**Library of Congress Cataloging-in-Publication Data**

Embree, Paul M.
C algorithms for real-time DSP / Paul M. Embree.
p. cm.
Includes bibliographical references.
ISBN 0-13-337353-3
1. C (Computer program language) 2. Computer algorithms. 3. Real-time data processing I. Title.
QA76.73.C15E63 1995
621.382'2'028552—dc20 95-4077
CIP

Acquisitions editor: Karen Gettman
Cover designer: Judith Leeds Design
Manufacturing buyer: Alexis R. Heydt
Compositor/Production services: Pine Tree Composition, Inc.



The publisher offers discounts on this book when ordered in bulk quantities.

For more information contact:
Corporate Sales Department
Prentice Hall PTR
One Lake Street
Upper Saddle River, New Jersey 07458

Phone: 800-382-3419
Fax: 201-236-7141
email: Corpsales@prenhall.com

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN: 0-13-337353-3

Prentice Hall International (UK) Limited, *London*
Prentice Hall of Australia Pty. Limited, *Sydney*
Prentice Hall Canada, Inc., *Toronto*
Prentice Hall Hispanoamericana, S.A., *Mexico*
Prentice Hall of India Private Limited, *New Delhi*
Prentice Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice Hall do Brasil, Ltda., *Rio de Janeiro*

# CONTENTS

**PREFACE** vii

**CHAPTER 1 DIGITAL SIGNAL PROCESSING FUNDAMENTALS** 1

**1.1 SEQUENCES** 2
1.1.1 The Sampling Function 3
1.1.2 Samples Signal Spectra 4
1.1.3 Spectra of Continuous Time and Discrete Time Signals 5

**1.2 LINEAR TIME-INVARIANT OPERATORS** 8
1.2.1 Causality 10
1.2.2 Difference Equations 10
1.2.3 The $z$-Transform Description of Linear Operators 11
1.2.4 Frequency Domain Transfer Function of an Operator 14
1.2.5 Frequency Response from the $z$-Transform Description 15

**1.3 DIGITAL FILTERS** 17
1.3.1 Finite Impulse Response (FIR) Filters 18
1.3.2 Infinite Impulse Response (IIR) Filters 21
1.3.3 Examples of Filter Responses 22
1.3.4 Filter Specifications 23

**1.4 DISCRETE FOURIER TRANSFORMS** 25
1.4.1 Form 25
1.4.2 Properties 26
1.4.3 Power Spectrum 27

1.4.4 Averaged Periodograms 28
1.4.5 The Fast Fourier Transform (FFT) 28
1.4.6 An Example of the FFT 30

**1.5 NONLINEAR OPERATORS 32**
1.5.1 μ-Law and A-Law Compression 33

**1.6 PROBABILITY AND RANDOM PROCESSES 35**
1.6.1 Basic Probability 35
1.6.2 Random Variables 36
1.6.3 Mean, Variance, and Gaussian Random Variables 37
1.6.4 Quantization of Sequences 40
1.6.5 Random Processes, Autocorrelation, and Spectral Density 42
1.6.6 Modeling Real-World Signals with AR Processes 43

**1.7 ADAPTIVE FILTERS AND SYSTEMS 46**
1.7.1 Wiener Filter Theory 48
1.7.2 LMS Algorithms 50

**1.8 REFERENCES 51**

**CHAPTER 2 C PROGRAMMING FUNDAMENTALS 53**

**2.1 THE ELEMENTS OF REAL-TIME DSP PROGRAMMING 53**

**2.2 VARIABLES AND DATA TYPES 56**
2.2.1 Types of Numbers 56
2.2.2 Arrays 58

**2.3 OPERATORS 59**
2.3.1 Assignment Operators 59
2.3.2 Arithmetic and Bitwise Operators 60
2.3.3 Combined Operators 61
2.3.4 Logical Operators 61
2.3.5 Operator Precedence and Type Conversion 62

**2.4 PROGRAM CONTROL 63**
2.4.1 Conditional Execution: `if-else` 63
2.4.2 The `switch` Statement 64
2.4.3 Single-Line Conditional Expressions 65
2.4.4 Loops: `while`, `do-while`, and `for` 66
2.4.5 Program Jumps: `break`, `continue`, and `goto` 67

**2.5 FUNCTIONS 69**
2.5.1 Defining and Declaring Functions 69
2.5.2 Storage Class, Privacy, and Scope 71
2.5.3 Function Prototypes 73

**2.6 MACROS AND THE C PREPROCESSOR 74**
2.6.1 Conditional Preprocessor Directives 74
2.6.2 Aliases and Macros 75



**2.7 POINTERS AND ARRAYS 77**
2.7.1 Special Pointer Operators 77
2.7.2 Pointers and Dynamic Memory Allocation 78
2.7.3 Arrays of Pointers 80

**2.8 STRUCTURES 82**
2.8.1 Declaring and Referencing Structures 82
2.8.2 Pointers to Structures 84
2.8.3 Complex Numbers 85

**2.9 COMMON C PROGRAMMING PITFALLS 87**
2.9.1 Array Indexing 87
2.9.2 Failure to Pass-by-Address 87
2.9.3 Misusing Pointers 88

**2.10 NUMERICAL C EXTENSIONS 90**
2.10.1 Complex Data Types 90
2.10.2 Iteration Operators 91

**2.11 COMMENTS ON PROGRAMMING STYLE 92**
2.11.1 Software Quality 93
2.11.2 Structured Programming 95

**2.12 REFERENCES 97**

**CHAPTER 3 DSP MICROPROCESSORS IN EMBEDDED SYSTEMS 98**

**3.1 TYPICAL FLOATING-POINT DIGITAL SIGNAL PROCESSORS 99**
3.1.1 AT&T DSP32C and DSP3210 100
3.1.2 Analog Devices ADSP-210XX 104
3.1.3 Texas Instruments TMS320C3X and TMS320C40 108

**3.2 TYPICAL PROGRAMMING TOOLS FOR DSP 111**
3.2.1 Basic C Compiler Tools 111
3.2.2 Memory Map and Memory Bandwidth Considerations 113
3.2.3 Assembly Language Simulators and Emulators 114

**3.3 ADVANCED C SOFTWARE TOOLS FOR DSP 117**
3.3.1 Source Level Debuggers 117
3.3.2 Assembly-C Language Interfaces 120
3.3.3 Numeric C Compilers 121

**3.4 REAL-TIME SYSTEM DESIGN CONSIDERATIONS 124**
3.4.1 Physical Input/Output (Memory Mapped, Serial, Polled) 124
3.4.2 Interrupts and Interrupt-Driven I/O 125
3.4.3 Efficiency of Real-Time Compiled Code 128
3.4.4 Multiprocessor Architectures 130

CHAPTER 4 REAL-TIME FILTERING 132

4.1 REAL-TIME FIR AND IIR FILTERS 132
4.1.1 FIR Filter Function 134
4.1.2 FIR Filter Coefficient Calculation 136
4.1.3 IIR Filter Function 145
4.1.4 Real-Time Filtering Example 151
4.2 FILTERING TO REMOVE NOISE 158
4.2.1 Gaussian Noise Generation 158
4.2.2 Signal-to-Noise Ratio Improvement 160
4.3 SAMPLE RATE CONVERSION 160
4.3.1 FIR Interpolation 163
4.3.2 Real-Time Interpolation Followed by Decimation 163
4.3.3 Real-Time Sample Rate Conversion 167
4.4 FAST FILTERING ALGORITHMS 168
4.4.1 Fast Convolution Using FFT Methods 170
4.4.2 Interpolation Using the FFT 176
4.5 OSCILLATORS AND WAVEFORM SYNTHESIS 178
4.5.1 IIR Filters as Oscillators 178
4.5.2 Table-Generated Waveforms 179
4.6 REFERENCES 184

CHAPTER 5 REAL-TIME DSP APPLICATIONS 186

5.1 FFT POWER SPECTRUM ESTIMATION 186
5.1.1 Speech Spectrum Analysis 187
5.1.2 Doppler Radar Processing 190
5.2 PARAMETRIC SPECTRAL ESTIMATION 193
5.2.1 ARMA Modeling of Signals 193
5.2.2 AR Frequency Estimation 198
5.3 SPEECH PROCESSING 200
5.3.1 Speech Compression 201
5.3.2 ADPCM (G.722) 202
5.4 MUSIC PROCESSING 218
5.4.1 Equalization and Noise Removal 218
5.4.2 Pitch-Shifting 220
5.4.3 Music Synthesis 225
5.5 ADAPTIVE FILTER APPLICATIONS 228
5.5.1 LMS Signal Enhancement 228
5.5.2 Frequency Tracking with Noise 233
5.6 REFERENCES 237

APPENDIX—DSP FUNCTION LIBRARY AND PROGRAMS 238

INDEX 241

# PREFACE

Digital signal processing techniques have become the method of choice in signal processing as digital computers have increased in speed, convenience, and availability. As microprocessors have become less expensive and more powerful, the number of DSP applications which have become commonly available has exploded. Thus, some DSP microprocessors can now be considered commodity products. Perhaps the most visible high volume DSP applications are the so called "multimedia" applications in digital audio, speech processing, digital video, and digital communications. In many cases, these applications contain embedded digital signal processors where a host CPU works in a loosely coupled way with one or more DSPs to control the signal flow or DSP algorithm behavior at a real-time rate. Unfortunately, the development of signal processing algorithms for these specialized embedded DSPs is still difficult and often requires specialized training in a particular assembly language for the target DSP.

The tools for developing new DSP algorithms are slowly improving as the need to design new DSP applications more quickly becomes important. The C language is proving itself to be a valuable programming tool for real-time computationally intensive software tasks. C has high-level language capabilities (such as structures, arrays, and functions) as well as low-level assembly language capabilities (such as bit manipulation, direct hardware input/output, and macros) which makes C an ideal language for embedded DSP. Most of the manufacturers of digital signal processing devices (such as Texas Instruments, AT&T, Motorola, and Analog Devices) provide C compilers, simulators, and emulators for their parts. These C compilers offer standard C language with extensions for DSP to allow for very efficient code to be generated. For example, an inline assembly language capability is usually provided in order to optimize the performance of time critical parts of an application. Because the majority of the code is C, an application can be transferred to another processor much more easily than an all assembly language program.

This book is constructed in such a way that it will be most useful to the engineer who is familiar with DSP and the C language, but who is not necessarily an expert in both. All of the example programs in this book have been tested using standard C compil-

ers in the UNIX and MS-DOS programming environments. In addition, the examples have been compiled utilizing the real-time programing tools of specific real-time embedded DSP microprocessors (Analog Devices' ADSP-21020 and ADSP-21062; Texas Instrument's TMS320C30 and TMS320C40; and AT&T's DSP32C) and then tested with real-time hardware using real world signals. All of the example programs presented in the text are provided in source code form on the IBM PC floppy disk included with the book.

The text is divided into several sections. Chapters 1 and 2 cover the basic principles of digital signal processing and C programming. Readers familiar with these topics may wish to skip one or both chapters. Chapter 3 introduces the basic real-time DSP programming techniques and typical programming environments which are used with DSP microprocessors. Chapter 4 covers the basic real-time filtering techniques which are the cornerstone of one-dimensional real-time digital signal processing. Finally, several real-time DSP applications are presented in Chapter 5, including speech compression, music signal processing, radar signal processing, and adaptive signal processing techniques.

The floppy disk included with this text contains C language source code for all of the DSP programs discussed in this book. The floppy disk has a high density format and was written by MS-DOS. The appendix and the READ.ME files on the floppy disk provide more information about how to compile and run the C programs. These programs have been tested using Borland's TURBO C (version 3 and greater) as well as Microsoft C (versions 6 and greater) for the IBM PC. Real-time DSP platforms using the Analog Devices ADSP-21020 and the ADSP-21062, the Texas Instruments TMS320C30, and the AT&T DSP32C have been used extensively to test the real-time performance of the algorithms.

## ACKNOWLEDGMENTS

I thank the following people for their generous help: Laura Mercs for help in preparing the electronic manuscript and the software for the DSP32C; the engineers at Analog Devices (in particular Steve Cox, Marc Hoffman, and Hans Rempel) for their review of the manuscript as well as hardware and software support; Texas Instruments for hardware and software support; Jim Bridges at Communication Automation & Control, Inc., and Talal Itani at Domain Technologies, Inc.

**Paul M. Embree**

## TRADEMARKS



# CHAPTER 1

# DIGITAL SIGNAL PROCESSING FUNDAMENTALS

Digital signal processing begins with a digital signal which appears to the computer as a sequence of digital values. Figure 1.1 shows an example of a digital signal processing operation or simple DSP system. There is an input sequence $x(n)$, the operator $\mathcal{O}\{\ \}$ and an output sequence, $y(n)$. A complete digital signal processing system may consist of many operations on the same sequence as well as operations on the result of operations. Because digital sequences are processed, all operators in DSP are discrete time operators (as opposed to continuous time operators employed by analog systems). Discrete time operators may be classified as *time-varying* or *time-invariant* and *linear* or *nonlinear*. Most of the operators described in this text will be time-invariant with the exception of adaptive filters which are discussed in Section 1.7. Linearity will be discussed in Section 1.2 and several nonlinear operators will be introduced in Section 1.5.

Operators are applied to sequences in order to effect the following results:

**(1)** Extract parameters or features from the sequence.
**(2)** Produce a similar sequence with particular features enhanced or eliminated.
**(3)** Restore the sequence to some earlier state.
**(4)** Encode or compress the sequence.

This chapter is divided into several sections. Section 1.1 deals with *sequences* of numbers: where and how they originate, their spectra, and their relation to continuous signals. Section 1.2 describes the common characteristics of *linear time-invariant operators* which are the most often used in DSP. Section 1.3 discusses the class of operators called *digital filters*. Section 1.4 introduces the *discrete Fourier transform* (DFTs and

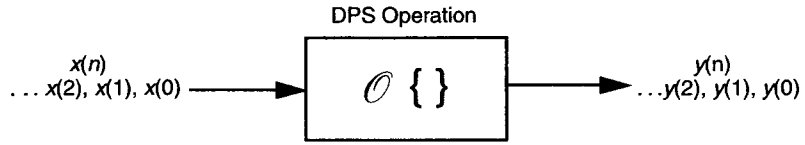


**FIGURE 1.1** DSP operation.

FFTs). Section 1.5 describes the properties of commonly used *nonlinear operators.* Section 1.6 covers basic *probability theory* and *random processes* and discusses their application to signal processing. Finally, Section 1.7 discusses the subject of *adaptive digital filters.*

## 1.1 SEQUENCES

In order for the digital computer to manipulate a signal, the signal must have been sampled at some interval. Figure 1.2 shows an example of a continuous function of time which has been sampled at intervals of $T$ seconds. The resulting set of numbers is called a *sequence.* If the continuous time function was $x(t)$, then the samples would be $x(nT)$ for $n$, an integer extending over some finite range of values. It is common practice to normalize the sample interval to 1 and drop it from the equations. The sequence then becomes $x(n)$. Care must be taken, however, when calculating power or energy from the sequences. The sample interval, including units of time, must be reinserted at the appropriate points in the power or energy calculations.

A sequence as a representation of a continuous time signal has the following important characteristics:

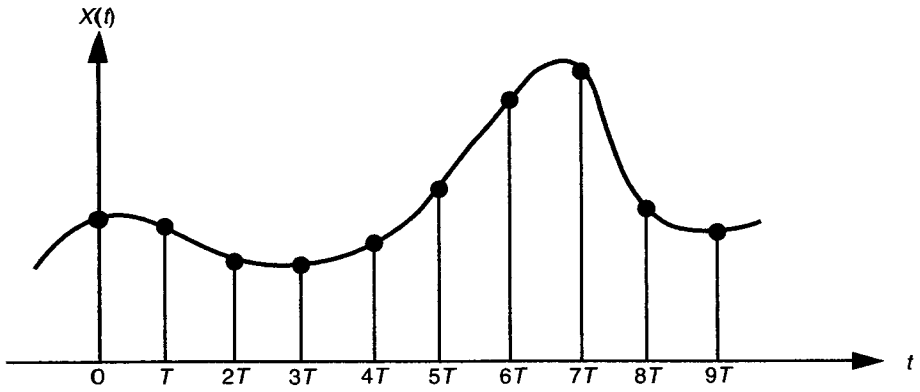


**FIGURE 1.2** Sampling.



(1) The signal is sampled. It has finite value at only discrete points in time.
(2) The signal is truncated outside some finite length representing a finite time interval.
(3) The signal is quantized. It is limited to discrete steps in amplitude, where the step size and, therefore, the accuracy (or *signal fidelity*) depends on how many steps are available in the A/D converter and on the arithmetic precision (number of bits) of the digital signal processor or computer.

In order to understand the nature of the results that DSP operators produce, these characteristics must be taken into account. The effect of sampling will be considered in Section 1.1.1. Truncation will be considered in the section on the discrete Fourier transform (Section 1.4) and quantization will be discussed in Section 1.7.4.

### 1.1.1 The Sampling Function

The *sampling function* is the key to traveling between the continuous time and discrete time worlds. It is called by various names: the *Dirac delta function,* the *sifting function,* the *singularity function,* and the *sampling function* among them. It has the following properties:

$$\text{Property 1.} \int_{-\infty}^{\infty} f(t)\delta(t-\tau)dt = f(\tau). \tag{1.1}$$

$$\text{Property 2.} \int_{-\infty}^{\infty} \delta(t-\tau)dt = 1. \tag{1.2}$$

In the equations above, $\tau$ can be any real number.

To see how this function can be thought of as the ideal sampling function, first consider the realizable sampling function, $\Delta(t)$, illustrated in Figure 1.3. Its pulse width is one unit of time and its amplitude is one unit of amplitude. It clearly exhibits Property 2 of the sampling function. When $\Delta(t)$ is multiplied by the function to be sampled, however, the $\Delta(t)$ sampling function chooses not a single instant in time but a range from $-\frac{1}{2}$ to $+\frac{1}{2}$. As a result, Property 1 of the sampling function is not met. Instead the following integral would result:

$$\int_{-\infty}^{\infty} f(t)\Delta(t-\tau)dt = \int_{\tau-\frac{1}{2}}^{\tau+\frac{1}{2}} f(t)dt. \tag{1.3}$$

This can be thought of as a kind of smearing of the sampling process across a band which is related to the pulse width of $\Delta(t)$. A better approximation to the sampling function would be a function $\Delta(t)$ with a narrower pulse width. As the pulse width is narrowed, however, the amplitude must be increased. In the limit, the ideal sampling function must have infinitely narrow pulse width so that it samples at a single instant in time, and infinitely large amplitude so that the sampled signal still contains the same finite energy.

Figure 1.2 illustrates the sampling process at sample intervals of $T$. The resulting time waveform can be written

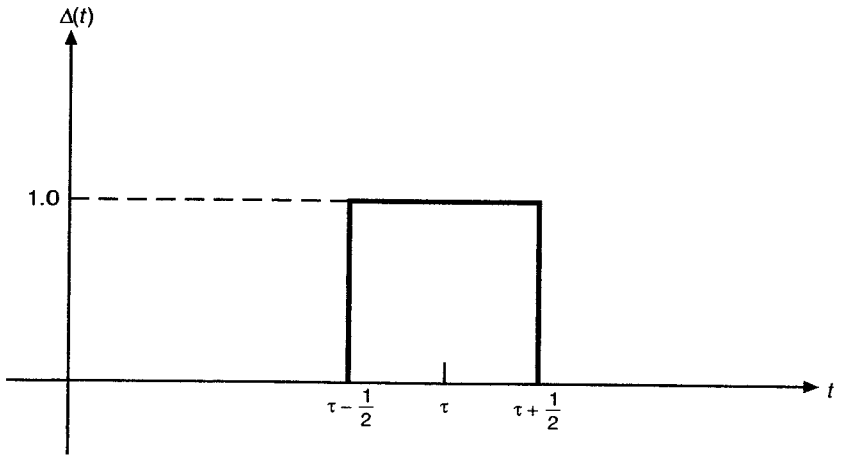


**FIGURE 1.3** Realizable sampling function.

$$x_s(t) = \sum_{n=-\infty}^{\infty} x(t)\delta(t - nT). \tag{1.4}$$

The waveform that results from this process is impossible to visualize due to the infinite amplitude and zero width of the ideal sampling function. It may be easier to picture a somewhat less than ideal sampling function (one with very small width and very large amplitude) multiplying the continuous time waveform.

It should be emphasized that $x_s(t)$ is a continuous time waveform made from the superposition of an infinite set of continuous time signals $x(t)\delta(t - nT)$. It can also be written

$$x_s(t) = \sum_{n=-\infty}^{\infty} x(nT)\delta(t - nT) \tag{1.5}$$

since the sampling function gives a nonzero multiplier only at the values $t = nT$. In this last equation, the sequence $x(nT)$ makes its appearance. This is the set of numbers or *samples* on which almost all DSP is based.

### 1.1.2 Sampled Signal Spectra

Using *Fourier transform theory,* the frequency spectrum of the continuous time waveform $x(t)$ can be written

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}\,dt \tag{1.6}$$



and the time waveform can be expressed in terms of its spectrum as

$$x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi ft}\,df. \tag{1.7}$$

Since this is true for any continuous function of time, $x(t)$, it is also true for $x_s(t)$.

$$X_s(f) = \int_{-\infty}^{\infty} x_s(t)e^{-j2\pi ft}\,dt. \tag{1.8}$$

Replacing $x_s(t)$ by the sampling representation

$$X_s(f) = \int_{-\infty}^{\infty} \left[\sum_{n=-\infty}^{\infty} x(t)\delta(t - nT)\right] e^{-j2\pi ft}\,dt. \tag{1.9}$$

The order of the summation and integration can be interchanged and Property 1 of the sampling function applied to give

$$X_s(f) = \sum_{n=-\infty}^{\infty} x(nT)e^{-j2\pi fnT}. \tag{1.10}$$

This equation is the exact form of a Fourier series representation of $X_s(f)$, a periodic function of frequency having period $1/T$. The coefficients of the Fourier series are $x(nT)$ and they can be calculated from the following integral:

$$x(nT) = T\int_{\frac{-1}{2T}}^{\frac{1}{2T}} X_2(f)e^{j2\pi fnT}\,df. \tag{1.11}$$

The last two equations are a Fourier series pair which allow calculation of either the time signal or frequency spectrum in terms of the opposite member of the pair. Notice that the use of the problematic signal $x_s(t)$ is eliminated and the sequence $x(nT)$ can be used instead.

### 1.1.3 Spectra of Continuous Time and Discrete Time Signals

By evaluating Equation (1.7) at $t = nT$ and setting the result equal to the right-hand side of Equation (1.11) the following relationship between the two spectra is obtained:

$$x(nT) = \int_{-\infty}^{\infty} X(f)e^{j2\pi fnT}\,df = T\int_{\frac{-1}{2T}}^{\frac{1}{2T}} X_s(f)e^{j2\pi fnT}\,df. \tag{1.12}$$

The right-hand side of Equation (1.7) can be expressed as the infinite sum of a set of integrals with finite limits

$$x(nT) = \sum_{m=-\infty}^{\infty} T\int_{\frac{2m-1}{2T}}^{\frac{2m+1}{2T}} X(f)e^{j2\pi fnT}\,df. \tag{1.13}$$

By changing variables to $\lambda = f - m/T$ (substituting $f = \lambda + m/T$ and $df = d\lambda$)

$$x(nT) = \sum_{m=-\infty}^{\infty} \int_{-\frac{1}{2T}}^{\frac{1}{2T}} X(\lambda + \frac{m}{T}) e^{j2\pi\lambda nT} e^{j2\pi\frac{m}{T}nT} d\lambda. \tag{1.14}$$

Moving the summation inside the integral, recognizing that $e^{j2\pi mn}$ (for all integers $m$ and $n$) is equal to 1, and equating everything inside the integral to the similar part of Equation (1.11) give the following relation:

$$X_s(f) = \sum_{m=-\infty}^{\infty} X(f + \frac{m}{T}). \tag{1.15}$$

Equation (1.15) shows that the sampled time frequency spectrum is equal to an infinite sum of shifted replicas of the continuous time frequency spectrum overlaid on each other. The shift of the replicas is equal to the sample frequency, $1/T$. It is interesting to examine the conditions under which the two spectra are equal to each other, at least for a limited range of frequencies. In the case where there are no spectral components of frequency greater than $1/2T$ in the original continuous time waveform, the two spectra

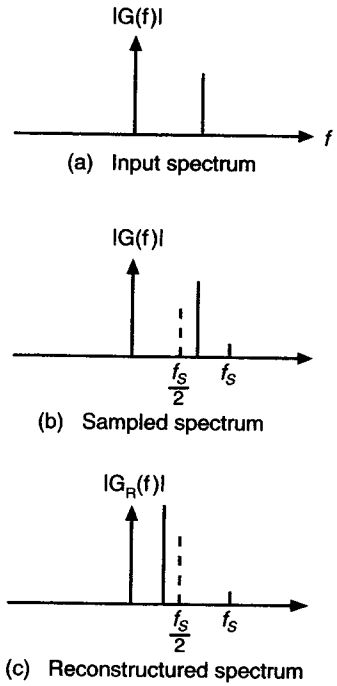


**FIGURE 1.4** Aliasing in the frequency domain. **(a)** Input spectrum. **(b)** Sampled spectrum. **(c)** Reconstructed spectrum.



are equal over the frequency range $f = -1/2T$ to $f = +1/2T$. Of course, the sampled time spectrum will repeat this same set of amplitudes periodically for all frequencies, while the continuous time spectrum is identically zero for all frequencies outside the specified range.

The *Nyquist sampling criterion* is based on the derivation just presented and asserts that a continuous time waveform, when sampled at a frequency greater than twice the maximum frequency component in its spectrum, can be reconstructed completely from the sampled waveform. Conversely, if a continuous time waveform is sampled at a frequency lower than twice its maximum frequency component a phenomenon called *aliasing* occurs. If a continuous time signal is reconstructed from an aliased representation, distortions will be introduced into the result and the degree of distortion is dependent on the degree of aliasing. Figure 1.4 shows the spectra of sampled signals without aliasing and with aliasing. Figure 1.5 shows the reconstructed waveforms of an aliased signal.

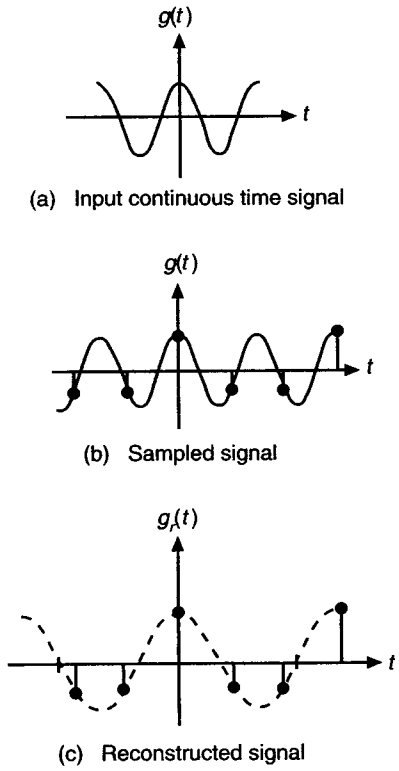


**FIGURE 1.5** Aliasing in the time domain. **(a)** Input continuous time signal. **(b)** Sampled signal. **(c)** Reconstructed signal.

## 1.2 LINEAR TIME-INVARIANT OPERATORS

The most commonly used DSP operators are *linear* and *time-invariant* (or LTI). The linearity property is stated as follows:

Given $x(n)$, a finite sequence, and $\mathcal{O}\{\ \}$, an operator in $n$-space, let

$$y(n) = \mathcal{O}\{x(n)\}. \tag{1.16}$$

If

$$x(n) = ax_1(n) + bx_2(n) \tag{1.17}$$

where $a$ and $b$ are constant with respect to $n$, then, if $\mathcal{O}\{\ \}$ is a linear operator

$$y(n) = a\mathcal{O}\{x_1(n)\} + b\mathcal{O}\{x_2(n)\}. \tag{1.18}$$

The time-invariant property means that if

$$y(n) = \mathcal{O}\{x(n)\}$$

then the shifted version gives the same response or

$$y(n-m) = \mathcal{O}\{x(n-m)\}. \tag{1.19}$$

Another way to state this property is that if $x(n)$ is periodic with period $N$ such that

$$x(n+N) = x(n)$$

then if $\mathcal{O}\{\ \}$ is a time-invariant operator in $n$ space

$$\mathcal{O}\{x(n+N)\} = \mathcal{O}x(n)\}.$$

Next, the LTI properties of the operator $\mathcal{O}\{\ \}$ will be used to derive an expression and method of calculation for $\mathcal{O}\{x(n)\}$. First, the impulse sequence can be used to represent $x(n)$ in a different manner,

$$x(n) = \sum_{m=-\infty}^{\infty} x(m)u_0(n-m). \tag{1.20}$$

This is because

$$u_0(n-m) = \begin{cases} 1, & n = m \\ 0, & \text{otherwise.} \end{cases} \tag{1.21}$$

The impulse sequence acts as a sampling or sifting function on the function $x(m)$, using the dummy variable $m$ to sift through and find the single desired value $x(n)$. Now this somewhat devious representation of $x(n)$ is substituted into the operator Equation (1.16):



$$y(n) = \mathcal{O}\left\{\sum_{m=-\infty}^{\infty} x(m)u_0(n-m)\right\}. \tag{1.22}$$

Recalling that $\mathcal{O}\{\ \}$ operates only on functions of $n$ and using the linearity property

$$y(n) = \sum_{m=-\infty}^{\infty} x(m)\mathcal{O}\{u_0(n-m)\}. \tag{1.23}$$

Every operator has a set of outputs that are its response when an impulse sequence is applied to its input. The impulse response is represented by $h(n)$ so that

$$h(n) = \mathcal{O}\{u_0(n)\}. \tag{1.24}$$

This impulse response is a sequence that has special significance for $\mathcal{O}\{\ \}$, since it is the sequence that occurs at the output of the block labeled $\mathcal{O}\{\ \}$ in Figure 1.1 when an impulse sequence is applied at the input. By time invariance it must be true that

$$h(n-m) = \mathcal{O}\{u_0(n-m)\} \tag{1.25}$$

so that

$$y(n) = \sum_{m=-\infty}^{\infty} x(m)h(n-m). \tag{1.26}$$

Equation (1.26) states that $y(n)$ is equal to the convolution of $x(n)$ with the impulse response $h(n)$. By substituting $m = n - p$ into Equation (1.26) an equivalent form is derived

$$y(n) = \sum_{p=-\infty}^{\infty} h(p)x(n-p). \tag{1.27}$$

It must be remembered that $m$ and $p$ are dummy variables and are used for purposes of the summation only. From the equations just derived it is clear that the impulse response completely characterizes the operator $\mathcal{O}\{\ \}$ and can be used to label the block representing the operator as in Figure 1.6.

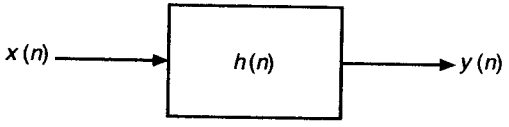


**FIGURE 1.6** Impulse response representation of an operator.

### 1.2.1 Causality

In the mathematical descriptions of sequences and operators thus far, it was assumed that the impulse responses of operators may include values that occur before any applied input stimulus. This is the most general form of the equations and has been suitable for the development of the theory to this point. However, it is clear that no physical system can produce an output in response to an input that has not yet been applied. Since DSP operators and sequences have their basis in physical systems, it is more useful to consider that subset of operators and sequences that can exist in the real world.

The first step in representing realizable sequences is to acknowledge that any sequence must have started at some time. Thus, it is assumed that any element of a sequence in a realizable system whose time index is less than zero has a value of zero. Sequences which start at times later than this can still be represented, since an arbitrary number of their beginning values can also be zero. However, the earliest true value of any sequence must be at a value of $n$ that is greater than or equal to zero. This attribute of sequences and operators is called *causality*, since it allows all attributes of the sequence to be caused by some physical phenomenon. Clearly, a sequence that has already existed for infinite time lacks a cause, as the term is generally defined.

Thus, the convolution relation for causal operators becomes:

$$y(n) = \sum_{m=0}^{\infty} h(m)\,x(n-m). \tag{1.28}$$

This form follows naturally since the impulse response is a sequence and can have no values for $m$ less than zero.

### 1.2.2 Difference Equations

All discrete time, linear, causal, time-invariant operators can be described in theory by the $N$th order difference equation

$$\sum_{m=0}^{N-1} a_m y(n-m) = \sum_{p=0}^{N-1} b_p x(n-p) \tag{1.29}$$

where $x(n)$ is the stimulus for the operator and $y(n)$ is the results or output of the operator. The equation remains completely general if all coefficients are normalized by the value of $a_0$ giving

$$y(n) + \sum_{m=1}^{N-1} a_m y(n-m) = \sum_{p=0}^{N-1} b_p x(n-p) \tag{1.30}$$

and the equivalent form

$$y(n) = \sum_{p=0}^{N-1} b_p x(n-p) - \sum_{m=1}^{N-1} a_m y(n-m) \tag{1.31}$$



or

$$\begin{aligned} y(n) = {} & b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) \ldots \\ & + b_{N-1} x(n-N+1) - a_1 y(n-1) - a_2 y(n-2) \\ & - \ldots - a_{N-1} y(n-N+1). \end{aligned} \tag{1.32}$$

To represent an operator properly may require a very high value of $N$, and for some complex operators $N$ may have to be infinite. In practice, the value of $N$ is kept within limits manageable by a computer; there are often approximations made of a particular operator to make $N$ an acceptable size.

In Equations (1.30) and (1.31) the terms $y(n-m)$ and $x(n-p)$ are shifted or delayed versions of the functions $y(n)$ and $x(n)$, respectively. For instance, Figure 1.7 shows a sequence $x(n)$ and $x(n-3)$, which is the same sequence delayed by three sample periods. Using this delaying property and Equation (1.32), a structure or flow graph can be constructed for the general form of a discrete time LTI operator. This structure is shown in Figure 1.8. Each of the boxes is a delay element with unity gain. The coefficients are shown next to the legs of the flow graph to which they apply. The circles enclosing the summation symbol ($\Sigma$) are adder elements.

### 1.2.3 The $z$-Transform Description of Linear Operators

There is a linear transform—called the *z-transform*—which is as useful to discrete time analysis as the Laplace transform is to continuous time analysis. Its definition is

$$\mathcal{Z}\{x(n)\} = \sum_{n=0}^{\infty} x(n) z^{-n} \tag{1.33}$$

where the symbol $\mathcal{Z}\{\ \}$ stands for "z-transform of," and the $z$ in the equation is a complex number. One of the most important properties of the z-transform is its relationship to time

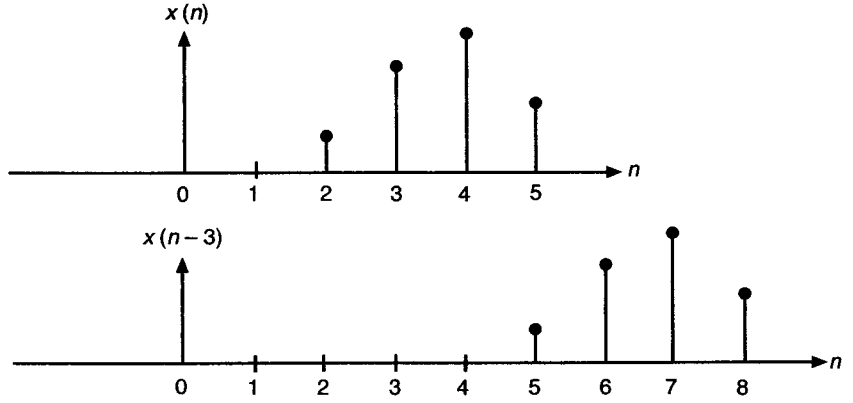


**FIGURE 1.7** Shifting of a sequence.

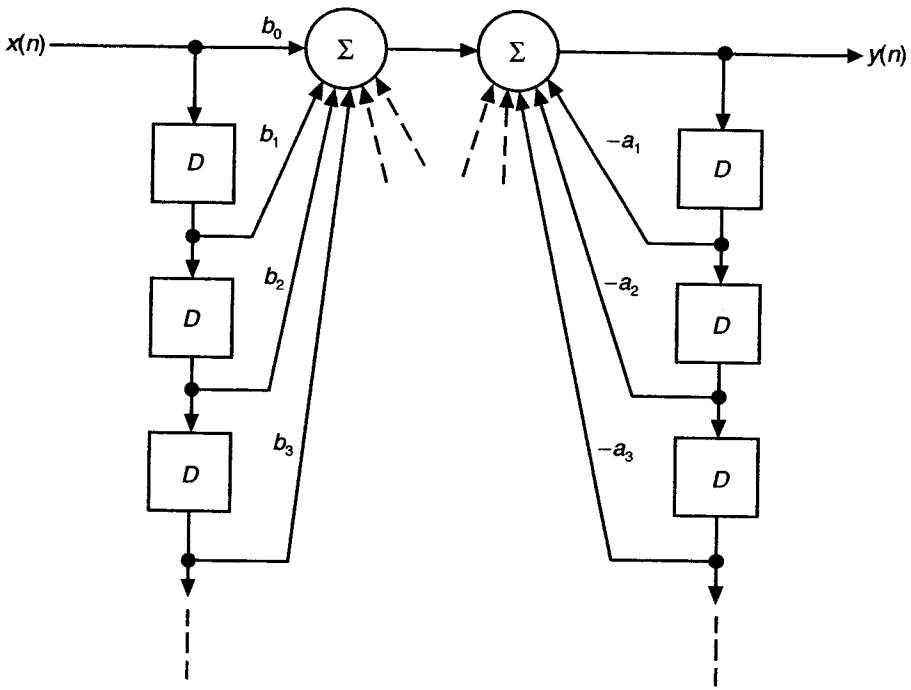


**FIGURE 1.8** Flow graph structure of linear operators.

delay in sequences. To show this property take a sequence, $x(n)$, with a $z$-transform as follows:

$$\mathcal{Z}\{x(n)\} = X(z) = \sum_{n=0}^{\infty} x(n)z^{-n}. \tag{1.34}$$

A shifted version of this sequence has a $z$-transform:

$$\mathcal{Z}\{x(n-p)\} = \sum_{n=0}^{\infty} x(n-p)z^{-n}. \tag{1.35}$$

By letting $m = n - p$ substitution gives:

$$\mathcal{Z}\{x(n-p)\} = \sum_{m=0}^{\infty} x(m)z^{-(m+p)} \tag{1.36}$$



$$= z^{-p} \sum_{m=0}^{\infty} x(m)z^{-m}. \tag{1.37}$$

But comparing the summation in this last equation to Equation (1.33) for the $z$-transform of $x(n)$, it can be seen that

$$\mathcal{Z}\{x(n-p)\} = z^{-p}\mathcal{Z}\{x(n)\} = z^{-p}X(z). \tag{1.38}$$

This property of the $z$-transform can be applied to the general equation for LTI operators as follows:

$$\mathcal{Z}\left\{y(n) + \sum_{p=1}^{\infty} a_p y(n-p)\right\} = z^{-p}\,\mathcal{Z}\left\{\sum_{q=0}^{\infty} b_q x(n-q)\right\}. \tag{1.39}$$

Since the $z$-transform is a linear transform, it possesses the distributive and associative properties. Equation (1.39) can be simplified as follows:

$$\mathcal{Z}\{y(n)\} + \sum_{p=1}^{\infty} a_p \mathcal{Z}\{y(n-p)\} = \sum_{q=0}^{\infty} b_q \mathcal{Z}\{x(n-p)\}. \tag{1.40}$$

Using the shift property of the $z$-transform (Equation (1.38))

$$Y(z) + \sum_{p=1}^{\infty} a_p z^{-p} Y(z) = \sum_{q=0}^{\infty} b_q z^{-q} X(z) \tag{1.41}$$

$$Y(z)\left[1 + \sum_{p=1}^{\infty} a_p z^{-p}\right] = X(z)\left[\sum_{q=0}^{\infty} b_q z^{-q}\right]. \tag{1.42}$$

Finally, Equation (1.42) can be rearranged to give the transfer function in the $z$-transform domain:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{q=0}^{\infty} b_q z^{-q}}{1 + \sum_{p=1}^{\infty} a_p z^{-p}}. \tag{1.43}$$

Using Equation (1.41), Figure 1.8 can be redrawn in the $z$-transform domain and this structure is shown in Figure 1.9. The flow graphs are identical if it is understood that a multiplication by $z^{-1}$ in the transform domain is equivalent to a delay of one sampling time interval in the time domain.

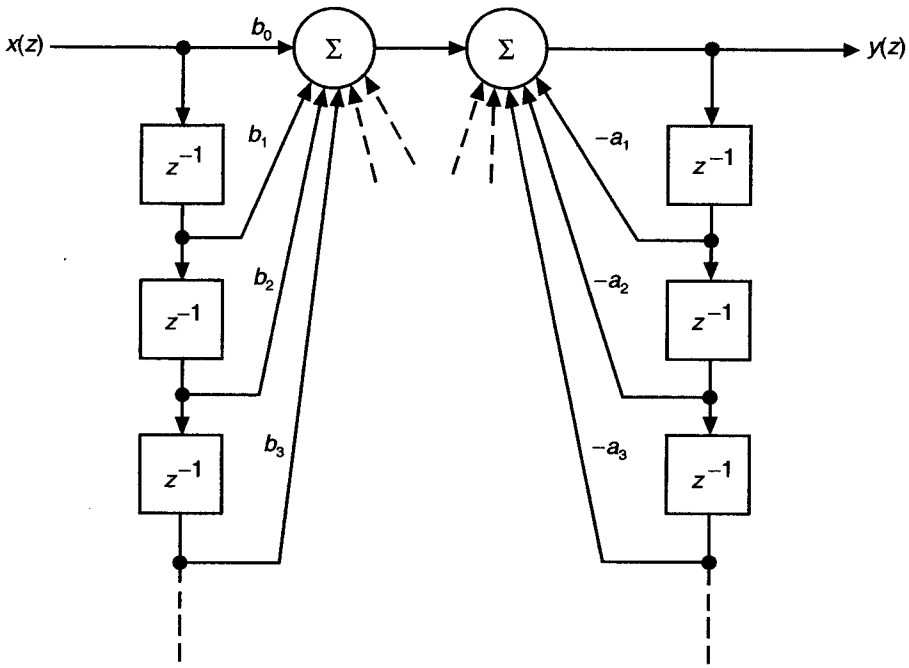


**FIGURE 1.9** Flow graph structure for the $z$-transform of an operator.

### 1.2.4 Frequency Domain Transfer Function of an Operator

Taking the Fourier transform of both sides of Equation (1.28) (which describes any LTI causal operator) results in the following:

$$\mathcal{F}\{y(n)\} = \sum_{m=0}^{\infty} h(m)\mathcal{F}\{x(n-m)\}. \tag{1.44}$$

Using one of the properties of the Fourier transform

$$\mathcal{F}\{x(n-m)\} = e^{-j2\pi fm}\mathcal{F}\{x(n)\}. \tag{1.45}$$

From Equation (1.45) it follows that

$$Y(f) = \sum_{m=0}^{\infty} h(m)e^{-j2\pi fm}X(f), \tag{1.46}$$



or dividing both sides by $X(f)$

$$\frac{Y(f)}{X(f)} = \sum_{m=0}^{\infty} h(m)e^{-j2\pi fm}, \tag{1.47}$$

which is easily recognized as the Fourier transform of the series $h(m)$. Rewriting this equation

$$\frac{Y(f)}{X(f)} = H(f) = \mathcal{F}\{h(m)\}. \tag{1.48}$$

Figure 1.10 shows the time domain block diagram of Equation (1.48) and Figure 1.11 shows the Fourier transform (or frequency domain) block diagram and equation. The frequency domain description of a linear operator is often used to describe the operator. Most often it is shown as an amplitude and a phase angle plot as a function of the variable $f$ (sometimes normalized with respect to the sampling rate, $1/T$).

### 1.2.5 Frequency Response from the $z$-Transform Description

Recall the Fourier transform pair

$$X_s(f) = \sum_{n=-\infty}^{\infty} x(nT)e^{-j2\pi fnT} \tag{1.49}$$

and

$$x(nT) = \int_{-\infty}^{\infty} X_s(f)e^{j2\pi fnT}df. \tag{1.50}$$

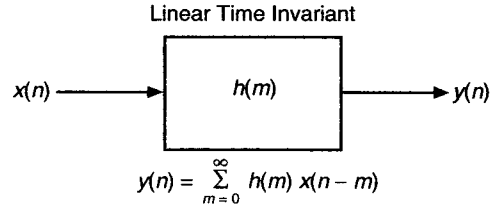


**FIGURE 1.10** Time domain block diagram of LTI system.

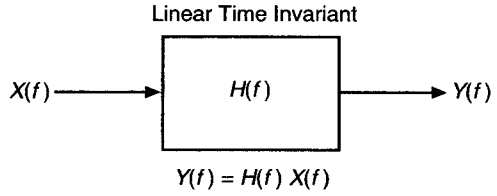


**FIGURE 1.11** Frequency block diagram of LTI system.

In order to simplify the notation, the value of $T$, the period of the sampling waveform, is normalized to be equal to one.

Now compare Equation (1.49) to the equation for the $z$-transform of $x(n)$ as follows:

$$X(z) = \sum_{n=0}^{\infty} x(n) z^{-n}. \tag{1.51}$$

Equations (1.49) and (1.51) are equal for sequences $x(n)$ which are causal (i.e., $x(n) = 0$ for all $n < 0$) if $z$ is set as follows:

$$z = e^{j2\pi f}. \tag{1.52}$$

A plot of the locus of values for $z$ in the complex plane described by Equation (1.52) is shown in Figure 1.12. The plot is a circle of unit radius. Thus, the $z$-transform of a causal sequence, $x(n)$, when evaluated on the unit circle in the complex plane, is equivalent to the frequency domain representation of the sequence. This is one of the properties of the $z$-transform which make it very useful for discrete signal analysis.

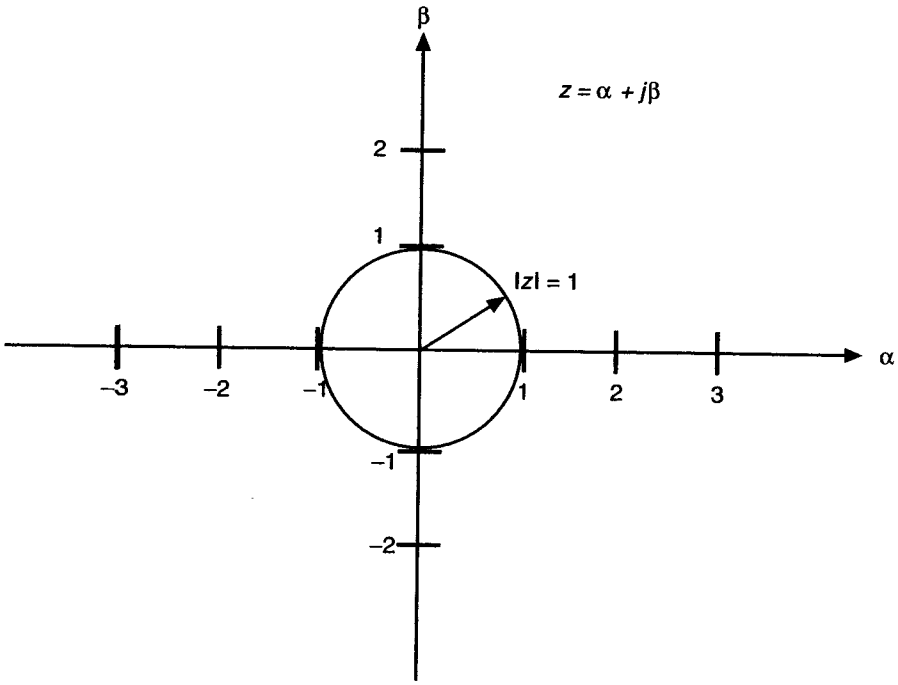


**FIGURE 1.12** The unit circle in the $z$-plane.



Summarizing the last few paragraphs, the impulse response of an operator is simply a sequence, $h(m)$, and the Fourier transform of this sequence is the frequency response of the operator. The $z$-transform of the sequence $h(m)$, called $H(z)$, can be evaluated on the unit circle to yield the frequency domain representation of the sequence. This can be written as follows:

$$H(z)\Big|_{z=e^{j2\pi f}} = H(f). \tag{1.53}$$

## 1.3 DIGITAL FILTERS

The linear operators that have been presented and analyzed in the previous sections can be thought of as *digital filters*. The concept of *filtering* is an analogy between the action of a physical strainer or sifter and the action of a linear operator on sequences when the operator is viewed in the frequency domain. Such a filter might allow certain frequency components of the input to pass unchanged to the output while blocking other components. Naturally, any such action will have its corresponding result in the time domain. This view of linear operators opens a wide area of theoretical analysis and provides increased understanding of the action of digital systems.

There are two broad classes of digital filters. Recall the difference equation for a general operator:

$$y(n) = \sum_{q=0}^{Q-1} b_q x(n-q) - \sum_{p=1}^{P-1} a_p y(n-p). \tag{1.54}$$

Notice that the infinite sums have been replaced with finite sums. This is necessary in order that the filters can be physically realizable.

The first class of digital filters have $a_p$ equal to 0 for all $p$. The common name for filters of this type is *finite impulse response* (FIR) filters, since their response to an impulse dies away in a finite number of samples. These filters are also called *moving average* (or MA) filters, since the output is simply a weighted average of the input values.

$$y(n) = \sum_{q=0}^{Q-1} b_q x(n-q). \tag{1.55}$$

There is a window of these weights ($b_q$) that takes exactly the $Q$ most recent values of $x(n)$ and combines them to produce the output.

The second class of digital filters are *infinite impulse response* (IIR) filters. This class includes both *autoregressive* (AR) filters and the most general form, autoregressive moving average (*ARMA*) filters. In the AR case all $b_q$ for $q = 1$ to $Q - 1$ are set to 0.

$$y(n) = x(n) - \sum_{p=1}^{P-1} a_p y(n-p) \tag{1.56}$$

For ARMA filters, the more general Equation (1.54) applies. In either type of IIR filter, a single-impulse response at the input can continue to provide output of infinite duration with a given set of coefficients. Stability can be a problem for IIR filters, since with poorly chosen coefficients, the output can grow without bound for some inputs.

### 1.3.1 Finite Impulse Response (FIR) Filters

Restating the general equation for FIR filters

$$y(n) = \sum_{q=0}^{Q-1} b_q x(n-q). \tag{1.57}$$

Comparing this equation with the convolution relation for linear operators

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

one can see that the coefficients in an FIR filter are identical to the elements in the impulse response sequence if this impulse response is finite in length.

$$b_q = h(q) \qquad \text{for } q = 0,1,2,3,\ldots,Q-1.$$

This means that if one is given the impulse response sequence for a linear operator with a finite impulse response one can immediately write down the FIR filter coefficients. However, as was mentioned at the start of this section, filter theory looks at linear operators primarily from the frequency domain point of view. Therefore, one is most often given the desired frequency domain response and asked to determine the FIR filter coefficients.

There are a number of methods for determining the coefficients for FIR filters given the frequency domain response. The two most popular FIR filter design methods are listed and described briefly below.

**1.** *Use of the DFT on the sampled frequency response.* In this method the required frequency response of the filter is sampled at a frequency interval of $1/T$ where $T$ is the time between samples in the DSP system. The inverse discrete Fourier transform (see section 1.4) is then applied to this sampled response to produce the impulse response of the filter. Best results are usually achieved if a smoothing window is applied to the frequency response before the inverse DFT is performed. A simple method to obtain FIR filter coefficients based on the Kaiser window is described in section 4.1.2 in chapter 4.

**2.** *Optimal mini-max approximation using linear programming techniques.* There is a well-known program written by Parks and McClellan (1973) that uses the REMEZ exchange algorithm to produce an optimal set of FIR filter coefficients, given the required frequency response of the filter. The Parks-McClellan program is available on the IEEE digital signal processing tape or as part of many of the filter design packages available for personal computers. The program is also printed in several DSP texts (see Elliot 1987 or



Rabiner and Gold 1975). The program REMEZ.C is a C language implementation of the Parks-McClellan program and is included on the enclosed disk. An example of a filter designed using the REMEZ program is shown at the end of section 4.1.2 in chapter 4.

The design of digital filters will not be considered in detail here. Interested readers may wish to consult references listed at the end of this chapter giving complete descriptions of all the popular techniques.

The frequency response of FIR filters can be investigated by using the transfer function developed for a general linear operator:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{q=0}^{Q-1} b_q z^{-q}}{1 + \sum_{p=1}^{P-1} a_p z^{-p}}. \tag{1.58}$$

Notice that the sums have been made finite to make the filter realizable. Since for FIR filters the $a_p$ are all equal to 0, the equation becomes:

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{q=0}^{Q-1} b_q z^{-q}. \tag{1.59}$$

The Fourier transform or frequency response of the transfer function is obtained by letting $z = e^{j2\pi f}$, which gives

$$H(f) = H(z)\big|_{z=e^{j2\pi f}} = \sum_{q=0}^{Q-1} b_q e^{-j2\pi fq}. \tag{1.60}$$

This is a polynomial in powers of $z^{-1}$ or a sum of products of the form

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + \ldots + b_{Q-1} z^{-(Q-1)}.$$

There is an important class of FIR filters for which this polynomial can be factored into a product of sums from

$$H(z) = \prod_{m=0}^{M-1} (z^{-2} + \alpha_m z^{-1} + \beta_m) \prod_{n=0}^{N-1} (z^{-1} + \gamma_n). \tag{1.61}$$

This expression for the transfer function makes explicit the values of the variable $z^{-1}$ which cause $H(z)$ to become zero. These points are simply the roots of the quadratic equation

$$0 = z^{-2} + \alpha_m z^{-1} + \beta_m,$$

which in general provides complex conjugate zero pairs, and the values $\gamma_n$ which provide single zeros.

In many communication and image processing applications it is essential to have filters whose transfer functions exhibit a phase characteristic that changes linearly with a change in frequency. This characteristic is important because it is the phase transfer relationship that gives minimum distortion to a signal passing through the filter. A very useful feature of FIR filters is that for a simple relationship of the coefficients, $b_q$, the resulting filter is guaranteed to have a linear phase response. The derivation of the relationship which provides a linear phase filter follows.

A linear phase relationship to frequency means that

$$H(f) = |H(f)|\, e^{j[\alpha f + \beta]},$$

where $\alpha$ and $\beta$ are constants. If the transfer function of a filter can be separated into a real function of $f$ multiplied by a phase factor $e^{j[\alpha f + \beta]}$, then this transfer function will exhibit linear phase.

Taking the FIR filter transfer function:

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + \ldots + b_{Q-1} z^{-(Q-1)}$$

and replacing $z$ by $e^{j2\pi f}$ to give the frequency response

$$H(f) = b_0 + b_1 e^{-j2\pi f} + b_2 e^{-j2\pi(2f)} + \ldots + b_{Q-1} e^{-j2\pi(Q-1)f}.$$

Factoring out the factor $e^{-j2\pi(Q-1)f/2}$ and letting $\zeta$ equal $(Q-1)/2$ gives

$$H(f) = e^{-j2\pi\zeta f}\left\{b_0 e^{j2\pi\zeta f} + b_1 e^{j2\pi(\zeta-1)f} + b_2 e^{j2\pi(\zeta-2)f} + \ldots + b_{Q-2} e^{-j2\pi(\zeta-1)f} + b_{Q-1} e^{-j2\pi\zeta f}\right\}.$$

Combining the coefficients with complex conjugate phases and placing them together in brackets

$$H(f) = e^{-j2\pi\zeta f}\left\{\left[b_0 e^{j2\pi\zeta} + b_{Q-1} e^{-j2\pi\zeta f}\right] + \left[b_1 e^{j2\pi(\zeta-1)f} + b_{Q-2} e^{-j2\pi(\zeta-1)f}\right] + \left[b_2 e^{j2\pi(\zeta-2)f} + b_{Q-3} e^{-j2\pi(\zeta-2)f}\right] + \ldots\right\}$$

If each pair of coefficients inside the brackets is set equal as follows:

$$b_0 = b_{Q-1}$$
$$b_1 = b_{Q-2}$$
$$b_2 = b_{Q-3}, \text{ etc.}$$

Each term in brackets becomes a cosine function and the linear phase relationship is achieved. This is a common characteristic of FIR filter coefficients.



### 1.3.2 Infinite Impulse Response (IIR) Filters

Repeating the general equation for IIR filters

$$y(n) = \sum_{q=0}^{Q-1} b_q x(n-q) - \sum_{p=1}^{P-1} a_p y(n-p).$$

The $z$-transform of the transfer function of an IIR filter is

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{q=0}^{Q-1} b_q z^{-q}}{1 + \sum_{p=1}^{P-1} a_p z^{-p}}.$$

No simple relationship exists between the coefficients of the IIR filter and the impulse response sequence such as that which exists in the FIR case. Also, obtaining linear phase IIR filters is not a straightforward coefficient relationship as is the case for FIR filters. However, IIR filters have an important advantage over FIR structures: In general, IIR filters require fewer coefficients to approximate a given filter frequency response than do FIR filters. This means that results can be computed faster on a general purpose computer or with less hardware in a special purpose design. In other words, IIR filters are computationally efficient. The disadvantage of the recursive realization is that IIR filters are much more difficult to design and implement. Stability, roundoff noise, and sometimes phase nonlinearity must be considered carefully in all but the most trivial IIR filter designs.

The direct form IIR filter realization shown in Figure 1.9, though simple in appearance, can have severe response sensitivity problems because of coefficient quantization, especially as the order of the filter increases. To reduce these effects, the transfer function is usually decomposed into second order sections and then realized as cascade sections. The C language implementation given in section 4.1.3 uses single precision floating-point numbers in order to avoid coefficient quantization effects associated with fixed-point implementations that can cause instability and significant changes in the transfer function.

IIR digital filters can be designed in many ways, but by far the most common IIR design method is the *bilinear transform.* This method relies on the existence of a known $s$-domain transfer function (or Laplace transform) of the filter to be designed. The $s$-domain filter coefficients are transformed into equivalent $z$-domain coefficients for use in an IIR digital filter. This might seem like a problem, since $s$-domain transfer functions are just as hard to determine as $z$-domain transfer functions. Fortunately, Laplace transform methods and $s$-domain transfer functions were developed many years ago for designing analog filters as well as for modeling mechanical and even biological systems. Thus, many tables of $s$-domain filter coefficients are available for almost any type of filter function (see the references for a few examples). Also, computer programs are available to generate coefficients for many of the common filter types (see the books by Jong, Anoutino, Stearns (1993), Embree (1991), or one of the many filter design packages

available for personal computers). Because of the vast array of available filter tables, the large number of filter types, and because the design and selection of a filter requires careful examination of all the requirements (passband ripple, stopband attenuation as well as phase response in some cases), the subject of $s$-domain IIR filter design will not be covered in this book. However, several IIR filter designs with exact z-domain coefficients are given in the examples in section 4.1 and on the enclosed disk.

### 1.3.3 Examples of Filter Responses

As an example of the frequency response of an FIR filter with very simple coefficients, take the following moving average difference equation:

$$y(n) = 0.11\,x(n) + 0.22\,x(n-1) + 0.34\,x(n-2) + 0.22\,x(n-3) + 0.11\,x(n-4).$$

One would suspect that this filter would be a lowpass type by inspection of the coefficients, since a constant (DC) value at the input will produce that same value at the output. Also, since all coefficients are positive, it will tend to average adjacent values of the signal.

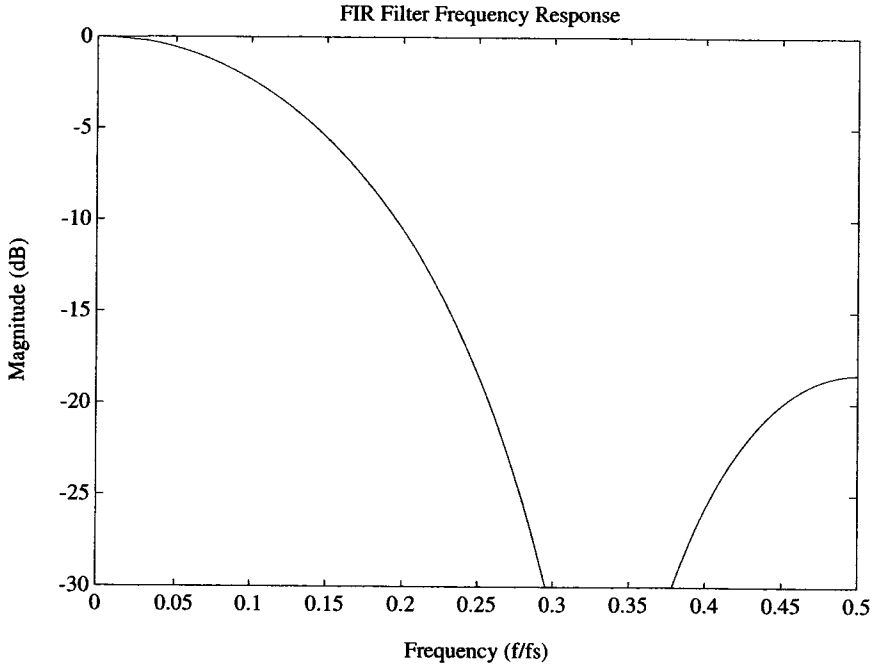


**FIGURE 1.13** FIR low pass response.



The response of this FIR filter is shown in Figure 1.13. It is indeed lowpass and the nulls in the stop band are characteristic of discrete time filters in general.

As an example of the simplest IIR filter, take the following difference equation:

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

Some contemplation of this filter's response to some simple inputs (like constant values, 0, 1, and so on) will lead to the conclusion that it is an integrator. For zero input, the output holds at a constant value forever. For any constant positive input greater than zero, the output grows linearly with time. For any constant negative input, the output decreases linearly with time. The frequency response of this filter is shown in Figure 1.14.

### 1.3.4 Filter Specifications

As mentioned previously, filters are generally specified by their performance in the frequency domain, both amplitude and phase response as a function of frequency. Figure 1.15 shows a lowpass filter magnitude response characteristic. The filter gain has

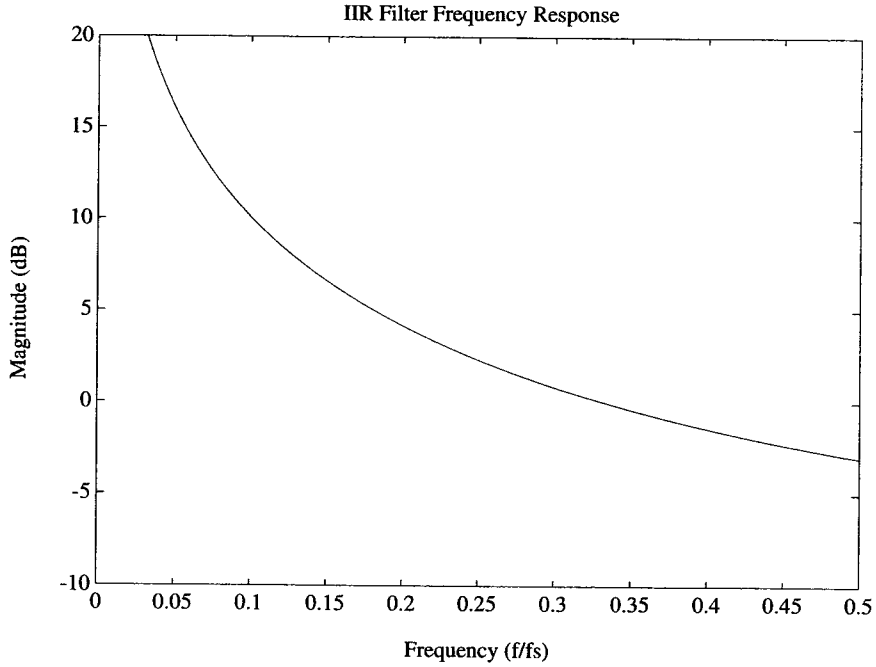


**FIGURE 1.14** IIR integrator response.

been normalized to be roughly 1.0 at low frequencies and the sampling rate is normalized to unity. The figure illustrates the most important terms associated with filter specifications.

The region where the filter allows the input signal to pass to the output with little or no attenuation is called the *passband.* In a lowpass filter, the passband extends from frequency $f = 0$ to the start of the transition band, marked as frequency $f_{pass}$ in Figure 1.15. The *transition band* is that region where the filter smoothly changes from passing the signal to stopping the signal. The end of the transition band occurs at the stopband frequency, $f_{stop}$. The *stopband* is the range of frequencies over which the filter is specified to attenuate the signal by a given factor. Typically, a filter will be specified by the following parameters:

(1) Passband ripple—$2\delta$ in the figure.
(2) Stopband attenuation—$1/\lambda$.
(3) Transition start and stop frequencies—$f_{pass}$ and $f_{stop}$.
(4) Cutoff frequency—$f_{pass}$. The frequency at which the filter gain is some given factor lower than the nominal passband gain. This may be −1 dB, −3 dB or other gain value close to the passband gain.

Computer programs that calculate filter coefficients from frequency domain magnitude response parameters use the above list or some variation as the program input.

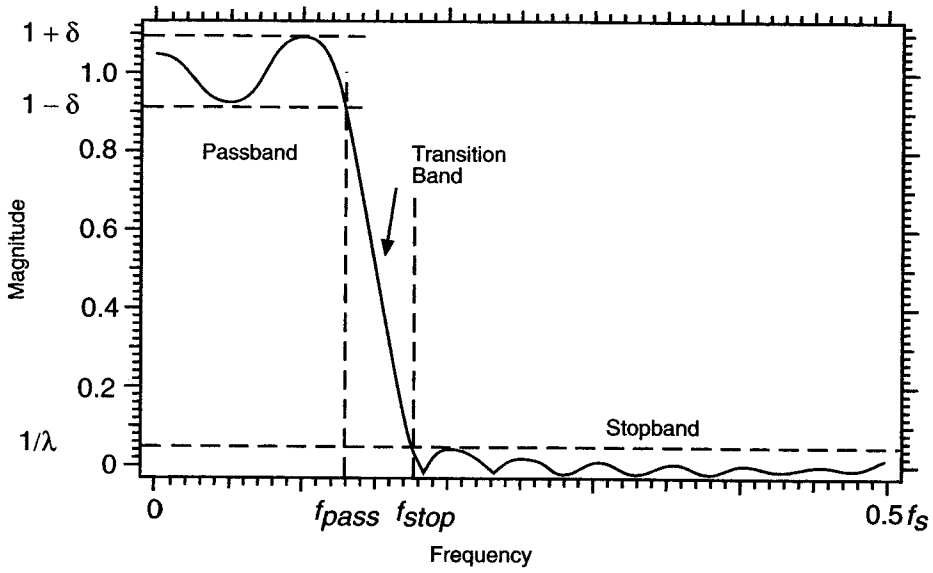


**FIGURE 1.15** Magnitude response of normalized lowpass filter.



## 1.4 DISCRETE FOURIER TRANSFORMS

So far, the *Fourier transform* has been used several times to develop the characteristics of sequences and linear operators. The Fourier transform of a causal sequence is:

$$\mathcal{F}\{x(n)\} = X(f) = \sum_{n=0}^{\infty} x(n)e^{-j2\pi fn} \tag{1.62}$$

where the sample time period has been normalized to 1 ($T = 1$). If the sequence is of limited duration (as must be true to be of use in a computer) then

$$X(f) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi fn} \tag{1.63}$$

where the sampled time domain waveform is $N$ samples long. The inverse Fourier transform is

$$\mathcal{F}^{-1}\{X(f)\} = x(n) = \int_{-1/2}^{1/2} X(f)e^{-j2\pi fn}df \tag{1.64}$$

since $X(f)$ is periodic with period $1/T = 1$, the integral can be taken over any full period. Therefore,

$$x(n) = \int_{0}^{1} X(f)e^{-j2\pi fn}df. \tag{1.65}$$

### 1.4.1 Form

These representations for the Fourier transform are accurate but they have a major drawback for digital applications—the frequency variable is continuous, not discrete. To overcome this problem, both the time and frequency representations of the signal must be approximated.

To create a *discrete Fourier transform* (DFT) a sampled version of the frequency waveform is used. This sampling in the frequency domain is equivalent to convolution in the time domain with the following time waveform:

$$h_1(t) = \sum_{r=-\infty}^{\infty} \delta(t - rT).$$

This creates duplicates of the sampled time domain waveform that repeats with period $T$. This $T$ is equal to the $T$ used above in the time domain sequence. Next, by using the same number of samples in one period of the repeating frequency domain waveform as in one period of the time domain waveform, a DFT pair is obtained that is a good approximation to the continuous variable Fourier transform pair. The forward discrete Fourier transform is

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N} \tag{1.66}$$

and the inverse discrete Fourier transform is

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{-j2\pi kn/N}. \tag{1.67}$$

For a complete development of the DFT by both graphical and theoretical means, see the text by Brigham (chapter 6).

### 1.4.2 Properties

This section describes some of the properties of the DFT. The corresponding paragraph numbers in the book *The Fast Fourier Transform* by Brigham (1974) are indicated. Due to the sampling theorem it is clear that no frequency higher than $1/2T$ can be represented by $X(k)$. However, the values of $k$ extend to $N$–1, which corresponds to a frequency nearly equal to the sampling frequency $1/T$. This means that for a real sequence, the values of $k$ from $N/2$ to $N$–1 are aliased and, in fact, the amplitudes of these values of $X(k)$ are

$$|X(k)| = |X(N-k)|, \text{ for } k = N/2 \text{ to } N-1. \tag{1.68}$$

This corresponds to Properties 8-11 and 8-14 in Brigham.

The DFT is a linear transform as is the $z$-transform so that the following relationships hold:

If

$$x(n) = \alpha\, a(n) + \beta\, b(n),$$

where $\alpha$ and $\beta$ are constants, then

$$X(k) = \alpha\, A(k) + \beta\, B(k),$$

where $A(k)$ and $B(k)$ are the DFTs of the time functions $a(n)$ and $b(n)$, respectively. This corresponds to Property 8-1 in Brigham.

The DFT also displays a similar attribute under time shifting as the $z$-transform. If $X(k)$ is the DFT of $x(n)$ then

$$\text{DFT}\{x(n-p)\} = \sum_{n=0}^{N-1} x(n-p)e^{-j2kn/N}$$

Now define a new variable $m = r - p$ so that $n = m + p$. This gives

$$\text{DFT}\{x(n-p)\} = \sum_{m=-p}^{m=N-1-p} x(m)e^{-j\pi km/N}e^{-j2\pi kp/N},$$

which is equivalent to the following:

$$\text{DFT}\{x(n-p)\} = e^{-j2\pi kp/N}X(k). \tag{1.69}$$



This corresponds to Property 8-5 in Brigham. Remember that for the DFT it is assumed that the sequence $x(m)$ goes on forever repeating its values based on the period $n = 0$ to $N - 1$. So the meaning of the negative time arguments is simply that

$$x(-p) = x(N-p), \text{ for } p = 0 \text{ to } N-1.$$

### 1.4.3 Power Spectrum

The DFT is often used as an analysis tool for determining the spectra of input sequences. Most often the amplitude of a particular frequency component in the input signal is desired. The DFT can be broken into amplitude and phase components as follows:

$$X(f) = X_{\text{real}}(f) + j\, X_{\text{imag}}(f) \tag{1.70}$$

$$X(f) = |X(f)|e^{j\theta(f)} \tag{1.71}$$

$$\text{where } |X(f)| = \sqrt{X_{\text{real}}^2 + X_{\text{imag}}^2}$$

$$\text{and } \theta(f) = \tan^{-1}\left[\frac{X_{\text{imag}}}{X_{\text{real}}}\right].$$

The power spectrum of the signal can be determined using the signal spectrum times its conjugate as follows:

$$X(k)X^*(k) = |X(k)|^2 = X_{\text{real}}^2 + X_{\text{imag}}^2. \tag{1.72}$$

There are some problems with using the DFT as a spectrum analysis tool, however. The problem of interest here concerns the assumption made in deriving the DFT that the sequence was a single period of a periodically repeating waveform. For almost all sequences there will be a discontinuity in the time waveform at the boundaries between these pseudo periods. This discontinuity will result in very high-frequency components in the resulting waveform. Since these components can be much higher than the sampling theorem limit of $1/2T$ (or half the sampling frequency) they may be aliased into the middle of the spectrum developed by the DFT.

The technique used to overcome this difficulty is called *windowing*. The problem to be overcome is the possible discontinuity at the edges of each period of the waveform. Since for a general purpose DFT algorithm there is no way to know the degree of discontinuity at the boundaries, the windowing technique simply reduces the sequence amplitude at the boundaries. It does this in a gradual and smooth manner so that no new discontinuities are produced, and the result is a substantial reduction in the aliased frequency components. This improvement does not come without a cost. Because the window is modifying the sequence before a DFT is performed, some reduction in the fidelity of the spectral representation must be expected. The result is somewhat reduced resolution of closely spaced frequency components. The best windows achieve the maximum reduction of *spurious* (or aliased) signals with the minimum degradation of spectral resolution.

There are a variety of windows, but they all work essentially the same way:

Attenuate the sequence elements near the boundaries (near $n = 0$ and $n = N - 1$) and compensate by increasing the values that are far away from the boundaries. Each window has its own individual transition from the center region to the outer elements. For a comparison of window performance see the references listed at the end of this chapter. (For example, see Harris (1983)).

### 1.4.4 Averaged Periodograms

Because signals are always associated with noise—either due to some physical attribute of the signal generator or external noise picked up by the signal source—the DFT of a single sequence from a continuous time process is often not a good indication of the true spectrum of the signal. The solution to this dilemma is to take multiple DFTs from successive sequences from the same signal source and take the time average of the power spectrum. If a new DFT is taken each $NT$ seconds and successive DFTs are labeled with superscripts:

$$\text{Power Spectrum} = \sum_{i=0}^{M-1}\left[(X^i_{\text{real}})^2 + (X^i_{\text{imag}})^2\right]. \tag{1.73}$$

Clearly, the spectrum of the signal cannot be allowed to change significantly during the interval $t = 0$ to $t = \text{M}$ $(NT)$.

### 1.4.5 The Fast Fourier Transform (FFT)

The *fast Fourier transform* (or FFT) is a very efficient algorithm for computing the DFT of a sequence. It takes advantage of the fact that many computations are repeated in the DFT due to the periodic nature of the discrete Fourier kernel: $e^{-j2\pi kn/N}$. The form of the DFT is

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}. \tag{1.74}$$

By letting $W^{nk} = e^{-j2\pi kn/N}$ Equation (1.74) becomes

$$X(k) = \sum_{n=0}^{N-1} x(n)W^{nk}. \tag{1.75}$$

Now, $W^{(N+qN)(k+rN)} = W^{nk}$ for all $q$, $r$ that are integers due to the periodicity of the Fourier kernel.

Next break the DFT into two parts as follows:

$$X(k) = \sum_{n=0}^{N/2-1} x(2n)W_N^{2nk} + \sum_{n=0}^{N/2-1} x(2n+1)W_N^{(2n+1)k}, \tag{1.76}$$

where the subscript $N$ on the Fourier kernel represents the size of the sequence.



By representing the even elements of the sequence $x(n)$ by $x_{\text{ev}}$ and the odd elements by $x_{\text{od}}$, the equation can be rewritten

$$X(k) = \sum_{n=0}^{N/2-1} x_{ev}(n)W_{N/2}^{nk} + W_{N/2}^{k}\sum_{n=0}^{N/2-1} x_{od}(n)W_{N/2}^{nk}. \tag{1.77}$$

Now there are two expressions in the form of DFTs so Equation (1.77) can be simplified as follows:

$$X(k) = X_{ev}(n) + W_{N/2}^{k}X_{od}(n). \tag{1.78}$$

Notice that only DFTs of $N/2$ points need be calculated to find the value of $X(k)$. Since the index $k$ must go to $N - 1$, however, the periodic property of the even and odd DFTs is used. In other words,

$$X_{ev}(k) = X_{ev}(k - \tfrac{N}{2}) \qquad \text{for } \tfrac{N}{2} \le k \le N-1. \tag{1.79}$$

The process of dividing the resulting DFTs into even and odd halves can be repeated until one is left with only two point DFTs to evaluate

$$\begin{aligned} \Lambda(k) &= \lambda(0) + \lambda(1)e^{-j2\pi k/2} && \text{for all } k \\ &= \lambda(0) + \lambda(1) && \text{for } k \text{ even} \\ &= \lambda(0) - \lambda(1) && \text{for } k \text{ odd.} \end{aligned}$$

Therefore, for 2 point DFTs no multiplication is required, only additions and subtractions. To compute the complete DFT still requires multiplication of the individual 2-point DFTs by appropriate factors of $W$ ranging from $W^0$ to $W^{N/2-1}$. Figure 1.16 shows a flow graph of a complete 32-point FFT. The savings in computation due to the FFT algorithm is as follows.

For the original DFT, $N$ complex multiplications are required for each of $N$ values of $k$. Also, $N - 1$ additions are required for each $k$.

In an FFT each function of the form

$$\lambda(0) \pm W^P\lambda(1)$$

(called a *butterfly* due to its flow graph shape) requires one multiplication and two additions. From the flow graph in Figure 1.16 the number of butterflies is

$$\text{Number of butterflies} = \frac{N}{2}\log_2(N).$$

This is because there are $N/2$ rows of butterflies (since each butterfly has two inputs) and there are $\log_2(N)$ columns of butterflies.

Table 1.1 gives a listing of additions and multiplications for various sizes of FFTs and DFTs. The dramatic savings in time for larger DFTs provided in the FFT has made this method of spectral analysis practical in many cases where a straight DFT computa-

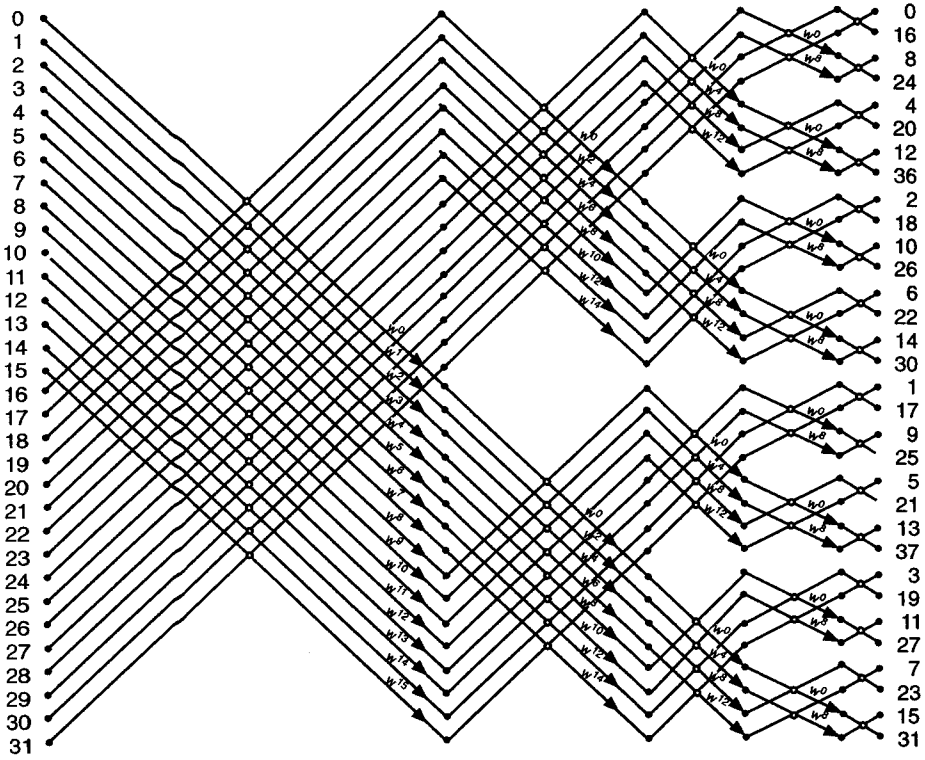


**FIGURE 1.16** 32-Point, radix 2, in-place FFT. (From Rabiner and Gold, 1975, p. 380.)

tion would be much too time consuming. Also, the FFT can be used for performing operations in the frequency domain that would require much more time consuming computations in the time domain.

### 1.4.6 An Example of the FFT

In order to help the reader gain more understanding of spectrum analysis with the FFT, a simple example is presented here. An input signal to a 16-point FFT processor is as follows:

$$x(n) = \cos[2\pi\,(4n/16)].$$

The argument of the cosine has been written in an unusual way to emphasize the frequency of the waveform when processed by a 16-point FFT. The amplitude of this signal is 1.0 and it is clearly a real signal, the imaginary component having zero amplitude. Figure 1.17 shows the 16 samples that comprise $x(0)$ to $x(15)$.



**TABLE 1.1** Comparison of Number of Butterfly Operations in the DFT and FFT, (each operation is one complex multiply/accumulate calculation).

| Transform Length ($N$) | DFT Operations ($N^2$) | FFT Operations $N\text{LOG}_2(N)$ |
|---|---|---|
| 8 | 64 | 24 |
| 16 | 256 | 64 |
| 32 | 1024 | 160 |
| 64 | 4096 | 384 |
| 128 | 16384 | 896 |
| 256 | 65536 | 1024 |
| 512 | 262144 | 4608 |
| 1024 | 1048576 | 10240 |
| 2048 | 4194304 | 22528 |

With this input a 16-point FFT will produce a very simple output. This output is shown in Figure 1.18. It is a spike at $k = 4$ of amplitude 0.5 and a spike at $k = 12$ of amplitude $-0.5$. The spike nature in the FFT output in this example occurs because for a cosine waveform of arbitrary frequency the Fourier transform is

$$X(f) = \int_{-\infty}^{+\infty} \cos(2\pi f_0 t) e^{-j2\pi ft}\, dt.$$

Representing the cosine by exponentials

$$X(f) = \frac{1}{2}\int_{-\infty}^{+\infty} e^{j2\pi(f_0 - f)t}\, dt - \frac{1}{2}\int_{-\infty}^{+\infty} e^{-j2\pi(f_0 + f)t}\, dt.$$

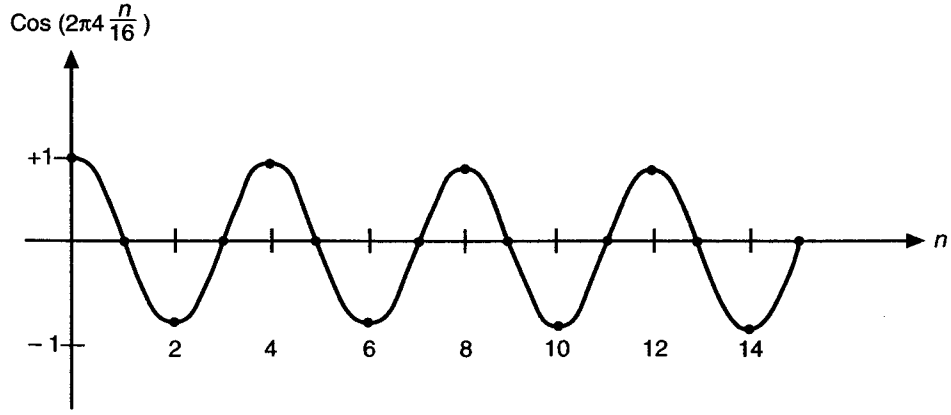


**FIGURE 1.17** Input to 16 point FFT.

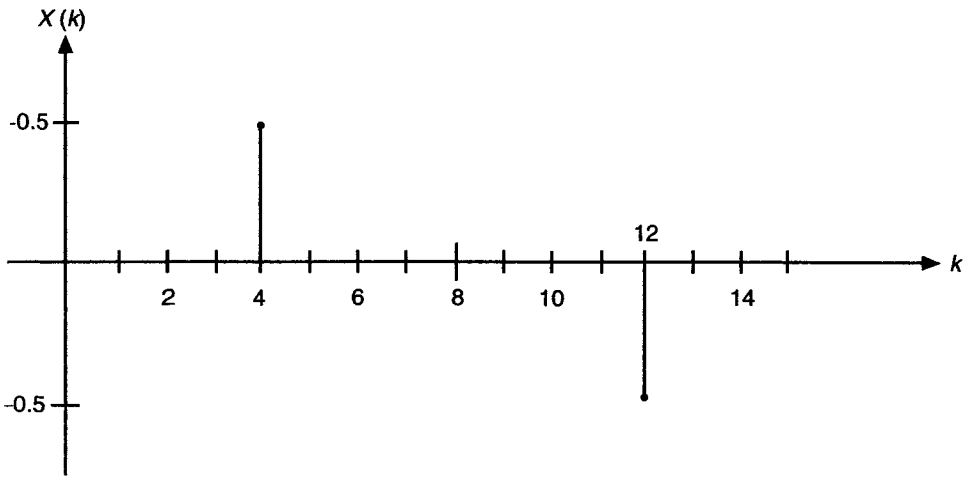


**FIGURE 1.18** Output of 16-point FFT.

It can be shown that the integrand in the two integrals above integrates to 0 unless the argument of the exponential is 0. If the argument of the exponential is zero, the result is two infinite spikes, one at $f = f_0$ and the other at $f = -f_0$. These are delta functions in the frequency domain.

Based on these results, and remembering that the impulse sequence is the digital analog of the delta function, the results for the FFT seem more plausible. It is still left to explain why $k = 12$ should be equivalent to $f = -f_0$. Referring back to the development of the DFT, it was necessary at one point for the frequency spectrum to become periodic with period $f_s$. Also, in the DFT only positive indices are used. Combining these two facts one can obtain the results shown in Figure 1.18.

## 1.5 NONLINEAR OPERATORS

Most of this book is devoted to linear operators and linear-signal processing because these are the most commonly used techniques in DSP. However, there are several nonlinear operators that are very useful in one-dimensional DSP. This section introduces the simple class of nonlinear operators that compress or clip the input to derive the output sequence.

There is often a need to reduce the number of significant bits in a quantized sequence. This is sometimes done by truncation of the least significant bits. This process is advantageous because it is linear: The quantization error is increased uniformly over the entire range of values of the sequence. There are many applications, however, where the need for accuracy in quantization is considerably less at high-signal values than at low-signal values. This is true in telephone voice communications where the human ear's



ability to differentiate between amplitudes of sound waves decreases with the amplitude of the sound. In these cases, a nonlinear function is applied to the signal and the resulting output range of values is quantized uniformly with the available bits.

This process is illustrated in Figure 1.19. First, the input signal is shown in Figure 1.19(a). The accuracy is 12 bits and the range is 0 to 4.095 volts, so each quantization level represents 1 mV. It is necessary because of some system consideration (such as transmission bandwidth) to reduce the number bits in each word to 8. Figure 1.19(b) shows that the resulting quantization levels are 16 times as coarse. Figure 1.19(c) shows the result of applying a linear-logarithmic compression to the input signal. In this type of compression the low-level signals (out to some specified value) are unchanged from the input values. Beginning at a selected level, say $f_{in} = a$, a logarithmic function is applied. The form of the function might be

$$f_{out} = a + A \log_{10}(1 + f_{in} - a)$$

so that at $f_{in} = a$ the output also equals $a$ and $A$ is chosen to place the maximum value of $f_{out}$ at the desired point.

A simpler version of the same process is shown in Figure 1.20. Instead of applying a logarithmic function from the point $f = a$ onward, the output values for $f \geq a$ are all the same. This is an example of clipping. A region of interest is defined and any values outside the region are given a constant output.

### 1.5.1 μ-Law and A-Law Compression

There are two other compression laws worth listing because of their use in telephony—the μ-*law* and *A-law* conversions. The μ-*law* conversion is defined as follows:

$$f_{out} = \text{sgn}(f_{in}) \frac{\ln(1 + \mu|f_{in}|)}{\ln(1 + \mu)}, \tag{1.80}$$

where sgn() is a function that takes the sign of its argument, and μ is the compression parameter (255 for North American telephone transmission). The input value $f_{in}$ must be normalized to lie between −1 and +1. The *A*-law conversion equations are as follows:

$$f_{out} = \text{sgn}(f_{in}) \frac{A|f_{in}|}{1 + \ln(A)}$$

for $|f_{in}|$ between 0 and $1/A$ and

$$f_{out} = \text{sgn}(f_{in}) \frac{1 + \ln(A|f_{in}|)}{1 + \ln(A)} \tag{1.81}$$

for $|f_{in}|$ between $1/A$ and 1.

In these equations, $A$ is the compression parameter (87.6 for European telephone transmission).

An extreme version of clipping is used in some applications of image processing to produce binary pictures. In this technique a threshold is chosen (usually based on a his-

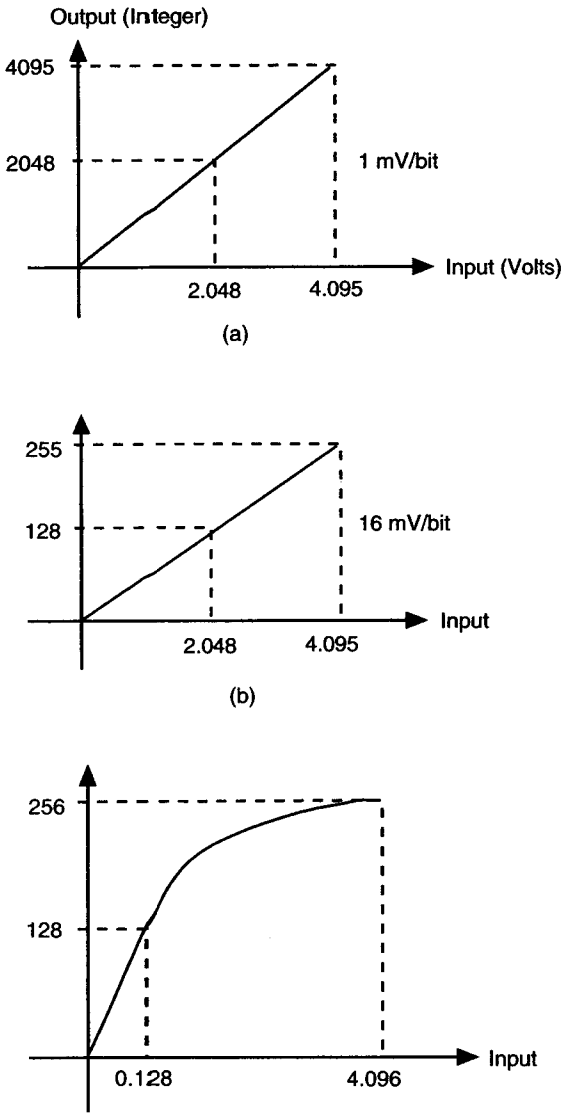


**FIGURE 1.19** **(a)** Linear 12-bit ADC. **(b)** Linear 8-bit ADC. **(c)** Nonlinear conversion.

togram of the picture elements) and any image element with a value higher than threshold is set to 1 and any element with a value lower than threshold is set to zero. In this way the significant bits are reduced to only one. Pictures properly thresholded can produce excellent outlines of the most interesting objects in the image, which simplifies further processing considerably.



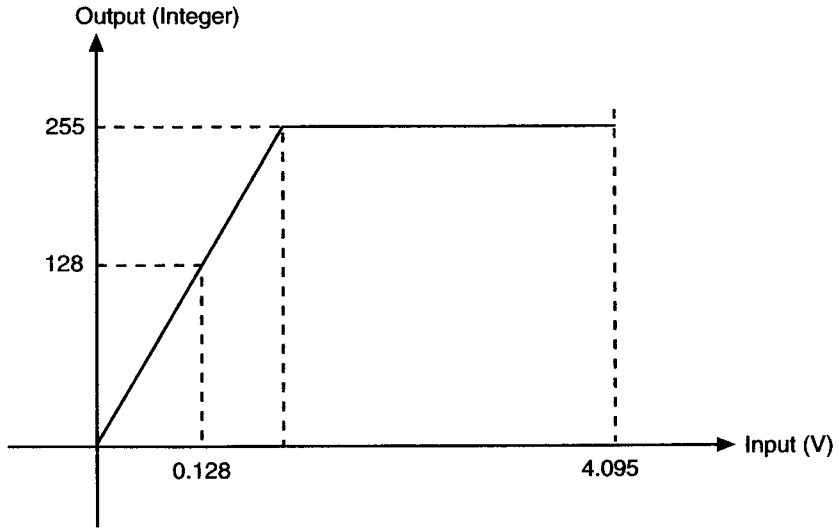


**FIGURE 1.20** Clipping to 8 bits.

## 1.6 PROBABILITY AND RANDOM PROCESSES

The signals of interest in most signal-processing problems are embedded in an environment of noise and interference. The noise may be due to spurious signals picked up during transmission (interference), or due to the noise characteristics of the electronics that receives the signal or a number of other sources. To deal effectively with noise in a signal, some model of the noise or of the signal plus noise must be used. Most often a probabilistic model is used, since the noise is, by nature, unpredictable. This section introduces the concepts of probability and randomness that are basic to digital signal processing and gives some examples of the way a composite signal of interest plus noise is modeled.

### 1.6.1 Basic Probability

Probability begins by defining the probability of an event labeled $A$ as $P(A)$. Event $A$ can be the result of a coin toss, the outcome of a horse race, or any other result of an activity that is not completely predictable. There are three attributes of this probability $P(A)$:

(1) $P(A) >= 0$. This simply means that any result will either have a positive chance of occurrence or no chance of occurrence.

(2) $P$ (All possible outcomes) = 1. This indicates that some result among those possible is bound to occur, a probability of 1 being certainty.

(3) For $\{A_i\}$, where $(A_i \cap A_i) = 0$, $P(\cup A_i) = \Sigma_i P(A_i)$. For a set of events, $\{A_i\}$, where the events are mutually disjoint (no two can occur as the result of a single trial of

the activity), the probability of any one of the events occurring is equal to the sum of their individual probabilities.

With probability defined in this way, the discussion can be extended to joint and conditional probabilities. Joint probability is defined as the probability of occurrence of a specific set of two or more events as the result of a single trial of an activity. For instance, the probability that horse $A$ will finish third and horse $B$ will finish first in a particular horse race is a joint probability. This is written:

$$P(A \cap B) = P(A \text{ and } B) = P(AB). \tag{1.82}$$

Conditional probability is defined as the probability of occurrence of an event $A$ given that $B$ has occurred. The probability assigned to event $A$ is conditioned by some knowledge of event $B$. This is written

$$P(A \text{ given } B) = P(A|B). \tag{1.83}$$

If this conditional probability, $P(A|B)$, and the probability of $B$ are both known, the probability of both of these events occurring (joint probability) is

$$P(AB) = P(A|B)P(B). \tag{1.84}$$

So the conditional probability is multiplied by the probability of the condition (event $B$) to get the joint probability. Another way to write this equation is

$$P(A|B) = \frac{P(AB)}{P(B)} \tag{1.85}$$

This is another way to define conditional probability once joint probability is understood.

### 1.6.2 Random Variables

In signal processing, the probability of a signal taking on a certain value or lying in a certain range of values is often desired. The signal in this case can be thought of as a *random variable* (an element whose set of possible values is the set of outcomes of the activity). For instance, for the random variable $X$, the following set of events, which could occur, may exist:

Event $A$ $X$ takes on the value of 5 ($X = 5$)
Event $B$ $X = 19$
Event $C$ $X = 1.66$
etc.

This is a useful set of events for discrete variables that can only take on certain specified values. A more practical set of events for continuous variables associates each event with



the variable lying within a range of values. A cumulative distribution function (or CDF) for a random variable can be defined as follows:

$$F(x) = P(X \leq x). \tag{1.86}$$

This cumulative distribution, function, then, is a monotonically increasing function of the independent variable $x$ and is valid only for the particular random variable, $X$. Figure 1.21 shows an example of a distribution function for a random variable. If $F(x)$ is differentiated with respect to $x$ the probability density function (or PDF) for $X$ is obtained, represented as follows:

$$p(x) = \frac{dF(x)}{dx}. \tag{1.87}$$

Integrating $p(x)$ gives the distribution function back again as follows:

$$F(x) = \int_{-\infty}^{x} p(\lambda) d\lambda. \tag{1.88}$$

Since $F(x)$ is always monotonically increasing, $p(x)$ must be always positive or zero. Figure 1.22 shows the density function for the distribution of Figure 1.21. The utility of these functions can be illustrated by determining the probability that the random variable $X$ lies between $a$ and $b$. By using probability Property 3 from above

$$P(X \leq b) = P(a < X \leq b) + P(X \leq a). \tag{1.89}$$

This is true because the two conditions on the right-hand side are independent (mutually exclusive) and $X$ must meet one or the other if it meets the condition on the left-hand side. This equation can be expressed using the definition of the distribution:

$$\begin{aligned} P(a < X \leq b) &= F(b) - F(a) \\ &= \int_{a}^{b} p(x) dx. \end{aligned} \tag{1.90}$$

In this way, knowing the distribution or the density function allows the calculation of the probability that $X$ lies within any given range.

### 1.6.3 Mean, Variance, and Gaussian Random Variables

There is an operator in random variable theory called the *expectation operator* usually written $E[x]$. This expression is pronounced "the expected value of $x$." The expectation operator extracts from a random variable the value that the variable is most likely to take

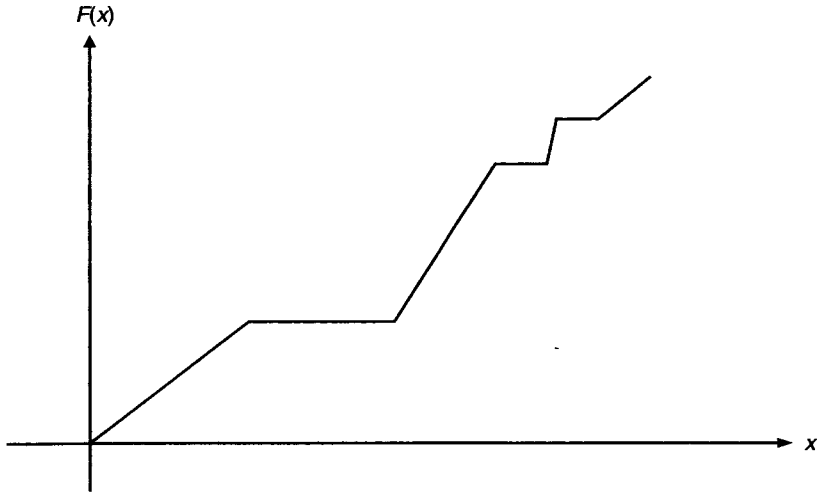


**FIGURE 1.21** An example of cumulative distribution function (CDF).

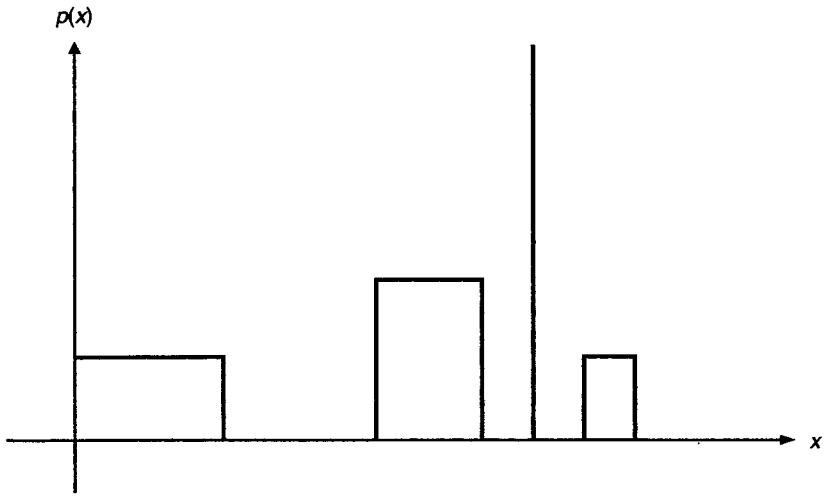


**FIGURE 1.22** Density function.



on. The expected value is sometimes called the *mean, average,* or *first moment* of the variable and is calculated from the density function as follows:

$$E[x] = \int_{-\infty}^{\infty} xp(x)dx. \tag{1.91}$$

A typical density function for a random variable is shown in Figure 1.23. The most likely value of variable $x$ is also indicated in the figure. The expected value can be thought of as a "center of gravity" or first moment of the random variable $x$.

The variance of a random variable is defined as

$$\sigma^2 = \text{Var}\{x\} = E\left[(x - E[x])^2\right], \tag{1.92}$$

where $\sigma$ is the root mean square value of the variable's difference from the mean. The variance is sometimes called the *mean square value* of $x$.

By extending the use of the expectation operator to joint probability densities, a variable $Y$ can be a function of two random variables, $s$ and $t$ such that

$$Y = \mathcal{O}\{s, t\}.$$

Then the expected value of $Y$ will be

$$E[Y] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathcal{O}[s, t]p(s, t)dsdt \tag{1.93}$$

where the joint probability density of $s$ and $t$ ($p(s,t)$), is required in the equation. The correlation of two random variables is defined to be the expected value of their product

$$E[st] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} st\ p(s, t)dsdt. \tag{1.94}$$

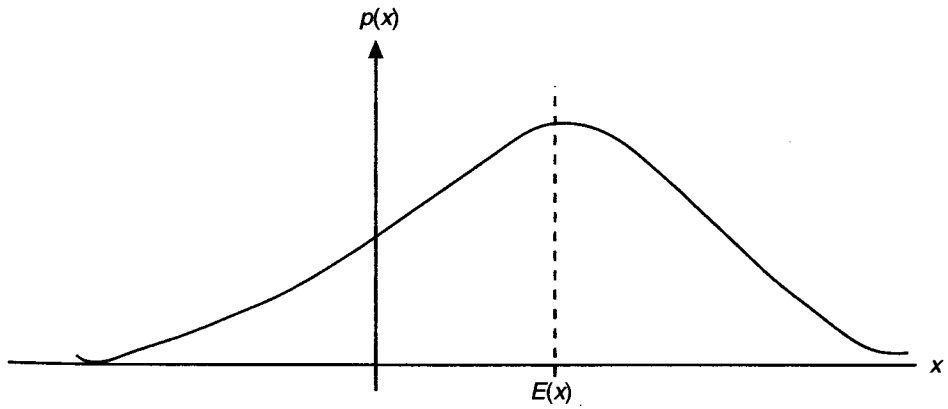


**FIGURE 1.23** Continuous PDF showing $E[x]$.

This definition will be used in the development of autocorrelation in section 1.6.5.

There is a set of random variables called *Gaussian random variables* whose density functions have special characteristics that make them particularly easy to analyze. Also, many physical processes give rise to approximately this sort of density function. A *Gaussian density function* has the following form:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], \tag{1.95}$$

where $\mu$ is the mean value of $x$ and $\sigma^2$ is the variance.

### 1.6.4 Quantization of Sequences

*Quantization* is to the amplitude domain of a continuous analog signal as sampling is to the time domain. It is the step that allows a continuous amplitude signal to be represented in the discrete amplitude increments available in a digital computer. To analyze the process of quantization, it is useful to diagram a system as shown in Figure 1.24. The illustration shows a continuous amplitude input signal, $f$, which is sampled and quantized, then reconstructed in the continuous amplitude domain. The output signal is $f$. By comparing the input and output of this process the effect of quantization can be illustrated.

The action of the box marked quantization in Figure 1.24 is illustrated in Figure 1.25. A set of decision levels is applied to each input signal, and the two levels which bracket the signal above and below are determined. A digital code is assigned to the region between each levels. In Figure 1.25, the digital code consists of 6 bits and runs from binary 0 to binary 63. The application of these decision levels and the assignment of a code to the input signal sample is the complete process of quantization. Reconstruction of the signal is accomplished by assigning a reconstruction level to each digital code.

The task that remains is to assign actual values to the decision levels and the reconstruction levels. Referring to Figure 1.25, the minimum value of the input signal is labeled $a_L$ and the maximum value is labeled $a_U$. If the signal $f$ has a probability density of $p(f)$, then the mean squared error due to the quantization and reconstruction process is

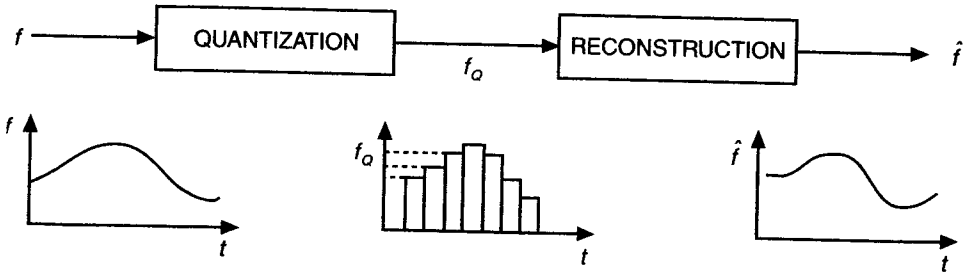


**FIGURE 1.24** Quantization and reconstruction of a signal.



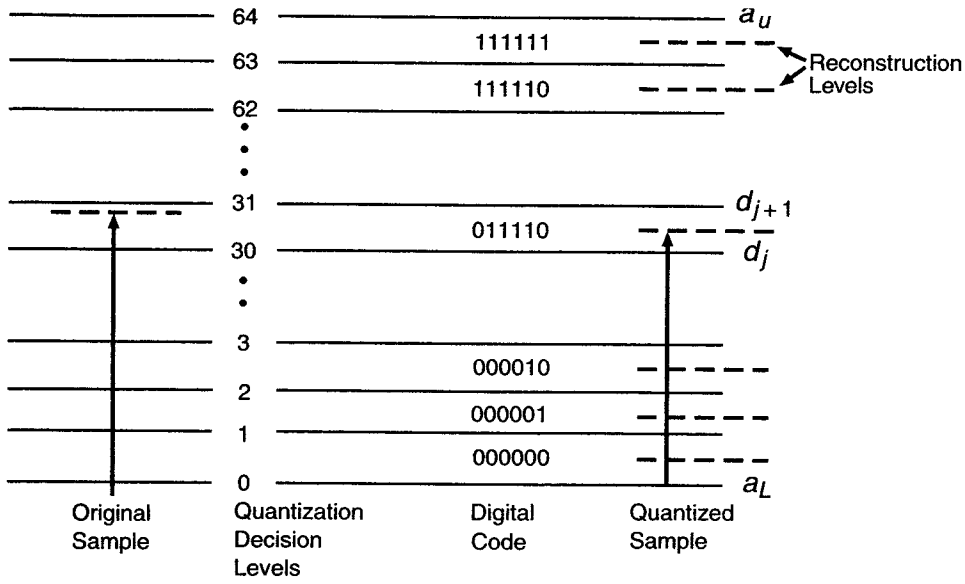


**FIGURE 1.25** Quantization operation showing decision and reconstruction levels.

$$\epsilon = E\left\{(f-\hat{f})^2\right\} = \int_{a_L}^{a_U} (f-\hat{f})^2 p(f)df,$$

and if the signal range is broken up into the segments between decision levels $d_j$ and $d_{j+1}$, then

$$\epsilon = E\left\{(f-\hat{f})^2\right\} = \sum_{j=0}^{J-1} \int_{d_j}^{d_{j+1}} (f-r_j)^2 p(f)df.$$

Numerical solutions can be determined that minimize $\epsilon$ for several common probability densities. The most common assumption is a uniform density ($p(f)$ equals $1/N$ for all values of $f$, where $N$ is the number of decision intervals). In this case, the decision levels are uniformly spaced throughout the interval and the reconstruction levels are centered between decision levels. This method of quantization is almost universal in commercial analog-to-digital converters. For this case the error in the analog-to-digital converter output is uniformly distributed from $-\frac{1}{2}$ of the least significant bit to $+\frac{1}{2}$ of the least significant bit. If it is assumed that the value of the least significant bit is unity, then the mean squared error due to this uniform quantization is given by:

$$\text{var}\{\epsilon\} = \int_{-\frac{1}{2}}^{+\frac{1}{2}} (f-\hat{f})^2 p(f)df = \int_{-\frac{1}{2}}^{+\frac{1}{2}} f^2 df = \frac{1}{12},$$

since $p(f) = 1$ from $-1/2$ to $+1/2$. This mean squared error gives the equivalent variance, or noise power, added to the original continuous analog samples as a result of the uniform quantization. If it is further assumed that the quantization error can be modeled as a stationary, uncorrelated white noise process (which is a good approximation when the number of quantization levels is greater than 16), then a maximum signal-to-noise ratio (SNR) can be defined for a quantization process of $B$ bits ($2^B$ quantization levels) as follows:

$$\text{SNR} = 10\log_{10}(V^2 / \text{var}\{\epsilon\}) = 10\log_{10}(12V^2),$$

where $V^2$ is the total signal power. For example, if a sinusoid is sampled with a peak amplitude of $2^{B-1}$, then $V^2 = 2^{2B}/8$ giving the signal to noise ratio for a full scale sinusoid as

$$\text{SNR} = 10\log_{10}((1.5)(2^{2B})) = 6.02B + 1.76.$$

This value of SNR is often referred to as the theoretical signal-to-noise ratio for a $B$ bit analog-to-digital converter. Because the analog circuits in a practical analog-to-digital converter always add some additional noise, the SNR of a real-world converter is always less than this value.

### 1.6.5 Random Processes, Autocorrelation, and Spectral Density

A *random process* is a function composed of random variables. An example is the random process $f(t)$. For each value of $t$, the process $f(t)$ can be considered a random variable. For $t = a$ there is a random variable $f(a)$ that has a probability density, an expected value (or mean), and a variance as defined in section 1.6.3. In a two-dimensional image, the function would be $f(x,y)$, where $x$ and $y$ are spatial variables. A two-dimensional random process is usually called a *random field.* Each $f(a,b)$ is a random variable.

One of the important aspects of a random process is the way in which the random variables at different points in the process are related to each other. The concept of joint probability is extended to distribution and density functions. A *joint probability distribution* is defined as

$$F(s,t) = P(S \le s, T \le t) \text{ (where } S \text{ and } T \text{ are some constants)},$$

and the corresponding density function is defined as

$$p(s,t) = \frac{\partial^2 F(s,t)}{\partial s \partial t}. \tag{1.96}$$

The integral relationship between distribution and density in this case is

$$F(s,t) = \int_{-\infty}^{s}\int_{-\infty}^{t} p(\alpha,\beta)d\alpha\, d\beta. \tag{1.97}$$

In section 1.6.3 it was shown that the correlation of two random variables is the expected value of their product. The *autocorrelation* of a random process is the expected value of the products of the random variables which make up the process. The symbol for autocorrelation is $R_{ff}(t_1, t_2)$ for the function $f(t)$ and the definition is



$$R_{ff}(t_1, t_2) = E[f(t_1)f(t_2)] \tag{1.98}$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \alpha\,\beta\, p_f(\alpha, \beta; t_1, t_2)d\alpha\, d\beta, \tag{1.99}$$

where $p_f(\alpha, \beta; t_1, t_2)$ is the joint probability density $f(t_1)$ and $f(t_2)$. By including $\alpha$ and $\beta$ in the parentheses the dependence of $p_f$ on these variables is made explicit.

In the general case, the autocorrelation can have different values for each value of $t_1$ and $t_2$. However, there is an important special class of random processes called *stationary processes* for which the form of the autocorrelation is somewhat simpler. In stationary random processes, the autocorrelation is only a function of the difference between the two time variables. For stationary processes

$$R_{ff}(t_2 - t_1) = R_{ff}(\xi) = E[f(t-\xi)f(t)]. \tag{1.100}$$

In section 1.6.6 the continuous variable theory presented here is extended to discrete variables and the concept of modeling real world signals is introduced.

### 1.6.6 Modeling Real-World Signals with AR Processes

By its nature, a noise process cannot be specified as a function of time in the way a deterministic signal can. Usually a noise process can be described with a probability function and the first and second moments of the process. Although this is only a partial characterization, a considerable amount of analysis can be performed using moment parameters alone. The first moment of a process is simply its average or mean value. In this section, all processes will have zero mean, simplifying the algebra and derivations but providing results for the most common set of processes.

The second moment is the autocorrelation of the process

$$r(n, n-k) = E[u(n)u^*(n-k)], \qquad \text{for } k = 0, \pm 1, \pm 2, \ldots.$$

The processes considered here are *stationary* to *second order.* This means that the first and second order statistics do not change with time. This allows the autocorrelation to be represented by

$$r(n, n-k) = r(k), \qquad \text{for } k = 0, \pm 1, \pm 2, \ldots$$

since it is a function only of the time *difference* between samples and not the time variable itself. In any process, an important member of the set of autocorrelation values is $r(0)$, which is

$$r(0) = E\{u(n)u^*(n)\} = E\{|u(n)|^2\}, \tag{1.101}$$

which is the mean square value of the process. For a zero mean process this is equal to the variance of the signal

$$r(0) = \text{var}\{u\}. \tag{1.102}$$

The process can be represented by a vector $\mathbf{u}(n)$ where

$$\mathbf{u}(n) = \begin{bmatrix} u(n) \\ u(n-1) \\ u(n-2) \\ \vdots \\ u(n-M+1) \end{bmatrix} \tag{1.103}$$

Then the autocorrelation can be represented in matrix form

$$\mathbf{R} = E\{\mathbf{u}(n)\mathbf{u}^H(n)\} \tag{1.104}$$

$$\mathbf{R} = \begin{bmatrix} r(0) & r(1) & r(2)\ldots & r(m-1) \\ r(-1) & r(0) & r(1)\ldots & \vdots \\ r(-2) & r(-1) & r(0)\ldots & \vdots \\ \vdots & \vdots & \vdots & r(-1) \\ \vdots & \vdots & \vdots & r(0) \\ r(-M+1) & r(-M+2) & \ldots\ldots & r(1) \end{bmatrix}$$

The second moment of a noise process is important because it is directly related to the power spectrum of the process. The relationship is

$$S(f) = \sum_{k=-M+1}^{M-1} r(k)e^{-j2\pi fk}, \tag{1.105}$$

which is the discrete Fourier transform (DFT) of the autocorrelation of the process ($r(k)$). Thus, the autocorrelation is the time domain description of the second order statistics, and the power spectral density, $S(f)$, is the frequency domain representation. This power spectral density can be modified by discrete time filters.

Discrete time filters may be classified as *autoregressive* (AR), *moving average* (MA), or a *combination of the two* (ARMA). Examples of these filter structures and the $z$-transforms of each of their impulse responses are shown in Figure 1.26. It is theoretically possible to create any arbitrary output stochastic process from an input white noise Gaussian process using a filter of sufficiently high (possibly infinite) order.

Referring again to the three filter structures in Figure 1.26, it is possible to create any arbitrary transfer function $H(z)$ with any one of the three structures. However, the orders of the realizations will be very different for one structure as compared to another. For instance, an infinite order MA filter may be required to duplicate an $M^{th}$ order AR filter.

One of the most basic theorems of adaptive and optimal filter theory is the *Wold*



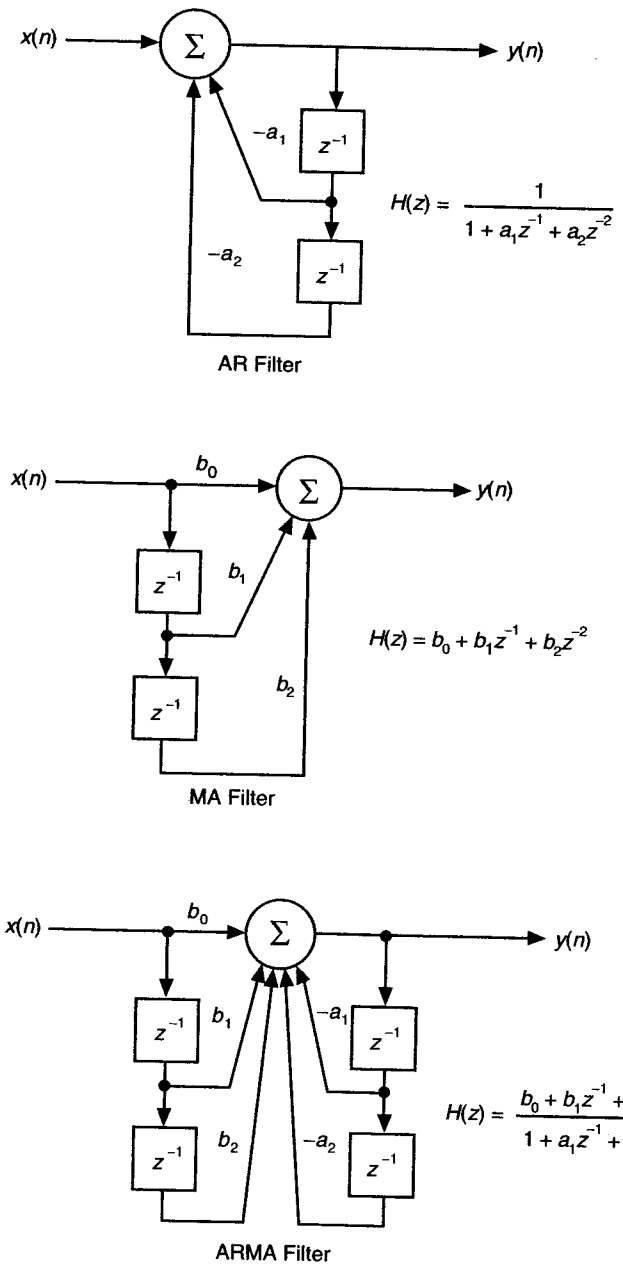


**FIGURE 1.26** AR, MA, and ARMA filter structures.

*decomposition*. This theorem states that any real-world process can be decomposed into a deterministic component (such as a sum of sine waves at specified amplitudes, phases, and frequencies) and a noise process. In addition, the theorem states that the noise process can be modeled as the output of a linear filter excited at its input by a white noise signal.

## ADAPTIVE FILTERS AND SYSTEMS

The problem of determining the optimum linear filter was solved by Norbert Wiener and others. The solution is referred to as the *Wiener filter* and is discussed in section 1.7.1. Adaptive filters and adaptive systems attempt to find an optimum set of filter parameters (often by approximating the Wiener optimum filter) based on the time varying input and output signals. In this section, adaptive filters and their application in closed loop adaptive systems are discussed briefly. Closed-loop adaptive systems are distinguished from open-loop systems by the fact that in a closed-loop system the adaptive processor is controlled based on information obtained from the input signal and the output signal of the processor. Figure 1.27 illustrates a basic adaptive system consisting of a processor that is controlled by an adaptive algorithm, which is in turn controlled by a performance calculation algorithm that has direct knowledge of the input and output signals.

Closed-loop adaptive systems have the advantage that the performance calculation algorithm can continuously monitor the input signal ($d$) and the output signal ($y$) and determine if the performance of the system is within acceptable limits. However, because several feedback loops may exist in this adaptive structure, the automatic optimization algorithm may be difficult to design, the system may become unstable or may result in nonunique and/or nonoptimum solutions. In other situations, the adaptation process may not converge and lead to a system with grossly poor performance. In spite of these possible drawbacks, closed-loop adaptive systems are widely used in communications, digital storage systems, radar, sonar, and biomedical systems.

The general adaptive system shown in Figure 1.27(a) can be applied in several ways. The most common application is prediction, where the desired signal ($d$) is the application provided input signal and a delayed version of the input signal is provided to the input of the adaptive processor ($x$) as shown in Figure 1.27(b). The adaptive processor must then try to predict the current input signal in order to reduce the error signal ($\varepsilon$) toward a mean squared value of zero. Prediction is often used in signal encoding (for example, speech compression), because if the next values of a signal can be accurately predicted, then these samples need not be transmitted or stored. Prediction can also be used to reduce noise or interference and therefore enhance the signal quality if the adaptive processor is designed to only predict the signal and ignore random noise elements or known interference patterns.

As shown in Figure 1.27(c), another application of adaptive systems is system modeling of an unknown or difficult to characterize system. The desired signal ($d$) is the unknown system's output and the input to the unknown system and the adaptive processor ($x$) is a broadband test signal (perhaps white Gaussian noise). After adaptation, the



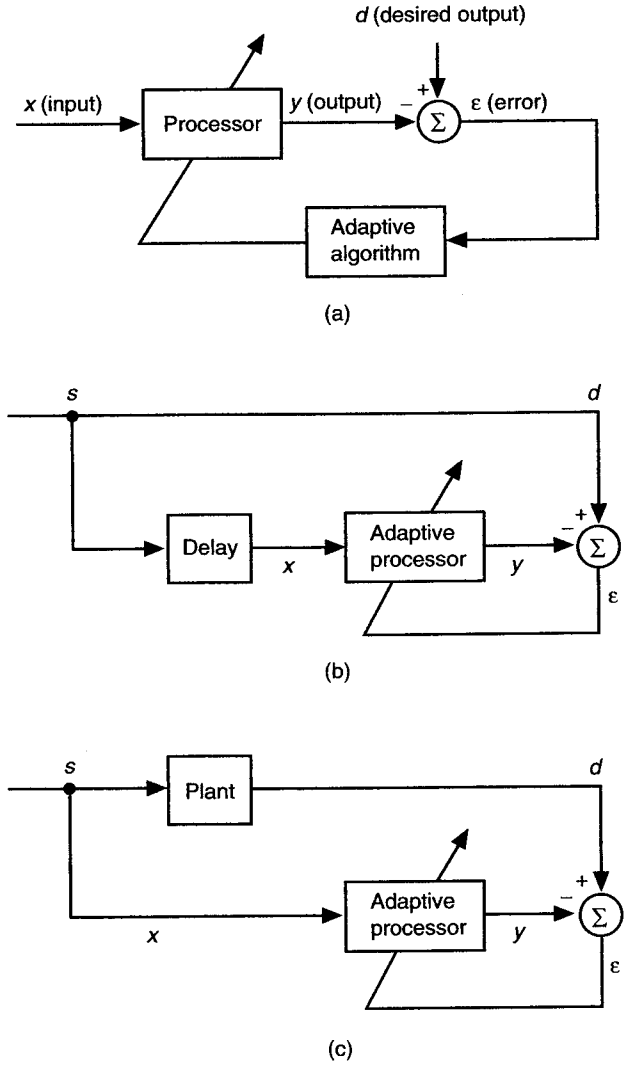


**FIGURE 1.27** **(a)** Closed-loop adaptive system; (b) prediction; (c) system modeling.

unknown system is modeled by the final transfer function of the adaptive processor. By using an AR, MA, or ARMA adaptive processor, different system models can be obtained. The magnitude of the error ($\varepsilon$) can be used to judge the relative success of each model.

### 1.7.1 Wiener Filter Theory

The problem of determining the optimum linear filter given the structure shown in Figure 1.28 was solved by Norbert Wiener and others. The solution is referred to as the *Wiener filter*. The statement of the problem is as follows:

Determine a set of coefficients, $w_k$, that minimize the mean of the squared error of the filtered output as compared to some desired output. The error is written

$$e(n) = d(n) - \sum_{k=1}^{M} w_k^* u(n-k+1), \tag{1.106}$$

or in vector form

$$e(n) = d(n) - \mathbf{w}^H \mathbf{u}(n). \tag{1.107}$$

The mean squared error is a function of the tap weight vector $\mathbf{w}$ chosen and is written

$$J(\mathbf{w}) = E\{e(n)e^*(n)\}. \tag{1.108}$$

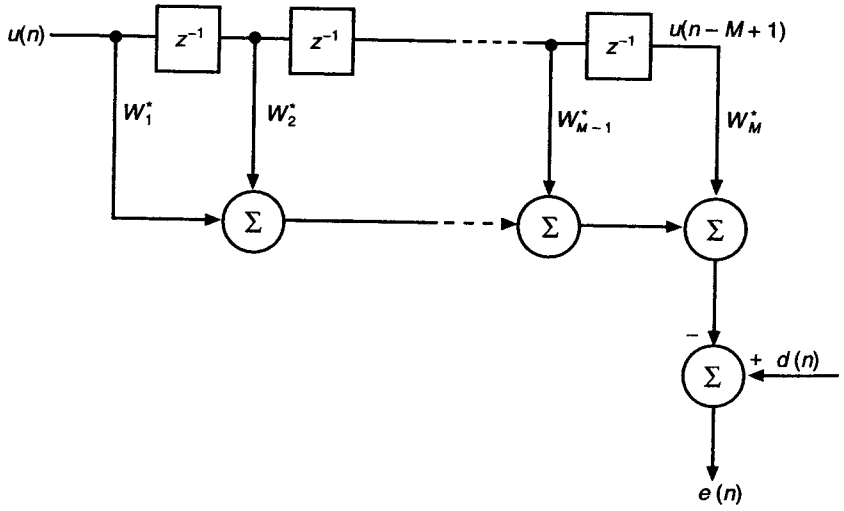


**FIGURE 1.28** Wiener filter problem.



Substituting in the expression for $e(n)$ gives

$$J(\mathbf{w}) = E\{d(n)d^*(n) - d(n)\mathbf{u}^H(n)\mathbf{w} - \mathbf{w}^H\mathbf{u}(n)d^*(n) + \mathbf{w}^H\mathbf{u}(n)\mathbf{u}^H(n)\mathbf{w}\} \tag{1.109}$$

$$J(\mathbf{w}) = \text{var}\{d\} - \mathbf{p}^H\mathbf{w} - \mathbf{w}^H\mathbf{p} + \mathbf{w}^H\mathbf{R}\mathbf{w}, \tag{1.110}$$

where $\mathbf{p} = E\{\mathbf{u}(n)d^*(n)\}$, the vector that is the product of the cross correlation between the desired signal and each element of the input vector.

In order to minimize $J(\mathbf{w})$ with respect to $\mathbf{w}$, the tap weight vector, one must set the derivative of $J(\mathbf{w})$ with respect to $\mathbf{w}$ equal to zero. This will give an equation which, when solved for $\mathbf{w}$, gives $\mathbf{w}_0$, the optimum value of $\mathbf{w}$. Setting the total derivative equal to zero gives

$$-2\mathbf{p} + 2\mathbf{R}\mathbf{w}_0 = 0 \tag{1.111}$$

or

$$\mathbf{R}\mathbf{w}_0 = \mathbf{p}. \tag{1.112}$$

If the matrix $\mathbf{R}$ is invertible (nonsingular) then $\mathbf{w}_0$ can be solved as

$$\mathbf{w}_0 = \mathbf{R}^{-1}\mathbf{p}. \tag{1.113}$$

So the optimum tap weight vector depends on the autocorrelation of the input process and the cross correlation between the input process and the desired output. Equation (1.113) is called the *normal equation* because a filter derived from this equation will produce an error that is orthogonal (or normal) to each element of the input vector. This can be written

$$E\{\mathbf{u}(n)e_0^*(n)\} = 0. \tag{1.114}$$

It is helpful at this point to consider what must be known to solve the Wiener filter problem:

**(1)** The $M$ x $M$ autocorrelation matrix of $\mathbf{u}(n)$, the input vector

**(2)** The cross correlation vector between $\mathbf{u}(n)$ and $d(n)$ the desired response.

It is clear that knowledge of any individual $\mathbf{u}(n)$ will not be sufficient to calculate these statistics. One must take the ensemble average, $E\{\ \}$, to form both the autocorrelation and the cross correlation. In practice, a model is developed for the input process and from this model the second order statistics are derived.

A legitimate question at this point is: What is $d(n)$? It depends on the problem. One example of the use of Wiener filter theory is in linear predictive filtering. In this case, the desired signal is the next value of $\mathbf{u}(n)$, the input. The actual $\mathbf{u}(n)$ is always available one sample after the prediction is made and this gives the ideal check on the quality of the prediction.

EMBREE, P. and KIMBLE, B. (1991). *C Language Algorithms for Digital Signal Processing.* Englewood Cliffs, NJ: Prentice Hall.

HARRIS, F. (1978). On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform. *Proceedings of the IEEE., 66,* (1), 51–83.

HAYKIN, S. (1986). *Adaptive Filter Theory.* Englewood Cliffs, NJ: Prentice Hall.

MCCLELLAN, J., PARKS, T. and RABINER, L.R. (1973). A Computer Program for Designing Optimum FIR Linear Phase Digital Filters. *IEEE Transactions on Audio and Electro-acoustics, AU-21.* (6), 506–526.

MOLER, C., LITTLE, J. and BANGERT, S. (1987). *PC-MATLAB User's Guide.* Sherbourne, MA: The Math Works.

OPPENHEIM, A. and SCHAFER, R. (1975). *Digital Signal Processing.* Englewood Cliffs, NJ: Prentice Hall.

OPPENHEIM, A. and SCHAFER, R. (1989). *Discrete-time Signal Processing.* Englewood Cliffs, NJ: Prentice Hall.

PAPOULIS, A. (1965). *Probability, Random Variables and Stochastic Processes.* New York: McGraw-Hill.

RABINER, L. and GOLD, B. (1975). *Theory and Application of Digital Signal Processing.* Englewood Cliffs, NJ: Prentice Hall.

STEARNS, S. and DAVID, R. (1988). *Signal Processing Algorithms.* Englewood Cliffs, NJ: Prentice Hall.

STEARNS, S. and DAVID, R. (1993). *Signal Processing Algorithms in FORTRAN and C.* Englewood Cliffs, NJ: Prentice Hall.

VAIDYANATHAN, P. (1993). *Multirate Systems and Filter Banks.* Englewood Cliffs, NJ: Prentice Hall.

WIDROW, B. and STEARNS, S. (1985). *Adaptive Signal Processing.* Englewood Cliffs, NJ: Prentice Hall.

# CHAPTER 2

# C PROGRAMMING FUNDAMENTALS

The purpose of this chapter is to provide the programmer with a complete overview of the fundamentals of the C programming language that are important in DSP applications. In particular, text manipulation, bitfields, enumerated data types, and unions are not discussed, because they have limited utility in the majority of DSP programs. Readers with C programming experience may wish to skip the bulk of this chapter with the possible exception of the more advanced concepts related to pointers and structures presented in sections 2.7 and 2.8. The proper use of pointers and data structures in C can make a DSP program easier to write and much easier for others to understand. Example DSP programs in this chapter and those which follow will clarify the importance of pointers and data structures in DSP programs.

## 2.1 THE ELEMENTS OF REAL-TIME DSP PROGRAMMING

The purpose of a *programming language* is to provide a tool so that a programmer can easily solve a problem involving the manipulation of some type of information. Based on this definition, the purpose of a DSP program is to manipulate a signal (a special kind of information) in such a way that the program solves a signal-processing problem. To do this, a DSP programming language must have five basic elements:

(1) A method of organizing different types of data (variables and data types)
(2) A method of describing the operations to be done (operators)
(3) A method of controlling the operations performed based on the results of operations (program control)