2

DSP Fundamentals and Implementation Considerations

This chapter introduces fundamental DSP principles and practical implementation considerations for the digital filters and algorithms [1–4]. DSP implementations, especially using fixed-point processors, require special consideration due to quantization and arithmetic errors.

2.1 Digital Signals and Systems

This section briefly introduces some basic terminologies of digital signals and systems that will be used in the book.

2.1.1 Elementary Digital Signals

A digital signal is a sequence of numbers $x(n)$, $-\infty < n < \infty$, where integer n is the time index. Signals can be classified as deterministic or random. Deterministic signals such as sinusoidal signals can be expressed mathematically. Random signals such as speech and noise cannot be described exactly by equations. Some basic deterministic signals will be introduced in this section along with the frequency concepts, while random signals will be discussed in Section 2.3.

The unit-impulse signal is defined as

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0, \end{cases} \tag{2.1}$$

where $\delta(n)$ is also called the Kronecker delta function. This unit-impulse signal is very useful for measuring, modeling, and analyzing the characteristics of DSP systems.

The unit-step signal is defined as

$$u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0. \end{cases} \tag{2.2}$$

Real-Time Digital Signal Processing: Fundamentals, Implementations and Applications, Third Edition.
Sen M. Kuo, Bob H. Lee and Wenshun Tian.

This function is very convenient for describing causal signals, which are the most commonly encountered signals in real-time DSP systems. For example, $x(n)u(n)$ clearly defines a causal signal $x(n)$ such that $x(n) = 0$ for $n < 0$.

Sinusoidal signals (also called sinusoids) can be expressed in simple mathematical formulas. For example, an analog sine wave can be expressed as

$$x(t) = A\sin(\Omega t + \phi) = A\sin(2\pi f t + \phi), \tag{2.3}$$

where A is the amplitude of the sine wave, f is the frequency in cycles per second (hertz or Hz),

$$\Omega = 2\pi f \tag{2.4}$$

is the frequency in radians per second (rad/s), and ϕ is the initial phase in radians.

Sampling the analog sinusoidal signal defined in (2.3) results in a digital sine wave expressed as

$$x(n) = A\sin(\Omega nT + \phi) = A\sin(2\pi f nT + \phi), \tag{2.5}$$

where $T = 1/f_s$ is the sampling period in seconds and f_s is the sampling rate (or sampling frequency) in Hz. This digital signal can also be expressed as

$$x(n) = A\sin(\omega n + \phi) = A\sin(F\pi n + \phi), \tag{2.6}$$

where

$$\omega = \Omega T = \frac{2\pi f}{f_s} \tag{2.7}$$

is the digital frequency in radians per sample, and

$$F = \frac{\omega}{\pi} = \frac{f}{(f_s/2)} \tag{2.8}$$

is the normalized digital frequency in cycles per sample.

The units, relationships, and ranges of these analog and digital frequency variables are summarized in Table 2.1. Sampling of analog signals results in the mapping of analog frequency variable f (or Ω) into a finite range of digital frequency variable ω (or F). The

Table 2.1 Units, relationships, and ranges of four frequency variables

Variables	Units	Relationships	Ranges
Ω	Radians per second	$\Omega = 2\pi f$	$-\infty < \Omega < \infty$
f	Cycles per second (Hz)	$f = \frac{\Omega}{2\pi} = \frac{\omega f_s}{2\pi}$	$-\infty < f < \infty$
ω	Radians per sample	$\omega = \Omega T = \frac{2\pi f}{f_s}$	$-\pi \leq \omega \leq \pi$
F	Cycles per sample	$F = \frac{f}{(f_s/2)} = \frac{\omega}{\pi}$	$-1 \leq F \leq 1$

highest frequency in a digital signal is $f = f_s/2, \omega = \pi$, or $F = 1$ based on the sampling theorem defined in (1.3). Therefore, the frequency contents of digital signals are restricted to the limited range as shown in Table 2.1.

Example 2.1

Generate 32 sine wave samples with $A = 2$, $f = 1000$ Hz, and $f_s = 8000$ Hz using MATLAB® [5–7].

From Table 2.1, we have $\omega = 2\pi f/f_s = 0.25\pi$. From (2.6), we can express the sine wave as $x(n) = 2\sin(0.25\pi n)$, $n = 0, 1, \ldots, 31$. The generated sine wave samples are plotted (as shown in Figure 2.1) and saved in a data file using ASCII format by the following MATLAB® script (`example2_1.m`):

```
n = [0:31];                  % Time index n=0,1, . . . ,31
omega = 0.25*pi;             % Digital frequency
xn = 2*sin(omega*n);         % Sine wave generation
plot(n, xn, 'o');            % Samples are marked by 'o'
xlabel('Time index, n');
ylabel('Amplitude');
axis([0 31 -2 2]);           % Define ranges of plot
save sine.dat xn -ascii;     % Save in ASCII data file
```

In the code, the MATLAB® function `save` is used to store the variable `xn` using eight-digit ASCII format in the data file `sine.dat`. This ASCII file can be read by other MATLAB® scripts using the function `load`. The default format for the `save` and `load` functions is a binary MAT-file with the extension `.mat`.

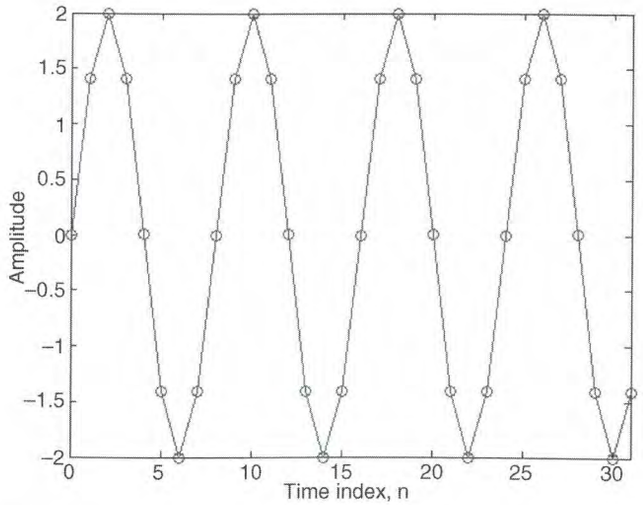

Figure 2.1 An example of a sine wave with $A = 2$ and $\omega = 0.25\pi$

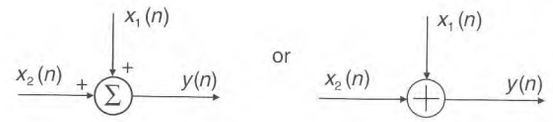

Figure 2.2 Block diagram of an adder

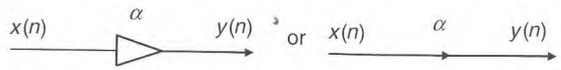

Figure 2.3 Block diagram of a multiplier

2.1.2 *Block Diagram Representation of Digital Systems*

A DSP system performs prescribed operations on signals. The processing of digital signals can be described as a combination of three basic operations: addition (or subtraction), multiplication, and time shift (or delay). Thus, a DSP system consists of the interconnection of three basic elements: adders, multipliers, and delay units.

Two signals, $x_1(n)$ and $x_2(n)$, can be added as illustrated in Figure 2.2, where the adder output is expressed as

$$y(n) = x_1(n) + x_2(n). \tag{2.9}$$

The adder could be drawn as a multi-input adder with more than two inputs, but the additions are typically performed with two signals at a time in practical DSP systems.

A given signal can be multiplied by a scalar, α, as illustrated in Figure 2.3, where $x(n)$ is the multiplier input and the multiplier's output is

$$y(n) = \alpha x(n). \tag{2.10}$$

Multiplication of a sequence by a gain factor, α, results in all samples in the sequence being scaled by α. The output signal is amplified if $|\alpha| > 1$, or attenuated if $|\alpha| < 1$.

The signal $x(n)$ can be delayed in time by one sampling period, T, as illustrated in Figure 2.4, where the box labeled z^{-1} represents the unit delay, $x(n)$ is the input signal, and the output signal

$$y(n) = x(n-1). \tag{2.11}$$

In fact, the signal $x(n-1)$ is actually the previous signal sample stored in memory before the current time n. Therefore, the delay unit is very easy to realize in a digital system with memory, but is difficult to implement in an analog system. A delay by more than one unit

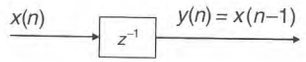

Figure 2.4 Block diagram of a unit delay

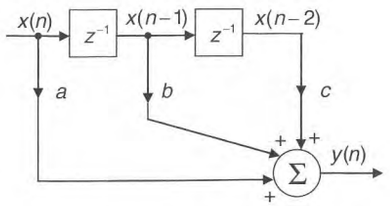

Figure 2.5 Signal-flow diagram of the DSP system described by (2.12)

can be implemented by cascading several delay units in a row. Therefore, an L-unit delay requires $L+1$ memory locations configured as a first-in, first-out buffer (tapped delay line) in memory.

Example 2.2

Consider a simple DSP system described by the difference equation

$$y(n) = ax(n) + bx(n-1) + cx(n-2), \tag{2.12}$$

where a, b, and c are real numbers. The signal-flow diagram of the system using three basic building blocks is sketched in Figure 2.5, which shows that the output signal $y(n)$ is computed using three multiplications and two additions. The difference equation given in (2.12) defines the I/O relationship of the system, and thus is also called the I/O equation.

2.2 System Concepts

This section introduces basic concepts and techniques to describe and analyze linear time-invariant (LTI) digital systems.

2.2.1 LTI Systems

If the input signal to an LTI system is the unit-impulse signal $\delta(n)$ defined in (2.1), then the output signal $y(n)$ is the impulse response of the system, $h(n)$.

Example 2.3

Consider a digital system defined by the I/O equation

$$y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2). \tag{2.13}$$

Applying the unit-impulse signal $\delta(n)$ to the input of the system, the output $y(n)$ is the system impulse response $h(n)$ that can be computed as follows:

$$\begin{aligned} h(0) &= y(0) = b_0 \cdot 1 + b_1 \cdot 0 + b_2 \cdot 0 = b_0 \\ h(1) &= y(1) = b_0 \cdot 0 + b_1 \cdot 1 + b_2 \cdot 0 = b_1 \\ h(2) &= y(2) = b_0 \cdot 0 + b_1 \cdot 0 + b_2 \cdot 1 = b_2 \\ h(3) &= y(3) = b_0 \cdot 0 + b_1 \cdot 0 + b_2 \cdot 0 = 0 \end{aligned}$$

Therefore, the impulse response of the system defined in (2.13) is $\{b_0, b_1, b_2, 0, 0, \ldots\}$.

The I/O equation given in (2.13) can be generalized with L coefficients as

$$\begin{aligned} y(n) &= b_0x(n) + b_1x(n-1) + \cdots + b_{L-1}x(n-L+1) \\ &= \sum_{l=0}^{L-1} b_l x(n-l). \end{aligned} \tag{2.14}$$

Substituting $x(n) = \delta(n)$ into (2.14), the output is the impulse response expressed as

$$\begin{aligned} h(n) &= \sum_{l=0}^{L-1} b_l \delta(n-l) \\ &= \begin{cases} b_n, & n = 0, 1, \ldots, L-1 \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{2.15}$$

Therefore, the length of the impulse response is L for the system defined in (2.14). This system is called a finite impulse response (FIR) filter. The parameters, $b_l, l = 0, 1, \ldots, L-1$, are filter coefficients (also referred as filter weights or taps). For FIR filters, the filter coefficients are identical to the non-zero samples of impulse response.

The signal-flow diagram of the system described by the I/O equation (2.14) is illustrated in Figure 2.6. The string of z^{-1} units is the tapped-delay line. The parameter, L, is the length of the FIR filter. Note that the order of the filter is $L-1$ for the FIR filter with length L since there are $L-1$ zeros. The design and implementation of FIR filters will be further discussed in Chapter 3.

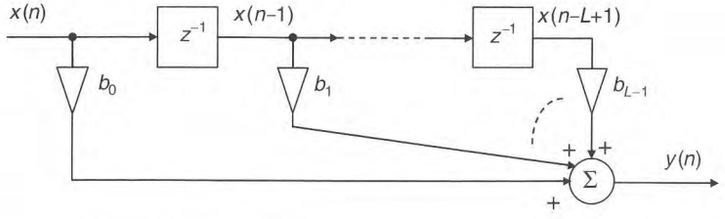

Figure 2.6 Detailed signal-flow diagram of an FIR filter

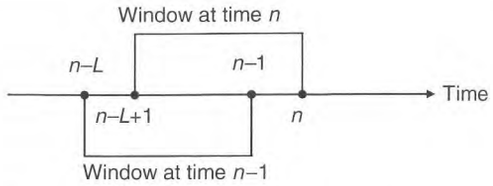

Figure 2.7 Time windows at current time n and previous time $n-1$

The moving (running) average filter is a simple example of the FIR filter. Consider the L-point moving-average filter defined as

$$\begin{aligned} y(n) &= \frac{1}{L}[x(n) + x(n-1) + \cdots + x(n-L+1)] \\ &= \frac{1}{L}\sum_{l=0}^{L-1} x(n-l), \end{aligned} \tag{2.16}$$

where each output signal sample is the average of L consecutive samples of current and passed input signal. Implementation of (2.16) requires $L-1$ additions and L memory locations for storing signal samples $x(n)$, $x(n-1)$, . . . , $x(n-L+1)$ in the memory buffer. Because most digital signal processors have a hardware multiplier, the division by a constant L can be implemented by multiplication of a constant α, where $\alpha = 1/L$.

The concept of a moving window is illustrated in Figure 2.7, where L signal samples inside the rectangular window at time n are used to compute the current output signal $y(n)$. Comparing the windows at time n and $n-1$, the oldest sample $x(n-L)$ for the window at time $n-1$ is replaced by the newest sample $x(n)$ for the window at time n, and the remaining $L-1$ samples are the same as those samples used by the previous window at time $n-1$ to compute $y(n-1)$. Therefore, the averaged signal, $y(n)$, can be computed recursively as

$$y(n) = y(n-1) + \frac{1}{L}[x(n) - x(n-L)]. \tag{2.17}$$

This recursive equation can be realized by using only two additions. Comparing it to the direct implementation of (2.16) that needs $L-1$ additions, the recursive equation (2.17) shows that with careful design (or optimization), the complexity of a system (or algorithm) can be reduced. However, we still need $L+1$ memory locations for keeping $L+1$ signal samples $\{x(n)\ x(n-1)\ \ldots\ x(n-L)\}$.

Consider the LTI system illustrated in Figure 2.8, where the output signal can be expressed as

$$\begin{aligned} y(n) &= x(n) * h(n) = h(n) * x(n) \\ &= \sum_{l=-\infty}^{\infty} x(l)h(n-l) = \sum_{l=-\infty}^{\infty} h(l)x(n-l), \end{aligned} \tag{2.18}$$

where $*$ denotes the linear convolution operation. The exact internal structure of the system is either unknown or ignored. The only way to interact with the system is by using its input and

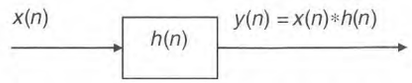

Figure 2.8 An LTI system expressed in the time domain

output terminals as shown in Figure 2.8. This "black box" representation is a very effective way to depict complicated DSP systems.

A digital system is a causal system if and only if

$$h(n) = 0, \quad n < 0. \tag{2.19}$$

A causal system does not provide a zero-state response prior to input application. That is, the output depends only on the current and past samples of the input. This is an obvious property for real-time DSP systems. However, if the data is a previous digitized signal saved in a file, the algorithm operating on the data set does not need to be causal. For a causal system, the lower bounds of the summation in (2.18) can be modified to reflect this restriction as

$$y(n) = \sum_{l=0}^{\infty} h(l)x(n-l). \tag{2.20}$$

Example 2.4

Consider the I/O equation of a digital system expressed as

$$y(n) = bx(n) - ay(n-1), \tag{2.21}$$

where each output signal $y(n)$ is dependent on the current input signal $x(n)$ and the past output signal $y(n-1)$. Assume that the system is causal, that is, $y(n) = 0$ for $n < 0$, and the input is a unit-impulse signal, that is, $x(n) = \delta(n)$. The output signal (impulse response) samples are computed as

$$\begin{aligned} y(0) &= bx(0) - ay(-1) = b \\ y(1) &= bx(1) - ay(0) = -ay(0) = -ab \\ y(2) &= bx(2) - ay(1) = -ay(1) = a^2 b. \end{aligned}$$

In general, we have $h(n) = y(n) = (-1)^n a^n b, n = 0, 1, 2, \ldots, \infty$. This system has infinite impulse response $h(n)$ if the coefficients a and b are non-zero, and thus is called an infinite impulse response (IIR) system.

A digital filter can be classified as either an FIR filter or IIR filter, depending on whether the impulse response of the filter has finite or infinite duration. The system defined in (2.21) is an IIR filter since it has infinite impulse response as shown by Example 2.4. The I/O equation of an IIR system can be generalized as

$$\begin{aligned} y(n) &= b_0 x(n) + b_1 x(n-1) + \cdots + b_{L-1} x(n-L+1) - a_1 y(n-1) - \cdots - a_M y(n-M) \\ &= \sum_{l=0}^{L-1} b_l x(n-l) - \sum_{m=1}^{M} a_m y(n-m), \end{aligned} \tag{2.22}$$

where M is the order of the system. This IIR system is defined by a set of feedforward coefficients $\{b_l, l = 0, 1, \ldots, L-1\}$ and feedback coefficients $\{a_m, m = 1, 2, \ldots, M\}$. Since the weighted output samples (by a_m) are fed back and combined with the weighted input samples (by b_l), IIR systems are feedback systems. Note that if all the a_m are zero, Equation (2.22) is identical to (2.14) which defines an FIR filter. Therefore, an FIR filter is a special case of an IIR filter when all feedback coefficients a_m are zero. The design and implementation of IIR filters will be further discussed in Chapter 4.

Example 2.5

The IIR filters given in (2.22) can be implemented using the MATLAB® function `filter` as

```
yn = filter(b, a, xn);
```

The vector `b` contains feedforward coefficients $\{b_l, l = 0, 1, \ldots, L-1\}$ and the vector `a` contains feedback coefficients $\{a_m, m = 0, 1, 2, \ldots, M$, where $a_0 = 1\}$. The signal vectors `xn` and `yn` are the input and output buffers of the system, respectively. The FIR filter defined in (2.14) can be implemented as

```
yn = filter(b, 1, xn);
```

This is because all a_m are zero except $a_0 = 1$ for the FIR filter.

Example 2.6

Assume the window length L is large so that the oldest sample $x(n-L)$ can be approximated by its average $y(n-1)$; then the moving-average filter defined in (2.17) can be approximated as

$$\begin{aligned} y(n) &\cong \left(1 - \frac{1}{L}\right) y(n-1) + \frac{1}{L} x(n) \\ &= (1-\alpha) y(n-1) + \alpha x(n) \end{aligned} \tag{2.23}$$

where $\alpha = 1/L$. This is a simple first-order IIR filter. Comparing (2.23) to (2.17), we need two multiplications instead of one, but only two memory locations instead of $L+1$. Thus, the recursive equation (2.23) is the most efficient technique for approximating a moving-average filter.

2.2.2 *The z-transform*

Continuous-time systems are commonly designed and analyzed using the Laplace transform, which will be briefly introduced in Chapter 4. For discrete-time systems, the transform

corresponding to the Laplace transform is the z-transform. The z-transform of a digital signal $x(n)$ is defined as

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) z^{-n}, \tag{2.24}$$

where $X(z)$ represents the z-transform of $x(n)$. The variable z is a complex number and can be expressed in polar form as

$$z = re^{j\theta}, \tag{2.25}$$

where r is the magnitude (radius) of z and θ is the angle of z. When $r = 1$, $|z| = 1$ is called the unit circle on the z-plane. Since the z-transform involves an infinite power series, it exists only for those values of z where the power series defined in (2.24) converges. The region on the complex z-plane in which the power series converges is called the region of convergence.

For causal signals, the two-sided z-transform defined in (2.24) becomes the one-sided z-transform expressed as

$$X(z) = \sum_{n=0}^{\infty} x(n) z^{-n}. \tag{2.26}$$

Example 2.7

Consider the exponential function

$$x(n) = a^n u(n).$$

The z-transform can be computed as

$$X(z) = \sum_{n=-\infty}^{\infty} a^n z^{-n} u(n) = \sum_{n=0}^{\infty} \left(az^{-1}\right)^n.$$

Using the infinite geometric series given in Appendix A, we have

$$X(z) = \frac{1}{1 - az^{-1}} = \frac{z}{z-a} \quad \text{if } |az^{-1}| < 1$$

Thus the equivalent condition for convergence (or region of convergence) is

$$|z| > |a|,$$

which is the region outside the circle with radius a.

Some z-transform properties are useful for the analysis of discrete-time LTI systems. These properties are summarized as follows:

1. *Linearity* (superposition). The z-transform (ZT) of the sum of two sequences is the sum of the z-transforms of the individual sequences. That is,

$$\begin{aligned} \mathrm{ZT}[a_1x_1(n) + a_2x_2(n)] &= a_1\mathrm{ZT}[x_1(n)] + a_2\mathrm{ZT}[x_2(n)] \\ &= a_1X_1(z) + a_2X_2(z), \end{aligned} \tag{2.27}$$

where a_1 and a_2 are constants.

2. *Time shifting* (delay). The z-transform of the shifted (delayed) signal $y(n) = x(n-k)$ is

$$Y(z) = \mathrm{ZT}[x(n-k)] = z^{-k}X(z). \tag{2.28}$$

For example, $\mathrm{ZT}[x(n-1)] = z^{-1}X(z)$. The unit delay z^{-1} corresponds to a time shift to the right of one sample.

3. *Convolution*. Considering the signal $x(n)$ as a linear convolution of two sequences

$$x(n) = x_1(n) * x_2(n), \tag{2.29}$$

we have

$$X(z) = X_1(z)X_2(z). \tag{2.30}$$

Thus, the z-transform converts the linear convolution in the time domain to multiplication in the z-domain.

2.2.3 *Transfer Functions*

Consider the LTI system illustrated in Figure 2.8. Using the convolution property, we have

$$Y(z) = X(z)H(z), \tag{2.31}$$

where $X(z) = \mathrm{ZT}[x(n)]$, $Y(z) = \mathrm{ZT}[y(n)]$, and $H(z) = \mathrm{ZT}[h(n)]$. The combination of time- and z-domain representations of the LTI system is illustrated in Figure 2.9, where ZT^{-1} denotes the inverse z-transform. This diagram shows that we can compute the output of a linear system by replacing the time-domain convolution with the z-domain multiplication.

The transfer function of an LTI system is defined in terms of the system's input and output. From (2.31), the transfer function is defined as

$$H(z) = \frac{Y(z)}{X(z)}. \tag{2.32}$$

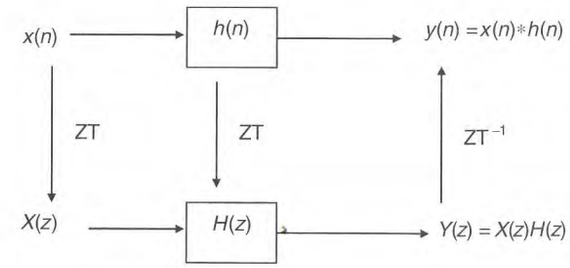

Figure 2.9 A block diagram of the LTI system in both the time domain and z domain

The transfer function can be used to create alternative filters that have exactly the same I/O behavior. An important example is the cascade or parallel connection of two or more low-order systems to realize a high-order system, as illustrated in Figure 2.10. In the cascade (series) interconnection shown in Figure 2.10(a), we have

$$Y_1(z) = X(z)H_1(z) \quad \text{and} \quad Y(z) = Y_1(z)H_2(z).$$

Thus,

$$Y(z) = X(z)H_1(z)H_2(z).$$

Therefore, the overall transfer function of the cascade of two systems is

$$H(z) = H_1(z)H_2(z) = H_2(z)H_1(z). \tag{2.33}$$

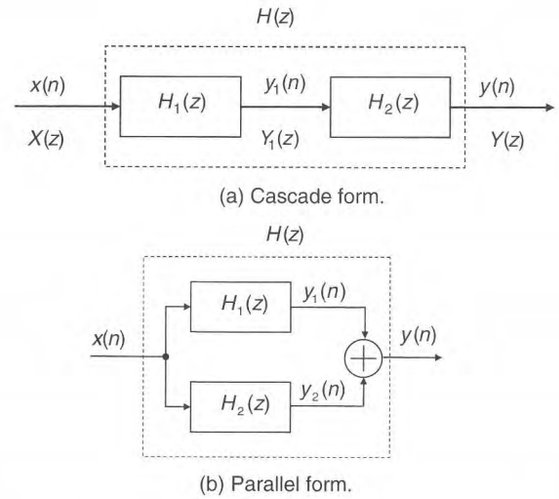

Figure 2.10 Interconnect of digital systems

Since multiplication is commutative, these two systems can be cascaded in either order to obtain the same overall system. Realization of IIR filters using a cascade structure for practical applications will be discussed in Chapter 4.

Similarly, the overall transfer function of the parallel connection of two LTI systems shown in Figure 2.10(b) is given by

$$H(z) = H_1(z) + H_2(z). \tag{2.34}$$

In practical IIR filtering applications, we factor polynomials to break down a high-order filter $H(z)$ into small sections, such as second-order or first-order filters, and use the cascade form. The concepts of parallel and cascade realizations of IIR filters will be further discussed in Chapter 4.

Example 2.8

The LTI system with transfer function

$$H(z) = \frac{1}{1 - 2z^{-1} + z^{-3}}$$

can be factored as

$$H(z) = \left(\frac{1}{1 - z^{-1}}\right)\left(\frac{1}{1 - z^{-1} - z^{-2}}\right) = H_1(z)H_2(z).$$

Thus, the overall system $H(z)$ can be realized as the cascade of the first-order system $H_1(z) = 1/(1 - z^{-1})$ and the second-order system $H_2(z) = 1/(1 - z^{-1} - z^{-2})$.

The I/O equation of an FIR filter is given in (2.14). Taking the z-transform of both sides using the delay property given in (2.28), we have

$$\begin{aligned} Y(z) &= b_0X(z) + b_1z^{-1}X(z) + \cdots + b_{L-1}z^{-(L-1)}X(z) \\ &= \left(b_0 + b_1z^{-1} + \cdots + b_{L-1}z^{-(L-1)}\right)X(z). \end{aligned} \tag{2.35}$$

Therefore, the transfer function of the FIR filter is expressed as

$$H(z) = b_0 + b_1z^{-1} + \cdots + b_{L-1}z^{-(L-1)} = \sum_{l=0}^{L-1} b_lz^{-l}. \tag{2.36}$$

Similarly, taking the z-transform of both sides of the IIR filter defined in (2.22) yields

$$\begin{aligned} Y(z) &= b_0X(z) + b_1z^{-1}X(z) + \cdots + b_{L-1}z^{-L+1}X(z) - a_1z^{-1}Y(z) - \cdots - a_Mz^{-M}Y(z) \\ &= \left(\sum_{l=0}^{L-1} b_lz^{-l}\right)X(z) - \left(\sum_{m=1}^{M} a_mz^{-m}\right)Y(z). \end{aligned} \tag{2.37}$$

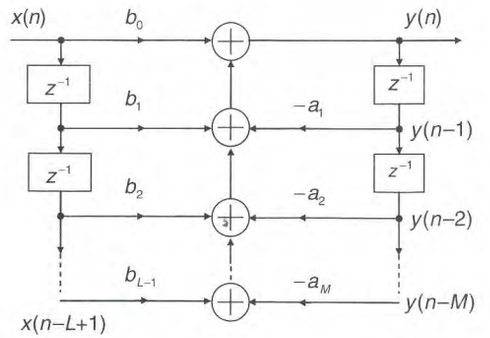

Figure 2.11 Detailed signal-flow diagram of IIR filter

By rearranging these terms, we can derive the transfer function of the IIR filter as

$$H(z) = \frac{\sum_{l=0}^{L-1} b_lz^{-l}}{1 + \sum_{m=1}^{M} a_mz^{-m}} = \frac{\sum_{l=0}^{L-1} b_lz^{-l}}{\sum_{m=0}^{M} a_mz^{-m}}, \tag{2.38}$$

where $a_0 = 1$. A detailed signal-flow diagram of the IIR filter is illustrated in Figure 2.11 for $M = L - 1$.

Example 2.9

Consider the moving-average filter given in (2.16). Taking the z-transform of both sides, we have

$$Y(z) = \frac{1}{L}\sum_{l=0}^{L-1} z^{-l}X(z).$$

Using the geometric series defined in Appendix A, the transfer function of the filter can be expressed as

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{L}\sum_{l=0}^{L-1} z^{-l} = \frac{1}{L}\left[\frac{1 - z^{-L}}{1 - z^{-1}}\right]. \tag{2.39}$$

This equation can be rearranged as

$$Y(z) = z^{-1}Y(z) + \frac{1}{L}\left[X(z) - z^{-L}X(z)\right].$$

Taking the inverse z-transform of both sides, we obtain

$$y(n) = y(n-1) + \frac{1}{L}[x(n) - x(n-L)]. \tag{2.40}$$

This is a formal way of deriving (2.17) from (2.16).

2.2.4 *Poles and Zeros*

Factoring the numerator and denominator polynomials of the rational function $H(z)$, (2.38) can be expressed as

$$H(z) = b_0 \frac{\prod_{l=1}^{L-1}(z - z_l)}{\prod_{m=1}^{M}(z - p_m)} = \frac{b_0(z - z_1)(z - z_2)\ldots(z - z_{L-1})}{(z - p_1)(z - p_2)\ldots(z - p_M)}. \tag{2.41}$$

The roots of the numerator polynomial are the zeros of the transfer function $H(z)$ since they are the values of z for which $H(z) = 0$. Thus, $H(z)$ given in (2.41) has $(L-1)$ zeros at $z = z_1, z_2, \ldots, z_{L-1}$. The roots of the denominator polynomial are the poles since they are the values of z such that $H(z) = \infty$, and there are M poles at $z = p_1, p_2, \ldots, p_M$. The LTI system defined in (2.41) is a pole–zero system of order M, while the system described in (2.36) is an all-zero system of order $L-1$.

Example 2.10

The roots of the numerator polynomial defined in (2.39) determine the zeros of $H(z)$, that is, $z^L - 1 = 0$. Using the complex arithmetic given in Appendix A, we have

$$z_l = e^{j(2\pi/L)l}, \quad l = 0, 1, \ldots, L-1. \tag{2.42}$$

Therefore, there are L zeros equally spaced on the unit circle $|z| = 1$.

Similarly, the poles of $H(z)$ are determined by the roots of the denominator $z^{L-1}(z-1)$. Thus, there are $L-1$ poles at the origin $z = 0$ and one pole at $z = 1$. A pole–zero diagram of $H(z)$ for $L = 8$ on the complex z-plane is illustrated in Figure 2.12. Note that the pole at $z = 1$ is canceled by the zero at $z = 1$. Therefore, the moving-average filter defined by (2.39) is an all-zero (or FIR) filter.

The pole–zero diagram provides important insight into the properties of LTI systems. To find the poles and zeros of a rational function $H(z)$, we can use the MATLAB® function `roots` on both the numerator and denominator polynomials. Another useful MATLAB® function for analyzing transfer functions is `zplane(b,a)`, which displays the pole–zero diagram of $H(z)$.

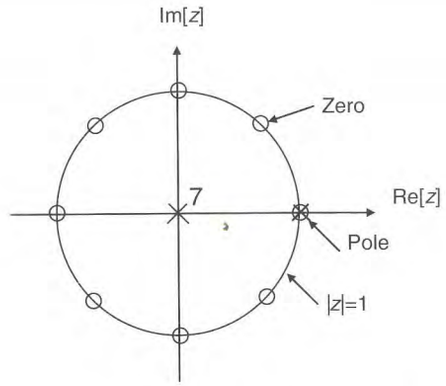

Figure 2.12 Pole–zero diagram of the moving-averaging filter, $L = 8$

Example 2.11

Consider the IIR filter with the transfer function

$$H(z) = \frac{1}{1 - z^{-1} + 0.9z^{-2}}.$$

We can plot the pole–zero diagram (Figure 2.13) using the following MATLAB® script (`example2_11a.m`):

```
b=[1];              % Define numerator
a=[1, -1, 0.9];     % Define denominator
zplane(b,a);        % Pole-zero plot
```

Similarly, we can plot the pole–zero diagram of a moving-average filter using the following MATLAB® script (`example2_11b`) for $L = 8$:

```
b=[1, 0, 0, 0, 0, 0, 0, 0, -1];
a=[1,  -1];
zplane(b,a);
```

Example 2.12

Consider the recursive approximation of the moving-average filter given in (2.23). Taking the z-transform of both sides and rearranging terms, we obtain the transfer function

$$H(z) = \frac{\alpha}{1 - (1-\alpha)z^{-1}}. \tag{2.43}$$

This is the simple first-order IIR filter with a zero at the origin and a pole at $z = 1 - \alpha$. Note that $\alpha = 1/L$ (where L is equivalent to the length of the window) results in

$1-\alpha=(L-1)/L$, which is slightly less than one. Thus for a long window, L is large, the value of $1-\alpha$ is close to one, and the pole is close to the unit circle.

An LTI system $H(z)$ is stable if and only if the poles

$$|p_m|<1, \quad \text{for all } m. \tag{2.44}$$

That is, all poles are inside the unit circle. In this case, $\lim_{n\to\infty}h(n)=0$, that is, the impulse response will converge to zero. A system is unstable if $H(z)$ has any pole outside the unit circle or any multiple-order pole on the unit circle. For example, if $H(z)=z/(z-1)^2$, $h(n)=n$, this system is unstable. A system is marginally stable, or oscillatory bounded, if $H(z)$ has a first-order pole that lies on the unit circle and the rest of the poles are inside the unit circle. For example, if $H(z)=z/(z+1)$, $h(n)=(-1)^n$, $n\geq 0$, this system is marginally stable.

Example 2.13

Consider the LTI system with transfer function

$$H(z)=\frac{z}{z-a}.$$

There is a zero at the origin $z=0$ and a pole at $z=a$. From Example 2.7, we have

$$h(n)=a^n, \quad n\geq 0. \tag{2.45}$$

When $|a|>1$, the pole at $z=a$ is outside the unit circle. Also, from (2.45) we have $\lim_{n\to\infty}h(n)\to\infty$, thus the system is unstable. However, when $|a|<1$, the pole is inside the unit circle and we have $\lim_{n\to\infty}h(n)\to 0$, which is a stable system.

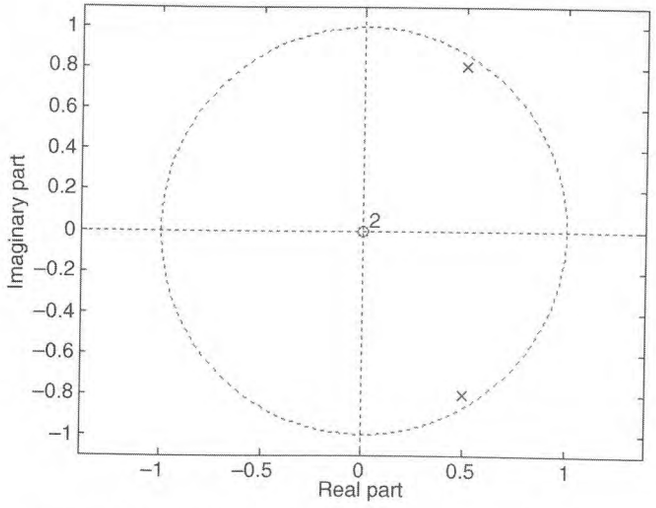

Figure 2.13 A pole–zero diagram generated by MATLAB®

2.2.5 *Frequency Responses*

The frequency response of a digital system $H(\omega)$ can be obtained from its transfer function $H(z)$ by setting $z=e^{j\omega}$. This is equivalent to computing the discrete-time Fourier transform (to be discussed in Chapter 5) of the infinite length impulse response $h(n)$ expressed as

$$H(\omega)=H(z)|_{z=e^{j\omega}}=\sum_{n=-\infty}^{\infty}h(n)e^{-j\omega n}. \tag{2.46}$$

Therefore, the frequency response $H(\omega)$ is obtained by evaluating the transfer function on the unit circle $|z|=|e^{j\omega}|=1$. As listed in Table 2.1, the digital frequency ω defined in (2.7) is in the range of $-\pi\leq\omega\leq\pi$.

The characteristics of the systems can be analyzed using the frequency response. In general, $H(\omega)$ is a complex-valued function expressed in polar form as

$$H(\omega)=\mathrm{Re}[H(\omega)]+j\,\mathrm{Im}[H(\omega)]=|H(\omega)|\,e^{j\phi(\omega)}, \tag{2.47}$$

where

$$|H(\omega)|=\sqrt{\{\mathrm{Re}[H(\omega)]\}^2+\{\mathrm{Im}[H(\omega)]\}^2} \tag{2.48}$$

is the magnitude (or amplitude) response and

$$\phi(\omega)=\begin{cases}\tan^{-1}\left\{\dfrac{\mathrm{Im}[H(\omega)]}{\mathrm{Re}[H(\omega)]}\right\}, & \text{if } \mathrm{Re}[H(\omega)]\geq 0\\ \pi+\tan^{-1}\left\{\dfrac{\mathrm{Im}[H(\omega)]}{\mathrm{Re}[H(\omega)]}\right\}, & \text{if } \mathrm{Re}[H(\omega)]<0\end{cases} \tag{2.49}$$

is the phase response (or angle) of the system $H(z)$. The magnitude response $|H(\omega)|$ is an even function of ω, that is, $|H(-\omega)|=|H(\omega)|$; and the phase response $\phi(\omega)$ is an odd function, that is, $\phi(-\omega)=-\phi(\omega)$. Thus, we only need to evaluate these functions in the frequency region $0\leq\omega\leq\pi$ since $|H(\omega)|$ is symmetric (or a mirror image) about $\omega=0$. Note that $|H(\omega_0)|$ is the gain and $\phi(\omega_0)$is the phase shift of the system at frequency ω_0.

Example 2.14

The two-point moving-average filter can be expressed as

$$y(n)=\frac{1}{2}[x(n)+x(n-1)], \quad n\geq 0.$$

Taking the z-transform of both sides and rearranging the terms, we obtain

$$H(z)=\frac{1}{2}(1+z^{-1}).$$

This is the simple first-order FIR filter. From (2.46), we have

$$H(\omega) = \frac{1}{2}\left(1 + e^{-j\omega}\right) = \frac{1}{2}(1 + \cos\omega - j\sin\omega),$$

$$|H(\omega)| = \sqrt{\{\mathrm{Re}[H(\omega)]\}^2 + \{\mathrm{Im}[H(\omega)]\}^2} = \sqrt{\frac{1}{2}(1 + \cos\omega)},$$

$$\phi(\omega) = \tan^{-1}\left\{\frac{\mathrm{Im}[H(\omega)]}{\mathrm{Re}[H(\omega)]}\right\} = \tan^{-1}\left(\frac{-\sin\omega}{1 + \cos\omega}\right).$$

From Appendix A,

$$\sin\omega = 2\sin\left(\frac{\omega}{2}\right)\cos\left(\frac{\omega}{2}\right) \quad \text{and} \quad \cos\omega = 2\cos^2\left(\frac{\omega}{2}\right) - 1.$$

Therefore, the phase response is

$$\phi(\omega) = \tan^{-1}\left[-\tan\left(\frac{\omega}{2}\right)\right] = -\frac{\omega}{2}.$$

For the transfer function $H(z)$ expressed in (2.38) where the numerator and denominator coefficients are given in the vectors `b` and `a`, respectively, the frequency response can be analyzed using the MATLAB® function

```
[H,w]=freqz(b,a)
```

which returns the complex frequency response vector `H` and the frequency vector `w`. Note that the function `freqz(b,a)` will plot both the magnitude response and the phase response of $H(z)$.

Example 2.15

Consider the IIR filter defined as

$$y(n) = x(n) + y(n-1) - 0.9y(n-2).$$

The transfer function is

$$H(z) = \frac{1}{1 - z^{-1} + 0.9z^{-2}}.$$

The MATLAB® script (`example2_15a.m`) for analyzing the magnitude and phase responses of this IIR filter is listed as follows:

```
b=[1]; a=[1, -1, 0.9];    % Define numerator and denominator
freqz(b,a);               % Plot magnitude and phase responses
```


Similarly, we can plot the magnitude and phase responses (shown in Figure 2.14) of the moving-average filter for $L = 8$ using the following script (`example2_15b.m`):

```
b=[1, 0, 0, 0, 0, 0, 0, 0, -1];  a=[1, -1];
freqz(b,a);
```

A useful method for obtaining the brief frequency response of LTI systems is based on the geometric evaluation of the poles and zeros. For example, consider the second-order IIR filter expressed as

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}, \tag{2.50}$$

where the filter coefficients are real valued. The roots of the characteristic equation

$$z^2 + a_1 z + a_2 = 0 \tag{2.51}$$

are the two poles of the filter, which may be either both real or complex-conjugate poles. Complex-conjugate poles can be expressed as

$$p_1 = r\,e^{j\theta} \quad \text{and} \quad p_2 = r\,e^{-j\theta}, \tag{2.52}$$

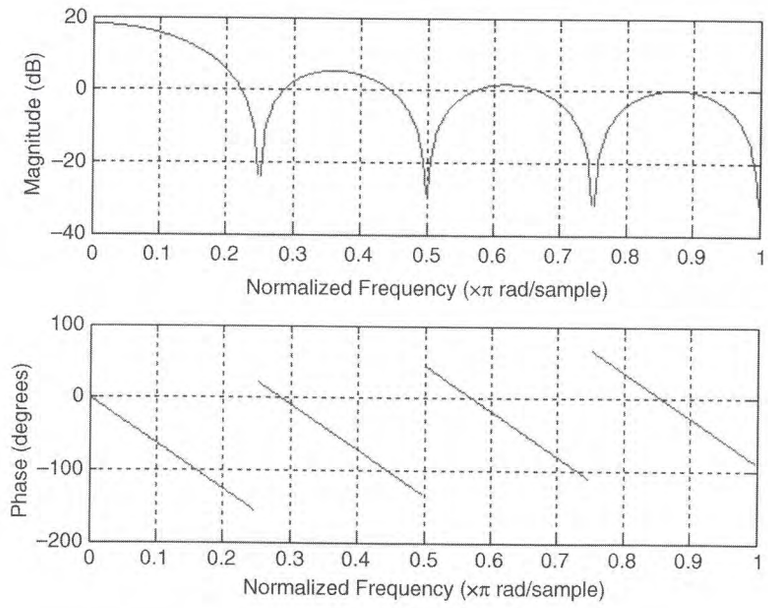

Figure 2.14 Magnitude (top) and phase responses of a moving-average filter, $L = 8$

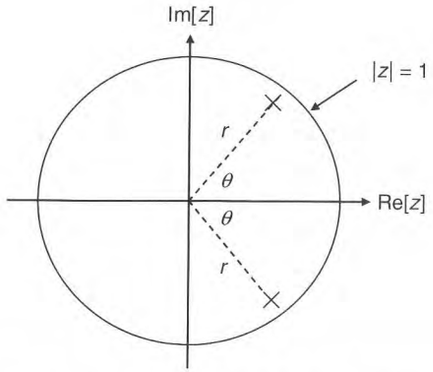

Figure 2.15 A second-order IIR filter with complex-conjugate poles

where r is the radius of the pole and θ is the angle of the pole. Therefore, (2.51) becomes

$$\left(z - r\,e^{j\theta}\right)\left(z - r\,e^{-j\theta}\right) = z^2 - 2r\cos(\theta)z + r^2 = 0. \tag{2.53}$$

Comparing this equation to (2.52), we obtain

$$r = \sqrt{a_2} \quad \text{and} \quad \theta = \cos^{-1}\left(\frac{-a_1}{2r}\right). \tag{2.54}$$

The system with the pair of complex-conjugate poles given in (2.52) is illustrated in Figure 2.15. This filter behaves as a digital resonator for r close to unity. The digital resonator is a bandpass filter with its passband centered at the resonant frequency θ. This issue will be further discussed in Chapter 4.

Similarly, we can obtain two zeros, z_1 and z_2, by evaluating $b_0z^2 + b_1z + b_2 = 0$. Thus, the transfer function defined in (2.50) can be expressed as

$$H(z) = \frac{b_0(z - z_1)(z - z_2)}{(z - p_1)(z - p_2)}. \tag{2.55}$$

In this case, the frequency response is given by

$$H(\omega) = \frac{b_0\left(e^{j\omega} - z_1\right)\left(e^{j\omega} - z_2\right)}{\left(e^{j\omega} - p_1\right)\left(e^{j\omega} - p_2\right)}. \tag{2.56}$$

The magnitude response can be obtained by evaluating $|H(\omega)|$ as the point z moves on the unit circle in a counterclockwise direction from $z = 1$ ($\theta = 0$) to $z = -1$ ($\theta = \pi$). As the point z moves closer to the pole p_1, the magnitude response increases. The closer r is to unity, the sharper and higher the peak. On the other hand, as the point z moves closer to the zero z_1, the magnitude response decreases. The magnitude response exhibits a peak at the pole angle (or frequency), whereas the magnitude response falls to a valley at an angle of zero. For example,

Figure 2.12 shows four zeros at $\theta = \pi/4$, $\pi/2$, $3\pi/4$, and π corresponding to four valleys shown in Figure 2.14.

2.2.6 *Discrete Fourier Transform*

To perform frequency analysis of $x(n)$, we can convert the time-domain signal into the frequency domain using the z-transform defined in (2.26), and substitute $z = e^{j\omega}$ in $X(z)$ as shown in (2.46). However, $X(\omega)$ is a continuous function of continuous frequency ω, and it also requires an infinite number of $x(n)$ samples for calculation. Therefore, it is difficult to compute $X(\omega)$ using digital hardware.

The discrete Fourier transform (DFT) of N-point signals $\{x(0), x(1), x(2), \ldots, x(N-1)\}$ can be obtained by sampling $X(\omega)$ on the unit circle at N equally spaced frequencies $\omega_k = 2\pi k/N$, $k = 0, 1, \ldots, N-1$. From (2.46), we have

$$X(k) = X(\omega)|_{\omega = 2\pi k/N} = \sum_{n=0}^{N-1} x(n)e^{-j(2\pi k/N)n}, \quad k = 0, 1, \ldots, N-1, \tag{2.57}$$

where n is the time index, k is the frequency index, and $X(k)$ is the kth DFT coefficient. The computation of the DFT can be manipulated to obtain very efficient algorithms called fast Fourier transforms (FFTs). The derivation, implementation, and application of DFTs and FFTs will be further discussed in Chapter 5.

MATLAB® provides the function `fft(x)` to compute the DFT of the signal vector `x`. The function `fft(x,N)` performs N-point DFT. If the length of `x` is less than `N`, then `x` is padded with zeros at the end to create an N-point sequence. If the length of `x` is greater than `N`, the function `fft(x,N)` truncates the sequence `x` and performs the DFT of the first `N` samples only.

The DFT generates N coefficients $X(k)$ equally spaced over the frequency range from 0 to 2π. Therefore, the frequency resolution of the N-point DFT is

$$\Delta_\omega = \frac{2\pi}{N} \tag{2.58}$$

or

$$\Delta_f = \frac{f_s}{N}. \tag{2.59}$$

The analog frequency f_k (in Hz) corresponding to the index k can be expressed as

$$f_k = k\Delta_f = \frac{kf_s}{N}, \quad k = 0, 1, \ldots, N-1. \tag{2.60}$$

Note that the Nyquist frequency $(f_s/2)$ corresponds to the frequency index $k = N/2$. Since the magnitude $|X(k)|$ is an even function of k, we only need to display the magnitude spectrum for $0 \le k \le N/2$ (or $0 \le \omega_k \le \pi$).

Example 2.16

Similar to Example 2.1, we can generate 100 sine wave samples with $A = 1, f = 1000$ Hz, and a sampling rate of 10 000 Hz. The magnitude spectrum of the signal can be computed and plotted (as in Figure 2.16) using the following MATLAB® script (`example2_16.m`):

```
N=100; f=1000; fs=10000;          % Define parameter values
n=[0:N-1]; k=[0:N-1];             % Define time and frequency indices
omega=2*pi*f/fs;                  % Frequency of sine wave
xn=sin(omega*n);                  % Generate sine wave
Xk=fft(xn,N);                     % Perform DFT
magXk=20*log10(abs(Xk));          % Compute magnitude spectrum
plot(k, magXk);                   % Plot magnitude spectrum
axis([0, N/2, -inf, inf]);        % Plot from 0 to pi
xlabel('Frequency index, k');
ylabel('Magnitude in dB');
```

From (2.59), the frequency resolution is 100 Hz. The peak of the spectrum shown in Figure 2.16 is located at the frequency index $k = 10$, which corresponds to 1000 Hz as indicated by (2.60). The magnitude spectrum is usually represented by a dB (decibel) scale, which is computed using `20*log10(abs(Xk))`, where the function `abs` calculates absolute value.

2.3 Introduction to Random Variables

The signals encountered in practice are often random signals such as speech, music, and ambient noise. In this section, we will give a brief introduction to the basic concepts of random variables that are useful for understanding quantization effects presented in this book [8].

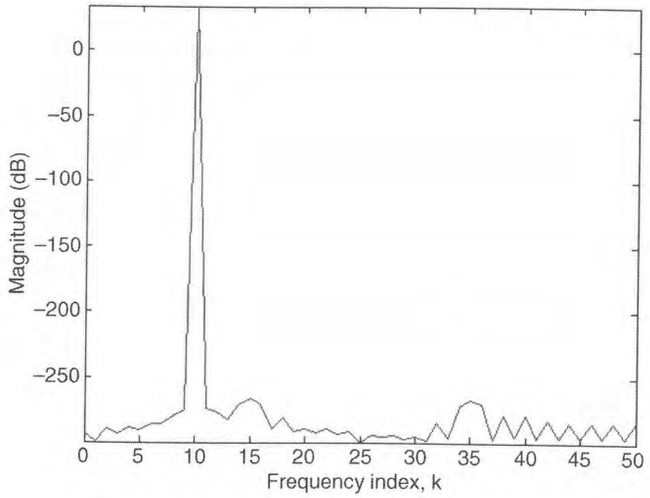

Figure 2.16 Magnitude spectrum of sine wave

Additional principles of random variables will be further discussed in Chapter 6 for introducing adaptive filters.

2.3.1 *Review of Random Variables*

An experiment that has at least two possible outcomes is fundamental to the concept of probability. The set of all possible outcomes in any given experiment is called the sample space S. A random variable, x, is defined as a function that maps all elements from the sample space S into points on the real line. For example, considering the outcomes of rolling of a fair die, we obtain a discrete random variable that can be any one of the discrete values from 1 through 6.

The cumulative probability distribution function of a random variable x is defined as

$$F(X) = P(x \leq X), \tag{2.61}$$

where X is a real number, and $P(x \leq X)$ is the probability of $\{x \leq X\}$. The probability density function of a random variable x is defined as

$$f(X) = \frac{\mathrm{d}F(X)}{\mathrm{d}X} \tag{2.62}$$

if the derivative exists. Two important properties of the probability density function $f(X)$ are summarized as follows:

$$\int_{-\infty}^{\infty} f(X)\,\mathrm{d}X = 1 \tag{2.63}$$

$$P(X_1 < x \leq X_2) = F(X_2) - F(X_1) = \int_{X_1}^{X_2} f(X)\,\mathrm{d}X. \tag{2.64}$$

If x is a discrete random variable that can be any one of the discrete values $X_i, i = 1, 2, \ldots,$ as the result of an experiment, we define the discrete probability function as

$$p_i = P(x = X_i). \tag{2.65}$$

Example 2.17

Consider a random variable x that has the following probability density function:

$$f(X) = \begin{cases} 0, & x < X_1 \quad \text{or} \quad x > X_2 \\ a, & X_1 \leq x \leq X_2, \end{cases}$$

which is uniformly distributed between X_1 and X_2. The constant value a can be computed using (2.63) as

$$\int_{-\infty}^{\infty} f(X)\mathrm{d}X = \int_{X_1}^{X_2} a \cdot \mathrm{d}X = a[X_2 - X_1] = 1.$$

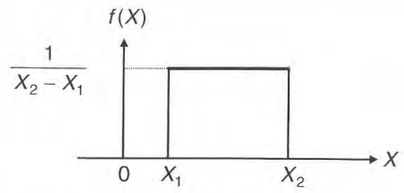

Figure 2.17 The uniform density function

Thus, we obtain

$$a = \frac{1}{X_2 - X_1}.$$

If a random variable x is equally likely to be any value between two limits X_1 and X_2, and cannot assume any value outside that range, it is uniformly distributed in the range $[X_1, X_2]$. As shown in Figure 2.17, the uniform density function is defined as

$$f(X) = \begin{cases} \dfrac{1}{X_2 - X_1}, & X_1 \le x \le X_2 \\ 0, & \text{otherwise.} \end{cases} \tag{2.66}$$

2.3.2 *Operations of Random Variables*

The statistics associated with random variables provide more meaningful information than the probability density function. The mean (expected value) of a random variable x is defined as

$$\begin{aligned} m_x = E[x] &= \int_{-\infty}^{\infty} Xf(X)\mathrm{d}X, \quad \text{continuous-time case} \\ &= \textstyle\sum_i X_i p_i, \quad \text{discrete-time case,} \end{aligned} \tag{2.67}$$

where $E[\cdot]$ denotes the expectation operation (or ensemble averaging). The mean m_x defines the level about which the random process x fluctuates.

The expectation is a linear operation. Two useful properties of the expectation operation are $E[\alpha] = \alpha$ and $E[\alpha x] = \alpha E[x]$, where α is a constant. If $E[x] = 0$, x is the zero-mean random variable. The MATLAB® function `mean` calculates the mean value. For example, the statement `mx = mean(x)` computes the mean `mx` of all the elements in the vector `x`.

Example 2.18

Consider the rolling of a fair die N times ($N \to \infty$), the probability of outcomes is listed as follows:

X_i	1	2	3	4	5	6
p_i	1/6	1/6	1/6	1/6	1/6	1/6

The mean of the outcomes can be computed as

$$m_x = \sum_{i=1}^{6} p_i X_i = \frac{1}{6}(1+2+3+4+5+6) = 3.5.$$

The variance is a measure of the spread about the mean, and is defined as

$$\begin{aligned} \sigma_x^2 &= E\left[(x - m_x)^2\right] \\ &= \int_{-\infty}^{\infty} (X - m_x)^2 f(X)\mathrm{d}X, \quad \text{continuous-time case} \\ &= \sum_i p_i (X_i - m_x)^2, \quad \text{discrete-time case,} \end{aligned} \tag{2.68}$$

where $(x - m_x)$ is the deviation of x from the mean value m_x. The positive square root of the variance is called the standard deviation σ_x. The MATLAB® function `std` calculates the standard deviation of the elements in the vector.

The variance defined in (2.68) can be expressed as

$$\begin{aligned} \sigma_x^2 &= E\left[(x - m_x)^2\right] = E\left[x^2 - 2xm_x + m_x^2\right] = E[x^2] - 2m_x E[x] + m_x^2 \\ &= E[x^2] - m_x^2. \end{aligned} \tag{2.69}$$

We call $E[x^2]$ the mean-square value of x. Thus, the variance is the difference between the mean-square value and the square of the mean value.

If the mean value is equal to zero, then the variance is equal to the mean-square value. For the zero-mean random variable x, that is, $m_x = 0$, we have

$$\sigma_x^2 = E\left[x^2\right] = P_x, \tag{2.70}$$

which is the power of x.

Consider the uniform density function defined in (2.66). The mean of the function can be computed by

$$\begin{aligned} m_x = E[x] &= \int_{-\infty}^{\infty} Xf(X)\mathrm{d}X = \frac{1}{X_2 - X_1}\int_{X_1}^{X_2} X\,\mathrm{d}X \\ &= \frac{X_2 - X_1}{2}. \end{aligned} \tag{2.71}$$

The variance of the function is

$$\begin{aligned} \sigma_x^2 &= E[x^2] - m_x^2 = \int_{-\infty}^{\infty} X^2 f(X)\mathrm{d}X - m_x^2 \\ &= \frac{1}{X_2 - X_1}\int_{X_1}^{X_2} X^2\,\mathrm{d}X - m_x^2 = \frac{1}{X_2 - X_1}\left(\frac{X_2^3 - X_1^3}{3}\right) - m_x^2 \\ &= \frac{(X_2 - X_1)^2}{12}. \end{aligned} \tag{2.72}$$

In general, if x is the random variable uniformly distributed in the interval $(-\Delta, \Delta)$, we have

$$m_x = 0 \quad \text{and} \quad \sigma_x^2 = \Delta^2/3. \tag{2.73}$$

Example 2.19

The MATLAB® function `rand` generates pseudo-random numbers uniformly distributed in the interval [0, 1]. From (2.71), the mean of the generated pseudo-random numbers is 0.5. From (2.72), the variance is 1/12.

To generate zero-mean random numbers, we subtract 0.5 from every generated random number. The numbers are now distributed in the interval [−0.5, 0.5]. To make these pseudo-random numbers have unit variance, that is, $\sigma_x^2 = \Delta^2/3 = 1$, the generated numbers must be equally distributed in the interval $[-\sqrt{3}, \sqrt{3}]$. This can be done by multiplying by $2\sqrt{3}$ every generated number from which 0.5 was subtracted.

The following MATLAB® statement can be used to generate the uniformly distributed random numbers with mean 0 and variance 1:

```
xn = 2*sqrt(3)*(rand-0.5);
```

The MATLAB® code of generating zero-mean, unit-variance ($\sigma_x^2 = 1$) white noise is given in `example2_19.m`.

Note that in earlier versions of MATLAB®, the integer seed `sd` is used in the syntax `rand('seed',sd)` to reproduce exactly the same random numbers each time. This syntax is no longer recommended in the newer versions of MATLAB®. In order to reproduce exactly the same (or repeatable) random numbers each time when we restart MATLAB® or rerun the program, we can reset the random number generator to its default startup settings by either using

```
rng('default')
```

or using the integer seed `sd` (a good choice of value is 12 357) as

```
rng(sd)
```

The principles of random number generators will be introduced in Chapter 7.

A sine wave $s(n)$ corrupted by white noise $v(n)$ can be expressed as

$$x(n) = A\sin(\omega n) + v(n). \tag{2.74}$$

When the signal $s(n)$ with power P_s is corrupted by the noise $v(n)$ with power P_v, the signal-to-noise ratio (SNR) in dB is defined as

$$\text{SNR} = 10\log_{10}\left(\frac{P_s}{P_v}\right) \text{dB}. \tag{2.75}$$

From (2.70), the power of the sine wave defined in (2.6) can be computed as

$$P_s = E\left[A^2\sin^2(\omega n)\right] = A^2/2. \tag{2.76}$$

Example 2.20

This example generates the signal $x(n)$ expressed in (2.74), where $v(n)$ is the zero-mean, unit-variance white noise. As shown in (2.76), when the sine wave amplitude $A = \sqrt{2}$, the power is equal to one. From (2.75), the SNR is 0 dB.

We can generate a sine wave corrupted by the zero-mean, unit-variance white noise with SNR = 0 dB using MATLAB® script `example2_20.m`.

Example 2.21

We can compute the N-point DFT of the signal $x(n)$ to obtain $X(k)$. The magnitude spectrum in the decibel scale can be calculated as $20\log_{10}|X(k)|$ for $k = 0, 1, \ldots, N/2$. Using the signal $x(n)$ generated by Example 2.20, the magnitude spectrum is computed and displayed in Figure 2.18 using the MATLAB® script `example2_21.m`. This magnitude spectrum shows that the power of white noise is uniformly distributed at frequency indices $k = 0, \ldots, 128$ (0 to π), while the power of the sine wave is concentrated at the frequency index $k = 26$ (0.2π).

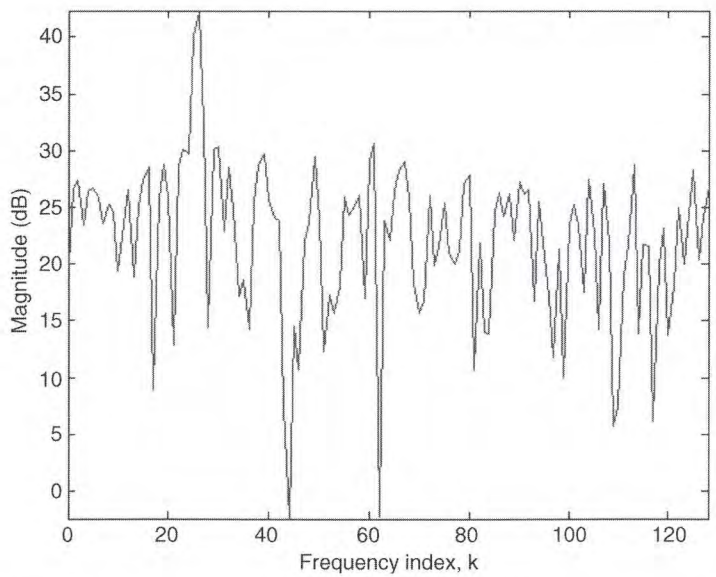

Figure 2.18 Spectrum of sine wave corrupted by white noise, SNR = 0 dB