta]| \leq \frac{1}{\sin\theta} \sum_{n=0}^{\infty} r^n = \frac{1}{(1-r)\sin\theta} \tag{10.59}$$

or the lower bound on $X_{\max}$ is given by

$$X_{\max} \geq (1-r)\sin\theta \tag{10.60}$$

The lower bound on the denominator of (10.58) is obtained by noting that

$$|H(\mathrm{e}^{\jmath\omega})| = \left| \sum_{n=0}^{\infty} h(n)\mathrm{e}^{-\jmath\omega} \right| \leq \sum_{n=0}^{\infty} |h(n)|$$

Now from (10.51), the magnitude $|H(\mathrm{e}^{\jmath\omega})|$ is given by

$$|H(\mathrm{e}^{\jmath\omega})| = \left| \frac{1}{1 - 2r\cos(\theta)\mathrm{e}^{-\jmath\omega} + r^2\mathrm{e}^{-\jmath 2\omega}} \right|$$

which has the maximum value at the resonant frequency $\omega = \theta$, which can be easily obtained. Hence

$$\sum_{n=0}^{\infty} |h(n)| \geq \left|H(\mathrm{e}^{\jmath\theta})\right| = \frac{1}{(1-r)\sqrt{1 + r^2 - 2r\cos(2\theta)}} \tag{10.61}$$

or the upper bound on $X_{\max}$ is given by

$$X_{\max} \leq (1-r)\sqrt{1 + r^2 - 2r\cos(2\theta)} \tag{10.62}$$

Substituting (10.60) and (10.62) in (10.56), the output SNR is upper and lower bounded by

$$2^{2(B+1)}(1-r)^2 \sin^2\theta \leq \mathrm{SNR} \leq 2^{2(B+1)}(1-r)^2(1+r^2-2r\cos 2\theta) \tag{10.63}$$

Substituting $1 - r = \delta \ll 1$ and after some simplification, we obtain

$$2^{2(B+1)}\delta^2 \sin^2\theta \leq \mathrm{SNR} \leq 4\left(2^{2(B+1)}\right)\delta^2 \sin^2\theta \tag{10.64}$$

or the difference between the upper and lower SNR bounds is about 6 dB. Once again the output SNR is directly proportional to B and δ. Furthermore, it also depends on the angle θ. Some of these observations are investigated in Example 10.12.

10.2.7 ANALYSIS USING MATLAB

We will again simulate round-off errors using the MATLAB function `QFix` with quantization mode `'round'` and overflow mode `'satur'`. Since a MAC architecture is assumed, we do not have to quantize the intermediate results and worry about overflow. Only the final sum needs to be quantized with saturation. These operations are also simulated in sequential fashion, which has an impact on execution speed. The simulation steps for the 2nd-order filter are detailed in the following example.

□ **EXAMPLE 10.12** Consider the model given in Figure 10.22d. We will simulate this model in MATLAB and investigate its output error characteristics. Let $r = 0.9$ and $\theta = \pi/3$, from which filter parameters are computed and quantized to B bits. The input signal is uniformly distributed over the $[-1, +1]$ interval and is also quantized to B bits prior to filtering. The scaling factor $X_{\max}$ is determined using (10.58), which can be obtained in MATLAB by computing the impulse response for a sufficiently large number of samples. Using $100,000$ signal samples and $B = 6$ bits, the following MATLAB script computes the true output SNR, the computed SNR, and the lower and upper bounds of the SNR.

```
close all; clc;

% Example Parameters
 B = 12;              % # of fractional bits
 N = 100000;          % # of samples
xn = (2*rand(1,N)-1); % Input sequence - Uniform
 r = 0.9; theta = pi/3;% Pole locations

% Computed Parameters
p1 = r*exp(j*theta);  % Poles
p2 = conj(p1);        %
 a = poly([p1,p2]);   % Filter parameters
hn = filter(1,a,[1,zeros(1,1000)]); % Imp res
Xm = 1/sum(abs(hn));  % Scaling factor
Xm_L = (1-r)*sin(theta); % Lower bound
Xm_U = (1-r)*sqrt(1+r*r-2*r*cos(2*theta)); % Upper bound

% Local variables
  bM = 7; DbM = 2^bM;    % bin parameter
  BB = 2^B;              % useful factor in quantization
   M = round(DbM/2);     % Half number of bins
bins = [-M+0.5:1:M-0.5]; % Bin values from -M to M
   Q = bins/DbM;         % Normalized bins
 YTN = 2^(-bM);          % Ytick marks interval
 YLM = 4*YTN;            % Yaxis limit

% Quantize the input and the filter coefficients
  xn = QFix(Xm*xn,B,'round','satur'); % Scaled Input quant B bits
   a = QFix(a,B,'round','satur');     % a quantized to B bits
  a1 = a(2); a2 = a(3);
```

```
% Filter output without multiplication quantization
  yn = filter(1,a,xn);  % output using filter routine

% Filter output with multiplication quantization
   yq = zeros(1,N); % Initialize quantized output array
yq(1) = xn(1);      % sample yq(1)
yq(2) = QFix((xn(2)-a1*yq(1)),B,'round','satur'); % sample yq(2)
for I = 3:N;
    yq(I) = xn(I)-a1*yq(I-1)-a2*yq(I-2); % Unquantized sample
    yq(I) = QFix(yq(I),B,'round','satur'); % Quantized sample
end

% Output Error Analysis
   en = yn-yq;                          % Output error sequence
varyn = var(yn); varen = var(en);       % Signal and noise power
eemax = max(en); eemin = min(en);       % Maximum and minimum of the error
enmax = max(abs([eemax,eemin]));        % Absolute maximum range of the error
enavg = mean(en); enstd = std(en);      % Mean and std dev of the error
   en = round(en*(2^bM)/(2*enmax)+0.5); % Normalized en (integer between -M & M)
   en = sort([en,-M:1:(M+1)]);          %
    H = diff(find(diff(en)))-1;         % Error histogram
    H = H/N;                            % Normalized histogram
 Hmax = max(H); Hmin = min(H);          % Max and Min of the normalized histogram

% Output SNRs
SNR_C = 10*log10(varyn/varen);          % Computed SNR
SNR_T = 6.02 + 6.02*B + 20*log10(Xm);   % Theoretical SNR
SNR_L = 6.02 + 6.02*B + 20*log10(Xm_L); % Lower SNR bound
SNR_U = 6.02 + 6.02*B + 20*log10(Xm_U); % Upper SNR bound
```

The part of the script not shown above also computes and plots the normalized histogram of the output error and prints the statistical values in the plot, as shown in Figure 10.23. The error again has a Gaussian distribution. The exact value of the output SNR is 25.22 dB, which agrees with the computed value of 25.11 dB and lies between the lower bound of 20.89 dB and the upper bound of 26.47 dB. Similar results done for $B = 12$ bits are shown in Figure 10.24. Again, the simulation results agree with the model results. □

10.2.8 HIGHER-ORDER FILTERS

The analysis of the quantization effects in a second-order filter can be applied directly to higher-order filters based on a parallel realization. In this case each 2nd-order filter section is independent of all the other sections, and therefore the total quantization noise power at the output of the parallel structure is simply the linear sum of the quantization noise powers of each of the individual sections. On the other hand, the cascade realization is more difficult to analyze because the noise generated in any

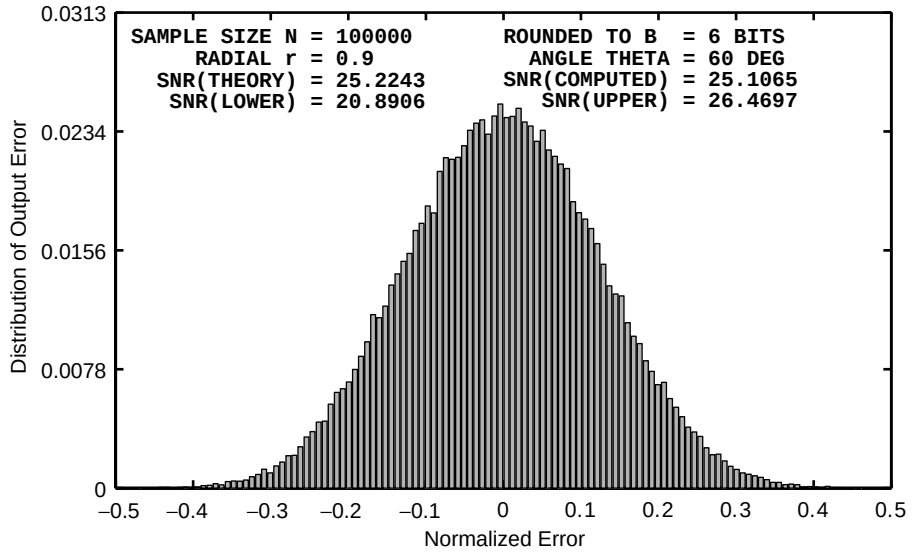


FIGURE 10.23 *Multiplication quantization effects in the 1st-order IIR filter in Example 10.12, $B = 6$ bits*

second-order filter section is filtered by the succeeding sections. To minimize the total noise power at the output of the high-order filter, a reasonable strategy is to place the sections in the order of decreasing maximum frequency gain. In this case the noise power generated in the early

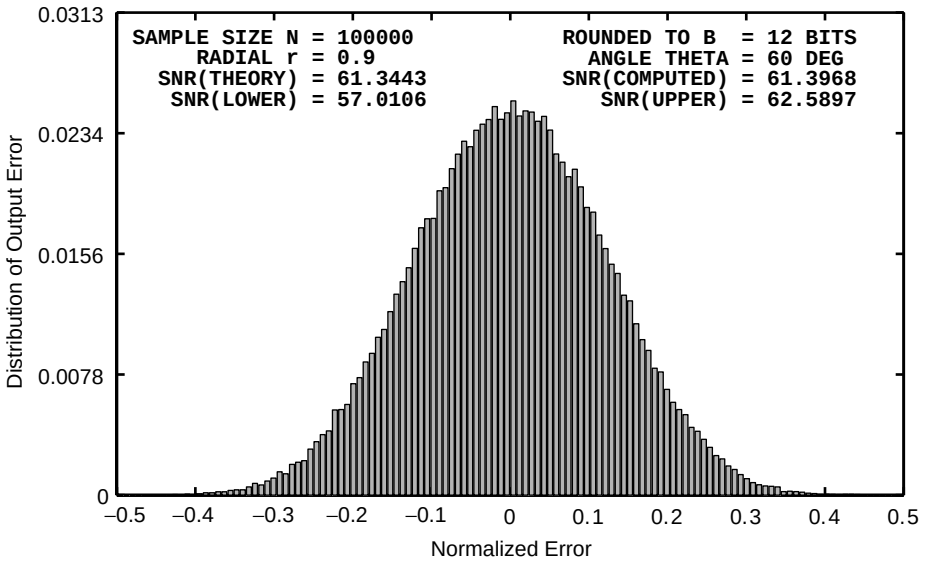


FIGURE 10.24 *Multiplication quantization effects in the 1st-order IIR filter in Example 10.12, $B = 12$ bits*

high-gain section is not boosted significantly by the latter sections. Using the MATLAB techniques developed in the previous sections, it is easier to simulate finite word-length implementations and determine the output SNR for a given cascade structure.

10.2.9 STATISTICAL ROUND-OFF NOISE—FLOATING-POINT ARITHMETIC

As stated in Chapter 6, the floating-point arithmetic gives an error that is relative to the magnitude rather than an absolute error. This results in a multiplicative noise rather than additive noise—that is, from (6.61)

$$\mathcal{Q}[x(n)] = x(n) + \varepsilon(n)x(n) = x(n)\{1 + \varepsilon(n)\} \tag{10.65}$$

with

$$-2^{-B} < \varepsilon(n) \leq 2^{-B} \tag{10.66}$$

for a $(B+1)$-bit mantissa. Hence the mean of the relative error is $m_\varepsilon = 0$ and its variance is

$$\sigma_\varepsilon^2 = \frac{2^{-2B}}{3} \tag{10.67}$$

Since MATLAB is implemented in IEEE-754 floating-point arithmetic, all simulations that we perform are IEEE-754 floating-point calculations. It is difficult (if not impossible) to simulate an arbitrary floating-point arithmetic in MATLAB. Therefore, we give theoretical results only.

1st-order filter Consider a 1st-order filter as before and shown in Figure 10.25a. For the finite word-length analysis with floating-point arithmetic we need quantizers after both multiplication and addition to account for rounding off in the mantissa, as shown in Figure 10.25b. Hence there are two noise sources in the the statistical model as shown in Figure 10.25c, where $e_1(n)$ is the noise source in the multiplier, $e_2(n)$ is the noise source in the adder, $\hat{g}(n)$ is an adder sequence prior to quantization, and $\hat{y}(n)$ is the quantized output. Now

$$e_1(n) = \varepsilon_1(n)\,\alpha\,\hat{y}(n-1) \tag{10.68a}$$

$$e_2(n) = \varepsilon_2(n)\,\hat{g}(n) \tag{10.68b}$$

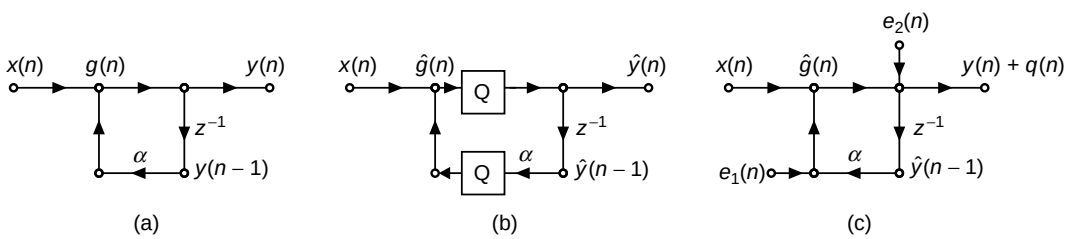


FIGURE 10.25 *First-order IIR filter: (a) structure, (b) finite word-length model for floating-point arithmetic, (c) statistical model for floating-point arithmetic*

where $\varepsilon_1(n)$ and $\varepsilon_2(n)$ are the relative errors in the corresponding quantizers. The exact analysis even for the 1st-order case is tedious; hence we make a few practically reasonable approximations. If the absolute values of the errors are small, then we have $\hat{y}(n-1) \approx y(n-1)$ and $\hat{g}(n) \approx y(n)$; hence from (10.68a) we obtain

$$e_1(n) \approx \alpha\, \varepsilon_1(n)\, y(n-1) \tag{10.69a}$$

$$e_2(n) \approx \varepsilon_2(n)\, y(n) \tag{10.69b}$$

Furthermore, we make the following assumption about the noise sources:

1. $\varepsilon_1(n)$ and $\varepsilon_2(n)$ are white noise sources.
2. $\varepsilon_1(n)$ and $\varepsilon_2(n)$ are uncorrelated with each other.
3. $\varepsilon_1(n)$ and $\varepsilon_2(n)$ are uncorrelated with the input $x(n)$.
4. $\varepsilon_1(n)$ and $\varepsilon_2(n)$ are uniformly distributed between -2^{-B} and 2^{-B}.

Let $x(n)$ be a zero-mean, stationary random sequence. Then $y(n)$ is also a zero-mean, stationary sequence. Hence from (10.69)

$$\sigma_{e_1}^2 = |\alpha|^2 \sigma_{\varepsilon_1}^2 \sigma_y^2 \tag{10.70a}$$

$$\sigma_{e_2}^2 = \sigma_{\varepsilon_2}^2 \sigma_y^2 \tag{10.70b}$$

Let the error in the output due $e_1(n)$ be $q_1(n)$ and that due to $e_2(n)$ be $q_2(n)$. Let $h_1(n)$ and $h_2(n)$ be the corresponding noise impulse responses. Note that $h_1(n) = h_2(n) = h(n) = \alpha^n u(n)$. Then the total error $q(n)$ is

$$q(n) = q_1(n) + q_2(n) \tag{10.71}$$

with

$$\sigma_q^2 = \sigma_{q_1}^2 + \sigma_{q_2}^2 \tag{10.72}$$

where

$$\sigma_{q_1}^2 = \sigma_{e_1}^2 \sum_0^\infty |h_1(n)|^2 \quad \text{and} \quad \sigma_{q_2}^2 = \sigma_{e_2}^2 \sum_0^\infty |h_2(n)|^2 \tag{10.73}$$

Hence using (10.72), (10.73), and (10.70),

$$\sigma_q^2 = \left(\sigma_{e_1}^2 + \sigma_{e_2}^2\right) \left(\frac{1}{1-|\alpha|^2}\right) = \sigma_y^2 \left(\frac{1}{1-|\alpha|^2}\right) \left(|\alpha|^2 \sigma_{\varepsilon_1}^2 + \sigma_{\varepsilon_2}^2\right) \tag{10.74}$$

Using $\sigma_{\varepsilon_1}^2 = \sigma_{\varepsilon_2}^2 = 2^{-2B}/3$, we obtain

$$\sigma_q^2 = \sigma_y^2 \left(\frac{2^{-2B}}{3}\right) \left(\frac{1+|\alpha|^2}{1-|\alpha|^2}\right) \tag{10.75}$$

Therefore

$$\text{SNR} = \frac{\sigma_y^2}{\sigma_q^2} = 3\left(2^{2B}\right) \left(\frac{1-|\alpha|^2}{1+|\alpha|^2}\right) \tag{10.76}$$

or

$$\text{SNR}_{\text{dB}} = 4.77 + 6.02B + 10\log_{10}(1-|\alpha|^2) - 10\log_{10}(1+|\alpha|^2) \quad \textbf{(10.77)}$$

which is also a very informative result. Some comments are in order.

1. The SNR in (10.76) was derived without assuming any input statistics, Hence the result is valid for a large class of inputs including white-noise, narrow-band, or wide-band signals. The floating-point arithmetic does not have to worry about the scaling or limiting input values since it can handle a large dynamic range.
2. Using $0 < \delta = 1 - |\alpha| \ll 1$, the SNR in (10.77) can be put in the form

$$\text{SNR}_{\text{dB}} \approx 4.77 + 6.02B + 10\log_{10}(\delta) = \text{O}(\delta) \quad \textbf{(10.78)}$$

   This is to be compared with the fixed-point result (10.50) where $\text{SNR} \approx \text{O}(\delta^2)$. Thus, the floating-point result is less sensitive to the distance of the pole to the unit circle.
3. In floating-point arithmetic, the output noise variance, σ_q^2, in (10.75) is proportional to σ_y^2. Thus, if the input signal is scaled up, so is the noise variance since σ_y^2 is also scaled up. Hence the SNR remains constant. This again should be compared with the fixed-point case (10.41), in which σ_q^2 is independent of the input signal. Hence if the signal level increases, then σ_y^2, increases, which increases the SNR.

2nd-order filter Similar analysis can be done for the 2nd-order filter with poles close to the unit circle. If the poles are given by $re^{\pm j\theta}$, then we can show that (see [18])

$$\text{SNR} = \frac{\sigma_y^2}{\sigma_q^2} \approx 3\left(2^{2B}\right)\frac{4\delta\sin^{2\theta}}{3+4\cos\theta} \approx \text{O}\left(\delta\right) \quad \textbf{(10.79)}$$

where $\delta = 1 - r$. This again is an approximate result that works very well in practice. In this case again, the SNR depends on δ rather than on δ^2 as in the fixed-point case.

10.3 ROUND-OFF EFFECTS IN FIR DIGITAL FILTERS

We will now turn our attention to the finite word-length effects in FIR digital filters. As before, we will consider the fixed-point and floating-point cases separately. We will then conclude this section with some representative examples.

10.3.1 FIXED-POINT ARITHMETIC

We will consider the effects on two realizations: direct-form and cascade-form. There is no parallel-form realization for FIR filters since we do not have a partial fraction expansion, except for the frequency sampling realization, which can be analyzed using IIR filter techniques. The analysis of FIR filters is much simpler than that for IIR because there are no feedback paths. One consequence of this is the absence of limit cycles.

Direct-form realization Consider an FIR filter of length M (i.e., there are M samples in the impulse response), which is realized using the direct form as shown in Figure 10.26a. The filter coefficients are the samples of the impulse response $h(n)$. We have to introduce quantizers in the vertical branches. If we use the implementation in which each multiplier output is quantized, then we obtain the model shown in Figure 10.26b. On the other hand if we implement the filter in a typical DSP chip, then the final sum is quantized as shown in Figure 10.26c. We will separately consider the effects of round-off noise and scaling (to avoid overflow).

Round-off noise Let the output of the filter in Figure 10.26b due to round-off errors be $\hat{y}(n) = y(n) + q(n)$. Then

$$q(n) = \sum_{k=0}^{M-1} e_k(n) \tag{10.80}$$

where $e_k(n)$ are the noise sources introduced in each vertical branch to account for the rounding operations. Since these noise sources are all independent and identical, the noise power in $q(n)$ is given by

$$\sigma_q^2 = \sum_{0}^{M-1} \sigma_{e_k}^2 = M\,\sigma_e^2 = M\left(\frac{2^{-2B}}{12}\right) = \frac{M}{3}2^{-2(B+1)} \tag{10.81}$$

In Figure 10.26c the output due to the rounding operation is $\hat{y}(n) = y(n) + e(n)$. Hence the noise power in this case is given by

$$\sigma_q^2 = \sigma_e^2 = \frac{1}{3}2^{-2(B+1)} \tag{10.82}$$

which is smaller by a factor of M compared to (10.81) as expected.

Scaling to avoid overflow We assume that the fixed-point numbers have the two's-complement form representation, which is a reasonable assumption. Then we will have to check only the overflow of the total sum. Thus, this analysis is the same for both implementations in Figure 10.26

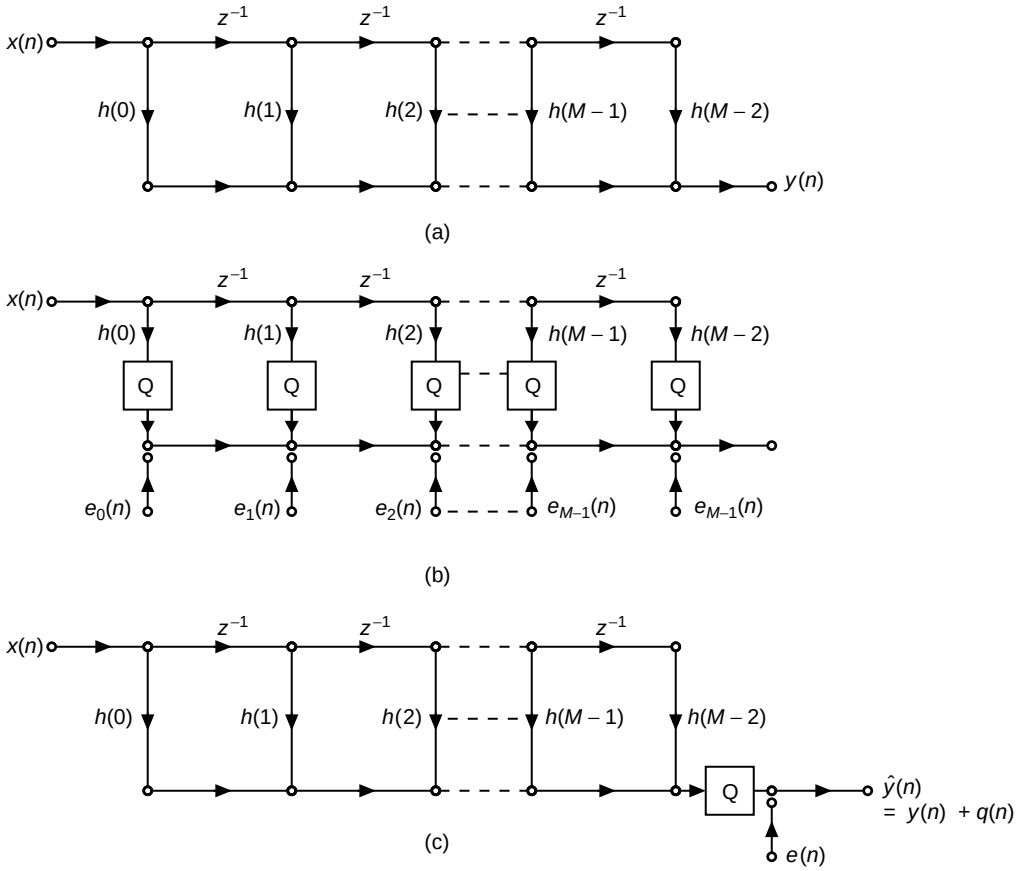


FIGURE 10.26 *Direct-form FIR filter: (a) structure, (b) round-off noise model with quantizers after each multiplier, (c) round-off noise mode with one quantizer after the final sum*

and is similar to that for the IIR filter in (10.42)–(10.44). The upper-bound on $y(n)$ is obtained as

$$|y(n)| = \left|\sum h(k)\, x(n-k)\right| \le X_{\max} \sum |h(n)| \qquad \textbf{(10.83)}$$

where $X_{\max}$ is the upper-bound on $x(n)$. To guarantee that $|y(n)| \le 1$, we need the scaling factor $X_{\max}$ on $x(n)$ as

$$X_{\max} \le \frac{1}{\sum |h(n)|} \qquad \textbf{(10.84)}$$

which is the most conservative scaling factor. There are other scaling factors, depending on the applications—for example, the narrowband

signals use

$$X_{\max} \leq \frac{1}{\max |H(e^{j\omega})|}$$

and wideband random signals use

$$X_{\max} \leq \frac{1}{4\sigma_x \sqrt{\sum |h(n)|^2}}.$$

Using (10.84) and assuming that $x(n)$ is uniformly distributed over $[-X_{\max}, +X_{\max}]$, the input signal power is given by

$$\sigma_x^2 = \frac{X_{\max}^2}{3} = \frac{1}{3\left(\sum |h(n)|\right)^2} \tag{10.85}$$

Furthermore, assuming that $x(n)$ is also a white sequence, the output signal power is given by

$$\sigma_y^2 = \sigma_x^2 \sum |h(n)|^2 = \frac{1}{3} \frac{\sum |h(n)|^2}{\left(\sum |h(n)|\right)^2} \tag{10.86}$$

Thus, the output SNR is

$$\text{SNR} = \frac{\sigma_y^2}{\sigma_q^2} = \frac{2^{2(B+1)}}{A} \left[\frac{\sum |h(n)|^2}{\left(\sum |h(n)|\right)^2} \right] \tag{10.87}$$

where $A = M$ for the model in Figure 10.26b or $A = 1$ for the model in Figure 10.26c. The corresponding SNR in dB is

$$\text{SNR}_{\text{dB}} = 6.02 + 6.02B + 10 \log_{10} \left(\frac{\sum |h(n)|^2}{\left(\sum |h(n)|\right)^2} \right) - 10 \log_{10} A \tag{10.88}$$

10.3.2 ANALYSIS USING MATLAB

This simulation in MATLAB can be done in parallel fashion since there is no feedback path for the multiplication quantization errors. Using the function `Qfix` function with `'round'` mode, we will compute the quantized multiplier output. In the case of M quantizers, assuming two's-complement format, we will use the `'twosc'` mode for each quantizer. Only the final sum will be quantized and saturated. In the case of one quantizer, we need the `'satur'` mode. These simulation steps are detailed in the following example.

□ **EXAMPLE 10.13** Let a fourth-order ($M = 5$) FIR filter be given by

$$H(z) = 0.1 + 0.2z^{-1} + 0.4z^{-2} + 0.2z^{-3} + 0.1z^{-4} \tag{10.89}$$

which is implemented as a direct form with $B = 12$ fractional bit quantizers. Compute SNRs for models in Figure 10.26b and c and verify them using MATLAB simulations.

Solution We will need the quantities $\sum |h(n)|^2$ and $(\sum |h(n)|)^2$. These quantities should be computed using 12-bit quantization of the filter coefficients. These values using the quantized numbers are $\sum |h(n)|^2 = 0.2599$ and $(\sum |h(n)|)^2 = 1$. Using (10.88), the output SNR is 65.42 dB for 5 multipliers and is 72.41 dB for 1 multiplier. The following MATLAB script evaluates these and other quantities.

```
% Example Parameters
B = 12;                    % # of fractional bits
N = 100000;                % # of samples
xn = (2*rand(1,N)-1);      % Input sequence - Uniform Distribution
h = [0.1,0.2,0.4,0.2,0.1]; % Filter parameters
M = length(h);

% Local variables
  bM = 7; DbM = 2^bM;    % bin parameter
  BB = 2^B;              % useful factor in quantization
   K = round(DbM/2);     % Half number of bins
bins = [-K+0.5:1:K-0.5]; % Bin values from -K to K
   Q = bins/DbM;         % Normalized bins
 YTN = 2^(-bM);          % Ytick marks interval
 YLM = 4*YTN;            % Yaxis limit

% Quantize the input and the filter coefficients
 h = QFix(h,B,'round','satur'); % h quantized to B bits
Xm = 1/sum(abs(h));             % Scaling factor
xn = QFix(Xm*xn,B,'round','satur');% Scaled Input quant to B bits

% Filter output without multiplication quantization
yn = filter(h,1,xn);     % output using filter routine

% Filter output with multi quant (5 multipliers)
x1 = [zeros(1,1),xn(1:N-1)]; x2 = [zeros(1,2),xn(1:N-2)];
x3 = [zeros(1,3),xn(1:N-3)]; x4 = [zeros(1,4),xn(1:N-4)];
h0x0 = QFix(h(1)*xn,B,'round','twosc');
h1x1 = QFix(h(2)*x1,B,'round','twosc');
h2x2 = QFix(h(3)*x2,B,'round','twosc');
h3x3 = QFix(h(4)*x3,B,'round','twosc');
h4x4 = QFix(h(5)*x4,B,'round','twosc');
yq = h0x0+h1x1+h2x2+h3x3+h4x4;
yq = QFix(yq,B,'round','satur');

% Output Error Analysis
   qn = yn-yq;                          % Outout error sequence
varyn = var(yn); varqn = var(qn);       % Signal and noise power
qqmax = max(qn); qqmin = min(qn);       % Maximun and minimum of the error
qnmax = max(abs([qqmax,qqmin]));        % Absolute maximum range of the error
qnavg = mean(qn); qnstd = std(qn);      % Mean and std dev of the error
```

```
   qn = round(qn*(2^bM)/(2*qnmax)+0.5); % Normalized en (interger between -K & K)
   qn = sort([qn,-K:1:(K+1)]);          %
    H = diff(find(diff(qn)))-1;         % Error histogram
    H = H/N;                            % Normalized histogram
 Hmax = max(H); Hmin = min(H);          % Max and Min of the normalized histogram

% Output SNRs
SNR_C = 10*log10(varyn/varqn); % Computed SNR
SNR_T = 6.02 + 6.02*B + 10*log10(sum(h.*h)/Xm^2) - 10*log10(M); % Theoretical SNR

% Filter output with multi quant (1 multiplier)
yq = QFix(yn,B,'round','satur');

% Output Error Analysis
   qn = yn-yq;                          % Outout error sequence
varyn = var(yn); varqn = var(qn);       % Signal and noise power
qqmax = max(qn); qqmin = min(qn);       % Maximun and minimum of the error
qnmax = max(abs([qqmax,qqmin]));        % Absolute maximum range of the error
qnavg = mean(qn); qnstd = std(qn);      % Mean and std dev of the error
   qn = round(qn*(2^bM)/(2*qnmax)+0.5); % Normalized en (interger between -K & K)
   qn = sort([qn,-K:1:(K+1)]);          %
    H = diff(find(diff(qn)))-1;         % Error histogram
    H = H/N;                            % Normalized histogram
 Hmax = max(H); Hmin = min(H);          % Max and Min of the normalized histogram

% Output SNRs
SNR_C = 10*log10(varyn/varqn); % Computed SNR
SNR_T = 6.02 + 6.02*B + 10*log10(sum(h.*h)/Xm^2); % Theoretical SNR
```

The computed and theoretical SNRs as well as output error histograms for the two models are shown in Figure 10.27. The top plot shows the histogram when five multipliers are used. The output error has Gaussian-like distribution with SNR equal to 65.42 dB, which agrees with the theoretical value. The bottom plot show the histogram when one multiplier is used. As expected, the error is uniformly distributed with SNR equal to 72.43 dB, which also agrees with the theoretical one. □

Cascade-form realization Let the filter be realized by a cascade of K, 2nd-order ($M = 3$) sections given by

$$H(z) = \sum_{i=1}^{K} H_i(z) \quad \text{where } H_i(z) = \beta_{0i} + \beta_{1i}\, z^{-1} + \beta_{2i}\, z^{-2} \qquad \textbf{(10.90)}$$

as shown in Figure 10.28. The overall length of the filter is $M = 2K + 1$. Figure 10.28 also shows the finite word-length model for the cascade form,

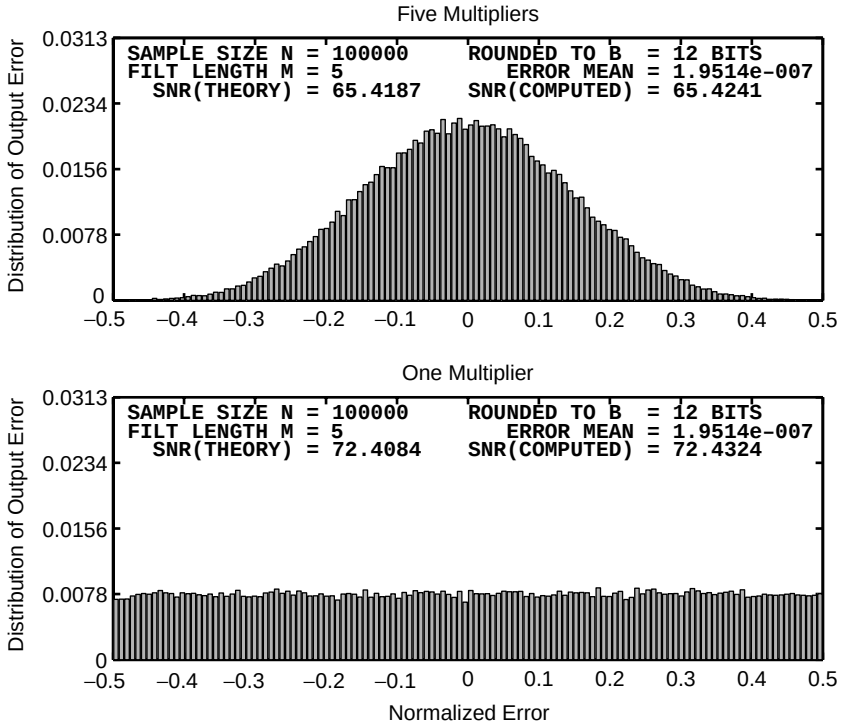


FIGURE 10.27 *Multiplication quantization effects for the direct-form FIR filter in Example 10.13*

in which quantization noise sources, $e_i(n)$ $1 \leq i \leq K$, at each section's output are incorporated. Let $y(n)$ be the output due to input $x(n)$, and let $q(n)$ be the output due to all noise sources. We make the following reasonable assumptions:

1. The sections are implemented using the MAC (multiply-accumulate) architecture so that there is only one independent noise source in each section that contributes to $e_i(n)$. The other possibility of three multipliers in each section is straightforward.

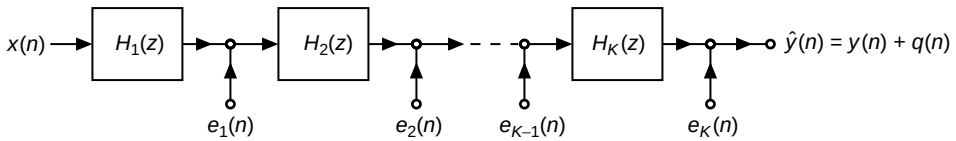


FIGURE 10.28 *Cascade form FIR filter structure with noise sources inserted for multiplication quantization*

2. The noise sources are independent of each other—that is,

$$e_i(n) \perp e_j(n) \quad \text{for } i \neq j$$

3. Each noise source is a white noise source with $\sigma_{e_i}^2 = 2^{-2B}/12$.

We will now consider the issues of round-off noise and scaling (to prevent overflow) for the cascade-form realization.

Round-off noise Let the noise impulse response at the output from the $e_i(n)$ node be denoted by $g_i(n)$. Then the length of $g_i(n)$ is equal to $(M - 2i)$. Let $q_i(n)$ be the output noise due to $e_i(n)$. Then its power is given by

$$\sigma_{q_i}^2 = \sigma_{e_i}^2 \sum_0^{M-2i} |g_i(n)|^2 = \frac{2^{-2B}}{12} \sum_0^{M-2i} |g_i(n)|^2 \tag{10.91}$$

Since $q(n) = \sum_{i=1}^{K} q_i(n)$ we obtain the total noise power as

$$\sigma_q^2 = \sum_{i=1}^{K} \sigma_{q_i}^2 = \frac{2^{-2B}}{12} \left(\sum_{i=1}^{K} \sum_{n=1}^{M-2i} |g_i(n)|^2 \right) \tag{10.92}$$

The expression $\sum_{i=1}^{K} \sum_{n=1}^{M-2i} |g_i(n)|^2$ shows that the error power depends on the order of the cascade connections. It has been shown that for the majority of the orderings the noise power is approximately the same.

Scaling to prevent overflow From Figure 10.28 we note that one must prevent overflow at each node. Let $h_k(n)$ be the impulse response at each node k; then we need a scaling constant $X_{\max}$ as

$$X_{\max} = \frac{1}{\max_k \sum |h_k(n)|}$$

so that $|y(n)| \leq 1$. Clearly, this is a very conservative value. A better approach is to scale the impulse responses of every section $\{h_i(n)\}$ so that $\sum |h_i| = 1$ for each i. Hence the output of every section is limited between -1 and $+1$ if the input $x(n)$ is distributed over the same interval. Assuming that $x(n)$ is uniformly distributed over $[-1, +1]$ and is white, the output signal power is

$$\sigma_y^2 = \sigma_x^2 \sum_0^{M-1} |h(n)|^2 = \frac{1}{3} \sum_0^{M-1} |h(n)|^2 \tag{10.93}$$

where $h(n)$ is the overall impulse response of the filter. Let $\hat{g}_i$ be the corresponding scaled impulse responses in (10.92). Now the output SNR can be computed as

$$\text{SNR} = \frac{\sigma_y^2}{\sigma_q^2} = 2^{2(B+1)} \frac{\sum_0^{M-1} |h(n)|^2}{\left(\sum_{i=1}^{K} \sum_{n=1}^{M-2i} |\hat{g}_i(n)|^2 \right)} \tag{10.94}$$

or

$$\text{SNR}_{\text{dB}} = 6.02(B+1)+10\log_{10}\left(\sum_{0}^{M-1}|h(n)|^2\right)-10\log_{10}\left(\sum_{i=1}^{K}\sum_{n=1}^{M-2i}|\hat{g}_i(n)|^2\right) \tag{10.95}$$

10.3.3 ANALYSIS USING MATLAB

Using the `casfiltr` function, we can compute the output of the infinite-precision cascade structure. Using the scaling approach outlined above, each second-order section can be scaled and used in the simulation of quantized outputs. Again, all calculations can be done in vector fashion, which improves the execution speed. These and other simulation steps are detailed in the following example.

□ **EXAMPLE 10.14** Consider the 4th-order FIR filter given in Example 10.13. Its cascade-form realization has two sections along with a gain constant b_0, which can be obtained using the `dir2cas` function:

$$H_1(z) = 1+1.4859z^{-1}+2.8901z^{-2}, \quad H_2(z) = 1+0.5141z^{-1}+0.3460z^{-2}, \text{ and } b_0 = 0.1 \tag{10.96}$$

Note that some of these coefficients are greater than 1, which will cause problems with coefficient quantization when only B fractional bits are used. Hence we need to scale each section as explained. The scaled values are

$$\hat{H}_1(z) = 0.1860+0.2764z^{-1}+0.5376z^{-2}, \quad \hat{H}_2(z) = 0.5376+0.2764z^{-1}+0.1860z^{-2} \tag{10.97}$$

and $\hat{b}_0 = 1$. Thus we do not need to scale the input. Now $\hat{g}_1(n) = \hat{h}_2(n)$ and $\hat{g}_2(n) = 1$ in (10.94). Thus, from (10.95) the output SNR is 70.96 dB, which compares well with the one-multiplier direct-form implementation (72.41 dB). These calculations and error histogram plotting are illustrated in the following MATLAB script.

```
% Example Parameters
B = 12;                    % # of fractional bits
N = 100000;                % # of samples
xn = (2*rand(1,N)-1);      % Input sequence - Uniform Distribution
h = [0.1,0.2,0.4,0.2,0.1]; % Filter parameters
M = length(h);             % Filter length
[b0,Bh,Ah] = dir2cas(h,1); % Cascade sections
h1 = Bh(1,:);              % Section-1
h2 = Bh(2,:);              % Section-2
h1 = h1/sum(h1);           % Scaled so Gain=1
h2 = h2/sum(h2);           % Scaled so Gain=1
% Local variables
  bM = 7; DbM = 2^bM;    % bin parameter
  BB = 2^B;              % useful factor in quantization
   K = round(DbM/2);     % Half number of bins
```

```
bins = [-K+0.5:1:K-0.5]; % Bin values from -K to K
   Q = bins/DbM;         % Normalized bins
 YTN = 2^(-bM);          % Ytick marks interval
 YLM = 20*YTN;           % Yaxis limit
% Quantize the input and the filter coefficients
h1 = QFix(h1,B,'round','satur'); % h1 quantized to B bits
h2 = QFix(h2,B,'round','satur'); % h1 quantized to B bits
xn = QFix(xn,B,'round','satur'); % Input quantized to B bits
% Filter output without multiplication quantization
yn = casfiltr(b0,Bh,Ah,xn); % output using Casfiltr routine
% Filter output with multi quant (1 multiplier/section)
xq = QFix(xn,B,'round','satur'); % Section-1 scaled input
wn = filter(h1,1,xq);            % Sec-1 unquantized output
wq = QFix(wn,B,'round','satur'); % Sec-1 quantized output
wq = QFix(wq,B,'round','satur'); % Section-2 scaled input
yq = filter(h2,1,wq);            % Sec-2 unquantized output
yq = QFix(yq,B,'round','satur'); % Sec-2 quantized output
% Output Error Analysis
   qn = yn-yq;                          % Outout error sequence
varyn = var(yn); varqn = var(qn);       % Signal and noise power
qqmax = max(qn); qqmin = min(qn);       % Maximun and minimum of the error
qnmax = max(abs([qqmax,qqmin]));        % Absolute maximum range of the error
qnavg = mean(qn); qnstd = std(qn);      % Mean and std dev of the error
   qn = round(qn*(2^bM)/(2*qnmax)+0.5); % Normalized en (interger between -K & K)
   qn = sort([qn,-K:1:(K+1)]);          %
    H = diff(find(diff(qn)))-1;         % Error histogram
    H = H/N;                            % Normalized histogram
 Hmax = max(H); Hmin = min(H);          % Max and Min of the normalized histogram
% Output SNRs
SNR_C = 10*log10(varyn/varqn); % Computed SNR
SNR_T = 6.02*(B+1) + 10*log10(sum(h.*h)) ...
    - 10*log10(1+sum(h2.*h2)); % Theoretical SNR
```

The plot is shown in Figure 10.29. The error distribution appears to have a Gaussian envelope, but the error is not continuously distributed. This behavior indicates that the output error takes only a fixed set of values, which is due to a particular set of coefficient values. The computed SNR is 70.85 dB, which agrees with the above theoretical value. Thus, our assumptions are reasonable. □

10.3.4 FLOATING-POINT ARITHMETIC

Analysis for the floating-point arithmetic is more complicated and tedious. Hence we will consider only the direct-form realization with simplified assumptions. Figure 10.30 shows a direct-form realization with a floating-point arithmetic model. In this realization, $\{\eta_i(n)\}$, $1 \le i \le M-1$ are

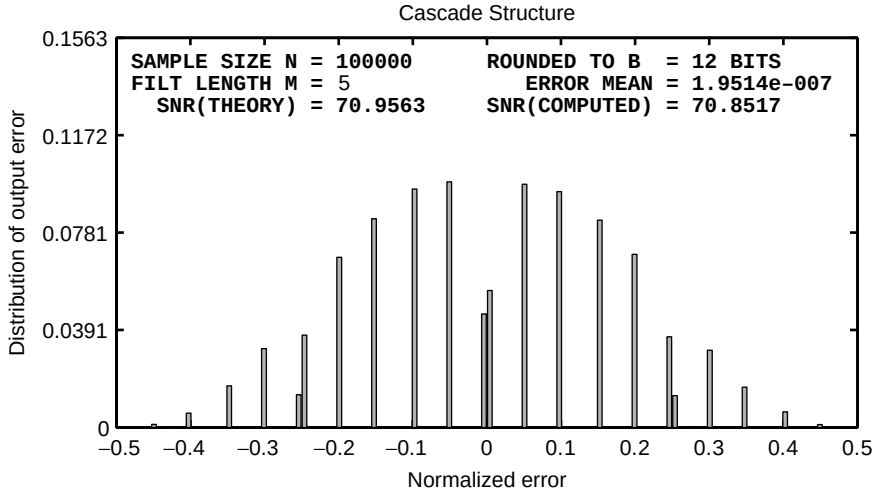


FIGURE 10.29 *Multiplication quantization effects for the cascade-form FIR filter in Example 10.14*

the relative errors in adders and $\{\varepsilon_i(n)\}$, $0 \le i \le M-1$ are the relative errors in multipliers, with $|\eta_i| \le 2^{-2B}$ and $|\varepsilon_i| \le 2^{-2B}$.

Let $A(n,k)$ be the gain from the kth multiplier to the output node, which is given by

$$A(n,k) = \begin{cases} (1+\varepsilon_k(n)) \prod_{r=k}^{M-1} (1+\eta_r(n)), & k \neq 0; \\ (1+\varepsilon_0(n)) \prod_{r=k}^{M-1} (1+\eta_r(n)), & k = 0. \end{cases} \tag{10.98}$$

Let $\hat{y}(n) \triangleq y(n) + q(n)$ be the overall output where $y(n)$ is the output due to the input $x(n)$ and $q(n)$ is the output due to noise sources. Then

$$\hat{y}(n) = \sum_{k=0}^{M-1} A(n,k)\, h(k)\, x(n-k) \tag{10.99}$$

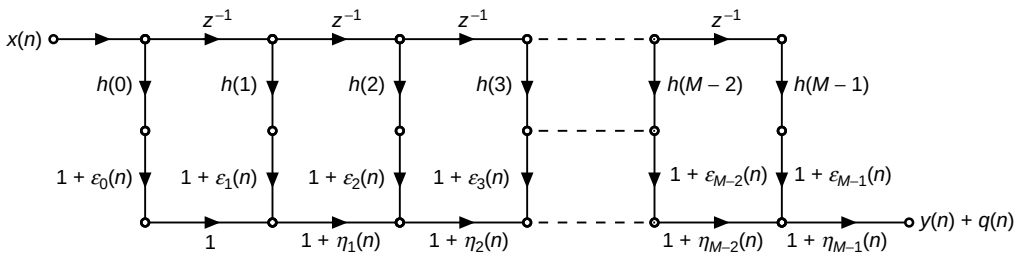


FIGURE 10.30 *Multiplication quantization model for direct-form floating-point implementation of an FIR filter*

Subtracting $y(n) = \sum_{k=0}^{M-1} h(k)\, x(n-k)$ from (10.99), we obtain

$$q(n) = \sum_{k=0}^{M-1} \{A(n,k) - 1\}\, h(k)\, x(n-k) \tag{10.100}$$

Now from (10.98), the average value of $A(n,k)$ is $\mathrm{E}A(n,k) = 1$ and the average power of $A(n,k)$ is

$$\begin{aligned} \mathrm{E}[A^2(n,k)] &= \left(1 + \frac{1}{3}\, 2^{-2B}\right)^{M+1-k} \\ &\approx 1 + (M+1-k)\frac{2^{-2B}}{3} \quad \text{for small } 2^{-2B} \end{aligned} \tag{10.101}$$

Assuming that the input signal $x(n)$ is also a white sequence with variance σ_x^2, then from (10.101) the noise power is given by

$$\sigma_q^2 = \frac{(M+1)2^{-2B}}{3}\, \sigma_x^2 \sum_{k=0}^{M-1} |h(k)|^2 \left(1 - \frac{k}{M+1}\right) \tag{10.102}$$

Since $(1 - \frac{k}{M+1}) \leq 1$ and using $\sigma_y^2 = \sigma_x^2 \sum_{k=0}^{M-1} |h(k)|^2$ the noise power σ_q^2 is upper bounded by

$$\sigma_q^2 \leq (M+1)\, \frac{2^{-2B}}{3}\, \sigma_y^2 \tag{10.103}$$

or the SNR is lower bounded by

$$\mathrm{SNR} \geq \frac{3}{M+1} 2^{2B} \tag{10.104}$$

Equation (10.104) shows that it is best to compute products in order of increasing magnitude.

□ **EXAMPLE 10.15** Again consider the 4th-order FIR filter given in Example 10.13 in which $M = 5$, $B = 12$, and $h(n) = \{0.1, 0.2, 0.4, 0.2, 0.1\}$. From (10.104), the SNR is lower bounded by

$$\mathrm{SNR}_{dB} \geq 10 \log_{10} \left(\frac{3}{M+1} 2^{24}\right) = 69.24 \text{ dB}$$

and the approximate value from (10.102) is 71 dB, which is comparable to the fixed-point value of 72 dB. Note that the fixed-point results would degrade with less than optimum scaling (e.g., if signal amplitude were 10 dB down), whereas the floating point SNR would remain the same. To counter this, one could put a variable scaling factor A on the fixed-point system, which is then getting close to the full floating-point system. In fact, floating-point is nothing but fixed-point with variable scaling—that is, a scaling by a power of two (or shifting) at each multiplication and addition. □

10.4 PROBLEMS

P10.1 Let $x(n) = 0.5[\cos(n/17) + \sin(n/23)]$. For the following parts, use $500,000$ samples of $x(n)$ and the `StatModelR` function.

1. Quantize $x(n)$ to $B = 2$ bits, and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.
2. Quantize $x(n)$ to $B = 4$ bits, and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.
3. Quantize $x(n)$ to $B = 6$ bits, and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.

P10.2 Let $x(n) = \frac{1}{3}[\cos(0.1\pi n) + \sin(0.2\pi n) + \sin(0.4\pi n)]$. For the following parts use $500,000$ samples of $x(n)$ and the `StatModelR` function.

1. Quantize $x(n)$ to $B = 2$ bits, and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.
2. Quantize $x(n)$ to $B = 4$ bits, and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.
3. Quantize $x(n)$ to $B = 6$ bits, and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.

P10.3 Let a real, causal, and stable IIR filter be given by

$$H(z) = R_0 + \sum_{k=1}^{N-1} \frac{R_k}{z - p_k} \tag{10.105}$$

where all poles are distinct. Using (10.16), (10.18a), and (10.105), show that

$$\frac{\sigma_q^2}{\sigma_e^2} = R_0^2 + \sum_{k=1}^{N-1} \sum_{\ell=1}^{N-1} \frac{R_k R_\ell^*}{1 - p_k p_\ell^*}$$

P10.4 Consider the lowpass digital filter designed in Problem P6.39. The input to this filter is an independent and identically distributed Gaussian sequence with zero-mean and variance equal to 0.1.

1. Determine the variance of the filter output process using the `VarGain` function.
2. Determine numerically the variance of the output process by generating 500,000 samples of the input sequence. Comment on your results.

P10.5 Design an elliptic bandpass digital filter that has a lower stopband of 0.3π, a lower passband of 0.4π, an upper passband of 0.5π, and an upper stopband of 0.65π. The passband ripple is 0.1 dB and the stopband attenuation is 50 dB. The input signal is a random sequence whose components are independent and uniformly distributed between -1 and 1.

1. Determine the variance of the filter output process using the `VarGain` function.
2. Determine numerically the variance of the output process by generating 500,000 samples of the input sequence. Comment on your results.

P10.6 Consider the 1st-order recursive system $y(n) = 0.75\,y(n-1) + 0.125\delta(n)$ with zero initial conditions. The filter is implemented in 4-bit (including sign) fixed-point two's-complement fractional arithmetic. Products are rounded to 3-bits.

1. Determine and plot the first 20 samples of the output using saturation limiter for the addition. Does the filter go into a limit cycle?
2. Determine and plot the first 20 samples of the output using two's-complement overflow for the addition. Does the filter go into a limit cycle?

P10.7 Repeat Problem P10.6 when products are truncated to 3 bits.

P10.8 Consider the 2nd-order recursive system $y(n) = 0.125\delta(n) - 0.875\,y(n-2)$ with zero initial conditions. The filter is implemented in 5-bit (including sign) fixed-point two's-complement fractional arithmetic. Products are rounded to 4-bits.

1. Determine and plot the first 30 samples of the output using a saturation limiter for the addition. Does the filter go into a limit cycle?
2. Determine and plot the first 30 samples of the output using two's-complement overflow for the addition. Does the filter go into a limit cycle?

P10.9 Repeat Problem P10.8 when products are truncated to 4 bits.

P10.10 Let $x(n) = \frac{1}{4}[\sin(n/11) + \cos(n/13) + \sin(n/17) + \cos(n/19)]$ and $c = 0.7777$. For the following parts use $500,000$ samples of $x(n)$ and the `StatModelR` function.

1. Quantize $c\,x(n)$ to $B = 4$ bits, and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.
2. Quantize $cx(n)$ to $B = 8$ bits, and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.
3. Quantize $cx(n)$ to $B = 12$ bits, and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.

P10.11 Let $x(n) =$ be a random sequence uniformly distributed between -1 and 1, and let $c = 0.7777$. For the following parts, use $500,000$ samples of $x(n)$ and the `StatModelR` function.

1. Quantize $c\,x(n)$ to $B = 4$ bits and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.
2. Quantize $cx(n)$ to $B = 8$ bits and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.
3. Quantize $cx(n)$ to $B = 12$ bits and plot the resulting distributions for the error signals $e_1(n)$ and $e_2(n)$. Comment on these plots.

P10.12 Consider an LTI system with the input $x(n)$ and output $y(n)$

$$y(n) = b_0x(n) + b_1x(n-1) + a_1y(n-1) \tag{10.106}$$

1. Draw the direct-form I structure for the above system.
2. Let $e_{b_0}(n)$ denote the multiplication quantization error resulting from the product $b_0x(n)$, $e_{b_1}(n-1)$ from the product $b_1x(n-1)$, and $e_{a_1}(n-1)$ from the product $a_1y(n-1)$ in the direct-form I realization. Draw an equivalent structure that contains only one noise source.
3. Draw an equivalent system that can be used to study multiplication quantization error for the system in (10.106). The input to this system should be the noise source in part 2, and the output should be the overall output error $q(n)$.
4. Using the model in part 3, determine an expression for the variance of the output error $e(n)$.

P10.13 Let the system be given by $y(n) = a\,y(n-1) + x(n)$. Let $a = 0.7$, which is quantized to B (fractional) bits in the filter realization. Let the input sequence be $x(n) = \sin(n/11)$, which is properly scaled to avoid overflow in the adder and quantized to B bits prior to filtering. The multiplications in the filtering operations are also quantized to B bits.

1. Let $B = 5$. Generate 100,000 samples of $x(n)$, and filter through the system with multiplication quantization. Compute the true output, the quantized output, the output error, and the output SNR. Provide a plot of normalized histogram, and comment on the results.
2. Let $B = 10$. Generate 100,000 samples of $x(n)$ and filter through the system with multiplication quantization. Compute the true output, the quantized output, the output error, and the output SNR. Provide a plot of normalized histogram, and comment on the results.

P10.14 Let the system be given by $y(n) = a\,y(n-1) + x(n)$. Let $a = 0.333$, which is quantized to B (fractional) bits in the filter realization. Let the input sequence be $x(n) = \sin(n/11)$, which is properly scaled to avoid overflow in the adder and quantized to B bits prior to filtering. The multiplications in the filtering operations are also quantized to B bits.

1. Let $B = 5$. Generate 100,000 samples of $x(n)$, and filter through the system with multiplication quantization. Compute the true output, the quantized output, the output error, and the output SNR. Provide a plot of normalized histogram and comment on the results.
2. Let $B = 10$. Generate 100,000 samples of $x(n)$, and filter through the system with multiplication quantization. Compute the true output, the quantized output, the output error, and the output SNR. Provide a plot of normalized histogram and comment on the results.

P10.15 Consider the 2nd-order IIR filter given in (10.51) with $r = 0.8$ and $\theta = \pi/4$. The input to this filter is $x(n) = \sin(n/23)$.

1. Investigate the multiplication quantization error behavior of this filter for $B = 5$ bits. Determine the true output SNR, the computed output SNR, and the upper and lower bounds of the SNR. Plot the normalized histogram of the output error.
2. Investigate the multiplication quantization error behavior of this filter for $B = 10$ bits. Determine the true output SNR, the computed output SNR, and the upper and lower bounds of the SNR. Plot the normalized histogram of the output error.

P10.16 Consider the second-order IIR filter given in (10.51) with $r = 0. - 8$ and $\theta = 2\pi/3$. The input to this filter is $x(n) = \sin(n/23)$.

1. Investigate the multiplication quantization error behavior of this filter for $B = 5$ bits. Determine the true output SNR, the computed output SNR, and the upper and lower bounds of the SNR. Plot the normalized histogram of the output error.
2. Investigate the multiplication quantization error behavior of this filter for $B = 10$ bits. Determine the true output SNR, the computed output SNR, and the upper and lower bounds of the SNR. Plot the normalized histogram of the output error.

P10.17 Consider a 5th-order FIR system given by

$$H(z) = 0.1 + 0.2z^{-1} + 0.3z^{-2} + 0.3z^{-3} + 0.2z^{-4} + 0.1z^{-5}$$

which is implemented in a direct form using $B = 10$ bits. Input to the filter is a random sequence whose samples are independent and identically distributed over $[-1, 1]$.

1. Investigate the multiplication quantization errors when all 6 multipliers are used in the implementation. Provide a plot of the normalized histogram of the output error.
2. Investigate the multiplication quantization errors when one multiplier is used in the implementation. Provide a plot of the normalized histogram of the output error.

P10.18 Consider a 4th-order FIR system given by

$$H(z) = 0.1 + 0.2z^{-1} + 0.3z^{-2} + 0.2z^{-3} + 0.1z^{-4}$$

which is implemented in a cascade form containing second-order sections. Input to the filter is a random sequence whose samples are independent and identically distributed over $[-1, 1]$.

1. Investigate the multiplication quantization errors when $B = 6$ bits is used in the implementation. Provide a plot of the normalized histogram of the output error.
2. Investigate the multiplication quantization errors when $B = 12$ bits is used in the implementation. Provide a plot of the normalized histogram of the output error.

CHAPTER 11

Applications in Adaptive Filtering

In Chapters 7 and 8 we described methods for designing FIR and IIR digital filters to satisfy some desired specifications. Our goal was to determine the coefficients of the digital filter that met the desired specifications.

In contrast to the filter design techniques considered in those two chapters, there are many digital signal processing applications in which the filter coefficients cannot be specified a priori. For example, let us consider a high-speed modem that is designed to transmit data over telephone channels. Such a modem employs a filter called a channel equalizer to compensate for the channel distortion. The modem must effectively transmit data through communication channels that have different frequency response characteristics and hence result in different distortion effects. The only way in which this is possible is if the channel equalizer has *adjustable coefficients* that can be optimized to minimize some measure of the distortion, on the basis of measurements performed on the characteristics of the channel. Such a filter with adjustable parameters is called an *adaptive filter*, in this case an *adaptive equalizer*.

Numerous applications of adaptive filters have been described in the literature. Some of the more noteworthy applications include (1) adaptive antenna systems, in which adaptive filters are used for beam steering and for providing nulls in the beam pattern to remove undesired interference [29]; (2) digital communication receivers, in which adaptive filters are used to provide equalization of intersymbol interference and for channel

identification [21]; (3) adaptive noise canceling techniques, in which an adaptive filter is used to estimate and eliminate a noise component in some desired signal [27, 9, 15]; and (4) system modeling, in which an adaptive filter is used as a model to estimate the characteristics of an unknown system. These are just a few of the best known examples on the use of adaptive filters.

Although both IIR and FIR filters have been considered for adaptive filtering, the FIR filter is by far the most practical and widely used. The reason for this preference is quite simple. The FIR filter has only adjustable zeros, and hence it is free of stability problems associated with adaptive IIR filters that have adjustable poles as well as zeros. We should not conclude, however, that adaptive FIR filters are always stable. On the contrary, the stability of the filter depends critically on the algorithm for adjusting its coefficients.

Of the various FIR filter structures that we may use, the direct form and the lattice form are the ones often used in adaptive filtering applications. The direct form FIR filter structure with adjustable coefficients $h(0), h(1), \ldots, h(N-1)$ is illustrated in Figure 11.1. On the other hand, the adjustable parameters in an FIR lattice structure are the reflection coefficients K_n shown in Figure 6.18.

An important consideration in the use of an adaptive filter is the criterion for optimizing the adjustable filter parameters. The criterion must not only provide a meaningful measure of filter performance, but it must also result in a practically realizable algorithm.

One criterion that provides a good measure of performance in adaptive filtering applications is the least-squares criterion, and its counterpart in a statistical formulation of the problem, namely, the mean-square-error (MSE) criterion. The least squares (and MSE) criterion results in a quadratic performance index as a function of the filter coefficients, and hence it possesses a single minimum. The resulting algorithms for adjusting the coefficients of the filter are relatively easy to implement.

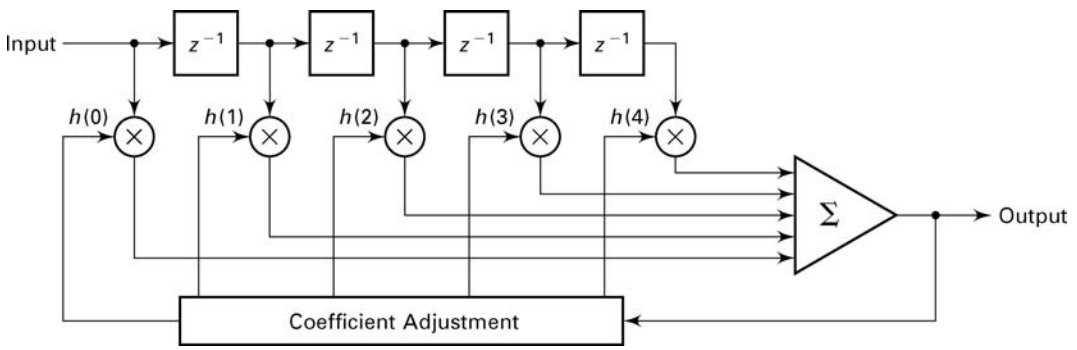


FIGURE 11.1 *Direct-form adaptive FIR filter*

In this chapter we describe a basic algorithm, called the *least-mean-square* (LMS) *algorithm*, to adaptively adjust the coefficients of an FIR filter. The adaptive filter structure that will be implemented is the direct form FIR filter structure with adjustable coefficients $h(0), h(1), \ldots, h(N-1)$, as illustrated in Figure 11.1. After we describe the LMS algorithm, we apply it to several practical systems in which adaptive filters are employed.

11.1 LMS ALGORITHM FOR COEFFICIENT ADJUSTMENT

Suppose we have an FIR filter with adjustable coefficients $\{h(k), 0 \leq k \leq N-1\}$. Let $\{x(n)\}$ denote the input sequence to the filter, and let the corresponding output be $\{y(n)\}$, where

$$y(n) = \sum_{k=0}^{N-1} h(k)x\,(n-k)\,, \quad n = 0, \ldots, M \tag{11.1}$$

Suppose that we also have a desired sequence $\{d(n)\}$ with which we can compare the FIR filter output. Then we can form the error sequence $\{e(n)\}$ by taking the difference between $d(n)$ and $y(n)$, that is,

$$e(n) = d(n) - y(n), \quad n = 0, \ldots, M \tag{11.2}$$

The coefficients of the FIR filter will be selected to minimize the sum of squared errors. Thus we have

$$\begin{aligned} \mathcal{E} &= \sum_{n=0}^{M} e^2(n) = \sum_{n=0}^{M} \left[d(n) - \sum_{k=0}^{N-1} h(k)x\,(n-k) \right]^2 \\ &= \sum_{n=0}^{M} d^2(n) - 2\sum_{k=0}^{N-1} h(k) r_{dx}(k) + \sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1} h(k)h\,(\ell)\, r_{xx}\,(k-\ell) \end{aligned} \tag{11.3}$$

where, by definition,

$$r_{dx}(k) = \sum_{n=0}^{M} d(n)x\,(n-k)\,, \quad 0 \leq k \leq N-1 \tag{11.4}$$

$$r_{xx}(k) = \sum_{n=0}^{M} x(n)x\,(n+k)\,, \quad 0 \leq k \leq N-1 \tag{11.5}$$

We call $\{r_{dx}(k)\}$ the crosscorrelation between the desired output sequence $\{d(n)\}$ and the input sequence $\{x(n)\}$, and $\{r_{xx}(k)\}$ is the autocorrelation sequence of $\{x(n)\}$.

The sum of squared errors $\mathcal{E}$ is a quadratic function of the FIR filter coefficients. Consequently, the minimization of $\mathcal{E}$ with respect to the filter coefficients $\{h(k)\}$ results in a set of linear equations. By differentiating $\mathcal{E}$ with respect to each of the filter coefficients, we obtain

$$\frac{\partial \mathcal{E}}{\partial h(m)} = 0, \quad 0 \le m \le N-1 \tag{11.6}$$

and hence

$$\sum_{k=0}^{N-1} h(k) r_{xx}(k-m) = r_{dx}(m), \quad 0 \le m \le N-1 \tag{11.7}$$

This is the set of linear equations that yield the optimum filter coefficients.

To solve the set of linear equations directly, we must first compute the autocorrelation sequence $\{r_{xx}(k)\}$ of the input signal and the crosscorrelation sequence $\{r_{dx}(k)\}$ between the desired sequence $\{d(n)\}$ and the input sequence $\{x(n)\}$.

The LMS algorithm provides an alternative computational method for determining the optimum filter coefficients $\{h(k)\}$ without explicitly computing the correlation sequences $\{r_{xx}(k)\}$ and $\{r_{dx}(k)\}$. The algorithm is basically a recursive gradient (steepest-descent) method that finds the minimum of $\mathcal{E}$ and thus yields the set of optimum filter coefficients.

We begin with any arbitrary choice for the initial values of $\{h(k)\}$, say $\{h_0(k)\}$. For example, we may begin with $h_0(k) = 0, \quad 0 \le k \le N-1$. Then after each new input sample $\{x(n)\}$ enters the adaptive FIR filter, we compute the corresponding output, say $\{y(n)\}$, form the error signal $e(n) = d(n) - y(n)$, and update the filter coefficients according to the equation

$$h_n(k) = h_{n-1}(k) + \triangle \cdot e(n) \cdot x(n-k), \quad 0 \le k \le N-1, \quad n = 0, 1, \ldots \tag{11.8}$$

where $\triangle$ is called the step size parameter, $x(n-k)$ is the sample of the input signal located at the kth tap of the filter at time n, and $e(n)x(n-k)$ is an approximation (estimate) of the negative of the gradient for the kth filter coefficient. This is the LMS recursive algorithm for adjusting the filter coefficients adaptively so as to minimize the sum of squared errors $\mathcal{E}$.

The step size parameter $\triangle$ controls the rate of convergence of the algorithm to the optimum solution. A large value of $\triangle$ leads to large step size adjustments and thus to rapid convergence, while a small value of $\triangle$ results in slower convergence. However, if $\triangle$ is made too large the algorithm becomes unstable. To ensure stability, $\triangle$ must be chosen [22]

to be in the range

$$0 < \triangle < \frac{1}{10NP_x} \tag{11.9}$$

where N is the length of the adaptive FIR filter and P_x is the power in the input signal, which can be approximated by

$$P_x \approx \frac{1}{1+M}\sum_{n=0}^{M} x^2(n) = \frac{r_{xx}(0)}{M+1} \tag{11.10}$$

The mathematical justification of equations (11.9) and (11.10) and the proof that the LMS algorithm leads to the solution for the optimum filter coefficients is given in more advanced treatments of adaptive filters. The interested reader may refer to the books by Haykin [8] and Proakis and Manolakis [23].

11.1.1 MATLAB IMPLEMENTATION

The LMS algorithm (11.8) can easily be implemented in MATLAB. Given the input sequence $\{x(n)\}$, the desired sequence $\{d(n)\}$, step size $\triangle$, and the desired length of the adaptive FIR filter N, we can use (11.1), (11.2), and (11.8) to determine the adaptive filter coefficients $\{h(n), 0 \le n \le N-1\}$ recursively. This is shown in the following function, called `lms`.

```
function [h,y] = lms(x,d,delta,N)
% LMS Algorithm for Coefficient Adjustment
% ----------------------------------------
% [h,y] = lms(x,d,delta,N)
%     h = estimated FIR filter
%     y = output array y(n)
%     x = input array x(n)
%     d = desired array d(n), length must be same as x
% delta = step size
%     N = length of the FIR filter
%
M = length(x); y = zeros(1,M);
h = zeros(1,N);
for n = N:M
    x1 = x(n:-1:n-N+1);
     y = h * x1';
     e = d(n) - y;
     h = h + delta*e*x1;
end
```

In addition, the `lms` function provides the output $\{y(n)\}$ of the adaptive filter.

We will apply the LMS algorithm to several practical applications involving adaptive filtering.

11.2 SYSTEM IDENTIFICATION OR SYSTEM MODELING

To formulate the problem, let us refer to Figure 11.2. We have an unknown linear system that we wish to identify. The unknown system may be an all-zero (FIR) system or a pole-zero (IIR) system. The unknown system will be approximated (modeled) by an FIR filter of length N. Both the unknown system and the FIR model are connected in parallel and are excited by the same input sequence $\{x(n)\}$. If $\{y(n)\}$ denotes the output of the model and $\{d(n)\}$ denotes the output of the unknown system, the error sequence is $\{e(n) = d(n) - y(n)\}$. If we minimize the sum of squared errors, we obtain the same set of linear equations as in (11.7). Therefore, the LMS algorithm given by (11.8) may be used to adapt the coefficients of the FIR model so that its output approximates the output of the unknown system.

11.2.1 PROJECT 11.1: SYSTEM IDENTIFICATION

There are three basic modules that are needed to perform this project.

1. A noise signal generator that generates a sequence of random numbers with zero mean value. For example, we may generate a sequence of uniformly distributed random numbers over the interval $[-a, a]$. Such a sequence of uniformly distributed numbers has an average value of zero and a variance of $a^2/3$. This signal sequence, call it $\{x(n)\}$, will be used as the input to the unknown system and the adaptive FIR model. In this case the input signal $\{x(n)\}$ has power $P_x = a^2/3$. In MATLAB this can be implemented using the `rand` function.

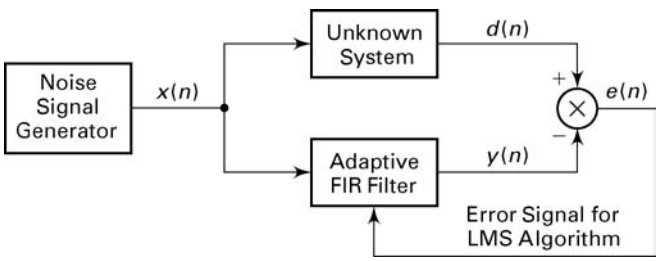


FIGURE 11.2 *Block diagram of system identification or system modeling problem*

2. An unknown system module that may be selected is an IIR filter and implemented by its difference equation. For example, we may select an IIR filter specified by the second-order difference equation

$$d(n) = a_1 d(n-1) + a_2 d(n-2) + x(n) + b_1 x(n-1) + b_2 x(n-2) \tag{11.11}$$

where the parameters $\{a_1, a_2\}$ determine the positions of the poles and $\{b_1, b_2\}$ determine the positions of the zeros of the filter. These parameters are input variables to the program. This can be implemented by the `filter` function.

3. An adaptive FIR filter module where the FIR filter has N tap coefficients that are adjusted by means of the LMS algorithm. The length N of the filter is an input variable to the program. This can be implemented using the `lms` function given in the previous section.

The three modules are configured as shown in Figure 11.2. From this project we can determine how closely the impulse response of the FIR model approximates the impulse response of the unknown system after the LMS algorithm has converged.

To monitor the convergence rate of the LMS algorithm, we may compute a short-term average of the squared error $e^2(n)$ and plot it. That is, we may compute

$$\text{ASE}(m) = \frac{1}{K} \sum_{k=n+1}^{n+K} e^2(k) \tag{11.12}$$

where $m = n/K = 1, 2, \ldots$. The averaging interval K may be selected to be (approximately) $K = 10N$. The effect of the choice of the step size parameter $\triangle$ on the convergence rate of the LMS algorithm may be observed by monitoring the ASE(m).

Besides the main part of the program, you should also include, as an aside, the computation of the impulse response of the unknown system, which can be obtained by exciting the system with a unit sample sequence $\delta(n)$. This actual impulse response can be compared with that of the FIR model after convergence of the LMS algorithm. The two impulse responses can be plotted for the purpose of comparison.

11.3 SUPPRESSION OF NARROWBAND INTERFERENCE IN A WIDEBAND SIGNAL

Let us assume that we have a signal sequence $\{x(n)\}$ that consists of a desired wideband signal sequence, say $\{w(n)\}$, corrupted by an additive narrowband interference sequence $\{s(n)\}$. The two sequences are uncorrelated. This problem arises in digital communications and in signal

detection, where the desired signal sequence $\{w(n)\}$ is a spread-spectrum signal, while the narrowband interference represents a signal from another user of the frequency band or some intentional interference from a jammer who is trying to disrupt the communication or detection system.

From a filtering point of view, our objective is to design a filter that suppresses the narrowband interference. In effect, such a filter should place a notch in the frequency band occupied by the interference. In practice, however, the frequency band of the interference might be unknown. Moreover, the frequency band of the interference may vary slowly in time.

The narrowband characteristics of the interference allow us to estimate $s(n)$ from past samples of the sequence $x(n) = s(n) + w(n)$ and to subtract the estimate from $x(n)$. Since the bandwidth of $\{s(n)\}$ is narrow compared to the bandwidth of $\{w(n)\}$, the samples of $\{s(n)\}$ are highly correlated. On the other hand, the wideband sequence $\{w(n)\}$ has a relatively narrow correlation.

The general configuration of the interference suppression system is shown in Figure 11.3. The signal $x(n)$ is delayed by D samples, where the delay D is chosen sufficiently large so that the wideband signal components $w(n)$ and $w(n-D)$, which are contained in $x(n)$ and $x(n-D)$, respectively, are uncorrelated. The output of the adaptive FIR filter is the estimate

$$\hat{s}(n) = \sum_{k=0}^{N-1} h(k)x(n-k-D) \tag{11.13}$$

The error signal that is used in optimizing the FIR filter coefficients is $e(n) = x(n) - \hat{s}(n)$. The minimization of the sum of squared errors again leads to a set of linear equations for determining the optimum coefficients. Due to the delay D, the LMS algorithm for adjusting the coefficients recursively becomes

$$h_n(k) = h_{n-1}(k) + \triangle e(n)x(n-k-D), \quad \begin{matrix} k = 0, 1, \ldots, N-1 \\ n = 1, 2, \ldots \end{matrix} \tag{11.14}$$

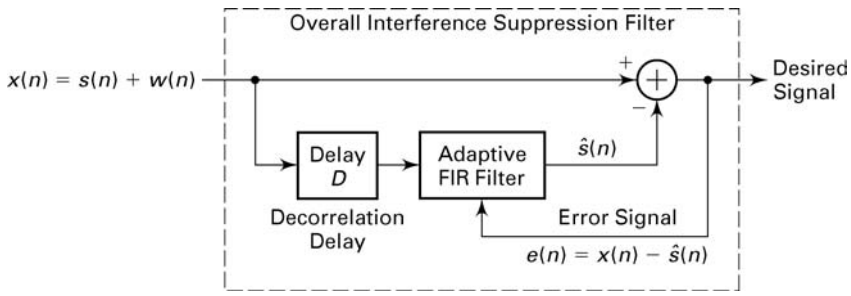


FIGURE 11.3 *Adaptive filter for estimating and suppressing a narrowband interference*

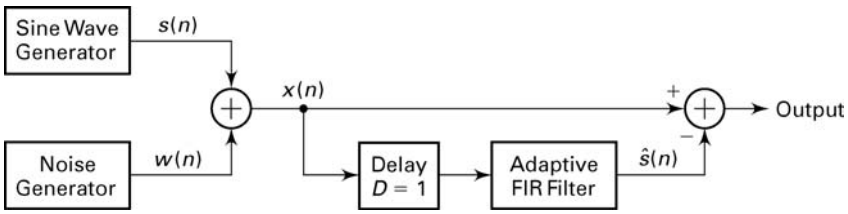


FIGURE 11.4 *Configuration of modules for experiment on interference suppression*

11.3.1 PROJECT 11.2: SUPPRESSION OF SINUSOIDAL INTERFERENCE

Three basic modules are required to perform this project.

1. A noise signal generator module that generates a wideband sequence $\{w(n)\}$ of random numbers with zero mean value. In particular, we may generate a sequence of uniformly distributed random numbers using the `rand` function as previously described in the project on system identification. The signal power is denoted as P_w.
2. A sinusoidal signal generator module that generates a sine wave sequence $s(n) = A \sin \omega_0 n$, where $0 < \omega_0 < \pi$ and A is the signal amplitude. The power of the sinusoidal sequence is denoted as P_s.
3. An adaptive FIR filter module using the `lms` function, where the FIR filter has N tap coefficients that are adjusted by the LMS algorithm. The length N of the filter is an input variable to the program.

The three modules are configured as shown in Figure 11.4. In this project the delay $D = 1$ is sufficient, since the sequence $\{w(n)\}$ is a white noise (spectrally flat or uncorrelated) sequence. The objective is to adapt the FIR filter coefficients and then to investigate the characteristics of the adaptive filter.

It is interesting to select the interference signal to be much stronger than the desired signal $w(n)$, for example, $P_s = 10P_w$. Note that the power P_x required in selecting the step size parameter in the LMS algorithm is $P_x = P_s + P_w$. The frequency response characteristic $H(e^{j\omega})$ of the adaptive FIR filter with coefficients $\{h(k)\}$ should exhibit a resonant peak at the frequency of the interference. The frequency response of the interference suppression filter is $H_s(e^{j\omega}) = 1 - H(e^{j\omega})$, which should then exhibit a notch at the frequency of the interference.

It is interesting to plot the sequences $\{w(n)\}$, $\{s(n)\}$, and $\{x(n)\}$. It is also interesting to plot the frequency responses $H(e^{j\omega})$ and $H_s(e^{j\omega})$ after the LMS algorithm has converged. The short-time average squared error ASE(m), defined by (11.12), may be used to monitor the convergence characteristics of the LMS algorithm. The effect of the length of the adaptive filter on the quality of the estimate should be investigated.

The project may be generalized by adding a second sinusoid of a different frequency. Then $H(e^{j\omega})$ should exhibit two resonant peaks, provided the frequencies are sufficiently separated. Investigate the effect of the filter length N on the resolution of two closely spaced sinusoids.

11.4 ADAPTIVE LINE ENHANCEMENT

In the preceding section we described a method for suppressing a strong narrowband interference from a wideband signal. An adaptive line enhancer (ALE) has the same configuration as the interference suppression filter in Figure 11.3, except that the objective is different.

In the adaptive line enhancer, $\{s(n)\}$ is the desired signal and $\{w(n)\}$ represents a wideband noise component that masks $\{s(n)\}$. The desired signal $\{s(n)\}$ may be a spectral line (a pure sinusoid) or a relatively narrowband signal. Usually, the power in the wideband signal is greater than that in the narrowband signal—that is, $P_w > P_s$. It is apparent that the ALE is a self-tuning filter that has a peak in its frequency response at the frequency of the input sinusoid or in the frequency band occupied by the narrowband signal. By having a narrow bandwidth FIR filter, the noise outside the frequency band of the signal is suppressed, and thus the spectral line is enhanced in amplitude relative to the noise power in $\{w(n)\}$.

11.4.1 PROJECT 11.3: ADAPTIVE LINE ENHANCEMENT

This project requires the same software modules as those used in the project on interference suppression. Hence the description given in the preceding section applies directly. One change is that in the ALE, the condition is that $P_w > P_s$. Second, the output signal from the ALE is $\{s(n)\}$. Repeat the project described in the previous section under these conditions.

11.5 ADAPTIVE CHANNEL EQUALIZATION

The speed of data transmission over telephone channels is usually limited by channel distortion that causes intersymbol interference (ISI). At data rates below 2400 bits the ISI is relatively small and is usually not a problem in the operation of a modem. However, at data rates above 2400 bits, an adaptive equalizer is employed in the modem to compensate for the channel distortion and thus to allow for highly reliable high-speed data

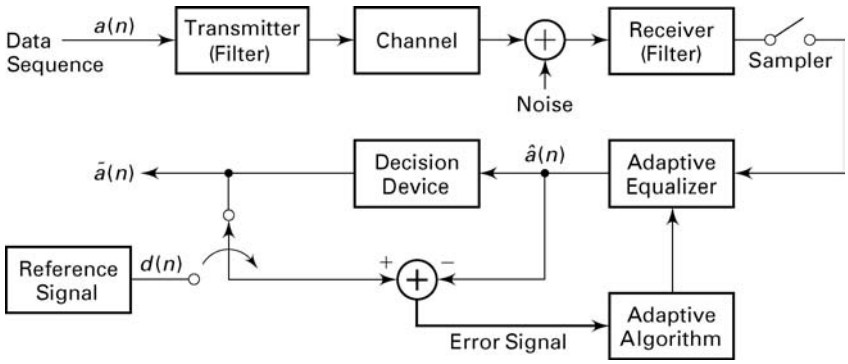


FIGURE 11.5 *Application of adaptive filtering to adaptive channel equalization*

transmission. In telephone channels, filters are used throughout the system to separate signals in different frequency bands. These filters cause amplitude and phase distortion. The adaptive equalizer is basically an adaptive FIR filter with coefficients that are adjusted by means of the LMS algorithm to correct for the channel distortion.

A block diagram showing the basic elements of a modem transmitting data over a channel is given in Figure 11.5. Initially, the equalizer coefficients are adjusted by transmitting a short training sequence, usually less than one second in duration. After the short training period, the transmitter begins to transmit the data sequence $\{a(n)\}$. To track the possible slow time variations in the channel, the equalizer coefficients must continue to be adjusted in an adaptive manner while receiving data. This is usually accomplished, as illustrated in Figure 11.5, by treating the decisions at the output of the decision device as correct and by using the decisions in place of the reference $\{d(n)\}$ to generate the error signal. This approach works quite well when decision errors occur infrequently, such as less than one error in 100 data symbols. The occasional decision errors cause only a small misadjustment in the equalizer coefficients.

11.5.1 PROJECT 11.4: ADAPTIVE CHANNEL EQUALIZATION

The objective of this project is to investigate the performance of an adaptive equalizer for data transmission over a channel that causes intersymbol interference. The basic configuration of the system to be simulated is shown in Figure 11.6. As we observe, five basic modules are required. Note that we have avoided carrier modulation and demodulation, which is required in a telephone channel modem. This is done to simplify the simulation program. However, all processing involves complex arithmetic operations.

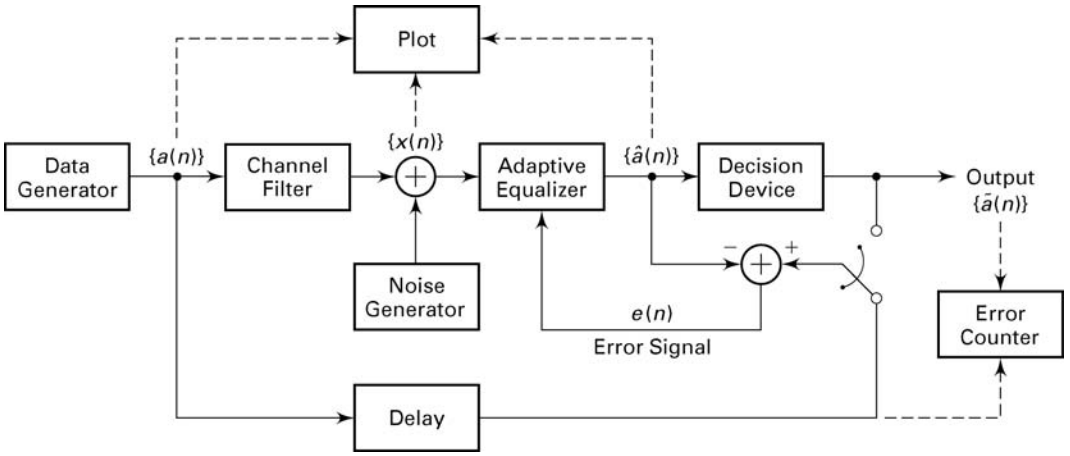


FIGURE 11.6 *Experiment for investigating the performance of an adaptive equalizer*

The five modules are as follows:

1. The data generator module is used to generate a sequence of complex-valued information symbols $\{a(n)\}$. In particular, employ four equally probable symbols $s + js$, $s - js$, $-s + js$, and $-s - js$, where s is a scale factor that may be set to $s = 1$, or it can be an input parameter.
2. The channel filter module is an FIR filter with coefficients $\{c(n), 0 \leq n \leq K - 1\}$ that simulates the channel distortion. For distortionless transmission, set $c(0) = 1$ and $c(n) = 0$ for $1 \leq n \leq K - 1$. The length K of the filter is an input parameter.
3. The noise generator module is used to generate additive noise that is usually present in any digital communication system. If we are modeling noise that is generated by electronic devices, the noise distribution should be Gaussian with zero mean. Use the `randu` function.
4. The adaptive equalizer module is an FIR filter with tap coefficients $\{h(k), 0 < k < N - 1\}$, which are adjusted by the LMS algorithm. However, due to the use of complex arithmetic, the recursive equation in the LMS algorithm is slightly modified to

$$h_n(k) = h_{n-1}(k) + \triangle\, e(n)x^*(n - k) \tag{11.15}$$

   where the asterisk denotes the complex conjugate.
5. The decision device module takes the estimate $\hat{a}(n)$ and quantizes it to one of the four possible signal points on the basis of the following

decision rule:

$$\begin{aligned}
\text{Re}\,[\hat{a}(n)] > 0 \text{ and } \text{Im}\,[\hat{a}(n)] > 0 &\longrightarrow \quad 1 + j \\
\text{Re}\,[\hat{a}(n)] > 0 \text{ and } \text{Im}\,[\hat{a}(n)] < 0 &\longrightarrow \quad 1 - j \\
\text{Re}\,[\hat{a}(n)] < 0 \text{ and } \text{Im}\,[\hat{a}(n)] > 0 &\longrightarrow -1 + j \\
\text{Re}\,[\hat{a}(n)] < 0 \text{ and } \text{Im}\,[\hat{a}(n)] < 0 &\longrightarrow -1 - j
\end{aligned}$$

The effectiveness of the equalizer in suppressing the ISI introduced by the channel filter may be seen by plotting the following relevant sequences in a two-dimensional (real–imaginary) display. The data generator output $\{a(n)\}$ should consist of four points with values $\pm 1 \pm j$. The effect of channel distortion and additive noise may be viewed by displaying the sequence $\{x(n)\}$ at the input to the equalizer. The effectiveness of the adaptive equalizer may be assessed by plotting its output $\{\hat{a}(n)\}$ after convergence of its coefficients. The short-time average squared error $\text{ASE}(n)$ may also be used to monitor the convergence characteristics of the LMS algorithm. Note that a delay must be introduced into the output of the data generator to compensate for the delays that the signal encounters due to the channel filter and the adaptive equalizer. For example, this delay may be set to the largest integer closest to $(N + K)/2$. Finally, an error counter may be used to count the number of symbol errors in the received data sequence, and the ratio for the number of errors to the total number of symbols (error rate) may be displayed. The error rate may be varied by changing the level of the ISI and the level of the additive noise.

It is suggested that simulations be performed for the following three channel conditions:

a. No ISI: $c(0) = 1, c(n) = 0,\ 1 \leq n \leq K - 1$
b. Mild ISI: $c(0) = 1,\ c(1) = 0.2,\ c(2) = -0.2,\ c(n) = 0,\ 3 \leq n \leq K - 1$
c. Strong ISI: $c(0) = 1,\ c(1) = 0.5,\ c(2) = 0.5,\ c(n) = 0,\ 3 \leq n \leq K - 1$

The measured error rate may be plotted as a function of the signal-to-noise ratio (SNR) at the input to the equalizer, where SNR is defined as P_s/P_n, where P_s is the signal power, given as $P_s = s^2$, and P_n is the noise power of the sequence at the output of the noise generator.

CHAPTER 12

Applications in Communications

Today MATLAB finds widespread use in the simulation of a variety of communication systems. In this chapter we shall focus on several applications dealing with waveform representation and coding, especially speech coding, and with digital communications. In particular, we shall describe several methods for digitizing analog waveforms, with specific application to speech coding and transmission. These methods are pulse-code modulation (PCM), differential PCM and adaptive differential PCM (ADPCM), delta modulation (DM) and adaptive delta modulation (ADM), and linear predictive coding (LPC). A project is formulated involving each of these waveform encoding methods for simulation using MATLAB.

The last three topics treated in this chapter deal with signal-detection applications that are usually encountered in the implementation of a receiver in a digital communication system. For each of these topics we describe a project that involves the implementations via simulation of the detection scheme in MATLAB.

12.1 PULSE-CODE MODULATION

Pulse-code modulation is a method for quantizing an analog signal for the purpose of transmitting or storing the signal in digital form. PCM is widely used for speech transmission in telephone communications and for telemetry systems that employ radio transmission. We shall concentrate our attention on the application of PCM to speech signal processing.

Speech signals transmitted over telephone channels are usually limited in bandwidth to the frequency range below 4 kHz. Hence the Nyquist rate for sampling such a signal is less than 8 kHz. In PCM the analog speech signal is sampled at the nominal rate of 8 kHz (samples per second), and each sample is quantized to one of 2^b levels, and represented digitally by a sequence of b bits. Thus the bit rate required to transmit the digitized speech signal is $8000\,b$ bits per second.

The quantization process may be modeled mathematically as

$$\tilde{s}(n) = s(n) + q(n) \tag{12.1}$$

where $\tilde{s}(n)$ represents the quantized value of $s(n)$, and $q(n)$ represents the quantization error, which we treat as an additive noise. Assuming that a uniform quantizer is used and the number of levels is sufficiently large, the quantization noise is well characterized statistically by the uniform probability density function,

$$p(q) = \frac{1}{\Delta}, \quad -\frac{\Delta}{2} \le q \le \frac{\Delta}{2} \tag{12.2}$$

where the step size of the quantizer is $\Delta = 2^{-b}$. The mean square value of the quantization error is

$$E(q^2) = \frac{\Delta^2}{12} = \frac{2^{-2b}}{12} \tag{12.3}$$

Measured in decibels, the mean square value of the noise is

$$10 \log \left(\frac{\Delta^2}{12} \right) = 10 \log \left(\frac{2^{-2b}}{12} \right) = -6b - 10.8 \text{ dB} \tag{12.4}$$

We observe that the quantization noise decreases by 6 dB/bit used in the quantizer. High-quality speech requires a minimum of 12 bits per sample and hence a bit rate of 96,000 bits per second (bps).

Speech signals have the characteristic that small signal amplitudes occur more frequently than large signal amplitudes. However, a uniform quantizer provides the same spacing between successive levels throughout the entire dynamic range of the signal. A better approach is to use a nonuniform quantizer, which provides more closely spaced levels at the low signal amplitudes and more widely spaced levels at the large signal amplitudes. For a nonuniform quantizer with b bits, the resulting quantization error has a mean square value that is smaller than that given by (12.4). A nonuniform quantizer characteristic is usually obtained by passing the signal through a nonlinear device that compresses the signal amplitude, followed by a uniform quantizer. For example, a logarithmic compressor employed in U.S. and Canadian telecommunications systems,

called a μ-law compressor, has an input-output magnitude characteristic of the form

$$y = \frac{\ln(1+\mu|s|)}{\ln(1+\mu)}\,\text{sgn}(s); \quad |s| \le 1,\ |y| \le 1 \tag{12.5}$$

where s is the normalized input, y is the normalized output, $\text{sgn}(\cdot)$ is the sign function, and μ is a parameter that is selected to give the desired compression characteristic.

In the encoding of speech waveforms the value of $\mu = 255$ has been adopted as a standard in the U.S. and Canada. This value results in about a 24 dB reduction in the quantization noise power relative to uniform quantization. Consequently, an 8-bit quantizer used in conjunction with a $\mu = 255$ logarithmic compressor produces the same quality speech as a 12-bit uniform quantizer with no compression. Thus the compressed PCM speech signal has a bit rate of 64,000 bps.

The logarithmic compressor standard used in European telecommunication systems is called A-law and is defined as

$$y = \begin{cases} \dfrac{1+\ln(A|s|)}{1+\ln A}\,\text{sgn}(s), & \frac{1}{A} \le |s| \le 1 \\ \dfrac{A|s|}{1+\ln A}\,\text{sgn}(s), & 0 \le |s| \le \frac{1}{A} \end{cases} \tag{12.6}$$

where A is chosen as 87.56. Although (12.5) and (12.6) are different nonlinear functions, the two compression characteristics are very similar. Figure 12.1 illustrates these two compression functions. Note their strong similarity.

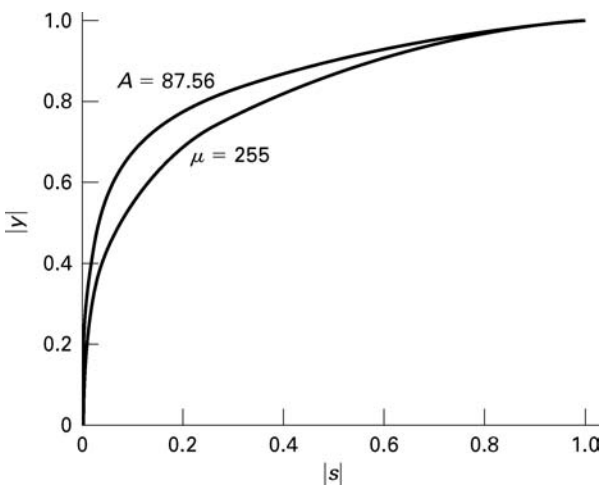


FIGURE 12.1 *Comparison of μ-law and A-law nonlinearities*

In the reconstruction of the signal from the quantized values, the decoder employs an inverse logarithmic relation to expand the signal amplitude. For example, in μ-law the inverse relation is given by

$$|s| = \frac{(1+\mu)^{|y|} - 1}{\mu}; \quad |y| \leq 1, \, |s| \leq 1 \tag{12.7}$$

The combined compressor-expander pair is termed a *compander*.

12.1.1 PROJECT 12.1: PCM

The purpose of this project is to gain an understanding of PCM compression (linear-to-logarithmic) and PCM expansion (logarithmic-to-linear). Write the following three MATLAB functions for this project:

1. a μ-law compressor function to implement (12.5) that accepts a zero-mean normalized ($|s| \leq 1$) signal and produces a compressed zero-mean signal with μ as a free parameter that can be specified,
2. a quantizer function that accepts a zero-mean input and produces an integer output after b-bit quantization that can be specified, and
3. a μ-law expander to implement (12.7) that accepts an integer input and produces a zero-mean output for a specified μ parameter.

For simulation purposes generate a large number of samples (10,000 or more) of the following sequences: (a) a sawtooth sequence, (b) an exponential pulse train sequence, (c) a sinusoidal sequence, and (d) a random sequence with small variance. Care must be taken to generate nonperiodic sequences by choosing their normalized frequencies as irrational numbers (i.e., sample values should not repeat). For example, a sinusoidal sequence can be generated using

$$s(n) = 0.5 \sin(n/33), \quad 0 \leq n \leq 10{,}000$$

From our discussions in Chapter 2, this sequence is nonperiodic, yet it has a periodic envelope. Other sequences can also be generated in a similar fashion. Process these signals through the above μ-law compressor, quantizer, and expander functions as shown in Figure 12.2, and compute

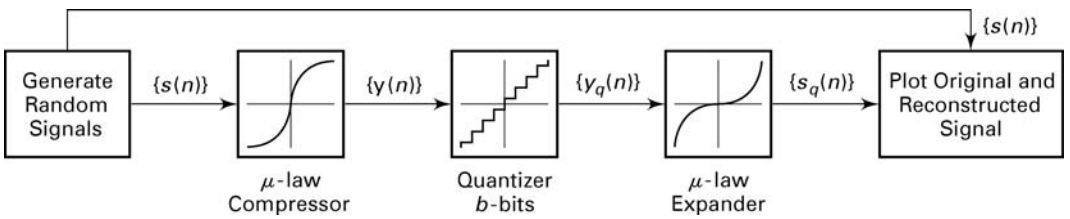


FIGURE 12.2 *PCM project*

the signal-to-quantization noise ratio (SQNR) in dB as

$$\text{SQNR} = 10 \log_{10} \left(\frac{\sum_{n=1}^{N} s^2(n)}{\sum_{n=1}^{N} \left(s(n) - s_q(n) \right)^2} \right).$$

For different b-bit quantizers, systematically determine the value of μ that maximizes the SQNR. Also plot the input and output waveforms and comment on the results.

12.2 DIFFERENTIAL PCM (DPCM)

In PCM each sample of the waveform is encoded independently of all the other samples. However, most signals, including speech, sampled at the Nyquist rate or faster exhibit significant correlation between successive samples. In other words, the average change in amplitude between successive samples is relatively small. Consequently, an encoding scheme that exploits the redundancy in the samples will result in a lower bit rate for the speech signal.

A relatively simple solution is to encode the differences between successive samples rather than the samples themselves. Since differences between samples are expected to be smaller than the actual sampled amplitudes, fewer bits are required to represent the differences. A refinement of this general approach is to predict the current sample based on the previous p samples. To be specific, let $s(n)$ denote the current sample of speech and let $\hat{s}(n)$ denote the predicted value of $s(n)$, defined as

$$\hat{s}(n) = \sum_{i=1}^{p} a(i)\, s(n-i) \tag{12.8}$$

Thus $\hat{s}(n)$ is a weighted linear combination of the past p samples, and the $a(i)$ are the predictor (filter) coefficients. The $a(i)$ are selected to minimize some function of the error between $s(n)$ and $\hat{s}(n)$.

A mathematically and practically convenient error function is the sum of squared errors. With this as the performance index for the predictor, we select the $a(i)$ to minimize

$$\begin{aligned} \mathcal{E}_p \triangleq \sum_{n=1}^{N} e^2(n) &= \sum_{n=1}^{N} \left[s(n) - \sum_{i=1}^{p} a(i)\, s(n-i) \right]^2 \\ &= r_{ss}(0) - 2 \sum_{i=1}^{p} a(i)\, r_{ss}(i) + \sum_{i=1}^{p} \sum_{j=1}^{p} a(i)\, a(j)\, r_{ss}(i-j) \end{aligned} \tag{12.9}$$

where $r_{ss}(m)$ is the autocorrelation function of the sampled signal sequence $s(n)$, defined as

$$r_{ss}(m) = \sum_{i=1}^{N} s(i)\, s(i+m) \tag{12.10}$$

Minimization of $\mathcal{E}_p$ with respect to the predictor coefficients $\{a_i(n)\}$ results in the set of linear equations, called the normal equations,

$$\sum_{i=1}^{p} a(i)\, r_{ss}(i-j) = r_{ss}(j), \quad j = 1, 2, \ldots, p \tag{12.11}$$

or in the matrix form,

$$\mathbf{Ra} = \mathbf{r} \Longrightarrow \mathbf{a} = \mathbf{R}^{-1}\mathbf{r} \tag{12.12}$$

where $\mathbf{R}$ is the autocorrelation matrix, $\mathbf{a}$ is the coefficient vector, and $\mathbf{r}$ is the autocorrelation vector. Thus the values of the predictor coefficients are established.

Having described the method for determining the predictor coefficients, let us now consider the block diagram of a practical DPCM system, shown in Figure 12.3. In this configuration the predictor is implemented with the feedback loop around the quantizer. The input to the predictor is denoted as $\tilde{s}(n)$, which represents the signal sample $s(n)$ modified by the quantization process, and the output of the predictor is

$$\hat{\tilde{s}} = \sum_{i=1}^{p} a(i)\, \tilde{s}(n-i) \tag{12.13}$$

The difference

$$e(n) = s(n) - \hat{\tilde{s}}(n) \tag{12.14}$$

is the input to the quantizer, and $\tilde{e}(n)$ denotes the output. Each value of the quantized prediction error $\tilde{e}(n)$ is encoded into a sequence of binary

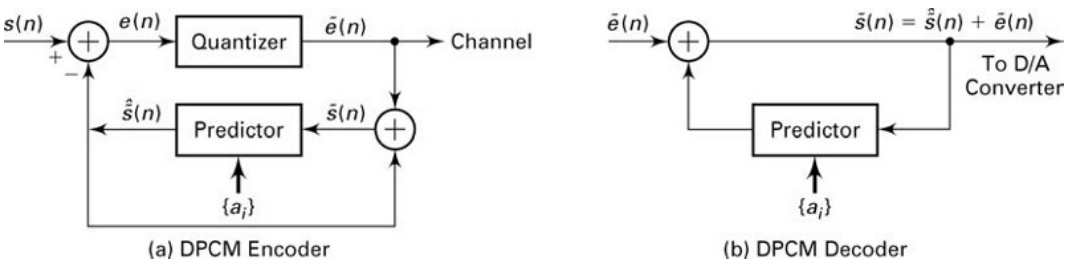


FIGURE 12.3 *Block diagram of a DPCM transcoder: (a) encoder, (b) decoder*

digits and transmitted over the channel to the receiver. The quantized error $\tilde{e}(n)$ is also added to the predicted value $\widehat{\tilde{s}}(n)$ to yield $\tilde{s}(n)$.

At the receiver the same predictor that was used at the transmitting end is synthesized, and its output $\widehat{\tilde{s}}(n)$ is added to $\tilde{e}(n)$ to yield $\tilde{s}(n)$. The signal $\tilde{s}(n)$ is the desired excitation for the predictor and also the desired output sequence from which the reconstructed signal $\tilde{s}\,(t)$ is obtained by filtering, as shown in Figure 12.3b.

The use of feedback around the quantizer, as described, ensures that the error in $\tilde{s}(n)$ is simply the quantization error $q(n) = \tilde{e}(n) - e(n)$ and that there is no accumulation of previous quantization errors in the implementation of the decoder. That is,

$$q(n) = \tilde{e}(n) - e(n) = \tilde{e}(n) - s(n) + \widehat{\tilde{s}}(n) = \tilde{s}(n) - s(n) \qquad \textbf{(12.15)}$$

Hence $\tilde{s}(n) = s(n) + q(n)$. This means that the quantized sample $\tilde{s}(n)$ differs from the input $s(n)$ by the quantization error $q(n)$ independent of the predictor used. Therefore the quantization errors do not accumulate.

In the DPCM system illustrated in Figure 12.3, the estimate or predicted value $\tilde{s}(n)$ of the signal sample $s(n)$ is obtained by taking a linear combination of past values $\tilde{s}\,(n-k)\,, \quad k = 1, 2, \ldots, p$, as indicated by (12.13). An improvement in the quality of the estimate is obtained by including linearly filtered past values of the quantized error. Specifically, the estimate of $s(n)$ may be expressed as

$$\widehat{\tilde{s}}(n) = \sum_{i=1}^{p} a\,(i)\,\tilde{s}\,(n-i) + \sum_{i=1}^{m} b\,(i)\,\tilde{e}\,(n-i) \qquad \textbf{(12.16)}$$

where $b\,(i)$ are the coefficients of the filter for the quantized error sequence $\tilde{e}(n)$. The block diagram of the encoder at the transmitter and the decoder at the receiver are shown in Figure 12.4. The two sets of coefficients $a\,(i)$ and $b\,(i)$ are selected to minimize some function of the error $e(n) = \tilde{s}(n) - s(n)$, such as the sum of squared errors.

By using a logarithmic compressor and a 4-bit quantizer for the error sequence $e(n)$, DPCM results in high-quality speech at a rate of 32,000 bps, which is a factor of two lower than logarithmic PCM.

12.2.1 PROJECT 12.2: DPCM

The objective of this project is to gain understanding of the DPCM encoding and decoding operations. For simulation purposes, generate correlated random sequences using a pole-zero signal model of the form

$$s(n) = a\,(1)\,s\,(n-1) + b_0 x(n) + b_1 x\,(n-1) \qquad \textbf{(12.17)}$$

where $x(n)$ is a zero-mean unit variance Gaussian sequence. This can be done using the `filter` function. The sequences developed in Project 12.1

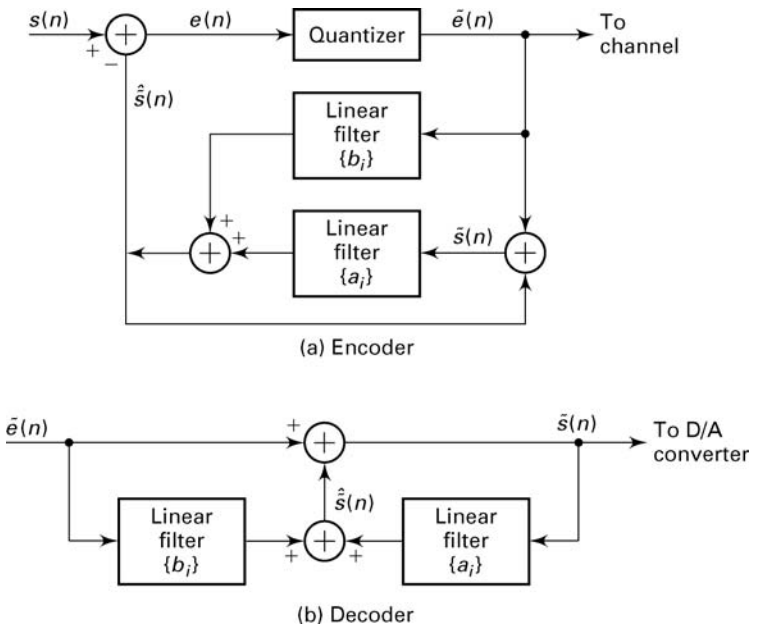


FIGURE 12.4 *DPCM modified by the linearly filtered error sequence*

can also be used for simulation. Develop the following three MATLAB modules for this project:

1. a model predictor function to implement (12.12), given the input signal $s(n)$;
2. a DPCM encoder function to implement the block diagram of Figure 12.3a, which accepts a zero-mean input sequence and produces a quantized b-bit integer error sequence, where b is a free parameter; and
3. a DPCM decoder function of Figure 12.3b, which reconstructs the signal from the quantized error sequence.

Experiment with several p-order prediction models for a given signal and determine the optimum order. Compare this DPCM implementation with the PCM system of Project 12.1 (at the end of the chapter) and comment on the results. Extend this implementation to include an mth-order moving average filter as indicated in (12.16).

12.3 ADAPTIVE PCM AND DPCM (ADPCM)

In general, the power in a speech signal varies slowly with time. PCM and DPCM encoders, however, are designed on the basis that the speech signal power is constant, and hence the quantizer is fixed. The efficiency

and performance of these encoders can be improved by having them adapt to the slowly time-variant power level of the speech signal.

In both PCM and DPCM the quantization error $q(n)$ resulting from a uniform quantizer operating on a slowly varying power level input signal will have a time-variant variance (quantization noise power). One improvement that reduces the dynamic range of the quantization noise is the use of an adaptive quantizer.

Adaptive quantizers can be classified as feedforward or feedback. A feedforward adaptive quantizer adjusts its step size for each signal sample, based on a measurement of the input speech signal variance (power). For example, the estimated variance, based as a sliding window estimator, is

$$\hat{\sigma}_{n+1}^2 = \frac{1}{M} \sum_{k=n+1-M}^{n+1} s^2(k) \tag{12.18}$$

Then the step size for the quantizer is

$$\Delta(n+1) = \Delta(n)\hat{\sigma}_{n+1} \tag{12.19}$$

In this case it is necessary to transmit $\Delta(n+1)$ to the decoder in order for it to reconstruct the signal.

A feedback adaptive quantizer employs the output of the quantizer in the adjustment of the step size. In particular, we may set the step size as

$$\Delta(n+1) = \alpha(n)\Delta(n) \tag{12.20}$$

where the scale factor $\alpha(n)$ depends on the previous quantizer output. For example, if the previous quantizer output is small, we may select $\alpha(n) < 1$ in order to provide for finer quantization. On the other hand, if the quantizer output is large, then the step size should be increased to reduce the possibility of signal clipping. Such an algorithm has been successfully used in the encoding of speech signals. Figure 12.5 illustrates such a (3-bit) quantizer in which the step size is adjusted recursively according to the relation

$$\Delta(n+1) = \Delta(n) \cdot M(n)$$

where $M(n)$ is a multiplication factor whose value depends on the quantizer level for the sample $s(n)$, and $\Delta(n)$ is the step size of the quantizer for processing $s(n)$. Values of the multiplication factors optimized for speech encoding have been given by [14]. These values are displayed in Table 12.1 for 2-, 3-, and 4-bit quantization for PCM and DPCM.

In DPCM the predictor can also be made adaptive. Thus in ADPCM the coefficients of the predictor are changed periodically to reflect the changing signal statistics of the speech. The linear equations given by (12.11) still apply, but the short-term autocorrelation function of $s(n)$, $r_{ss}(m)$ changes with time.

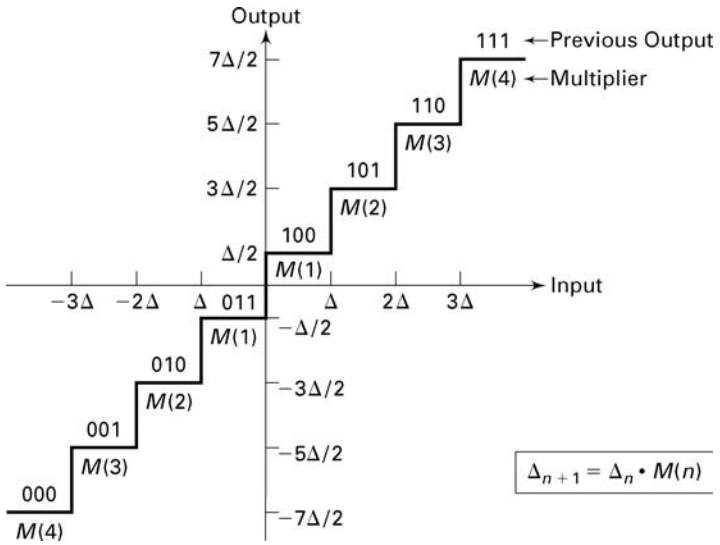


FIGURE 12.5 *Example of a quantizer with an adaptive step size ([10])*

12.3.1 ADPCM STANDARD

Figure 12.6 illustrates, in block diagram form, a 32,000 bps ADPCM encoder and decoder that has been adopted as an international (CCITT) standard for speech transmission over telephone channels. The ADPCM encoder is designed to accept 8-bit PCM compressed signal samples at 64,000 bps, and by means of adaptive prediction and adaptive 4-bit quantization to reduce the bit rate over the channel to 32,000 bps. The ADPCM decoder accepts the 32,000 bps data stream and reconstructs the signal in the form of an 8-bit compressed PCM at 64,000 bps. Thus we have a configuration shown in Figure 12.7, where the ADPCM encoder/decoder is embedded into a PCM system. Although the ADPCM encoder/

TABLE 12.1 *Multiplication factors for adaptive step size adjustment ([9])*

	PCM			*DPCM*		
	2	*3*	*4*	*2*	*3*	*4*
$M(1)$	0.60	0.85	0.80	0.80	0.90	0.90
$M(2)$	2.20	1.00	0.80	1.60	0.90	0.90
$M(3)$		1.00	0.80		1.25	0.90
$M(4)$		1.50	0.80		1.70	0.90
$M(5)$			0.80			1.20
$M(6)$			0.80			1.60
$M(7)$			0.80			2.00
$M(8)$			0.80			2.40

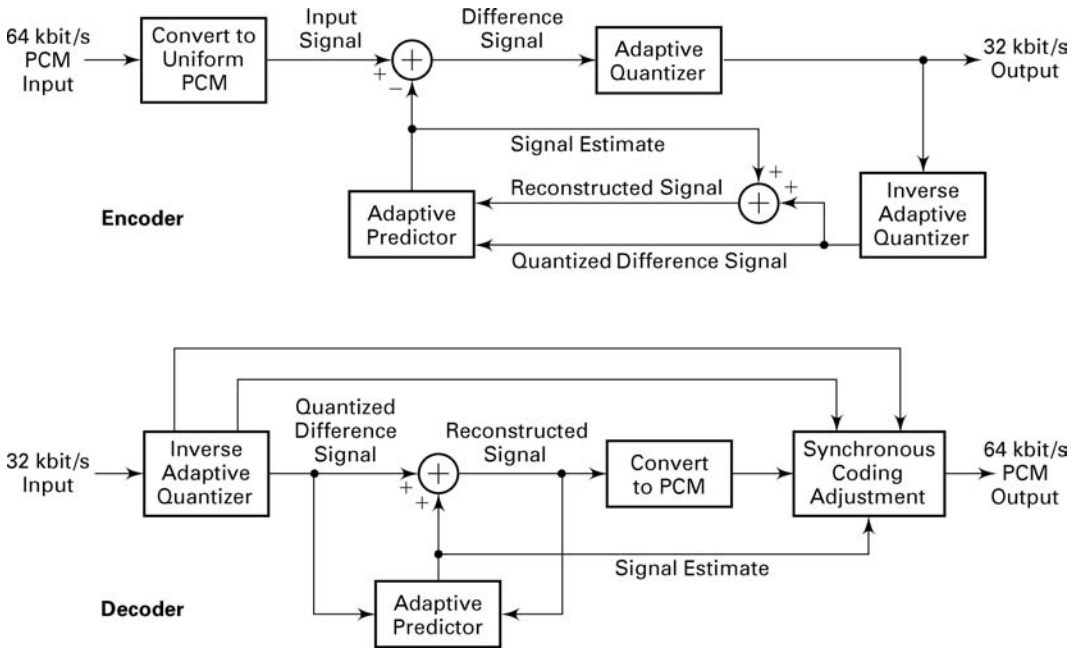


FIGURE 12.6 *ADPCM block diagram*

decoder could be used directly on the speech signal, the interface to the PCM system is necessary in practice in order to maintain compatibility with existing PCM systems that are widely used in the telephone network.

The ADPCM encoder accepts the 8-bit PCM compressed signal and expands it to a 14-bit-per-sample linear representation for processing. The predicted value is subtracted from this 14-bit linear value to produce a difference signal sample that is fed to the quantizer. Adaptive quantization is performed on the difference signal to produce a 4-bit output for transmission over the channel.

Both the encoder and decoder update their internal variables, based only on the ADPCM values that are generated. Consequently, an ADPCM

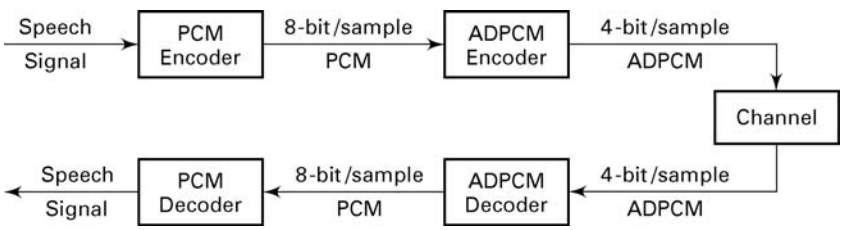


FIGURE 12.7 *ADPCM interface to PCM system*

decoder, including an inverse adaptive quantizer, is embedded in the encoder so that all internal variables are updated, based on the same data. This ensures that the encoder and decoder operate in synchronism without the need to transmit any information on the values of internal variables.

The adaptive predictor computes a weighted average of the last six dequantized difference values and the last two predicted values. Hence this predictor is basically a 2-pole ($p = 2$) and 6-zero ($m = 6$) filter governed by the difference equation given by (12.16). The filter coefficients are updated adaptively for every new input sample.

At the receiving decoder and at the decoder that is embedded in the encoder, the 4-bit transmitted ADPCM value is used to update the inverse adaptive quantizer, whose output is a dequantized version of the difference signal. This dequantized value is added to the value generated by the adaptive predictor to produce the reconstructed speech sample. This signal is the output of the decoder, which is converted to compressed PCM format at the receiver.

12.3.2 PROJECT 12.3: ADPCM

The objective of this project is to gain familiarity with, and understanding of, ADPCM and its interface with a PCM encoder/decoder (transcoder). As described, the ADPCM transcoder is inserted between the PCM compressor and the PCM expander as shown in Figure 12.7. Use the already developed MATLAB PCM and DPCM modules for this project.

The input to the PCM-ADPCM transcoder system can be supplied from internally generated waveform data files, just as in the case of the PCM project. The output of the transcoder can be plotted. Comparisons should be made between the output signal from the PCM-ADPCM transcoder with the signal from the PCM transcoder (PCM Project 12.1), and with the original input signal.

12.4 DELTA MODULATION (DM)

Delta modulation may be viewed as a simplified form of DPCM in which a 2-level (1-bit) quantizer is used in conjunction with a fixed 1st-order predictor. The block diagram of a DM encoder-decoder is shown in Figure 12.8. We note that

$$\widehat{\tilde{s}}(n) = \tilde{s}\,(n-1) = \widehat{\tilde{s}}\,(n-1) + \tilde{e}\,(n-1) \qquad \textbf{(12.21)}$$

Since

$$q(n) = \tilde{e}(n) - e(n) = \tilde{e}(n) - \left[s(n) - \widehat{\tilde{s}}(n)\right]$$

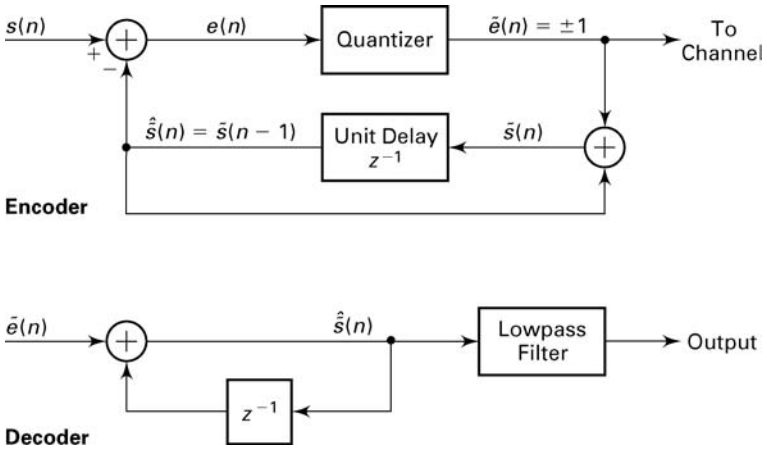


FIGURE 12.8 *Block diagram of a delta modulation system*

it follows that

$$\hat{\tilde{s}}(n) = s\,(n-1) + q\,(n-1) \tag{12.22}$$

Thus the estimated (predicted) value of $s(n)$ is really the previous sample $s\,(n-1)$ modified by the quantization noise $q\,(n-1)$. We also note that the difference equation in (12.21) represents an integrator with an input $\tilde{e}(n)$. Hence an equivalent realization of the 1-step predictor is an accumulator with an input equal to the quantized error signal $\tilde{e}(n)$. In general, the quantized error signal is scaled by some value, say Δ_1, which is called the step size. This equivalent realization is illustrated in Figure 12.9. In effect, the encoder shown in Figure 12.9 approximates a waveform $s\,(t)$

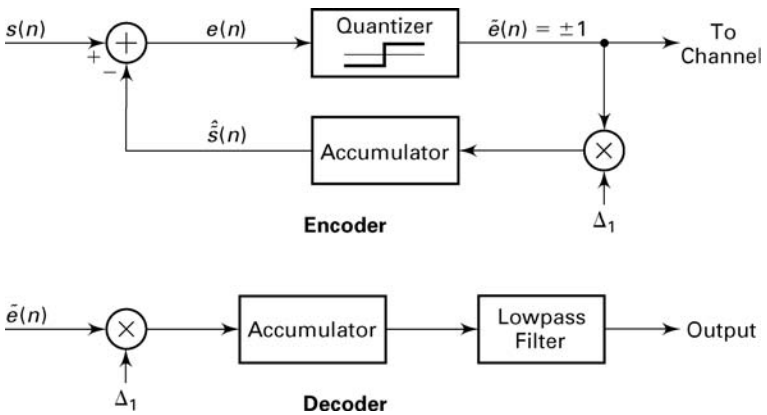


FIGURE 12.9 *An equivalent realization of a delta modulation system*

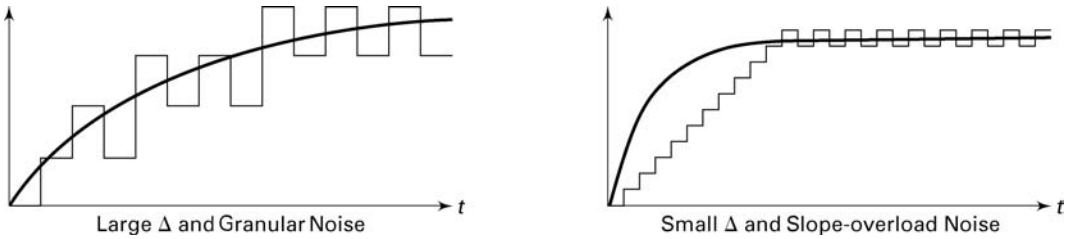


FIGURE 12.10 *Two types of distortion in the DM encoder*

by a linear staircase function. In order for the approximation to be relatively good, the waveform $s(t)$ must change slowly relative to the sampling rate. This requirement implies that the sampling rate must be several (a factor of at least 5) times the Nyquist rate. A lowpass filter is usually incorporated into the decoder to smooth out discontinuities in the reconstructed signal.

12.4.1 ADAPTIVE DELTA MODULATION (ADM)

At any given sampling rate, the performance of the DM encoder is limited by two types of distortion as shown in Figure 12.10. One is called slope-overload distortion. It is due to the use of a step size Δ_1 that is too small to follow portions of the waveform that have a steep slope. The second type of distortion, called granular noise, results from using a step size that is too large in parts of the waveform having a small slope. The need to minimize both of these two types of distortion results in conflicting requirements in the selection of the step size Δ_1.

An alternative solution is to employ a variable size that adapts itself to the short-term characteristics of the source signal. That is, the step size is increased when the waveform has a steep slope and decreased when the waveform has a relatively small slope.

A variety of methods can be used to set adaptively the step size in every iteration. The quantized error sequence $\tilde{e}(n)$ provides a good indication of the slope characteristics of the waveform being encoded. When the quantized error $\tilde{e}(n)$ is changing signs between successive iterations, this is an indication that the slope of the waveform in the locality is relatively small. On the other hand, when the waveform has a steep slope, successive values of the error $\tilde{e}(n)$ are expected to have identical signs. From these observations it is possible to devise algorithms that decrease or increase the step size, depending on successive values of $\tilde{e}(n)$. A relatively simple rule devised by [13] is to vary adaptively the step size according to the relation

$$\Delta(n) = \Delta(n-1)\, K^{\tilde{e}(n)\tilde{e}(n-1)}, \quad n = 1, 2, \ldots \tag{12.23}$$

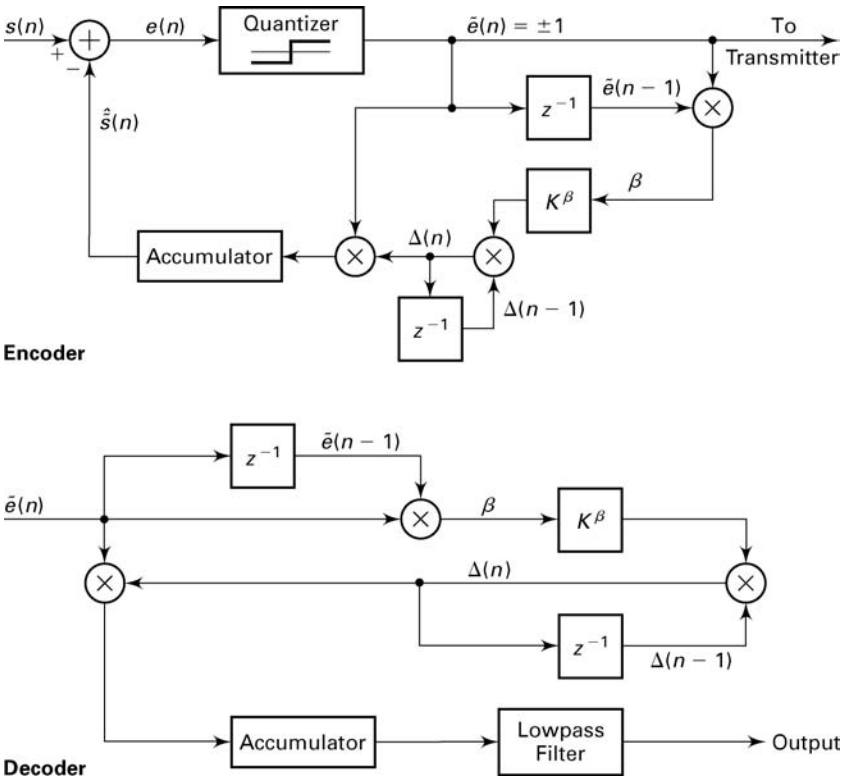


FIGURE 12.11 *An example of a delta modulation system with adaptive step size*

where $K \geq 1$ is a constant that is selected to minimize the total distortion. A block diagram of a DM encoder-decoder that incorporates this adaptive algorithm is illustrated in Figure 12.11.

Several other variations of adaptive DM encoding have been investigated and described in the technical literature. A particularly effective and popular technique first proposed by [6] is called *continuously variable slope delta modulation* (CVSD). In CVSD the adaptive step size parameter may be expressed as

$$\Delta(n) = \alpha\Delta\,(n-1) + k_1 \tag{12.24}$$

if $\tilde{e}(n)$, $\tilde{e}\,(n-1)$, and $\tilde{e}(n-2)$ have the same sign; otherwise

$$\Delta(n) = \alpha\Delta\,(n-1) + k_2 \tag{12.25}$$

The parameters α, k_1, and k_2 are selected such that $0 < \alpha < 1$ and $k_1 > k_2 > 0$. For more discussion on this and other variations of adaptive

DM, the interested reader is referred to the papers by Jayant [14] and Flanagan et al. [4] and to the extensive references contained in these papers.

12.4.2 PROJECT 12.4: DM AND ADM

The purpose of this project is to gain an understanding of delta modulation and adaptive delta modulation for coding of waveforms. This project involves writing MATLAB functions for the DM encoder and decoder as shown in Figure 12.9, and for the ADM encoder and decoder shown in Figure 12.11. The lowpass filter at the decoder can be implemented as a linear-phase FIR filter. For example, a Hanning filter that has the impulse response

$$h(n) = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{N-1}\right)\right], \quad 0 \le n \le N-1 \tag{12.26}$$

may be used, where the length N may be selected in the range $5 \le N \le 15$.

The input to the DM and ADM systems can be supplied from the waveforms generated in Project 12.1 except that the sampling rate should be higher by a factor of 5 to 10. The output of the decoder can be plotted. Comparisons should be made between the output signal from the DM and ADM decoders and the original input signal.

12.5 LINEAR PREDICTIVE CODING (LPC) OF SPEECH

The linear predictive coding (LPC) method for speech analysis and synthesis is based on modeling the vocal tract as a linear all-pole (IIR) filter having the system function

$$H(z) = \frac{G}{1 + \sum_{k=1}^{p} a_p(k)\, z^{-k}} \tag{12.27}$$

where p is the number of poles, G is the filter gain, and $\{a_p(k)\}$ are the parameters that determine the poles. There are two mutually exclusive excitation functions to model voiced and unvoiced speech sounds. On a short-time basis, voiced speech is periodic with a fundamental frequency F_0, or a pitch period $1/F_0$, which depends on the speaker. Thus voiced speech is generated by exciting the all-pole filter model by a periodic impulse train with a period equal to the desired pitch period. Unvoiced speech sounds are generated by exciting the all-pole filter model by the output of a random-noise generator. This model is shown in Figure 12.12.

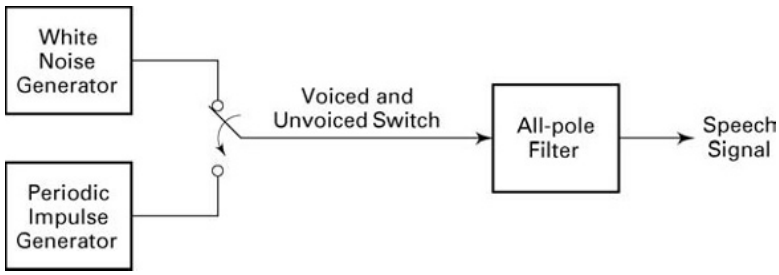


FIGURE 12.12 *Block diagram model for the generation of a speech signal*

Given a short-time segment of a speech signal, usually about 20 ms or 160 samples at an 8 kHz sampling rate, the speech encoder at the transmitter must determine the proper excitation function, the pitch period for voiced speech, the gain parameter G, and the coefficients $a_p(k)$. A block diagram that illustrates the speech encoding system is given in Figure 12.13. The parameters of the model are determined adaptively from the data and encoded into a binary sequence and transmitted to the receiver. At the receiver the speech signal is synthesized from the model and the excitation signal.

The parameters of the all-pole filter model are easily determined from the speech samples by means of linear prediction. To be specific, the output of the FIR linear prediction filter is

$$\hat{s}(n) = -\sum_{k=1}^{p} a_p(k)\, s(n-k) \tag{12.28}$$

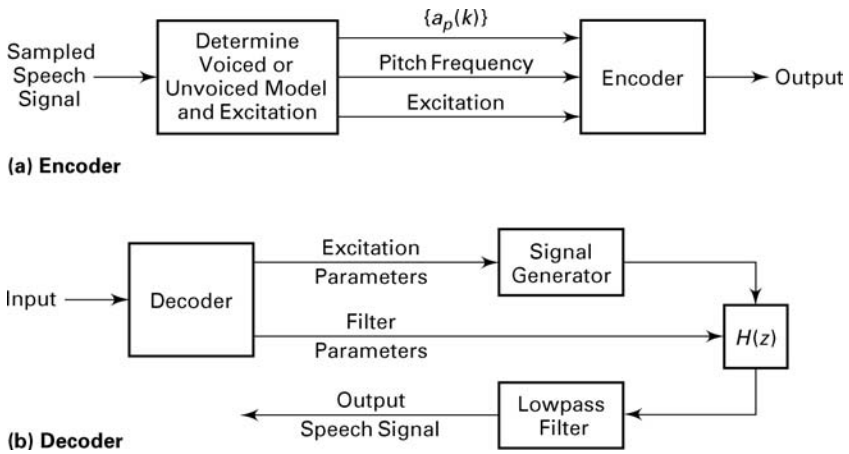


FIGURE 12.13 *Encoder and decoder for LPC*

and the corresponding error between the observed sample $s(n)$ and the predicted value $\hat{s}(n)$ is

$$e(n) = s(n) + \sum_{k=1}^{p} a_p(k)\, s(n-k) \tag{12.29}$$

By minimizing the sum of squared errors, that is,

$$\mathcal{E} = \sum_{n=0}^{N} e^2(n) = \sum_{n=0}^{N} \left[s(n) + \sum_{k=1}^{p} a_p(k)\, s(n-k) \right]^2 \tag{12.30}$$

we can determine the pole parameters $\{a_p(k)\}$ of the model. The result of differentiating $\mathcal{E}$ with respect to each of the parameters and equating the result to zero, is a set of p linear equations

$$\sum_{k=1}^{p} a_p(k)\, r_{ss}(m-k) = -r_{ss}(m), \quad m = 1, 2, \ldots, p \tag{12.31}$$

where $r_{ss}(m)$ is the autocorrelation of the sequence $s(n)$ defined as

$$r_{ss}(m) = \sum_{n=0}^{N} s(n) s(n+m) \tag{12.32}$$

The linear equation (12.31) can be expressed in matrix form as

$$\mathbf{R}_{ss}\mathbf{a} = -\mathbf{r}_{ss} \tag{12.33}$$

where $\mathbf{R}_{ss}$ is a $p \times p$ autocorrelation matrix, $\mathbf{r}_{ss}$ is a $p \times 1$ autocorrelation vector, and $\mathbf{a}$ is a $p \times 1$ vector of model parameters. Hence

$$\mathbf{a} = -\mathbf{R}_{ss}^{-1}\mathbf{r}_{ss} \tag{12.34}$$

These equations can also be solved recursively and most efficiently, without resorting to matrix inversion, by using the Levinson-Durbin algorithm [19]. However, in MATLAB it is convenient to use the matrix inversion. The all-pole filter parameters $\{a_p(k)\}$ can be converted to the all-pole lattice parameters $\{K_i\}$ (called the reflection coefficients) using the MATLAB function `dir2latc` developed in Chapter 6.

The gain parameter of the filter can be obtained by noting that its input-output equation is

$$s(n) = -\sum_{k=1}^{p} a_p(k)\, s(n-k) + Gx(n) \tag{12.35}$$

where $x(n)$ is the input sequence. Clearly,

$$Gx(n) = s(n) + \sum_{k=1}^{p} a_p(k)\, s(n-k) = e(n)$$

Then

$$G^2 \sum_{n=0}^{N-1} x^2(n) = \sum_{n=0}^{N-1} e^2(n) \tag{12.36}$$

If the input excitation is normalized to unit energy by design, then

$$G^2 = \sum_{n=0}^{N-1} e^2(n) = r_{ss}(0) + \sum_{k=1}^{p} a_p(k)\, r_{ss}(k) \tag{12.37}$$

Thus G^2 is set equal to the residual energy resulting from the least-squares optimization.

Once the LPC coefficients are computed, we can determine whether the input speech frame is voiced, and if so, what the pitch is. This is accomplished by computing the sequence

$$r_e(n) = \sum_{k=1}^{p} r_a(k)\, r_{ss}(n-k) \tag{12.38}$$

where $r_a(k)$ is defined as

$$r_a(k) = \sum_{i=1}^{p} a_p(i)\, a_p(i+k) \tag{12.39}$$

which is the autocorrelation sequence of the prediction coefficients. The pitch is detected by finding the peak of the normalized sequence $r_e(n)/r_e(0)$ in the time interval that corresponds to 3 to 15 ms in the 20-ms sampling frame. If the value of this peak is at least 0.25, the frame of speech is considered voiced with a pitch period equal to the value of $n = N_p$, where $r_e(N_p)/r_e(0)$ is a maximum. If the peak value is less than 0.25, the frame of speech is considered unvoiced and the pitch is zero.

The values of the LPC coefficients, the pitch period, and the type of excitation are transmitted to the receiver, where the decoder synthesizes the speech signal by passing the proper excitation through the all-pole filter model of the vocal tract. Typically, the pitch period requires 6 bits, and the gain parameter may be represented by 5 bits after its dynamic range is compressed logarithmically. If the prediction coefficients were to be coded, they would require between 8 to 10 bits per coefficient for accurate representation. The reason for such high accuracy is that relatively small changes in the prediction coefficients result in a large change in the pole positions of the filter model. The accuracy requirements are lessened by transmitting the reflection coefficients $\{K_i\}$, which have a smaller dynamic range—that is, $|K_i| < 1$. These are adequately represented by 6 bits per coefficient. Thus for a 10th-order predictor the total number of

bits assigned to the model parameters per frame is 72. If the model parameters are changed every 20 μsec. the resulting bit rate is $3,600$ bps. Since the reflection coefficients are usually transmitted to the receiver, the synthesis filter at the receiver is implemented as an all-pole lattice filter, described in Chapter 6.

12.5.1 PROJECT 12.5: LPC

The objective of this project is to analyze a speech signal through an LPC coder and then to synthesize it through the corresponding PLC decoder. Use several `.wav` sound files (sampled at 8000 sam/sec rate), which are available in MATLAB for this purpose. Divide speech signals into short-time segments (with lengths between 120 and 150 samples) and process each segment to determine the proper excitation function (voiced or unvoiced), the pitch period for voiced speech, the coefficients $\{a_p(k)\}$ $(p \leq 10)$, and the gain G. The decoder that performs the synthesis is an all-pole lattice filter whose parameters are the reflection coefficients that can be determined from $\{a_p(k)\}$. The output of this project is a synthetic speech signal that can be compared with the original speech signal. The distortion effects due to LPC analysis/synthesis may be assessed qualitatively.

12.6 DUAL-TONE MULTIFREQUENCY (DTMF) SIGNALS

DTMF is the generic name for push-button telephone signaling that is equivalent to the Touch Tone system in use within the Bell System. DTMF also finds widespread use in electronic mail systems and telephone banking systems in which the user can select options from a menu by sending DTMF signals from a telephone.

In a DTMF signaling system a combination of a high-frequency tone and a low-frequency tone represent a specific digit or the characters * and #. The eight frequencies are arranged as shown in Figure 12.14, to accommodate a total of 16 characters, 12 of which are assigned as shown, while the other four are reserved for future use.

DTMF signals are easily generated in software and detected by means of digital filters, also implemented in software, that are tuned to the 8 frequency tones. Usually, DTMF signals are interfaced to the analog world via a *codec* (coder/decoder) chip or by linear A/D and D/A converters. Codec chips contain all the necessary A/D and D/A, sampling, and filtering circuitry for a bidirectional analog/digital interface.

The DTMF tones may be generated either mathematically or from a look-up table. In a hardware implementation (e.g., in a digital signal

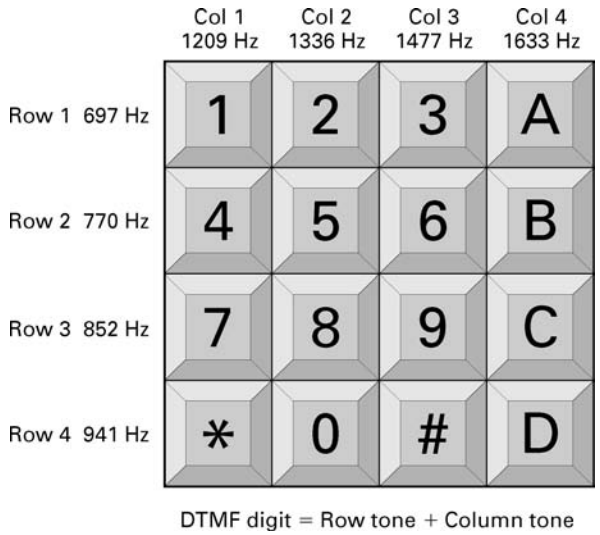


FIGURE 12.14 *DTMF digits*

processor), digital samples of two sine waves are generated mathematically, scaled, and added together. The sum is logarithmically compressed and sent to the codec for conversion to an analog signal. At an 8 kHz sampling rate the hardware must output a sample every 125 ms. In this case a sine look-up table is not used because the values of the sine wave can be computed quickly without using the large amount of data memory that a table look-up would require. For simulation and investigation purposes, the look-up table might be a good approach in MATLAB.

At the receiving end, the logarithmically compressed, 8-bit digital data words from the codec are received and logarithmically expanded to their 16-bit linear format. Then the tones are detected to decide on the transmitted digit. The detection algorithm can be a DFT implementation using the FFT algorithm or a filter bank implementation. For the relatively small number of tones to be detected, the filter bank implementation is more efficient. We now describe the use of the Goertzel algorithm to implement the 8 tuned filters.

Recall from the discussion in Chapter 5 that the DFT of an N-point data sequence $\{x(n)\}$ is

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad k = 0, 1, \ldots, N-1 \tag{12.40}$$

If the FFT algorithm is used to perform the computation of the DFT, the number of computations (complex multiplications and additions) is $N \log_2 N$. In this case we obtain all N values of the DFT at once. However,

if we desire to compute only M points of the DFT, where $M < \log_2 N$, then a direct computation of the DFT is more efficient. The Goertzel algorithm, which is now described, is basically a linear filtering approach to the computation of the DFT and provides an alternative to direct computation.

12.6.1 THE GOERTZEL ALGORITHM

The Goertzel algorithm exploits the periodicity of the phase factors $\{W_N^k\}$ and allows us to express the computation of the DFT as a linear filtering operation. Since $W_N^{-kN} = 1$, we can multiply the DFT by this factor. Thus

$$X(k) = W_N^{-kN} X(k) = \sum_{m=0}^{N-1} x(m) W_N^{-k(N-m)} \tag{12.41}$$

We note that (12.41) is in the form of a convolution. Indeed, if we define the sequence $y_k(n)$ as

$$y_k(n) = \sum_{m=0}^{N-1} x(m) W_N^{-k(n-m)} \tag{12.42}$$

then it is clear that $y_k(n)$ is the convolution of the finite-duration input sequence $x(n)$ of length N with a filter that has an impulse response

$$h_k(n) = W_N^{-kn} u(n) \tag{12.43}$$

The output of this filter at $n = N$ yields the value of the DFT at the frequency $\omega_k = 2\pi k/N$. That is,

$$X(k) = y_k(n)|_{n=N} \tag{12.44}$$

as can be verified by comparing (12.41) with (12.42).

The filter with impulse response $h_k(n)$ has the system function

$$H_k(z) = \frac{1}{1 - W_N^{-k} z^{-1}} \tag{12.45}$$

This filter has a pole on the unit circle at the frequency $\omega_k = 2\pi k/N$. Thus the entire DFT can be computed by passing the block of input data into a parallel bank of N single-pole filters (resonators), where each filter has a pole at the corresponding frequency of the DFT.

Instead of performing the computation of the DFT as in (12.42), via convolution, we can use the difference equation corresponding to the filter given by (12.45) to compute $y_k(n)$ recursively. Thus we have

$$y_k(n) = W_N^{-k} y_k(n-1) + x(n), \quad y_k(-1) = 0 \tag{12.46}$$

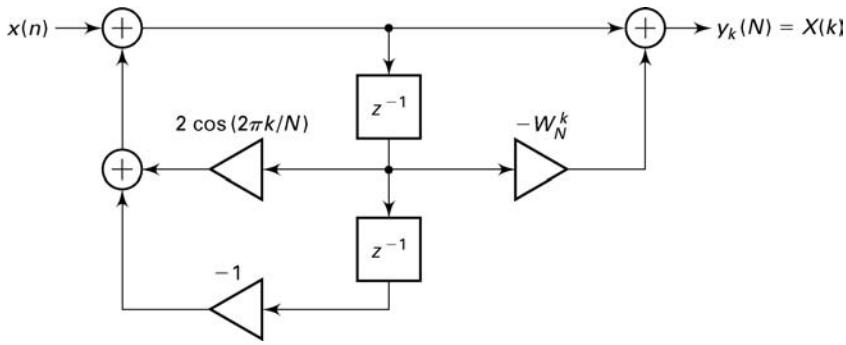


FIGURE 12.15 *Realization of two-pole resonator for computing the DFT*

The desired output is $X(k) = y_k(N)$. To perform this computation, we can compute once and store the phase factor W_N^{-k}.

The complex multiplications and additions inherent in (12.46) can be avoided by combining the pairs of resonators possessing complex conjugate poles. This leads to 2-pole filters with system functions of the form

$$H_k(z) = \frac{1 - W_N^k z^{-1}}{1 - 2\cos(2\pi k/N)\, z^{-1} + z^{-2}} \tag{12.47}$$

The realization of the system illustrated in Figure 12.15 is described by the difference equations

$$v_k(n) = 2\cos\frac{2\pi k}{N} v_k(n-1) - v_k(n-2) + x(n) \tag{12.48}$$

$$y_k(n) = v_k(n) - W_N^k v_k(n-1) \tag{12.49}$$

with initial conditions $v_k(-1) = v_k(-2) = 0$. This is the Goertzel algorithm.

The recursive relation in (12.48) is iterated for $n = 0, 1, \ldots, N$, but the equation in (12.49) is computed only once, at time $n = N$. Each iteration requires one real multiplication and two additions. Consequently, for a real input sequence $x(n)$, this algorithm requires $N + 1$ real multiplications to yield not only $X(k)$ but also, due to symmetry, the value of $X(N - k)$.

We can now implement the DTMF decoder by use of the Goertzel algorithm. Since there are eight possible tones to be detected, we require eight filters of the type given by (12.47), with each filter tuned to one of the eight frequencies. In the DTMF detector, there is no need to compute the complex value $X(k)$; only the magnitude $|X(k)|$ or the magnitude-squared value $|X(k)|^2$ will suffice. Consequently, the final step in the computation of the DFT value involving the numerator term (feedforward part of the

filter computation) can be simplified. In particular, we have

$$|X(k)|^2 = |y_k(N)|^2 = \left|v_k(N) - W_N^k v_k\,(N-1)\right|^2 \qquad \textbf{(12.50)}$$

$$= v_k^2(N) + v_k^2\,(N-1) - \left(2\cos\frac{2\pi k}{N}\right) v_k(N) v_k\,(N-1)$$

Thus complex-valued arithmetic operations are completely eliminated in the DTMF detector.

12.6.2 PROJECT 12.6: DTMF SIGNALING

The objective of this project is to gain an understanding of the DTMF tone generation software and the DTMF decoding algorithm (the Goertzel algorithm). Design the following MATLAB modules:

1. a tone generation function that accepts an array containing dialing digits and produces a signal containing appropriate tones (from Figure 12.14) of 0.5-second duration for each digit at 8 kHz sampling frequency
2. a dial-tone generator generating samples of $(350 + 440)$ Hz frequency at 8 kHz sampling interval for a specified amount of duration
3. a decoding function to implement (12.50) that accepts a DTMF signal and produces an array containing dialing digits

Generate several dialing list arrays containing a mix of digits and dial tones. Experiment with the tone generation and detection modules and comment on your observations. Use MATLAB's sound generation capabilities to listen to the tones and to observe the frequency components of the generated tones.

12.7 BINARY DIGITAL COMMUNICATIONS

Digitized speech signals that have been encoded via PCM, ADPCM, DM, and LPC are usually transmitted to the decoder by means of digital modulation. A binary digital communications system employs two signal waveforms, say $s_1(t) = s(t)$ and $s_2(t) = -s(t)$, to transmit the binary sequence representing the speech signal. The signal waveform $s(t)$, which is nonzero over the interval $0 \le t \le T$, is transmitted to the receiver if the data bit is a 1, and the signal waveform $-s(t)$, $0 \le t \le T$ is transmitted if the data bit is a 0. The time interval T is called the signal interval, and the bit rate over the channel is $R = 1/T$ bits per second. A typical signal waveform $s(t)$ is a rectangular pulse—that is, $s(t) = A$, $0 \le t \le T$—which has energy A^2T.

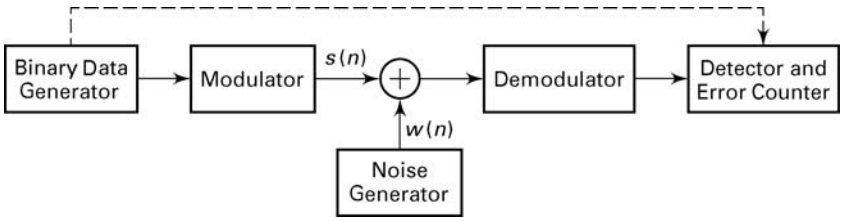


FIGURE 12.16 *Model of binary data communications system*

In practice the signal waveforms transmitted over the channel are corrupted by additive noise and other types of channel distortions that ultimately limit the performance of the communications system. As a measure of performance, we normally use the average probability of error, which is often called the bit error rate.

12.7.1 PROJECT 12.7: BINARY DATA COMMUNICATIONS SYSTEM

The purpose of this project is to investigate the performance of a binary data communications system on an additive noise channel by means of simulation. The basic configuration of the system to be simulated is shown in Figure 12.16. Five MATLAB functions are required.

1. A binary data generator module that generates a sequence of independent binary digits with equal probability.
2. A modulator module that maps a binary digit 1 into a sequence of M consecutive $+1$'s, and maps a binary digit 0 into a sequence of M consecutive -1's. Thus the M consecutive $+1$'s represent a sampled version of the rectangular pulse.
3. A noise generator that generates a sequence of uniformly distributed numbers over the interval $(-a, a)$. Each noise sample is added to a corresponding signal sample.
4. A demodulator module that sums the M successive outputs of the noise corrupted sequence $+1$'s or -1's received from the channel. We assume that the demodulator is time synchronized so that it knows the beginning and end of each waveform.
5. A detector and error-counting module. The detector compares the output of the modulator with zero and decides in favor of 1 if the output is greater than zero and in favor of zero if the output is less than zero. If the output of the detector does not agree with the transmitted bit from the transmitter, an error is counted by the counter. The error rate depends on the ratio (called signal-to-noise ratio) of the size of M to the additive noise power, which is $P_n = a^2/3$.

The measured error rate can be plotted for different signal-to-noise ratios, either by changing M and keeping P_n fixed or vice versa.

12.8 SPREAD-SPECTRUM COMMUNICATIONS

Spread-spectrum signals are often used in the transmission of digital data over communication channels that are corrupted by interference due to intentional jamming or from other users of the channel (e.g., cellular telephones and other wireless applications). In applications other than communications, spread-spectrum signals are used to obtain accurate range (time delay) and range rate (velocity) measurements in radar and navigation. For the sake of brevity we shall limit our discussion to the use of spread spectrum for digital communications. Such signals have the characteristic that their bandwidth is much greater than the information rate in bits per second.

In combatting intentional interference (jamming), it is important to the communicators that the jammer who is trying to disrupt their communication does not have prior knowledge of the signal characteristics. To accomplish this, the transmitter introduces an element of unpredictability or randomness (pseudo-randomness) in each of the possible transmitted signal waveforms, which is known to the intended receiver, but not to the jammer. As a consequence, the jammer must transmit an interfering signal without knowledge of the pseudo-random characteristics of the desired signal.

Interference from other users arises in multiple-access communications systems in which a number of users share a common communications channel. At any given time a subset of these users may transmit information simultaneously over a common channel to corresponding receivers. The transmitted signals in this common channel may be distinguished from one another by superimposing a different pseudo-random pattern, called a *multiple-access code*, in each transmitted signal. Thus a particular receiver can recover the transmitted data intended for it by knowing the pseudo-random pattern, that is, the key used by the corresponding transmitter. This type of communication technique, which allows multiple users to simultaneously use a common channel for data transmission, is called *code division multiple access* (CDMA).

The block diagram shown in Figure 12.17 illustrates the basic elements of a spread-spectrum digital communications system. It differs

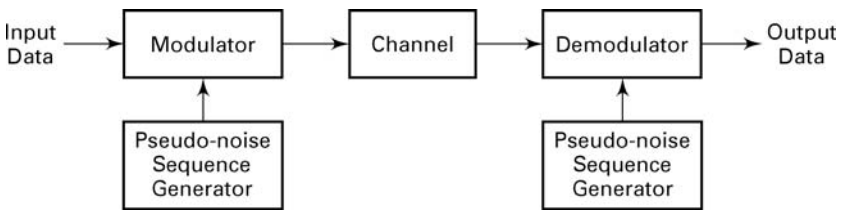


FIGURE 12.17 *Basic spread spectrum digital communications system*

from a conventional digital communications system by the inclusion of two identical pseudo-random pattern generators, one that interfaces with the modulator at the transmitting end and the second that interfaces with the demodulator at the receiving end. The generators generate a pseudo-random or *pseudo-noise* (PN) binary-valued sequence (±1's), which is impressed on the transmitted signal at the modulator and removed from the received signal at the demodulator.

Synchronization of the PN sequence generated at the demodulator with the PN sequence contained in the incoming received signal is required in order to demodulate the received signal. Initially, prior to the transmission of data, synchronization is achieved by transmitting a short fixed PN sequence to the receiver for purposes of establishing synchronization. After time synchronization of the PN generators is established, the transmission of data commences.

12.8.1 PROJECT 12.8: BINARY SPREAD-SPECTRUM COMMUNICATIONS

The objective of this project is to demonstrate the effectiveness of a PN spread-spectrum signal in suppressing sinusoidal interference. Let us consider the binary communication system described in Project 12.7, and let us multiply the output of the modulator by a binary (±1) PN sequence. The same binary PN sequence is used to multiply the input to the demodulator and thus to remove the effect of the PN sequence in the desired signal. The channel corrupts the transmitted signal by the addition of a

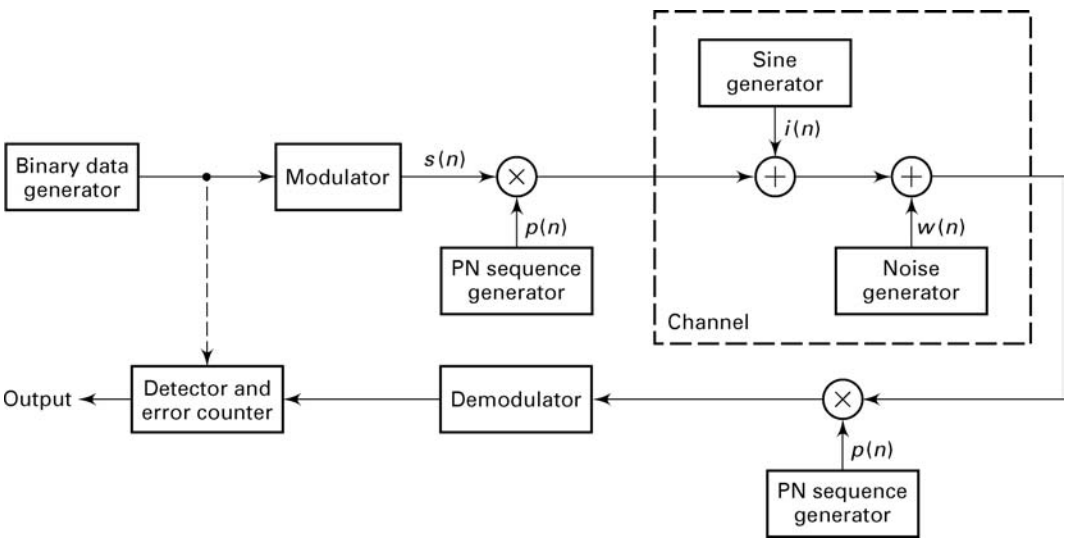


FIGURE 12.18 *Block diagram of binary PN spread-spectrum system for simulation experiment*

wideband noise sequence $\{w(n)\}$ and a sinusoidal interference sequence of the form $i(n) = A \sin \omega_0 n$, where $0 < \omega_0 < \pi$. We may assume that $A \geq M$, where M is the number of samples per bit from the modulator. The basic binary spread spectrum-system is shown in Figure 12.18. As can be observed, this is just the binary digital communication system shown in Figure 12.16, to which we have added the sinusoidal interference and the PN sequence generators. The PN sequence may be generated by using a random-number generator to generate a sequence of equally probable ± 1's.

Execute the simulated system with and without the use of the PN sequence, and measure the error rate under the condition that $A \geq M$ for different values of M, such as $M = 50$, 100, 500, 1000. Explain the effect of the PN sequence on the sinusoidal interference signal. Thus explain why the PN spread-spectrum system outperforms the conventional binary communication system in the presence of the sinusoidal jamming signal.