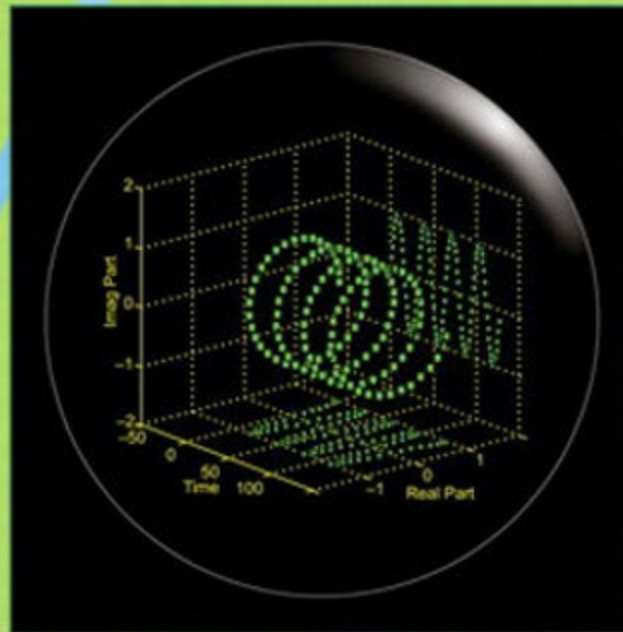


RICHARD G. LYONS

UNDERSTANDING DIGITAL SIGNAL PROCESSING



THIRD EDITION

Understanding Digital Signal Processing

Third Edition

Richard G. Lyons



Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/ph

Library of Congress Cataloging-in-Publication Data

Lyons, Richard G., 1948-

Understanding digital signal processing / Richard G. Lyons.—3rd ed.
p. cm.

Includes bibliographical references and index.

ISBN 0-13-702741-9 (hardcover : alk. paper)

1. Signal processing—Digital techniques. I. Title.

TK5102.9.L96 2011

621.382'2—dc22

2010035407

Copyright © 2011 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-13-702741-5

ISBN-10: 0-13-702741-9

Text printed in the United States on recycled paper at Edwards Brothers in Ann Arbor, Michigan.
Second printing, April 2011

I dedicate this book to my daughters, Julie and Meredith—I wish I could go with you; to my mother, Ruth, for making me finish my homework; to my father, Grady, who didn't know what he started when he built that workbench in the basement; to my brother Ray for improving us all; to my brother Ken who succeeded where I failed; to my sister Nancy for running interference for us; and to the skilled folks on the USENET newsgroup comp.dsp. They ask the good questions and provide the best answers. Finally, to Sigi Pardula (Batgirl): Without your keeping the place running, this book wouldn't exist.

Contents

PREFACE

ABOUT THE AUTHOR

1 DISCRETE SEQUENCES AND SYSTEMS

1.1 Discrete Sequences and Their Notation

1.2 Signal Amplitude, Magnitude, Power

1.3 Signal Processing Operational Symbols

1.4 Introduction to Discrete Linear Time-Invariant Systems

1.5 Discrete Linear Systems

1.6 Time-Invariant Systems

1.7 The Commutative Property of Linear Time-Invariant Systems

1.8 Analyzing Linear Time-Invariant Systems

References

Chapter 1 Problems

2 PERIODIC SAMPLING

2.1 Aliasing: Signal Ambiguity in the Frequency Domain

2.2 Sampling Lowpass Signals

2.3 Sampling Bandpass Signals

2.4 Practical Aspects of Bandpass Sampling

References

Chapter 2 Problems

3 THE DISCRETE FOURIER TRANSFORM

3.1 Understanding the DFT Equation

3.2 DFT Symmetry

3.3 DFT Linearity

3.4 DFT Magnitudes

3.5 DFT Frequency Axis

3.6 DFT Shifting Theorem

3.7 Inverse DFT

3.8 DFT Leakage

[3.9 Windows](#)

[3.10 DFT Scalloping Loss](#)

[3.11 DFT Resolution, Zero Padding, and Frequency-Domain Sampling](#)

[3.12 DFT Processing Gain](#)

[3.13 The DFT of Rectangular Functions](#)

[3.14 Interpreting the DFT Using the Discrete-Time Fourier Transform](#)

[References](#)

[Chapter 3 Problems](#)

4 THE FAST FOURIER TRANSFORM

[4.1 Relationship of the FFT to the DFT](#)

[4.2 Hints on Using FFTs in Practice](#)

[4.3 Derivation of the Radix-2 FFT Algorithm](#)

[4.4 FFT Input/Output Data Index Bit Reversal](#)

[4.5 Radix-2 FFT Butterfly Structures](#)

[4.6 Alternate Single-Butterfly Structures](#)

[References](#)

[Chapter 4 Problems](#)

5 FINITE IMPULSE RESPONSE FILTERS

[5.1 An Introduction to Finite Impulse Response \(FIR\) Filters](#)

[5.2 Convolution in FIR Filters](#)

[5.3 Lowpass FIR Filter Design](#)

[5.4 Bandpass FIR Filter Design](#)

[5.5 Highpass FIR Filter Design](#)

[5.6 Parks-McClellan Exchange FIR Filter Design Method](#)

[5.7 Half-band FIR Filters](#)

[5.8 Phase Response of FIR Filters](#)

[5.9 A Generic Description of Discrete Convolution](#)

[5.10 Analyzing FIR Filters](#)

[References](#)

[Chapter 5 Problems](#)

6 INFINITE IMPULSE RESPONSE FILTERS

[6.1 An Introduction to Infinite Impulse Response Filters](#)

[6.2 The Laplace Transform](#)

[6.3 The z-Transform](#)

[6.4 Using the z-Transform to Analyze IIR Filters](#)

[6.5 Using Poles and Zeros to Analyze IIR Filters](#)

[6.6 Alternate IIR Filter Structures](#)

[6.7 Pitfalls in Building IIR Filters](#)

[6.8 Improving IIR Filters with Cascaded Structures](#)

[6.9 Scaling the Gain of IIR Filters](#)

[6.10 Impulse Invariance IIR Filter Design Method](#)

[6.11 Bilinear Transform IIR Filter Design Method](#)

[6.12 Optimized IIR Filter Design Method](#)

[6.13 A Brief Comparison of IIR and FIR Filters](#)

[References](#)

[Chapter 6 Problems](#)

7 SPECIALIZED DIGITAL NETWORKS AND FILTERS

[7.1 Differentiators](#)

[7.2 Integrators](#)

[7.3 Matched Filters](#)

[7.4 Interpolated Lowpass FIR Filters](#)

[7.5 Frequency Sampling Filters: The Lost Art](#)

[References](#)

[Chapter 7 Problems](#)

8 QUADRATURE SIGNALS

[8.1 Why Care about Quadrature Signals?](#)

[8.2 The Notation of Complex Numbers](#)

[8.3 Representing Real Signals Using Complex Phasors](#)

[8.4 A Few Thoughts on Negative Frequency](#)

[8.5 Quadrature Signals in the Frequency Domain](#)

[8.6 Bandpass Quadrature Signals in the Frequency Domain](#)

[8.7 Complex Down-Conversion](#)

[8.8 A Complex Down-Conversion Example](#)

[8.9 An Alternate Down-Conversion Method](#)

[References](#)

[Chapter 8 Problems](#)

[9 THE DISCRETE HILBERT TRANSFORM](#)

[9.1 Hilbert Transform Definition](#)

[9.2 Why Care about the Hilbert Transform?](#)

[9.3 Impulse Response of a Hilbert Transformer](#)

[9.4 Designing a Discrete Hilbert Transformer](#)

[9.5 Time-Domain Analytic Signal Generation](#)

[9.6 Comparing Analytical Signal Generation Methods](#)

[References](#)

[Chapter 9 Problems](#)

[10 SAMPLE RATE CONVERSION](#)

[10.1 Decimation](#)

[10.2 Two-Stage Decimation](#)

[10.3 Properties of Downsampling](#)

[10.4 Interpolation](#)

[10.5 Properties of Interpolation](#)

[10.6 Combining Decimation and Interpolation](#)

[10.7 Polyphase Filters](#)

[10.8 Two-Stage Interpolation](#)

[10.9 z-Transform Analysis of Multirate Systems](#)

[10.10 Polyphase Filter Implementations](#)

[10.11 Sample Rate Conversion by Rational Factors](#)

[10.12 Sample Rate Conversion with Half-band Filters](#)

[10.13 Sample Rate Conversion with IFIR Filters](#)

[10.14 Cascaded Integrator-Comb Filters](#)

[References](#)

[Chapter 10 Problems](#)

[11 SIGNAL AVERAGING](#)

[11.1 Coherent Averaging](#)

[11.2 Incoherent Averaging](#)

[11.3 Averaging Multiple Fast Fourier Transforms](#)

[11.4 Averaging Phase Angles](#)

[11.5 Filtering Aspects of Time-Domain Averaging](#)

[11.6 Exponential Averaging](#)

[References](#)

[Chapter 11 Problems](#)

12 DIGITAL DATA FORMATS AND THEIR EFFECTS

[12.1 Fixed-Point Binary Formats](#)

[12.2 Binary Number Precision and Dynamic Range](#)

[12.3 Effects of Finite Fixed-Point Binary Word Length](#)

[12.4 Floating-Point Binary Formats](#)

[12.5 Block Floating-Point Binary Format](#)

[References](#)

[Chapter 12 Problems](#)

13 DIGITAL SIGNAL PROCESSING TRICKS

[13.1 Frequency Translation without Multiplication](#)

[13.2 High-Speed Vector Magnitude Approximation](#)

[13.3 Frequency-Domain Windowing](#)

[13.4 Fast Multiplication of Complex Numbers](#)

[13.5 Efficiently Performing the FFT of Real Sequences](#)

[13.6 Computing the Inverse FFT Using the Forward FFT](#)

[13.7 Simplified FIR Filter Structure](#)

[13.8 Reducing A/D Converter Quantization Noise](#)

[13.9 A/D Converter Testing Techniques](#)

[13.10 Fast FIR Filtering Using the FFT](#)

[13.11 Generating Normally Distributed Random Data](#)

[13.12 Zero-Phase Filtering](#)

[13.13 Sharpened FIR Filters](#)

[13.14 Interpolating a Bandpass Signal](#)

[13.15 Spectral Peak Location Algorithm](#)

[13.16 Computing FFT Twiddle Factors](#)

[13.17 Single Tone Detection](#)

[13.18 The Sliding DFT](#)

[13.19 The Zoom FFT](#)

[13.20 A Practical Spectrum Analyzer](#)

[13.21 An Efficient Arctangent Approximation](#)

[13.22 Frequency Demodulation Algorithms](#)

[13.23 DC Removal](#)

[13.24 Improving Traditional CIC Filters](#)

[13.25 Smoothing Impulsive Noise](#)

[13.26 Efficient Polynomial Evaluation](#)

[13.27 Designing Very High-Order FIR Filters](#)

[13.28 Time-Domain Interpolation Using the FFT](#)

[13.29 Frequency Translation Using Decimation](#)

[13.30 Automatic Gain Control \(AGC\)](#)

[13.31 Approximate Envelope Detection](#)

[13.32 A Quadrature Oscillator](#)

[13.33 Specialized Exponential Averaging](#)

[13.34 Filtering Narrowband Noise Using Filter Nulls](#)

[13.35 Efficient Computation of Signal Variance](#)

[13.36 Real-time Computation of Signal Averages and Variances](#)

[13.37 Building Hilbert Transformers from Half-band Filters](#)

[13.38 Complex Vector Rotation with Arctangents](#)

[13.39 An Efficient Differentiating Network](#)

[13.40 Linear-Phase DC-Removal Filter](#)

[13.41 Avoiding Overflow in Magnitude Computations](#)

[13.42 Efficient Linear Interpolation](#)

[13.43 Alternate Complex Down-conversion Schemes](#)

[13.44 Signal Transition Detection](#)

[13.45 Spectral Flipping around Signal Center Frequency](#)

[13.46 Computing Missing Signal Samples](#)

[13.47 Computing Large DFTs Using Small FFTs](#)

[13.48 Computing Filter Group Delay without Arctangents](#)

[13.49 Computing a Forward and Inverse FFT Using a Single FFT](#)

[13.50 Improved Narrowband Lowpass IIR Filters](#)

[13.51 A Stable Goertzel Algorithm](#)

[References](#)

A THE ARITHMETIC OF COMPLEX NUMBERS

[A.1 Graphical Representation of Real and Complex Numbers](#)

[A.2 Arithmetic Representation of Complex Numbers](#)

[A.3 Arithmetic Operations of Complex Numbers](#)

[A.4 Some Practical Implications of Using Complex Numbers](#)

B CLOSED FORM OF A GEOMETRIC SERIES

C TIME REVERSAL AND THE DFT

D MEAN, VARIANCE, AND STANDARD DEVIATION

[D.1 Statistical Measures](#)

[D.2 Statistics of Short Sequences](#)

[D.3 Statistics of Summed Sequences](#)

[D.4 Standard Deviation \(RMS\) of a Continuous Sinewave](#)

[D.5 Estimating Signal-to-Noise Ratios](#)

[D.6 The Mean and Variance of Random Functions](#)

[D.7 The Normal Probability Density Function](#)

E DECIBELS (DB AND DBM)

[E.1 Using Logarithms to Determine Relative Signal Power](#)

[E.2 Some Useful Decibel Numbers](#)

[E.3 Absolute Power Using Decibels](#)

F DIGITAL FILTER TERMINOLOGY

G FREQUENCY SAMPLING FILTER DERIVATIONS

[G.1 Frequency Response of a Comb Filter](#)

[G.2 Single Complex FSF Frequency Response](#)

[G.3 Multisection Complex FSF Phase](#)

[G.4 Multisection Complex FSF Frequency Response](#)

[G.5 Real FSF Transfer Function](#)

[G.6 Type-IV FSF Frequency Response](#)

[**H FREQUENCY SAMPLING FILTER DESIGN TABLES**](#)

[**I COMPUTING CHEBYSHEV WINDOW SEQUENCES**](#)

[I.1 Chebyshev Windows for FIR Filter Design](#)

[I.2 Chebyshev Windows for Spectrum Analysis](#)

[**INDEX**](#)

Preface

This book is an expansion of previous editions of *Understanding Digital Signal Processing*. Like those earlier editions, its goals are (1) to help beginning students understand the theory of digital signal processing (DSP) and (2) to provide practical DSP information, not found in other books, to help working engineers/scientists design and test their signal processing systems. Each chapter of this book contains new information beyond that provided in earlier editions.

It's traditional at this point in the preface of a DSP textbook for the author to tell readers why they should learn DSP. I don't need to tell you how important DSP is in our modern engineering world. You already know that. I'll just say that the future of electronics is DSP, and with this book you will not be left behind.

For Instructors

This third edition is appropriate as the text for a one- or two-semester undergraduate course in DSP. It follows the DSP material I cover in my corporate training activities and a signal processing course I taught at the University of California Santa Cruz Extension. To aid students in their efforts to learn DSP, this third edition provides additional explanations and examples to increase its tutorial value. To test a student's understanding of the material, homework problems have been included at the end of each chapter. (For qualified instructors, a Solutions Manual is available from Prentice Hall.)

For Practicing Engineers

To help working DSP engineers, the changes in this third edition include, but are not limited to, the following:

- Practical guidance in building discrete differentiators, integrators, and matched filters
- Descriptions of statistical measures of signals, variance reduction by way of averaging, and techniques for computing real-world signal-to-noise ratios (SNRs)
- A significantly expanded chapter on sample rate conversion (multirate systems) and its associated filtering
- Implementing fast convolution (FIR filtering in the frequency domain)
- IIR filter scaling
- Enhanced material covering techniques for analyzing the behavior and performance of digital filters
- Expanded descriptions of industry-standard binary number formats used in modern processing systems
- Numerous additions to the popular “[Digital Signal Processing Tricks](#)” chapter

For Students

Learning the fundamentals, and how to speak the language, of digital signal processing does not require profound analytical skills or an extensive background in mathematics. All you need is a little experience with elementary algebra, knowledge of what a sinewave is, this book, and enthusiasm.

This may sound hard to believe, particularly if you’ve just flipped through the pages of this book and seen figures and equations that look rather complicated. The content here, you say, looks suspiciously like material in technical journals and textbooks whose meaning has eluded you in the past. Well, this is not just another book on digital signal processing.

In this book I provide a gentle, but thorough, explanation of the theory and practice of DSP. The text is not written so that you *may* understand the material, but so that you *must* understand the material. I’ve attempted to avoid the traditional instructor–student relationship and have tried to make reading this book seem like talking to a friend while walking in the park. I’ve used just enough mathematics to help you develop a fundamental understanding of DSP theory and have illustrated that theory with practical examples.

I have designed the homework problems to be more than mere exercises that assign values to variables for the student to plug into some equation in order to compute a result. Instead, the homework problems are designed to be as educational as possible in the sense of expanding on and enabling further investigation of specific aspects of DSP topics covered in the text. Stated differently, the homework problems are not designed to induce “death by algebra,” but rather to improve your understanding of DSP. Solving the problems helps you become proactive in your own DSP education instead of merely being an inactive recipient of DSP information.

The Journey

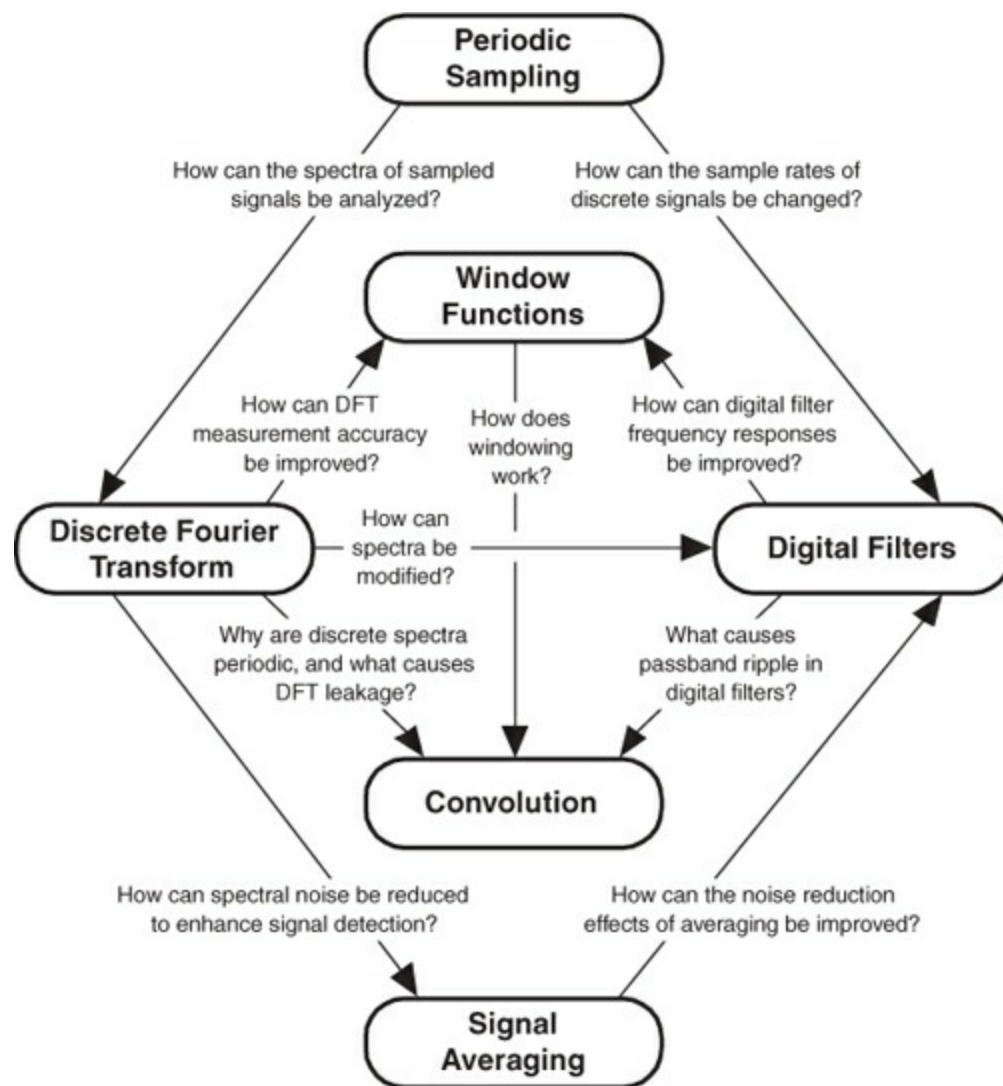
Learning digital signal processing is not something you accomplish; it’s a journey you take. When you gain an understanding of one topic, questions arise that cause you to investigate some other facet of digital signal processing.[†] Armed with more knowledge, you’re likely to begin exploring further aspects of digital signal processing much like those shown in the diagram on page [xviii](#). This book is your tour guide during the first steps of your journey.

[†]“You see I went on with this research just the way it led me. This is the only way I ever heard of research going. I asked a question, devised some method of getting an answer, and got—a fresh question. Was this possible, or that possible? You cannot imagine what this means to an investigator, what an intellectual passion grows upon him. You cannot imagine the strange colourless delight of these intellectual desires” (Dr. Moreau—infamous physician and vivisectionist from H.G. Wells’ *Island of Dr. Moreau*, 1896).

You don’t need a computer to learn the material in this book, but it would certainly help. DSP simulation software allows the beginner to verify signal processing theory through the time-tested trial and error process.[‡] In particular, software routines that plot signal data, perform the fast Fourier transforms, and analyze digital filters would be very useful.

[‡]“One must learn by doing the thing; for though you think you know it, you have no certainty until you try it” (Sophocles, 496–406 B.C.).

As you go through the material in this book, don’t be discouraged if your understanding comes slowly. As the Greek mathematician Menaechmus curtly remarked to Alexander the Great, when asked for a quick explanation of mathematics, “There is no royal road to mathematics.” Menaechmus was confident in telling Alexander the only way to learn mathematics is through careful study. The same applies to digital signal processing. Also, don’t worry if you need to read some of the material twice. While the concepts in this book are not as complicated as quantum physics, as mysterious as the lyrics of the song “Louie Louie,” or as puzzling as the assembly instructions of a metal shed, they can become a little involved. They deserve your thoughtful attention. So, go slowly and read the material twice if necessary; you’ll be glad you did. If you show persistence, to quote Susan B. Anthony, “Failure is impossible.”



Coming Attractions

[Chapter 1](#) begins by establishing the notation used throughout the remainder of the book. In that chapter we introduce the concept of discrete signal sequences, show how they relate to continuous signals, and illustrate how those sequences can be depicted in both the time and frequency domains. In addition, [Chapter 1](#) defines the operational symbols we'll use to build our signal processing system block diagrams. We conclude that chapter with a brief introduction to the idea of linear systems and see why linearity enables us to use a number of powerful mathematical tools in our analysis.

[Chapter 2](#) introduces the most frequently misunderstood process in digital signal processing, periodic sampling. Although the concept of sampling a continuous signal is not complicated, there are mathematical subtleties in the process that require thoughtful attention. Beginning gradually with simple examples of lowpass sampling, we then proceed to the interesting subject of bandpass sampling. [Chapter 2](#) explains and quantifies the frequency-domain ambiguity (aliasing) associated with these important topics.

[Chapter 3](#) is devoted to one of the foremost topics in digital signal processing, the discrete Fourier transform (DFT) used for spectrum analysis. Coverage begins with detailed examples illustrating the important properties of the DFT and how to interpret DFT spectral results, progresses to the topic of windows used to reduce DFT leakage, and discusses the processing gain afforded by the DFT. The chapter concludes with a detailed discussion of the various forms of the transform of rectangular functions that the reader is likely to encounter in the literature.

[Chapter 4](#) covers the innovation that made the most profound impact on the field of digital signal processing, the fast Fourier transform (FFT). There we show the relationship of the popular radix 2 FFT to the DFT, quantify the powerful processing advantages gained by using the FFT, demonstrate why the FFT functions as it does, and present various FFT implementation structures. [Chapter 4](#) also includes a list of recommendations to help the reader use the FFT in practice.

[Chapter 5](#) ushers in the subject of digital filtering. Beginning with a simple lowpass finite impulse response (FIR) filter example, we carefully progress through the analysis of that filter's frequency-domain magnitude and phase response. Next, we learn how window functions affect, and can be used to design, FIR filters. The methods for converting lowpass FIR filter designs to bandpass and highpass digital filters are presented, and the popular Parks-McClellan (Remez) Exchange FIR filter design technique is introduced and illustrated by example. In that chapter we acquaint the reader with, and take the mystery out of, the process called convolution. Proceeding through several simple convolution examples, we conclude [Chapter 5](#) with a discussion of the powerful convolution theorem and show why it's so useful as a qualitative tool in understanding digital signal processing.

[Chapter 6](#) is devoted to a second class of digital filters, infinite impulse response (IIR) filters. In discussing several methods for the design of IIR filters, the reader is introduced to the powerful digital signal processing analysis tool called the z -transform. Because the z -transform is so closely related to the continuous Laplace transform, [Chapter 6](#) starts by gently guiding the reader from the origin, through the properties, and on to the utility of the Laplace transform in preparation for learning the z -transform. We'll see how IIR filters are designed and implemented, and why their performance is so different from that of FIR filters. To indicate under what conditions these filters should be used, the chapter concludes with a qualitative comparison of the key properties of FIR and IIR filters.

[Chapter 7](#) introduces specialized networks known as *digital differentiators*, *integrators*, and *matched filters*. In addition, this chapter covers two specialized digital filter types that have not received their deserved exposure in traditional DSP textbooks. Called *interpolated FIR* and *frequency sampling* filters, providing improved lowpass filtering computational efficiency, they belong in our arsenal of filter design techniques. Although these are FIR filters, their introduction is delayed to this chapter because familiarity with the z -transform (in [Chapter 6](#)) makes the properties of these filters easier to understand.

[Chapter 8](#) presents a detailed description of quadrature signals (also called *complex* signals). Because quadrature signal theory has become so important in recent years, in both signal analysis and digital communications implementations, it deserves its own chapter. Using three-dimensional illustrations, this chapter gives solid physical meaning to the mathematical notation, processing advantages, and use of quadrature signals. Special emphasis is given to quadrature sampling (also called *complex down-conversion*).

[Chapter 9](#) provides a mathematically gentle, but technically thorough, description of the Hilbert transform—a process used to generate a quadrature (complex) signal from a real signal. In this chapter we describe the properties, behavior, and design of practical Hilbert transformers.

[Chapter 10](#) presents an introduction to the fascinating and useful process of sample rate conversion (changing the effective sample rate of discrete data sequences through decimation or interpolation). Sample rate conversion—so useful in improving the performance and reducing the computational complexity of many signal processing operations—is essentially an exercise in lowpass filter design. As such, polyphase and cascaded integrator-comb filters are described in detail in this chapter.

[Chapter 11](#) covers the important topic of signal averaging. There we learn how averaging increases the accuracy of signal measurement schemes by reducing measurement background noise. This accuracy enhancement is called *processing gain*, and the chapter shows how to predict the processing gain associated with averaging signals in both the time and frequency domains. In addition, the key differences between coherent and incoherent averaging techniques are explained and demonstrated with examples. To complete that chapter the popular scheme known as *exponential averaging* is covered in some detail.

[Chapter 12](#) presents an introduction to the various binary number formats the reader is likely to encounter in modern digital signal processing. We establish the precision and dynamic range afforded by these formats along with the inherent pitfalls associated with their use. Our exploration of the critical subject of binary data word width (in bits) naturally leads to a discussion of the numerical resolution limitations of analog-to-digital (A/D) converters and how to determine the optimum A/D converter word size for a given application. The problems of data value overflow roundoff errors are covered along with a statistical introduction to the two most popular remedies for overflow, truncation and rounding. We end that chapter by covering the interesting subject of floating-point binary formats that allow us to overcome most of the limitations induced by fixed-point binary formats, particularly in reducing the ill effects of data overflow.

[Chapter 13](#) provides the literature's most comprehensive collection of *tricks of the trade* used by DSP professionals to make their processing algorithms more efficient. These techniques are compiled into a chapter at the end of the book for two reasons. First, it seems wise to keep our collection of tricks in one chapter so that we'll know where to find them in the future. Second, many of these clever schemes require an understanding of the material from the previous chapters, making the last chapter an appropriate place to keep our arsenal of clever tricks. Exploring these techniques in detail verifies and reiterates many of the important ideas covered in previous chapters.

The appendices include a number of topics to help the beginner understand the nature and mathematics of digital signal processing. A comprehensive description of the arithmetic of complex numbers is covered in [Appendix A](#), and [Appendix B](#) derives the often used, but seldom explained, closed form of a geometric series. The subtle aspects and two forms of time reversal in discrete systems (of which zero-phase digital filtering is an application) are explained in [Appendix C](#). The statistical concepts of mean, variance, and standard deviation are introduced and illustrated in [Appendix D](#), and [Appendix E](#) provides a discussion of the origin and utility of the logarithmic decibel scale used to improve the magnitude resolution of spectral representations. [Appendix F](#), in a slightly different vein, provides a glossary of the terminology used in the field of digital filters. [Appendices G](#) and [H](#) provide supplementary information for designing and analyzing specialized digital filters. [Appendix I](#) explains the computation of Chebyshev window sequences.

Acknowledgments

Much of the new material in this edition is a result of what I've learned from those clever folk on the USENET newsgroup comp.dsp. (I could list a dozen names, but in doing so I'd make 12 friends and 500 enemies.) So, I say thanks to my DSP pals on comp.dsp for teaching me so much signal processing theory.

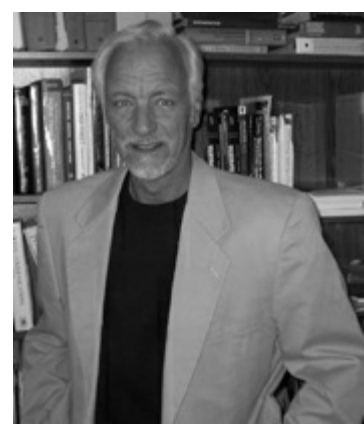
In addition to the reviewers of previous editions of this book, I thank Randy Yates, Clay Turner, and Ryan Groulx for their time and efforts to help me improve the content of this book. I am especially indebted to my eagle-eyed mathematician friend Antoine Trux for his relentless hard work to both

enhance this DSP material and create a homework Solutions Manual.

As before, I thank my acquisitions editor, Bernard Goodwin, for his patience and guidance, and his skilled team of production people, project editor Elizabeth Ryan in particular, at Prentice Hall.

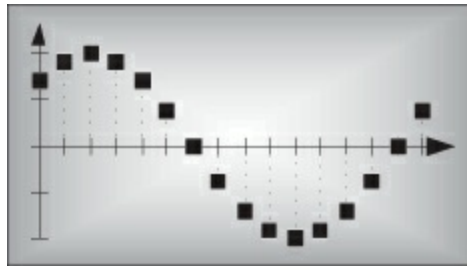
If you're still with me this far in this Preface, I end by saying I had a ball writing this book and sincerely hope you benefit from reading it. If you have any comments or suggestions regarding this material, or detect any errors no matter how trivial, please send them to me at R.Lyons@ieee.org. I promise I will reply to your e-mail.

About the Author



Richard Lyons is a consulting systems engineer and lecturer with Besser Associates in Mountain View, California. He has been the lead hardware engineer for numerous signal processing systems for both the National Security Agency (NSA) and Northrop Grumman Corp. Lyons has taught DSP at the University of California Santa Cruz Extension and authored numerous articles on DSP. As associate editor for the *IEEE Signal Processing Magazine* he created, edits, and contributes to the magazine's "DSP Tips & Tricks" column.

Chapter One. Discrete Sequences and Systems



Digital signal processing has never been more prevalent or easier to perform. It wasn't that long ago when the fast Fourier transform (FFT), a topic we'll discuss in [Chapter 4](#), was a mysterious mathematical process used only in industrial research centers and universities. Now, amazingly, the FFT is readily available to us all. It's even a built-in function provided by inexpensive spreadsheet software for home computers. The availability of more sophisticated commercial signal processing software now allows us to analyze and develop complicated signal processing applications rapidly and reliably. We can perform spectral analysis, design digital filters, develop voice recognition, data communication, and image compression processes using software that's interactive both in the way algorithms are defined and how the resulting data are graphically displayed. Since the mid-1980s the same integrated circuit technology that led to affordable home computers has produced powerful and inexpensive hardware development systems on which to implement our digital signal processing designs.[‡] Regardless, though, of the ease with which these new digital signal processing development systems and software can be applied, we still need a solid foundation in understanding the basics of digital signal processing. The purpose of this book is to build that foundation.

[‡] During a television interview in the early 1990s, a leading computer scientist stated that had automobile technology made the same strides as the computer industry, we'd all have a car that would go a half million miles per hour and get a half million miles per gallon. The cost of that car would be so low that it would be cheaper to throw it away than pay for one day's parking in San Francisco.

In this chapter we'll set the stage for the topics we'll study throughout the remainder of this book by defining the terminology used in digital signal processing, illustrating the various ways of graphically representing discrete signals, establishing the notation used to describe sequences of data values, presenting the symbols used to depict signal processing operations, and briefly introducing the concept of a linear discrete system.

1.1 Discrete Sequences and Their Notation

In general, the term *signal processing* refers to the science of analyzing time-varying physical processes. As such, signal processing is divided into two categories, analog signal processing and digital signal processing. The term *analog* is used to describe a waveform that's continuous in time and can take on a continuous range of amplitude values. An example of an analog signal is some voltage that can be applied to an oscilloscope, resulting in a continuous display as a function of time. Analog signals can also be applied to a conventional spectrum analyzer to determine their frequency content. The term *analog* appears to have stemmed from the analog computers used prior to 1980. These computers solved linear differential equations by means of connecting physical (electronic) differentiators and integrators using old-style telephone operator patch cords. That way, a continuous voltage or current in the actual circuit was *analogous* to some variable in a differential equation, such as speed, temperature, air pressure, etc. (Although the flexibility and speed of modern-day digital computers have since made analog computers obsolete, a good description of the short-lived utility of

analog computers can be found in reference [1].) Because present-day signal processing of continuous radio-type signals using resistors, capacitors, operational amplifiers, etc., has nothing to do with analogies, the term *analog* is actually a misnomer. The more correct term is *continuous signal processing* for what is today so commonly called analog signal processing. As such, in this book we'll minimize the use of the term *analog signals* and substitute the phrase *continuous signals* whenever appropriate.

The term *discrete-time signal* is used to describe a signal whose independent time variable is quantized so that we know only the value of the signal at discrete instants in time. Thus a discrete-time signal is not represented by a continuous waveform but, instead, a sequence of values. In addition to quantizing time, a discrete-time signal quantizes the signal amplitude. We can illustrate this concept with an example. Think of a continuous sinewave with a peak amplitude of 1 at a frequency f_0 described by the equation

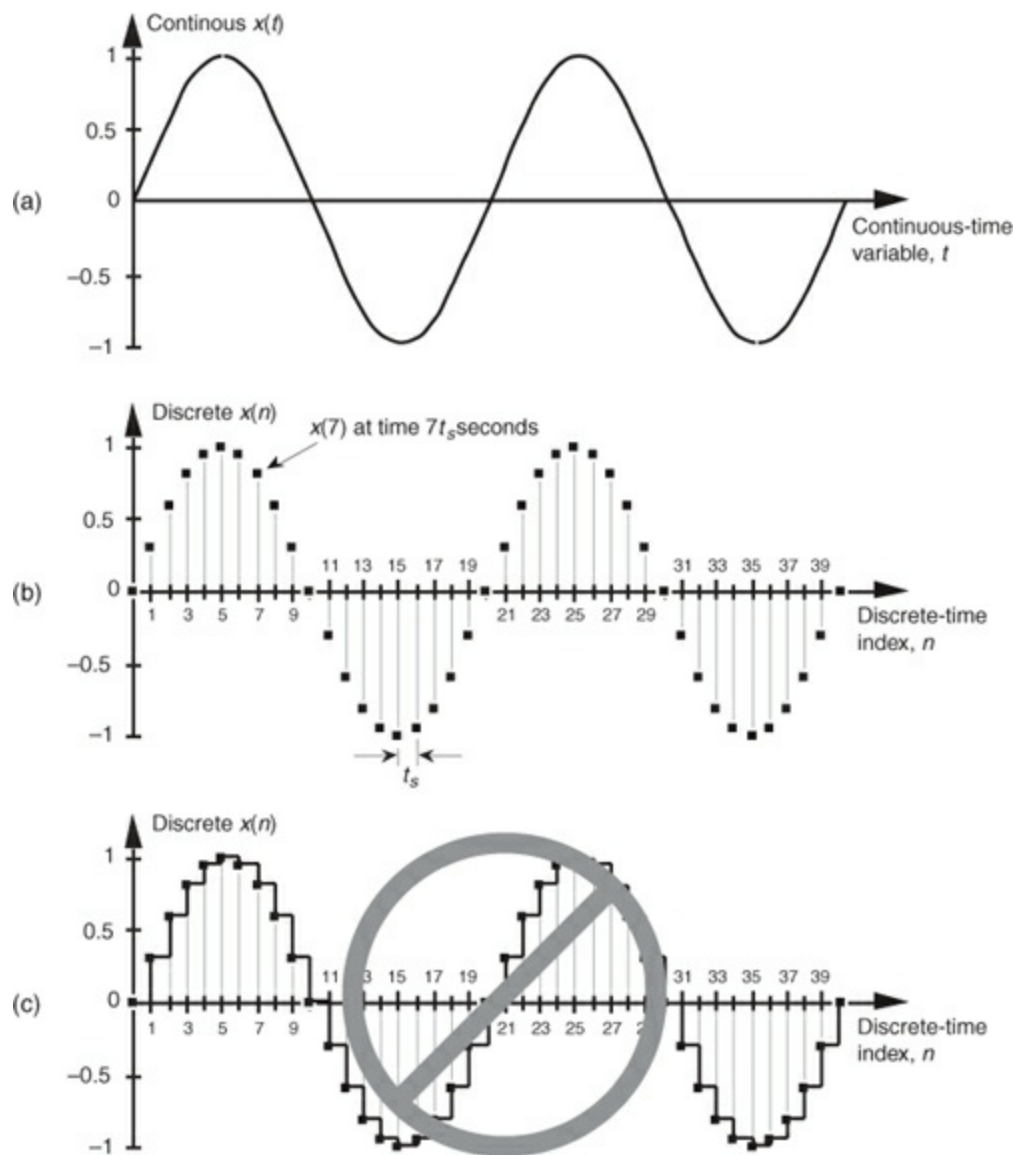
$$(1-1) \\ x(t) = \sin(2\pi f_0 t).$$

The frequency f_0 is measured in hertz (Hz). (In physical systems, we usually measure frequency in units of hertz. One Hz is a single oscillation, or cycle, per second. One kilohertz (kHz) is a thousand Hz, and a megahertz (MHz) is one million Hz.[‡]) With t in [Eq. 1-1](#) representing time in seconds, the $f_0 t$ factor has dimensions of cycles, and the complete $2\pi f_0 t$ term is an angle measured in radians.

[‡] The dimension for frequency used to be *cycles/second*; that's why the tuning dials of old radios indicate frequency as kilocycles/second (kcps) or megacycles/second (Mcps). In 1960 the scientific community adopted hertz as the unit of measure for frequency in honor of the German physicist Heinrich Hertz, who first demonstrated radio wave transmission and reception in 1887.

Plotting [Eq. \(1-1\)](#), we get the venerable continuous sinewave curve shown in [Figure 1-1\(a\)](#). If our continuous sinewave represents a physical voltage, we could *sample* it once every t_s seconds using an analog-to-digital converter and represent the sinewave as a sequence of discrete values. Plotting those individual values as dots would give us the discrete waveform in [Figure 1-1\(b\)](#). We say that [Figure 1-1\(b\)](#) is the “discrete-time” version of the continuous signal in [Figure 1-1\(a\)](#). The independent variable t in [Eq. \(1-1\)](#) and [Figure 1-1\(a\)](#) is continuous. The independent *index* variable n in [Figure 1-1\(b\)](#) is discrete and can have only integer values. That is, index n is used to identify the individual elements of the discrete sequence in [Figure 1-1\(b\)](#).

Figure 1-1 A time-domain sinewave: (a) continuous waveform representation; (b) discrete sample representation; (c) discrete samples with connecting lines.



Do not be tempted to draw lines between the dots in [Figure 1-1\(b\)](#). For some reason, people (particularly those engineers experienced in working with continuous signals) want to connect the dots with straight lines, or the stair-step lines shown in [Figure 1-1\(c\)](#). Don't fall into this innocent-looking trap. Connecting the dots can mislead the beginner into forgetting that the $x(n)$ sequence is nothing more than a list of numbers. Remember, $x(n)$ is a discrete-time sequence of individual values, and each value in that sequence plots as a single dot. It's not that we're ignorant of what lies between the dots of $x(n)$; there *is* nothing between those dots.

We can reinforce this discrete-time sequence concept by listing those [Figure 1-1\(b\)](#) sampled values as follows:

(1-2)

$x(0) = 0$	(1st sequence value, index $n = 0$)
$x(1) = 0.31$	(2nd sequence value, index $n = 1$)
$x(2) = 0.59$	(3rd sequence value, index $n = 2$)
$x(3) = 0.81$	(4th sequence value, index $n = 3$)
...	...
and so on,	

where n represents the time index integer sequence 0, 1, 2, 3, etc., and t_s is some constant time period between samples. Those sample values can be represented collectively, and concisely, by the discrete-time expression

(1-3)

$$x(n) = \sin(2\pi f_0 n t_s).$$

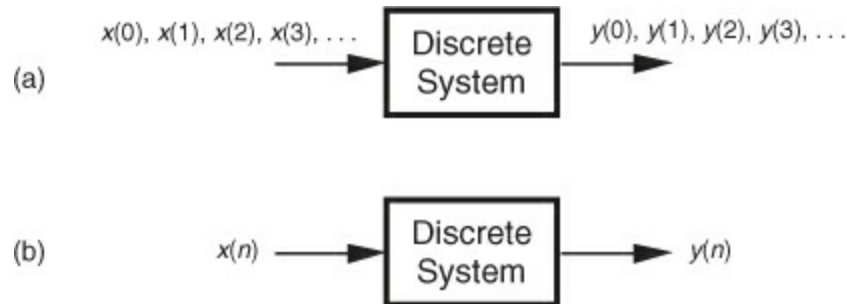
(Here again, the $2\pi f_0 n t_s$ term is an angle measured in radians.) Notice that the index n in [Eq. \(1-2\)](#) started with a value of 0, instead of 1. There's nothing sacred about this; the first value of n could just as well have been 1, but we start the index n at zero out of habit because doing so allows us to describe the sinewave starting at time zero. The variable $x(n)$ in [Eq. \(1-3\)](#) is read as "the sequence x of n ." [Equations \(1-1\)](#) and [\(1-3\)](#) describe what are also referred to as *time-domain* signals because the independent variables, the continuous time t in [Eq. \(1-1\)](#), and the discrete-time $n t_s$ values used in [Eq. \(1-3\)](#) are measures of time.

With this notion of a discrete-time signal in mind, let's say that a discrete system is a collection of hardware components, or software routines, that operate on a discrete-time signal sequence. For example, a discrete system could be a process that gives us a discrete output sequence $y(0)$, $y(1)$, $y(2)$, etc., when a discrete input sequence of $x(0)$, $x(1)$, $x(2)$, etc., is applied to the system input as shown in [Figure 1-2\(a\)](#). Again, to keep the notation concise and still keep track of individual elements of the input and output sequences, an abbreviated notation is used as shown in [Figure 1-2\(b\)](#) where n represents the integer sequence 0, 1, 2, 3, etc. Thus, $x(n)$ and $y(n)$ are general variables that represent two separate sequences of numbers. [Figure 1-2\(b\)](#) allows us to describe a system's output with a simple expression such as

$$(1-4)$$

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

Figure 1-2 With an input applied, a discrete system provides an output: (a) the input and output are sequences of individual values; (b) input and output using the abbreviated notation of $x(n)$ and $y(n)$.



Illustrating [Eq. \(1-4\)](#), if $x(n)$ is the five-element sequence $x(0) = 1$, $x(1) = 3$, $x(2) = 5$, $x(3) = 7$, and $x(4) = 9$, then $y(n)$ is the five-element sequence $y(0) = 1$, $y(1) = 5$, $y(2) = 9$, $y(3) = 13$, and $y(4) = 17$.

[Equation \(1-4\)](#) is formally called a *difference equation*. (In this book we won't be working with differential equations or partial differential equations. However, we will, now and then, work with partially difficult equations.)

The fundamental difference between the way time is represented in continuous and discrete systems leads to a very important difference in how we characterize frequency in continuous and discrete systems. To illustrate, let's reconsider the continuous sinewave in [Figure 1-1\(a\)](#). If it represented a voltage at the end of a cable, we could measure its frequency by applying it to an oscilloscope, a spectrum analyzer, or a frequency counter. We'd have a problem, however, if we were merely given the list of values from [Eq. \(1-2\)](#) and asked to determine the frequency of the waveform they represent. We'd graph those discrete values, and, sure enough, we'd recognize a single sinewave as in [Figure 1-1\(b\)](#). We can say that the sinewave repeats every 20 samples, but there's no way to determine the exact sinewave frequency from the discrete sequence values alone. You can probably see the point

we're leading to here. If we knew the time between samples—the sample period t_s —we'd be able to determine the absolute frequency of the discrete sinewave. Given that the t_s sample period is, say, 0.05 milliseconds/sample, the period of the sinewave is

(1-5)

$$\text{sinewave period} = \frac{20 \text{ samples}}{\text{period}} \cdot \frac{0.05 \text{ milliseconds}}{\text{sample}} = 1 \text{ millisecond.}$$

Because the frequency of a sinewave is the reciprocal of its period, we now know that the sinewave's absolute frequency is $1/(1 \text{ ms})$, or 1 kHz. On the other hand, if we found that the sample period was, in fact, 2 milliseconds, the discrete samples in [Figure 1-1\(b\)](#) would represent a sinewave whose period is 40 milliseconds and whose frequency is 25 Hz. The point here is that when dealing with discrete systems, absolute frequency determination in Hz is dependent on the sampling frequency

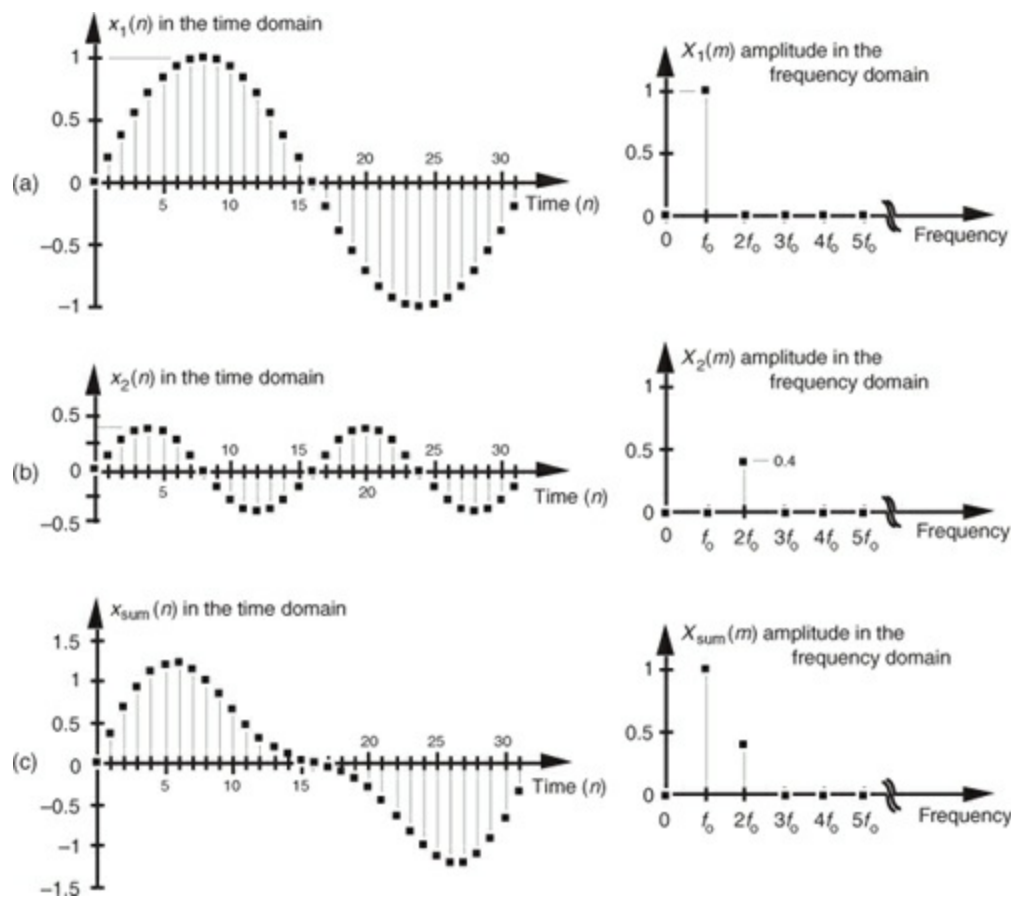
(1-5')

$$f_s = 1/t_s.$$

We'll be reminded of this dependence throughout the remainder of this book.

In digital signal processing, we often find it necessary to characterize the frequency content of discrete time-domain signals. When we do so, this frequency representation takes place in what's called the *frequency domain*. By way of example, let's say we have a discrete sinewave sequence $x_1(n)$ with an arbitrary frequency f_0 Hz as shown on the left side of [Figure 1-3\(a\)](#). We can also characterize $x_1(n)$ by showing its spectral content, the $X_1(m)$ sequence on the right side of [Figure 1-3\(a\)](#), indicating that it has a single spectral component, and no other frequency content. Although we won't dwell on it just now, notice that the frequency-domain representations in [Figure 1-3](#) are themselves discrete.

Figure 1-3 Time- and frequency-domain graphical representations: (a) sinewave of frequency f_0 ; (b) reduced amplitude sinewave of frequency $2f_0$; (c) sum of the two sinewaves.



To illustrate our time- and frequency-domain representations further, [Figure 1-3\(b\)](#) shows another discrete sinewave $x_2(n)$, whose peak amplitude is 0.4, with a frequency of $2f_0$. The discrete sample values of $x_2(n)$ are expressed by the equation

$$(1-6)$$

$$x_2(n) = 0.4 \cdot \sin(2\pi 2f_0 n t_s).$$

When the two sinewaves, $x_1(n)$ and $x_2(n)$, are added to produce a new waveform $x_{\text{sum}}(n)$, its time-domain equation is

$$(1-7)$$

$$x_{\text{sum}}(n) = x_1(n) + x_2(n) = \sin(2\pi f_0 n t_s) + 0.4 \cdot \sin(2\pi 2f_0 n t_s),$$

and its time- and frequency-domain representations are those given in [Figure 1-3\(c\)](#). We interpret the $X_{\text{sum}}(m)$ frequency-domain depiction, the *spectrum*, in [Figure 1-3\(c\)](#) to indicate that $x_{\text{sum}}(n)$ has a frequency component of f_0 Hz and a reduced-amplitude frequency component of $2f_0$ Hz.

Notice three things in [Figure 1-3](#). First, time sequences use lowercase variable names like the “ x ” in $x_1(n)$, and uppercase symbols for frequency-domain variables such as the “ X ” in $X_1(m)$. The term $X_1(m)$ is read as “the spectral sequence X sub one of m .” Second, because the $X_1(m)$ frequency-domain representation of the $x_1(n)$ time sequence is itself a sequence (a list of numbers), we use the index “ m ” to keep track of individual elements in $X_1(m)$. We can list frequency-domain sequences just as we did with the time sequence in [Eq. \(1-2\)](#). For example, $X_{\text{sum}}(m)$ is listed as

$$\begin{array}{ll}
X_{\text{sum}}(0) = 0 & \text{(1st } X_{\text{sum}}(m) \text{ value, index } m = 0) \\
X_{\text{sum}}(1) = 1.0 & \text{(2nd } X_{\text{sum}}(m) \text{ value, index } m = 1) \\
X_{\text{sum}}(2) = 0.4 & \text{(3rd } X_{\text{sum}}(m) \text{ value, index } m = 2) \\
X_{\text{sum}}(3) = 0 & \text{(4th } X_{\text{sum}}(m) \text{ value, index } m = 3) \\
\vdots & \vdots
\end{array}$$

and so on,

where the frequency index m is the integer sequence 0, 1, 2, 3, etc. Third, because the $x_1(n) + x_2(n)$ sinewaves have a phase shift of zero degrees relative to each other, we didn't really need to bother depicting this phase relationship in $X_{\text{sum}}(m)$ in [Figure 1-3\(c\)](#). In general, however, phase relationships in frequency-domain sequences are important, and we'll cover that subject in [Chapters 3](#) and [5](#).

A key point to keep in mind here is that we now know three equivalent ways to describe a discrete-time waveform. Mathematically, we can use a time-domain equation like [Eq. \(1-6\)](#). We can also represent a time-domain waveform graphically as we did on the left side of [Figure 1-3](#), and we can depict its corresponding, discrete, frequency-domain equivalent as that on the right side of [Figure 1-3](#).

As it turns out, the discrete time-domain signals we're concerned with are not only quantized in time; their amplitude values are also quantized. Because we represent all digital quantities with binary numbers, there's a limit to the resolution, or granularity, that we have in representing the values of discrete numbers. Although signal amplitude quantization can be an important consideration—we cover that particular topic in [Chapter 12](#)—we won't worry about it just now.

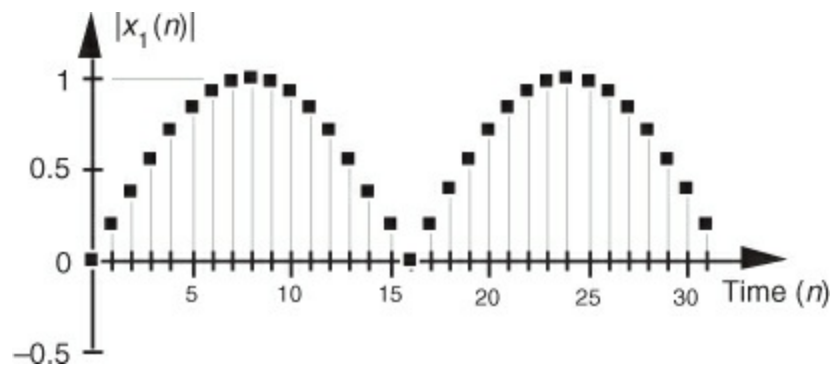
1.2 Signal Amplitude, Magnitude, Power

Let's define two important terms that we'll be using throughout this book: *amplitude* and *magnitude*. It's not surprising that, to the layman, these terms are typically used interchangeably. When we check our thesaurus, we find that they are synonymous.[‡] In engineering, however, they mean two different things, and we must keep that difference clear in our discussions. The amplitude of a variable is the measure of how far, and in what direction, that variable differs from zero. Thus, signal amplitudes can be either positive or negative. The time-domain sequences in [Figure 1-3](#) presented the sample value amplitudes of three different waveforms. Notice how some of the individual discrete amplitude values were positive and others were negative.

[‡] Of course, laymen are "other people." To the engineer, the brain surgeon is the layman. To the brain surgeon, the engineer is the layman.

The magnitude of a variable, on the other hand, is the measure of how far, regardless of direction, its quantity differs from zero. So magnitudes are always positive values. [Figure 1-4](#) illustrates how the magnitude of the $x_1(n)$ time sequence in [Figure 1-3\(a\)](#) is equal to the amplitude, but with the sign always being positive for the magnitude. We use the modulus symbol ($||$) to represent the magnitude of $x_1(n)$. Occasionally, in the literature of digital signal processing, we'll find the term *magnitude* referred to as the *absolute value*.

Figure 1-4 Magnitude samples, $|x_1(n)|$, of the time waveform in [Figure 1-3\(a\)](#).



When we examine signals in the frequency domain, we'll often be interested in the power level of those signals. The power of a signal is proportional to its amplitude (or magnitude) squared. If we assume that the proportionality constant is one, we can express the power of a sequence in the time or frequency domains as

(1-8)

$$x_{\text{pwr}}(n) = |x(n)|^2,$$

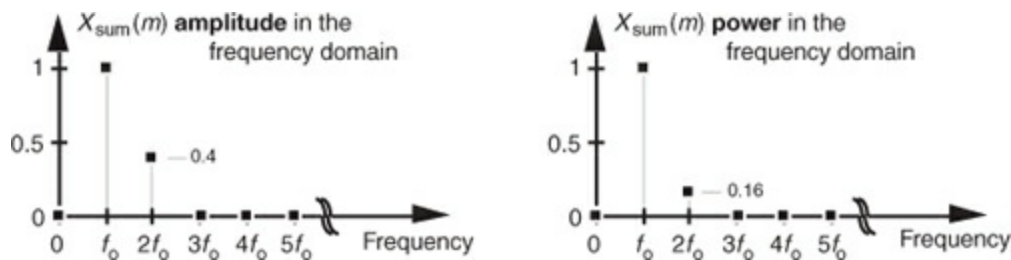
or

(1-8')

$$X_{\text{pwr}}(m) = |X(m)|^2.$$

Very often we'll want to know the difference in power levels of two signals in the frequency domain. Because of the squared nature of power, two signals with moderately different amplitudes will have a much larger difference in their relative powers. In [Figure 1-3](#), for example, signal $x_1(n)$'s amplitude is 2.5 times the amplitude of signal $x_2(n)$, but its power level is 6.25 that of $x_2(n)$'s power level. This is illustrated in [Figure 1-5](#) where both the amplitude and power of $X_{\text{sum}}(m)$ are shown.

Figure 1-5 Frequency-domain amplitude and frequency-domain power of the $x_{\text{sum}}(n)$ time waveform in [Figure 1-3\(c\)](#).

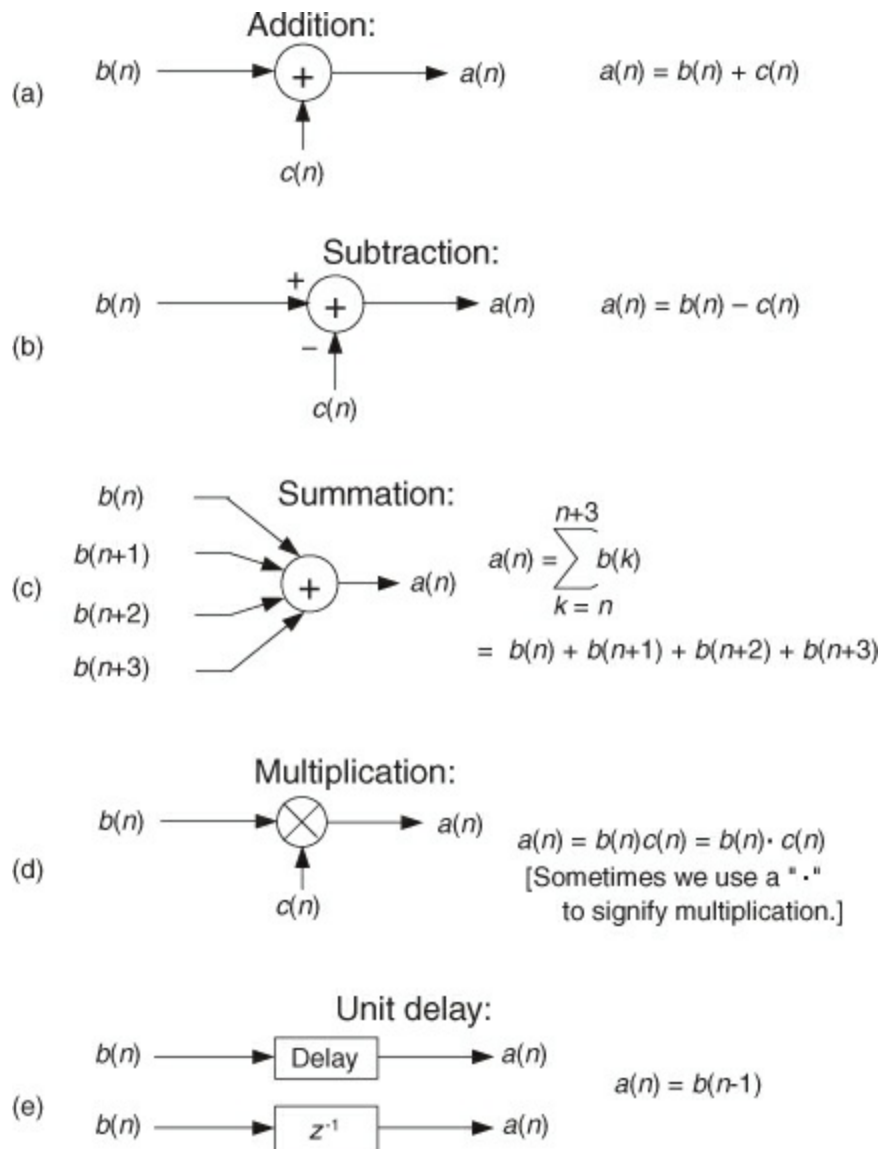


Because of their squared nature, plots of power values often involve showing both very large and very small values on the same graph. To make these plots easier to generate and evaluate, practitioners usually employ the decibel scale as described in [Appendix E](#).

1.3 Signal Processing Operational Symbols

We'll be using block diagrams to graphically depict the way digital signal processing operations are implemented. Those block diagrams will comprise an assortment of fundamental processing symbols, the most common of which are illustrated and mathematically defined in [Figure 1-6](#).

Figure 1-6 Terminology and symbols used in digital signal processing block diagrams.



[Figure 1-6\(a\)](#) shows the addition, element for element, of two discrete sequences to provide a new sequence. If our sequence index n begins at 0, we say that the first output sequence value is equal to the sum of the first element of the b sequence and the first element of the c sequence, or $a(0) = b(0) + c(0)$. Likewise, the second output sequence value is equal to the sum of the second element of the b sequence and the second element of the c sequence, or $a(1) = b(1) + c(1)$. [Equation \(1-7\)](#) is an example of adding two sequences. The subtraction process in [Figure 1-6\(b\)](#) generates an output sequence that's the element-for-element difference of the two input sequences. There are times when we must calculate a sequence whose elements are the sum of more than two values. This operation, illustrated in [Figure 1-6\(c\)](#), is called *summation* and is very common in digital signal processing. Notice how the lower and upper limits of the summation index k in the expression in [Figure 1-6\(c\)](#) tell us exactly which elements of the b sequence to sum to obtain a given $a(n)$ value. Because we'll encounter summation operations so often, let's make sure we understand their notation. If we repeat the summation equation from [Figure 1-6\(c\)](#) here, we have

(1-9)

$$a(n) = \sum_{k=n}^{n+3} b(k).$$

This means that

(1-10)

when $n = 0$, index k goes from 0 to 3, so	$a(0) = b(0) + b(1) + b(2) + b(3)$
when $n = 1$, index k goes from 1 to 4, so	$a(1) = b(1) + b(2) + b(3) + b(4)$
when $n = 2$, index k goes from 2 to 5, so	$a(2) = b(2) + b(3) + b(4) + b(5)$
when $n = 3$, index k goes from 3 to 6, so	$a(3) = b(3) + b(4) + b(5) + b(6)$
...	...
and so on.	

We'll begin using summation operations in earnest when we discuss digital filters in [Chapter 5](#).

The multiplication of two sequences is symbolized in [Figure 1-6\(d\)](#). Multiplication generates an output sequence that's the element-for-element product of two input sequences: $a(0) = b(0)c(0)$, $a(1) = b(1)c(1)$, and so on. The last fundamental operation that we'll be using is called the *unit delay* in [Figure 1-6\(e\)](#). While we don't need to appreciate its importance at this point, we'll merely state that the unit delay symbol signifies an operation where the output sequence $a(n)$ is equal to a delayed version of the $b(n)$ sequence. For example, $a(5) = b(4)$, $a(6) = b(5)$, $a(7) = b(6)$, etc. As we'll see in [Chapter 6](#), due to the mathematical techniques used to analyze digital filters, the unit delay is very often depicted using the term z^{-1} .

The symbols in [Figure 1-6](#) remind us of two important aspects of digital signal processing. First, our processing operations are always performed on sequences of individual discrete values, and second, the elementary operations themselves are very simple. It's interesting that, regardless of how complicated they appear to be, the vast majority of digital signal processing algorithms can be performed using combinations of these simple operations. If we think of a digital signal processing algorithm as a recipe, then the symbols in [Figure 1-6](#) are the ingredients.

1.4 Introduction to Discrete Linear Time-Invariant Systems

In keeping with tradition, we'll introduce the subject of linear time-invariant (LTI) systems at this early point in our text. Although an appreciation for LTI systems is not essential in studying the next three chapters of this book, when we begin exploring digital filters, we'll build on the strict definitions of linearity and time invariance. We need to recognize and understand the notions of linearity and time invariance not just because the vast majority of discrete systems used in practice are LTI systems, but because LTI systems are very accommodating when it comes to their analysis. That's good news for us because we can use straightforward methods to predict the performance of any digital signal processing scheme as long as it's linear and time invariant. Because linearity and time invariance are two important system characteristics having very special properties, we'll discuss them now.

1.5 Discrete Linear Systems

The term *linear* defines a special class of systems where the output is the superposition, or sum, of the individual outputs had the individual inputs been applied separately to the system. For example, we can say that the application of an input $x_1(n)$ to a system results in an output $y_1(n)$. We symbolize this situation with the following expression:

$$(1-11)$$

$$x_1(n) \xrightarrow{\text{results in}} y_1(n).$$

Given a different input $x_2(n)$, the system has a $y_2(n)$ output as

$$(1-12)$$

$$x_2(n) \xrightarrow{\text{results in}} y_2(n).$$

For the system to be linear, when its input is the sum $x_1(n) + x_2(n)$, its output must be the sum of the individual outputs so that

(1-13)

$$x_1(n) + x_2(n) \xrightarrow{\text{results in}} y_1(n) + y_2(n).$$

One way to paraphrase expression (1-13) is to state that a linear system's output is the sum of the outputs of its parts. Also, part of this description of linearity is a proportionality characteristic. This means that if the inputs are scaled by constant factors c_1 and c_2 , then the output sequence parts are also scaled by those factors as

(1-14)

$$c_1x_1(n) + c_2x_2(n) \xrightarrow{\text{results in}} c_1y_1(n) + c_2y_2(n).$$

In the literature, this proportionality attribute of linear systems in expression (1-14) is sometimes called the *homogeneity property*. With these thoughts in mind, then, let's demonstrate the concept of system linearity.

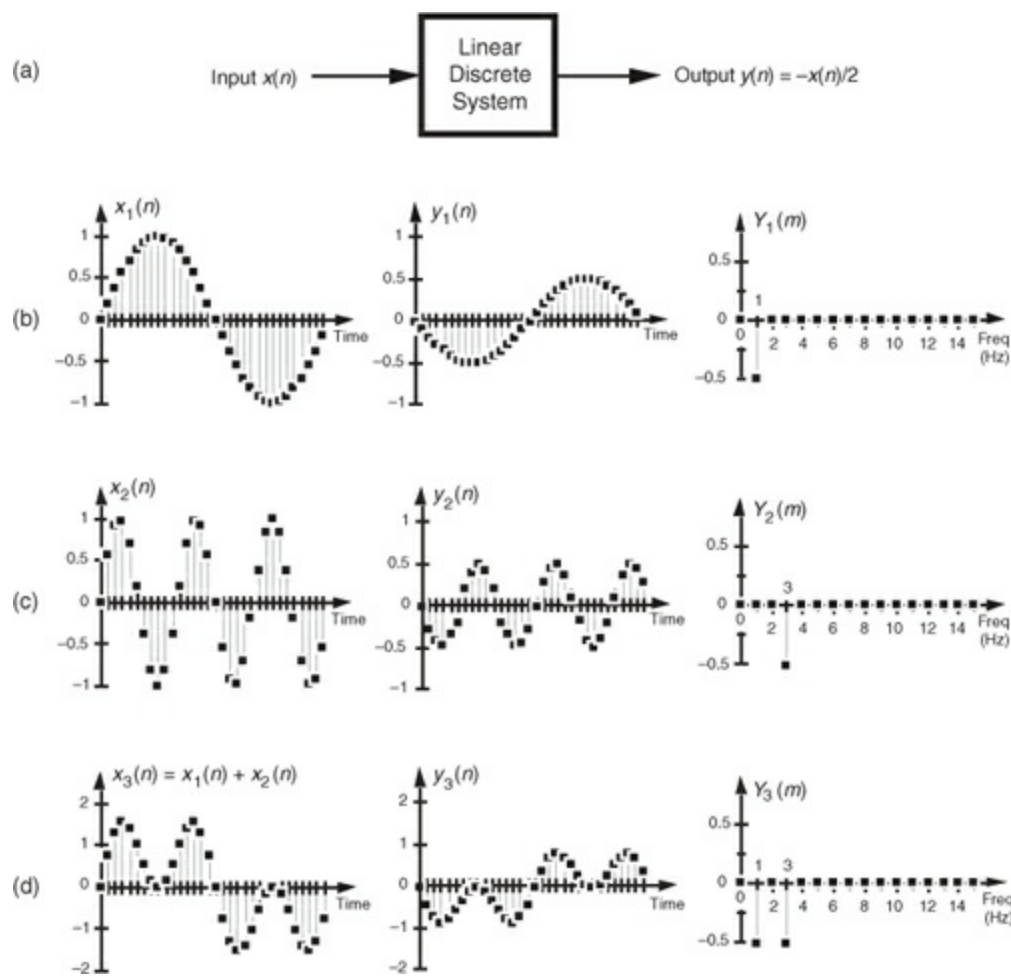
1.5.1 Example of a Linear System

To illustrate system linearity, let's say we have the discrete system shown in Figure 1-7(a) whose output is defined as

(1-15)

$$y(n) = \frac{-x(n)}{2},$$

Figure 1-7 Linear system input-to-output relationships: (a) system block diagram where $y(n) = -x(n)/2$; (b) system input and output with a 1 Hz sinewave applied; (c) with a 3 Hz sinewave applied; (d) with the sum of 1 Hz and 3 Hz sinewaves applied.



that is, the output sequence is equal to the negative of the input sequence with the amplitude reduced by a factor of two. If we apply an $x_1(n)$ input sequence representing a 1 Hz sinewave sampled at a rate of 32 samples per cycle, we'll have a $y_1(n)$ output as shown in the center of [Figure 1-7\(b\)](#). The frequency-domain spectral amplitude of the $y_1(n)$ output is the plot on the right side of [Figure 1-7\(b\)](#), indicating that the output comprises a single tone of peak amplitude equal to -0.5 whose frequency is 1 Hz. Next, applying an $x_2(n)$ input sequence representing a 3 Hz sinewave, the system provides a $y_2(n)$ output sequence, as shown in the center of [Figure 1-7\(c\)](#). The spectrum of the $y_2(n)$ output, $Y_2(m)$, confirming a single 3 Hz sinewave output is shown on the right side of [Figure 1-7\(c\)](#). Finally—here's where the linearity comes in—if we apply an $x_3(n)$ input sequence that's the sum of a 1 Hz sinewave and a 3 Hz sinewave, the $y_3(n)$ output is as shown in the center of [Figure 1-7\(d\)](#). Notice how $y_3(n)$ is the sample-for-sample sum of $y_1(n)$ and $y_2(n)$. [Figure 1-7\(d\)](#) also shows that the output spectrum $Y_3(m)$ is the sum of $Y_1(m)$ and $Y_2(m)$. That's linearity.

1.5.2 Example of a Nonlinear System

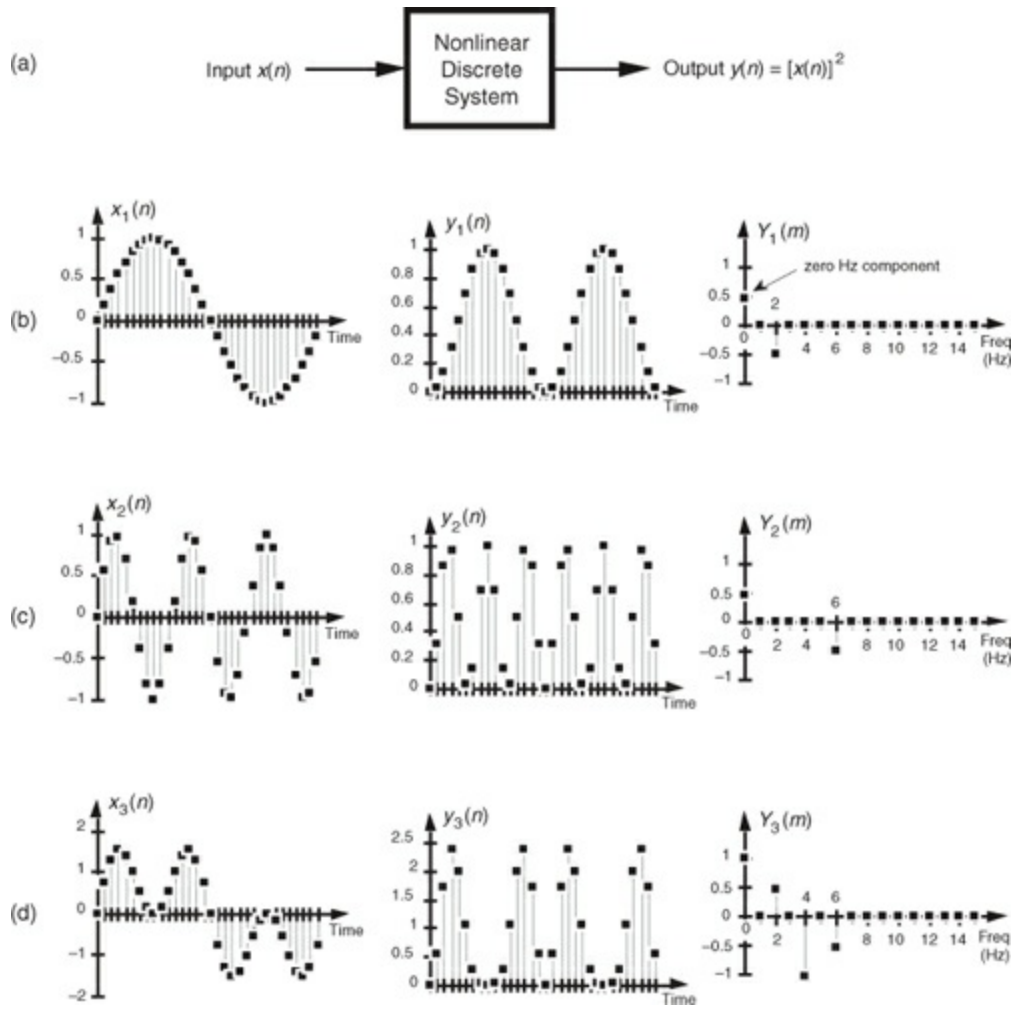
It's easy to demonstrate how a nonlinear system yields an output that is not equal to the sum of $y_1(n)$ and $y_2(n)$ when its input is $x_1(n) + x_2(n)$. A simple example of a nonlinear discrete system is that in [Figure 1-8\(a\)](#) where the output is the square of the input described by

$$(1-16)$$

$$y(n) = [x(n)]^2.$$

Figure 1-8 Nonlinear system input-to-output relationships: (a) system block diagram where $y(n) = [x(n)]^2$; (b) system input and output with a 1 Hz sinewave applied; (c) with a 3 Hz sinewave applied;

(d) with the sum of 1 Hz and 3 Hz sinewaves applied.



We'll use a well-known trigonometric identity and a little algebra to predict the output of this nonlinear system when the input comprises simple sinewaves. Following the form of [Eq. \(1-3\)](#), let's describe a sinusoidal sequence, whose frequency $f_o = 1$ Hz, by

$$(1-17)$$

$$x_1(n) = \sin(2\pi f_o n t_s) = \sin(2\pi \cdot 1 \cdot n t_s).$$

[Equation \(1-17\)](#) describes the $x_1(n)$ sequence on the left side of [Figure 1-8\(b\)](#). Given this $x_1(n)$ input sequence, the $y_1(n)$ output of the nonlinear system is the square of a 1 Hz sinewave, or

$$(1-18)$$

$$y_1(n) = [x_1(n)]^2 = \sin(2\pi \cdot 1 \cdot n t_s) \cdot \sin(2\pi \cdot 1 \cdot n t_s).$$

We can simplify our expression for $y_1(n)$ in [Eq. \(1-18\)](#) by using the following trigonometric identity:

$$(1-19)$$

$$\sin(\alpha) \cdot \sin(\beta) = \frac{\cos(\alpha - \beta)}{2} - \frac{\cos(\alpha + \beta)}{2}.$$

Using [Eq. \(1-19\)](#), we can express $y_1(n)$ as

$$(1-20)$$

$$y_1(n) = \frac{\cos(2\pi \cdot 1 \cdot nt_s - 2\pi \cdot 1 \cdot nt_s)}{2} - \frac{\cos(2\pi \cdot 1 \cdot nt_s + 2\pi \cdot 1 \cdot nt_s)}{2}$$

$$= \frac{\cos(0)}{2} - \frac{\cos(4\pi \cdot 1 \cdot nt_s)}{2} = \frac{1}{2} - \frac{\cos(2\pi \cdot 2 \cdot nt_s)}{2},$$

which is shown as the all-positive sequence in the center of [Figure 1-8\(b\)](#). Because [Eq. \(1-19\)](#) results in a frequency sum ($\alpha + \beta$) and frequency difference ($\alpha - \beta$) effect when multiplying two sinusoids, the $y_1(n)$ output sequence will be a cosine wave of 2 Hz and a peak amplitude of -0.5 , added to a constant value of $1/2$. The constant value of $1/2$ in [Eq. \(1-20\)](#) is interpreted as a zero Hz frequency component, as shown in the $Y_1(m)$ spectrum in [Figure 1-8\(b\)](#). We could go through the same algebraic exercise to determine that when a 3 Hz sinewave $x_2(n)$ sequence is applied to this nonlinear system, the output $y_2(n)$ would contain a zero Hz component and a 6 Hz component, as shown in [Figure 1-8\(c\)](#).

System nonlinearity is evident if we apply an $x_3(n)$ sequence comprising the sum of a 1 Hz and a 3 Hz sinewave as shown in [Figure 1-8\(d\)](#). We can predict the frequency content of the $y_3(n)$ output sequence by using the algebraic relationship

$$(1-21)$$

$$(a+b)^2 = a^2 + 2ab + b^2,$$

where a and b represent the 1 Hz and 3 Hz sinewaves, respectively. From [Eq. \(1-19\)](#), the a^2 term in [Eq. \(1-21\)](#) generates the zero Hz and 2 Hz output sinusoids in [Figure 1-8\(b\)](#). Likewise, the b^2 term produces in $y_3(n)$ another zero Hz and the 6 Hz sinusoid in [Figure 1-8\(c\)](#). However, the $2ab$ term yields additional 2 Hz and 4 Hz sinusoids in $y_3(n)$. We can show this algebraically by using [Eq. \(1-19\)](#) and expressing the $2ab$ term in [Eq. \(1-21\)](#) as

$$(1-22)$$

$$2ab = 2 \cdot \sin(2\pi \cdot 1 \cdot nt_s) \cdot \sin(2\pi \cdot 3 \cdot nt_s)$$

$$= \frac{2 \cos(2\pi \cdot 1 \cdot nt_s - 2\pi \cdot 3 \cdot nt_s)}{2} - \frac{2 \cos(2\pi \cdot 1 \cdot nt_s + 2\pi \cdot 3 \cdot nt_s)}{2}$$

$$= \cos(2\pi \cdot 2 \cdot nt_s) - \cos(2\pi \cdot 4 \cdot nt_s).^\dagger$$

[†] The first term in [Eq. \(1-22\)](#) is $\cos(2\pi \cdot nt_s - 6\pi \cdot nt_s) = \cos(-4\pi \cdot nt_s) = \cos(-2\pi \cdot 2 \cdot nt_s)$. However, because the cosine function is even, $\cos(-\alpha) = \cos(\alpha)$, we can express that first term as $\cos(2\pi \cdot 2 \cdot nt_s)$.

[Equation \(1-22\)](#) tells us that two additional sinusoidal components will be present in $y_3(n)$ because of the system's nonlinearity, a 2 Hz cosine wave whose amplitude is $+1$ and a 4 Hz cosine wave having an amplitude of -1 . These spectral components are illustrated in $Y_3(m)$ on the right side of [Figure 1-8\(d\)](#).

Notice that when the sum of the two sinewaves is applied to the nonlinear system, the output contained sinusoids, [Eq. \(1-22\)](#), that were not present in either of the outputs when the individual sinewaves alone were applied. Those extra sinusoids were generated by an interaction of the two input sinusoids due to the squaring operation. That's nonlinearity; expression [\(1-13\)](#) was not satisfied. (Electrical engineers recognize this effect of internally generated sinusoids as *intermodulation*

distortion.) Although nonlinear systems are usually difficult to analyze, they are occasionally used in practice. References [2], [3], and [4], for example, describe their application in nonlinear digital filters. Again, expressions (1-13) and (1-14) state that a linear system's output resulting from a sum of individual inputs is the superposition (sum) of the individual outputs. They also stipulate that the output sequence $y_1(n)$ depends only on $x_1(n)$ combined with the system characteristics, and not on the other input $x_2(n)$; i.e., there's no interaction between inputs $x_1(n)$ and $x_2(n)$ at the output of a linear system.

1.6 Time-Invariant Systems

A time-invariant system is one where a time delay (or shift) in the input sequence causes an equivalent time delay in the system's output sequence. Keeping in mind that n is just an indexing variable we use to keep track of our input and output samples, let's say a system provides an output $y(n)$ given an input of $x(n)$, or

$$(1-23)$$

$$x(n) \xrightarrow{\text{results in}} y(n).$$

For a system to be time invariant, with a shifted version of the original $x(n)$ input applied, $x'(n)$, the following applies:

$$(1-24)$$

$$x'(n) = x(n+k) \xrightarrow{\text{results in}} y'(n) = y(n+k),$$

where k is some integer representing k sample period time delays. For a system to be time invariant, Eq. (1-24) must hold true for any integer value of k and any input sequence.

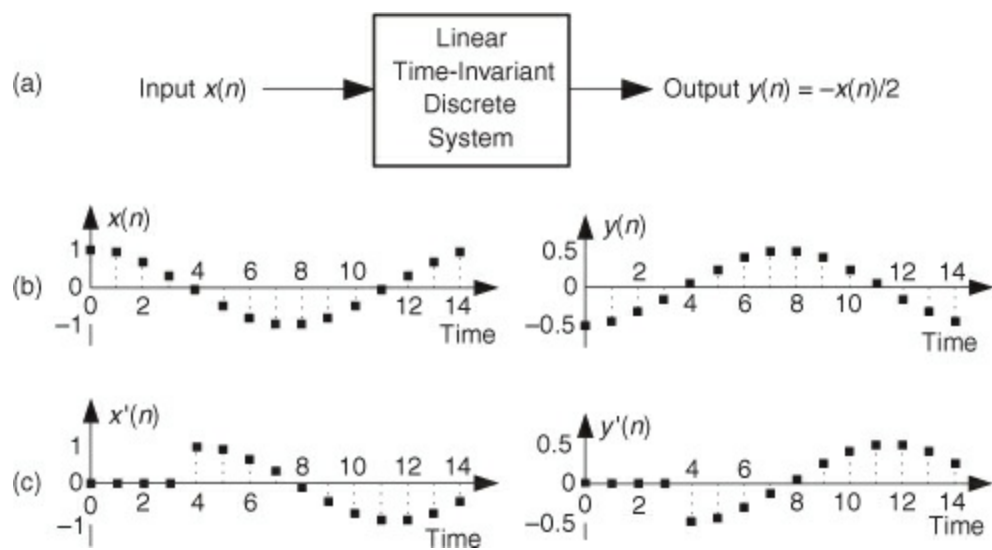
1.6.1 Example of a Time-Invariant System

Let's look at a simple example of time invariance illustrated in Figure 1-9. Assume that our initial $x(n)$ input is a unity-amplitude 1 Hz sinewave sequence with a $y(n)$ output, as shown in Figure 1-9(b). Consider a different input sequence $x'(n)$, where

$$(1-25)$$

$$x'(n) = x(n-4).$$

Figure 1-9 Time-invariant system input/output relationships: (a) system block diagram, $y(n) = -x(n)/2$; (b) system input/output with a sinewave input; (c) input/output when a sinewave, delayed by four samples, is the input.



Equation (1-25) tells us that the input sequence $x'(n)$ is equal to sequence $x(n)$ shifted to the right by $k = -4$ samples. That is, $x'(4) = x(0)$, $x'(5) = x(1)$, $x'(6) = x(2)$, and so on as shown in Figure 1-9(c). The discrete system is time invariant because the $y'(n)$ output sequence is equal to the $y(n)$ sequence shifted to the right by four samples, or $y'(n) = y(n-4)$. We can see that $y'(4) = y(0)$, $y'(5) = y(1)$, $y'(6) = y(2)$, and so on as shown in Figure 1-9(c). For time-invariant systems, the time shifts in $x'(n)$ and $y'(n)$ are equal. Take careful notice of the minus sign in Eq. (1-25). In later chapters, that is the notation we'll use to algebraically describe a time-delayed discrete sequence.

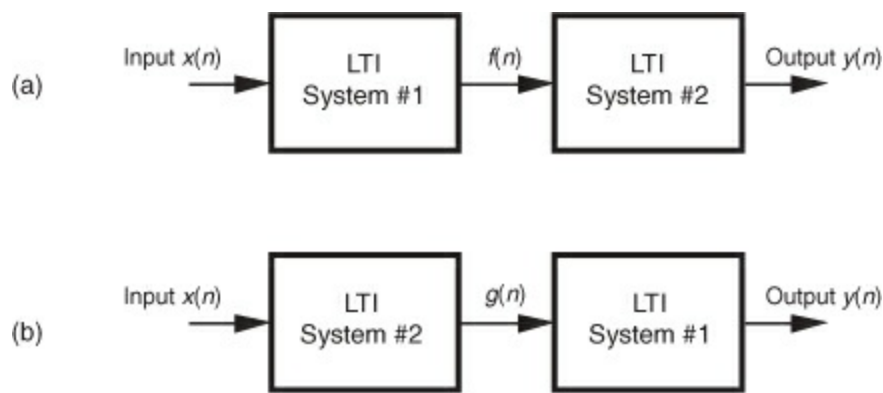
Some authors succumb to the urge to define a time-invariant system as one whose parameters do not change with time. That definition is incomplete and can get us in trouble if we're not careful. We'll just stick with the formal definition that a time-invariant system is one where a time shift in an input sequence results in an equal time shift in the output sequence. By the way, time-invariant systems in the literature are often called *shift-invariant* systems.[‡]

[‡] An example of a discrete process that's not time invariant is the downsampling, or decimation, process described in Chapter 10.

1.7 The Commutative Property of Linear Time-Invariant Systems

Although we don't substantiate this fact until we reach Section 6.11, it's not too early to realize that LTI systems have a useful commutative property by which their sequential order can be rearranged with no change in their final output. This situation is shown in Figure 1-10 where two different LTI systems are configured in series. Swapping the order of two cascaded systems does not alter the final output. Although the intermediate data sequences $f(n)$ and $g(n)$ will usually not be equal, the two pairs of LTI systems will have identical $y(n)$ output sequences. This commutative characteristic comes in handy for designers of digital filters, as we'll see in Chapters 5 and 6.

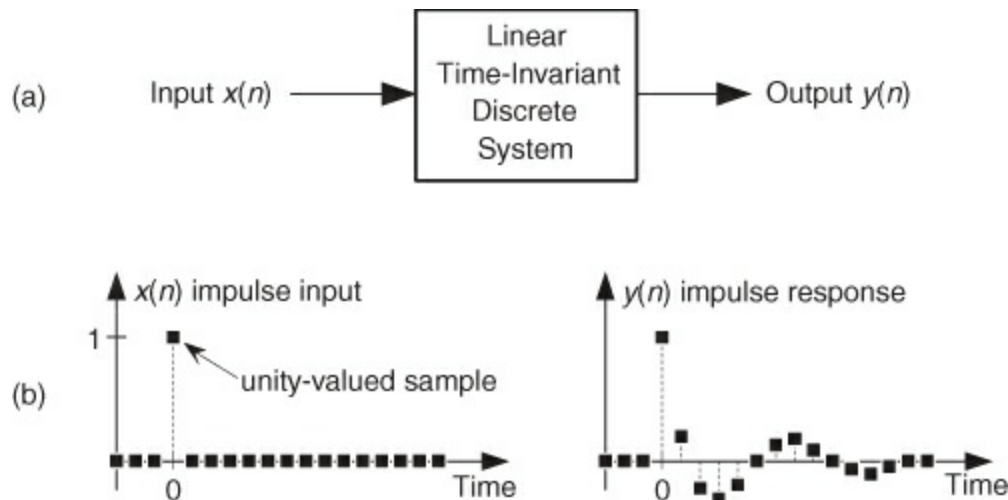
Figure 1-10 Linear time-invariant (LTI) systems in series: (a) block diagram of two LTI systems; (b) swapping the order of the two systems does not change the resultant output $y(n)$.



1.8 Analyzing Linear Time-Invariant Systems

As previously stated, LTI systems can be analyzed to predict their performance. Specifically, if we know the *unit impulse response* of an LTI system, we can calculate everything there is to know about the system; that is, the system's unit impulse response completely characterizes the system. By "unit impulse response" we mean the system's time-domain output sequence when the input is a single unity-valued sample (unit impulse) preceded and followed by zero-valued samples as shown in [Figure 1-11\(b\)](#).

Figure 1-11 LTI system unit impulse response sequences: (a) system block diagram; (b) impulse input sequence $x(n]$ and impulse response output sequence $y(n]$.



Knowing the (unit) impulse response of an LTI system, we can determine the system's output sequence for any input sequence because the output is equal to the *convolution* of the input sequence and the system's impulse response. Moreover, given an LTI system's time-domain impulse response, we can find the system's *frequency response* by taking the Fourier transform in the form of a *discrete Fourier transform* of that impulse response[5]. The concepts in the two previous sentences are among the most important principles in all of digital signal processing!

Don't be alarmed if you're not exactly sure what is meant by convolution, frequency response, or the discrete Fourier transform. We'll introduce these subjects and define them slowly and carefully as we need them in later chapters. The point to keep in mind here is that LTI systems can be designed and analyzed using a number of straightforward and powerful analysis techniques. These techniques will become tools that we'll add to our signal processing toolboxes as we journey through the subject of digital signal processing.

In the testing (analyzing) of continuous linear systems, engineers often use a narrow-in-time impulsive signal as an input signal to their systems. Mechanical engineers give their systems a little whack with

a hammer, and electrical engineers working with analog-voltage systems generate a very narrow voltage spike as an impulsive input. Audio engineers, who need an impulsive acoustic test signal, sometimes generate an audio impulse by firing a starter pistol.

In the world of DSP, an impulse sequence called a *unit impulse* takes the form

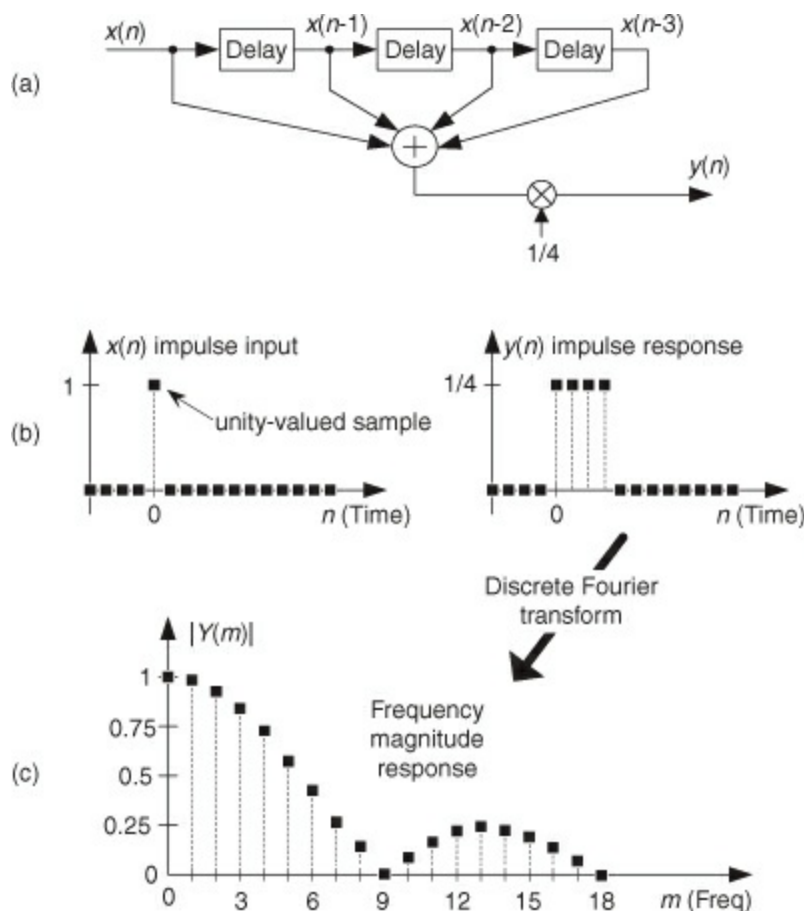
$$(1-26)$$

$$x(n) = \dots 0, 0, 0, 0, 0, A, 0, 0, 0, 0, 0, \dots$$

The value A is often set equal to one. The leading sequence of zero-valued samples, before the A -valued sample, must be a bit longer than the length of the transient response of the system under test in order to initialize the system to its zero state. The trailing sequence of zero-valued samples, following the A -valued sample, must be a bit longer than the transient response of the system under test in order to capture the system's entire $y(n)$ impulse response output sequence.

Let's further explore this notion of impulse response testing to determine the frequency response of a discrete system (and take an opportunity to start using the operational symbols introduced in [Section 1.3](#)). Consider the block diagram of a 4-point moving averager shown in [Figure 1-12\(a\)](#). As the $x(n)$ input samples march their way through the system, at each time index n four successive input samples are averaged to compute a single $y(n)$ output. As we'll learn in subsequent chapters, a *moving averager* behaves like a digital lowpass filter. However, we can quickly illustrate that fact now.

Figure 1-12 Analyzing a moving averager: (a) averager block diagram; (b) impulse input and impulse response; (c) averager frequency magnitude response.



If we apply an impulse input sequence to the system, we'll obtain its $y(n)$ impulse response output shown in [Figure 1-12\(b\)](#). The $y(n)$ output is computed using the following difference equation:

$$(1-27)$$

$$y(n) = \frac{1}{4}[x(n) + x(n-1) + x(n-2) + x(n-3)] = \frac{1}{4} \sum_{k=n-3}^n x(k).$$

If we then perform a discrete Fourier transform (a process we cover in much detail in [Chapter 3](#)) on $y(n)$, we obtain the $Y(m)$ frequency-domain information, allowing us to plot the frequency magnitude response of the 4-point moving averager as shown in [Figure 1-12\(c\)](#). So we see that a moving averager indeed has the characteristic of a lowpass filter. That is, the averager attenuates (reduces the amplitude of) high-frequency signal content applied to its input.

OK, this concludes our brief introduction to discrete sequences and systems. In later chapters we'll learn the details of discrete Fourier transforms, discrete system impulse responses, and digital filters.

References

- [1] Karplus, W. J., and Soroka, W. W. *Analog Methods*, 2nd ed., McGraw-Hill, New York, 1959, p. 117.
- [2] Mikami, N., Kobayashi, M., and Yokoyama, Y. "A New DSP-Oriented Algorithm for Calculation of the Square Root Using a Nonlinear Digital Filter," *IEEE Trans. on Signal Processing*, Vol. 40, No. 7, July 1992.
- [3] Heinen, P., and Neuvo, Y. "FIR-Median Hybrid Filters," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-35, No. 6, June 1987.
- [4] Oppenheim, A., Schaffer, R., and Stockham, T. "Nonlinear Filtering of Multiplied and Convolved Signals," *Proc. IEEE*, Vol. 56, August 1968.
- [5] Pickerd, John. "Impulse-Response Testing Lets a Single Test Do the Work of Thousands," *EDN*, April 27, 1995.

Chapter 1 Problems

1.1 This problem gives us practice in thinking about sequences of numbers. For centuries mathematicians have developed clever ways of computing π . In 1671 the Scottish mathematician James Gregory proposed the following very simple series for calculating π :

$$\pi \approx 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} \dots \right).$$

Thinking of the terms inside the parentheses as a sequence indexed by the variable n , where $n = 0, 1, 2, 3, \dots, 100$, write Gregory's algorithm in the form

$$\pi \approx 4 \cdot \sum_{n=0}^{100} (-1)^? \cdot ?$$

replacing the "?" characters with expressions in terms of index n .

1.2 One of the ways to obtain discrete sequences, for follow-on processing, is to digitize a continuous (analog) signal with an analog-to-digital (A/D) converter. A 6-bit A/D converter's output words (6-bit binary words) can only represent $2^6=64$ different numbers. (We cover this digitization, *sampling*, and A/D converters in detail in upcoming chapters.) Thus we say the A/D converter's "digital" output can only represent a finite number of amplitude values. Can you think of a continuous time-domain electrical signal that only has a finite number of amplitude values? If so, draw a graph of that continuous-time signal.

1.3 On the Internet, the author once encountered the following line of C-language code

$$\text{PI} = 2 * \text{asin}(1.0);$$

whose purpose was to define the constant π . In standard mathematical notation, that line of code can be described by

$$\pi = 2 \cdot \sin^{-1}(1).$$

Under what assumption does the above expression correctly define the constant π ?

1.4 Many times in the literature of signal processing you will encounter the identity

$$x^0 = 1.$$

That is, x raised to the zero power is equal to one. Using the Laws of Exponents, prove the above expression to be true.

1.5 Recall that for discrete sequences the t_s sample period (the time period between samples) is the reciprocal of the sample frequency f_s . Write the equations, as we did in the text's [Eq. \(1-3\)](#), describing time-domain sequences for unity-amplitude cosine waves whose f_o frequencies are

(a) $f_o = f_s/2$, one-half the sample rate,

(b) $f_o = f_s/4$, one-fourth the sample rate,

(c) $f_o = 0$ (zero) Hz.

1.6 Draw the three time-domain cosine wave sequences, where a sample value is represented by a dot, described in Problem 1.5. The correct solution to Part (a) of this problem is a useful sequence used to convert some lowpass digital filters into highpass filters. ([Chapter 5](#) discusses that topic.) The correct solution to Part (b) of this problem is an important discrete sequence used for *frequency translation* (both for signal *down-conversion* and *up-conversion*) in modern-day wireless communications systems. The correct solution to Part (c) of this problem should convince us that it's perfectly valid to describe a cosine sequence whose frequency is zero Hz.

1.7 Draw the three time-domain sequences of unity-amplitude sinewaves (not cosine waves) whose frequencies are

(a) $f_o = f_s/2$, one-half the sample rate,

(b) $f_o = f_s/4$, one-fourth the sample rate,

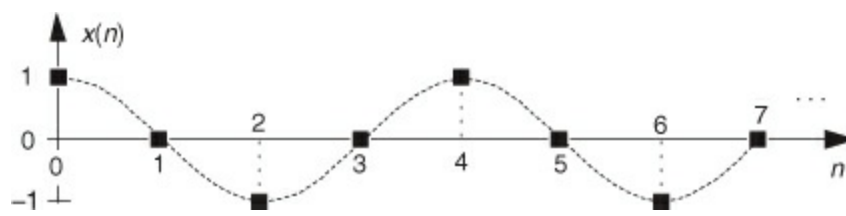
(c) $f_o = 0$ (zero) Hz.

The correct solutions to Parts (a) and (c) show us that the two frequencies, 0 Hz and $f_s/2$ Hz, are special frequencies in the world of discrete signal processing. What is *special* about the sinewave sequences obtained from the above Parts (a) and (c)?

1.8 Consider the infinite-length time-domain sequence $x(n)$ in [Figure P1-8](#). Draw the first eight samples of a shifted time sequence defined by

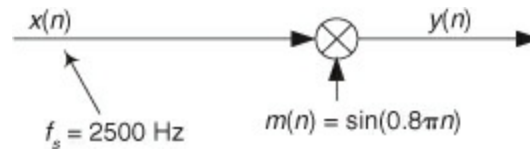
$$x_{\text{shift}}(n) = x(n+1).$$

Figure P1-8



- 1.9 Assume, during your reading of the literature of DSP, you encounter the process shown in [Figure P1-9](#). The $x(n)$ input sequence, whose f_s sample rate is 2500 Hz, is multiplied by a sinusoidal $m(n)$ sequence to produce the $y(n)$ output sequence. What is the frequency, measured in Hz, of the sinusoidal $m(n)$ sequence?

Figure P1-9



- 1.10 There is a process in DSP called an “ N -point running sum” (a kind of digital lowpass filter, actually) that is described by the following equation:

$$y(n) = \sum_{p=0}^{N-1} x(n-p).$$

Write out, giving the indices of all the $x()$ terms, the algebraic expression that describes the computations needed to compute $y(9)$ when $N=6$.

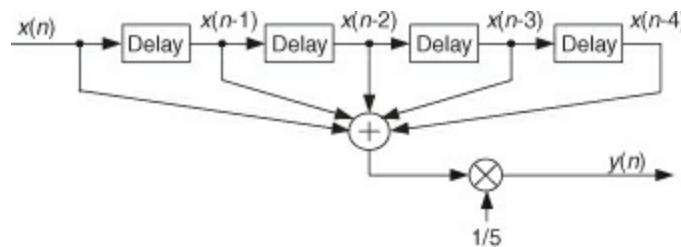
- 1.11 A 5-point *moving averager* can be described by the following difference equation:

(P1-1)

$$y(n) = \frac{1}{5}[x(n) + x(n-1) + x(n-2) + x(n-3) + x(n-4)] = \frac{1}{5} \sum_{k=n-4}^n x(k).$$

The averager’s signal-flow block diagram is shown in [Figure P1-11](#), where the $x(n)$ input samples flow through the averager from left to right.

Figure P1-11



[Equation \(P1-1\)](#) is equivalent to

(P1-2)

$$\begin{aligned} y(n) &= \frac{x(n)}{5} + \frac{x(n-1)}{5} + \frac{x(n-2)}{5} + \frac{x(n-3)}{5} + \frac{x(n-4)}{5} \\ &= \sum_{k=n-4}^n \frac{x(k)}{5}. \end{aligned}$$

(a) Draw the block diagram of the discrete system described by [Eq. \(P1-2\)](#).

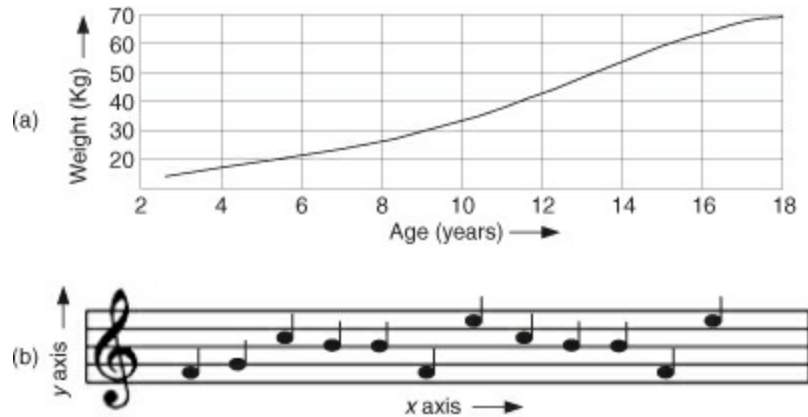
(b) The *moving average* processes described by [Eqs. \(P1-1\)](#) and [\(P1-2\)](#) have identical impulse

responses. Draw that impulse response.

(c) If you had to implement (using programmable hardware or assembling discrete hardware components) either [Eq. \(P1-1\)](#) or [Eq. \(P1-2\)](#), which would you choose? Explain why.

1.12 In this book we will look at many two-dimensional drawings showing the value of one variable (y) plotted as a function of another variable (x). Stated in different words, we'll graphically display what are the values of a y axis variable for various values of an x axis variable. For example, [Figure P1-12\(a\)](#) plots the weight of a male child as a function of the child's age. The dimension of the x axis is years and the dimension of the y axis is kilograms. What are the dimensions of the x and y axes of the familiar two-dimensional plot given in [Figure P1-12\(b\)](#)?

Figure P1-12

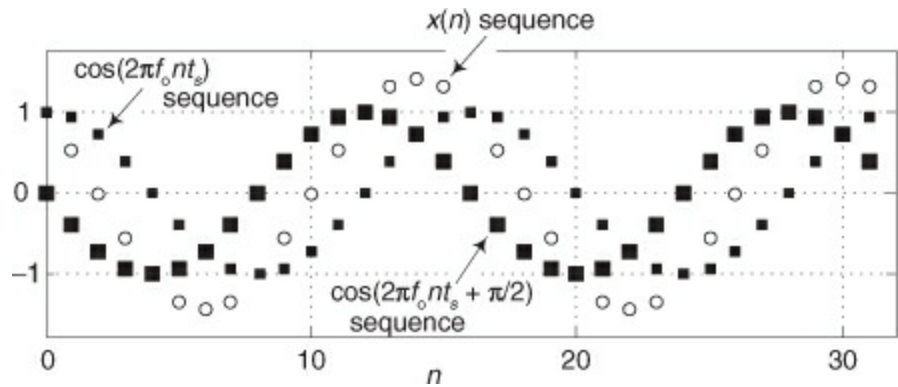


1.13 Let's say you are writing software code to generate an $x(n)$ test sequence composed of the sum of two equal-amplitude discrete cosine waves, as

$$x(n) = \cos(2\pi f_0 n t_s + \phi) + \cos(2\pi f_0 n t_s)$$

where t_s is the time between your $x(n)$ samples, and ϕ is a constant phase shift measured in radians. An example $x(n)$ when $\phi = \pi/2$ is shown in [Figure P1-13](#) where the $x(n)$ sequence, represented by the circular dots, is a single sinusoid whose frequency is f_0 Hz.

Figure P1-13



Using the trigonometric identity $\cos(\alpha+\beta) + \cos(\alpha-\beta) = 2\cos(\alpha)\cos(\beta)$, derive an equation for $x(n)$ that is of the form

$$x(n) = 2\cos(\alpha)\cos(\beta)$$

where variables α and β are in terms of $2\pi f_0 n t_s$ and ϕ .

1.14 In your engineering education you'll often read in some mathematical derivation, or hear

someone say, “For small α , $\sin(\alpha) = \alpha$.” (In fact, you’ll encounter that statement a few times in this book.) Draw two curves defined by

$$x = \alpha, \text{ and } y = \sin(\alpha)$$

over the range of $\alpha = -\pi/2$ to $\alpha = \pi/2$, and discuss why that venerable “For small α , $\sin(\alpha) = \alpha$ ” statement is valid.

1.15 Considering two continuous (analog) sinusoids, having initial phase angles of α radians at time $t = 0$, replace the following “?” characters with the correct angle arguments:

(a) $\sin(2\pi f_o t + \alpha) = \cos(?)$.

(b) $\cos(2\pi f_o t + \alpha) = \sin(?)$.

1.16 National Instruments Corp. manufactures an A/D converter, Model #NI USB-5133, that is capable of sampling an analog signal at an f_s sample rate of 100 megasamples per second (100 MHz). The A/D converter has internal memory that can store up to 4×10^6 discrete samples. What is the maximum number of cycles of a 25 MHz analog sinewave that can be stored in the A/D converter’s memory? Show your work.

1.17 In the first part of the text’s [Section 1.5](#) we stated that for a process (or system) to be *linear* it must satisfy a scaling property that we called the *proportionality* characteristic in the text’s [Eq. \(1-14\)](#). Determine if the following processes have that proportionality characteristic:

(a) $y_a(n) = x(n-1)/6$,

(b) $y_b(n) = 3 + x(n)$,

(c) $y_c(n) = \sin[x(n)]$.

This problem is *not* “busy work.” Knowing if a process (or system) is linear tells us what signal processing principles, and algorithms, can be applied in the analysis of that process (or system).

1.18 There is an often-used process in DSP called [decimation](#), and in that process we retain some samples of an $x(n)$ input sequence and discard other $x(n)$ samples. Decimation by a factor of two can be described algebraically by

$$(P1-3)$$

$$y(m) = x(2n)$$

where index $m = 0, 1, 2, 3, \dots$. The decimation defined by [Eq. \(P1-3\)](#) means that $y(m)$ is equal to alternate samples (every other sample) of $x(n)$. For example:

$$y(0) = x(0), y(1) = x(2), y(2) = x(4), y(3) = x(6), \dots$$

and so on. Here is the question: Is that decimation process time invariant? Illustrate your answer by decimating a simple sinusoidal $x(n)$ time-domain sequence by a factor of two to obtain $y(m)$. Next, create a shifted-by-one-sample version of $x(n)$ and call it $x_{\text{shift}}(n)$. That new sequence is defined by

$$(P1-4)$$

$$x_{\text{shift}}(n) = x(n+1).$$

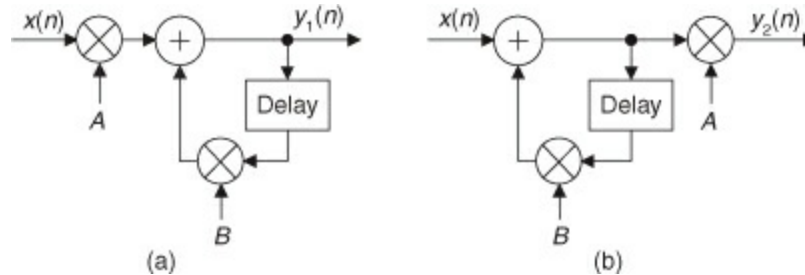
Finally, decimate $x_{\text{shift}}(n)$ according to [Eq. \(P1-3\)](#) to obtain $y_{\text{shift}}(m)$. The decimation process is

time invariant if $y_{\text{shift}}(m)$ is equal to a time-shifted version of $y(m)$. That is, decimation is time invariant if

$$y_{\text{shift}}(m) = y(m+1).$$

- 1.19** In [Section 1.7](#) of the text we discussed the commutative property of linear time-invariant systems. The two networks in [Figure P1-19](#) exhibit that property. Prove this to be true by showing that, given the same $x(n)$ input sequence, outputs $y_1(n)$ and $y_2(n)$ will be equal.

Figure P1-19



- 1.20** Here we investigate several simple discrete processes that turn out to be useful in a number of DSP applications. Draw the block diagrams, showing their inputs as $x(n)$, of the processes described by the following difference equations:

(a) a 4th-order comb filter: $y_C(n) = x(n) - x(n-4)$,

(b) an integrator: $y_I(n) = x(n) + y_I(n-1)$,

(c) a leaky integrator: $y_{LI}(n) = Ax(n) + (1-A)y_{LI}(n-1)$ [the scalar value A is a real-valued constant in the range $0 < A < 1$],

(d) a differentiator: $y_D(n) = 0.5x(n) - 0.5x(n-2)$.

- 1.21** Draw the unit impulse responses (the output sequences when the input is a unit sample impulse applied at time $n = 0$) of the four processes listed in Problem 1.20. Let $A = 0.5$ for the leaky integrator. Assume that all sample values within the systems are zero at time $n = 0$.

- 1.22** DSP engineers involved in building control systems often need to know what is the *step response* of a discrete system. The step response, $y_{\text{step}}(n)$, can be defined in two equivalent ways. One way is to say that $y_{\text{step}}(n)$ is a system's response to an input sequence of all unity-valued samples. A second definition is that $y_{\text{step}}(n)$ is the cumulative sum (the accumulation, discrete integration) of that system's unit impulse response $y_{\text{imp}}(n)$. Algebraically, this second definition of step response is expressed as

$$y_{\text{step}}(n) = \sum_{k=-\infty}^n y_{\text{imp}}(k).$$

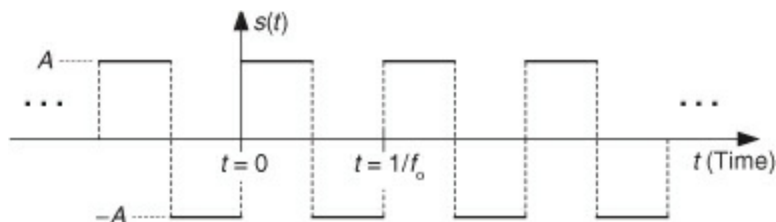
In words, the above $y_{\text{step}}(n)$ expression tells us: "The step response at time index n is equal to the sum of all the previous impulse response samples up to and including $y_{\text{imp}}(n)$." With that said, what are the step responses of the four processes listed in Problem 1.20? (Let $A = 0.5$ for the leaky integrator.) Assume that all sample values within the system are zero at time $n = 0$.

- 1.23** Thinking about the spectra of signals, the *ideal* continuous (analog) squarewave $s(t)$ in [Figure](#)

[P1-23](#), whose fundamental frequency is f_0 Hz, is equal to the sum of an f_0 Hz sinewave and all sinewaves whose frequencies are odd multiples of f_0 Hz. We call $s(t)$ “ideal” because we assume the amplitude transitions from plus and minus A occur instantaneously (zero seconds!). Continuous Fourier analysis of the $s(t)$ squarewave allows us to describe this sum of frequencies as the following infinite sum:

$$s(t) = \frac{4A}{\pi} \left[\sin(2\pi f_0 t) + \frac{\sin(6\pi f_0 t)}{3} + \frac{\sin(10\pi f_0 t)}{5} + \frac{\sin(14\pi f_0 t)}{7} + \dots \right].$$

Figure P1-23



Using a summation symbol, we can express squarewave $s(t)$ algebraically as

$$s(t) = \frac{4A}{\pi} \sum_{n=1}^{\infty} \sin(2\pi n f_0 t) / n,$$

for $n = \text{odd integers only}$, showing $s(t)$ to be an infinite sum of sinusoids.

- (a) Imagine applying $s(t)$ to a filter that completely removes $s(t)$ ’s lowest-frequency spectral component. Draw the time-domain waveform at the output of such a filter.
- (b) Assume $s(t)$ represents a voltage whose f_0 fundamental frequency is 1 Hz, and we wish to amplify that voltage to peak amplitudes of $\pm 2A$. Over what frequency range must an amplifier operate (that is, what must be the amplifier’s *passband width*) in order to exactly double the ideal 1 Hz squarewave’s peak-peak amplitude?

1.24 This interesting problem illustrates an *illegal* mathematical operation that we must learn to avoid in our future algebraic activities. The following claims to be a mathematical proof that $4 = 5$. Which of the following steps is illegal? Explain why.

Proof that $4 = 5$:

Step 1: $16 - 36 = 25 - 45$

Step 2: $4^2 - 9 \cdot 4 = 5^2 - 9 \cdot 5$

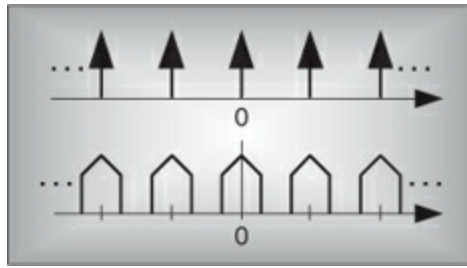
Step 3: $4^2 - 9 \cdot 4 + 81/4 = 5^2 - 9 \cdot 5 + 81/4$

Step 4: $(4 - 9/2)^2 = (5 - 9/2)^2$

Step 5: $4 - 9/2 = 5 - 9/2$

Step 6: $4 = 5$

Chapter Two. Periodic Sampling



Periodic sampling, the process of representing a continuous signal with a sequence of discrete data values, pervades the field of digital signal processing. In practice, sampling is performed by applying a continuous signal to an analog-to-digital (A/D) converter whose output is a series of digital values. Because sampling theory plays an important role in determining the accuracy and feasibility of any digital signal processing scheme, we need a solid appreciation for the often misunderstood effects of periodic sampling. With regard to sampling, the primary concern is just how fast a given continuous signal must be sampled in order to preserve its information content. We can sample a continuous signal at any sample rate we wish, and we'll obtain a series of discrete values—but the question is, how well do these values represent the original signal? Let's learn the answer to that question and, in doing so, explore the various sampling techniques used in digital signal processing.

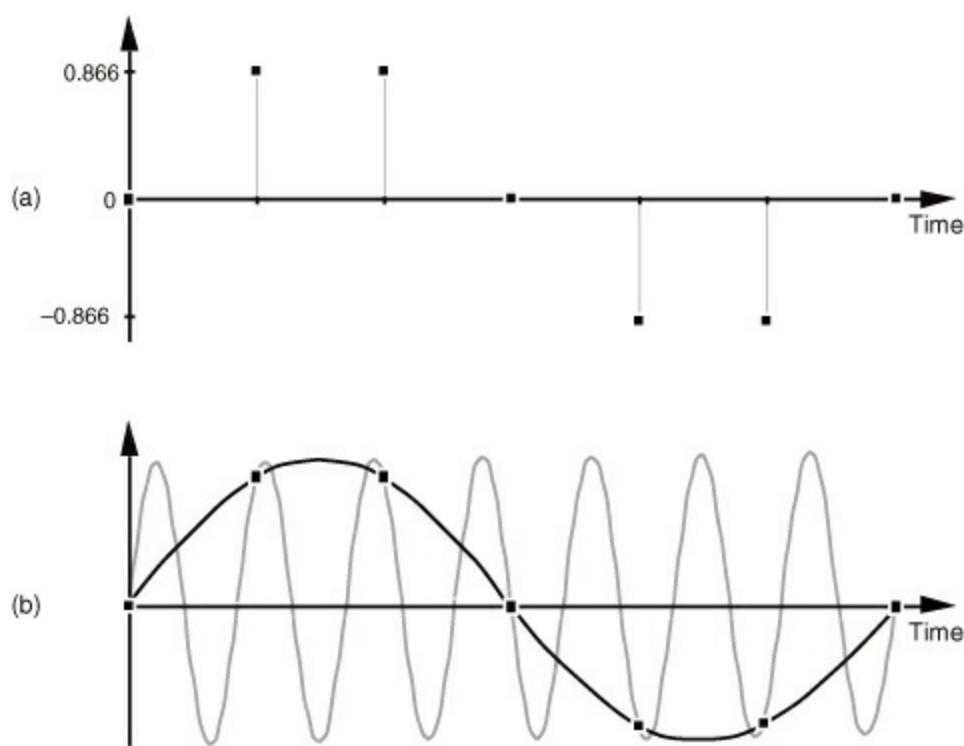
2.1 Aliasing: Signal Ambiguity in the Frequency Domain

There is a frequency-domain ambiguity associated with discrete-time signal samples that does not exist in the continuous signal world, and we can appreciate the effects of this uncertainty by understanding the sampled nature of discrete data. By way of example, suppose you were given the following sequence of values,

$$\begin{aligned}x(0) &= 0 \\x(1) &= 0.866 \\x(2) &= 0.866 \\x(3) &= 0 \\x(4) &= -0.866 \\x(5) &= -0.866 \\x(6) &= 0,\end{aligned}$$

and were told that they represent instantaneous values of a time-domain sinewave taken at periodic intervals. Next, you were asked to draw that sinewave. You'd start by plotting the sequence of values shown by the dots in [Figure 2-1\(a\)](#). Next, you'd be likely to draw the sinewave, illustrated by the solid line in [Figure 2-1\(b\)](#), that passes through the points representing the original sequence.

Figure 2-1 Frequency ambiguity: (a) discrete-time sequence of values; (b) two different sinewaves that pass through the points of the discrete sequence.



Another person, however, might draw the sinewave shown by the shaded line in [Figure 2-1\(b\)](#). We see that the original sequence of values could, with equal validity, represent sampled values of both sinewaves. The key issue is that if the data sequence represents periodic samples of a sinewave, we cannot unambiguously determine the frequency of the sinewave from those sample values alone.

Reviewing the mathematical origin of this frequency ambiguity enables us not only to deal with it, but to use it to our advantage. Let's derive an expression for this frequency-domain ambiguity and, then, look at a few specific examples. Consider the continuous time-domain sinusoidal signal defined as

(2-1)

$$x(t) = \sin(2\pi f_0 t).$$

This $x(t)$ signal is a garden-variety sinewave whose frequency is f_0 Hz. Now let's sample $x(t)$ at a rate of f_s samples/second, i.e., at regular periods of t_s seconds where $t_s = 1/f_s$. If we start sampling at time $t = 0$, we will obtain samples at times $0t_s$, $1t_s$, $2t_s$, and so on. So, from [Eq. \(2-1\)](#), the first n successive samples have the values

(2-2)

0th sample:	$x(0) = \sin(2\pi f_0 0t_s)$
1st sample:	$x(1) = \sin(2\pi f_0 1t_s)$
2nd sample:	$x(2) = \sin(2\pi f_0 2t_s)$
...	...
...	...
n th sample:	$x(n) = \sin(2\pi f_0 nt_s).$

[Equation \(2-2\)](#) defines the value of the n th sample of our $x(n)$ sequence to be equal to the original sinewave at the time instant nt_s . Because two values of a sinewave are identical if they're separated by an integer multiple of 2π radians, i.e., $\sin(\theta) = \sin(\theta + 2\pi m)$ where m is any integer, we can modify [Eq. \(2-2\)](#) as

(2-3)

$$x(n) = \sin(2\pi f_o n t_s) = \sin(2\pi f_o n t_s + 2\pi m) = \sin(2\pi(f_o + \frac{m}{n t_s}) n t_s).$$

If we let m be an integer multiple of n , $m = kn$, we can replace the m/n ratio in [Eq. \(2-3\)](#) with k so that

(2-4)

$$x(n) = \sin(2\pi(f_o + \frac{k}{t_s}) n t_s).$$

Because $f_s = 1/t_s$, we can equate the $x(n)$ sequences in [Eqs. \(2-2\)](#) and [\(2-4\)](#) as

(2-5)

$$x(n) = \sin(2\pi f_o n t_s) = \sin(2\pi(f_o + k f_s) n t_s).$$

The f_o and $(f_o + k f_s)$ factors in [Eq. \(2-5\)](#) are therefore equal. The implication of [Eq. \(2-5\)](#) is critical. It means that an $x(n)$ sequence of digital sample values, representing a sinewave of f_o Hz, also exactly represents sinewaves at other frequencies, namely, $f_o + k f_s$. This is one of the most important relationships in the field of digital signal processing. It's the thread with which all sampling schemes are woven. In words, [Eq. \(2-5\)](#) states:

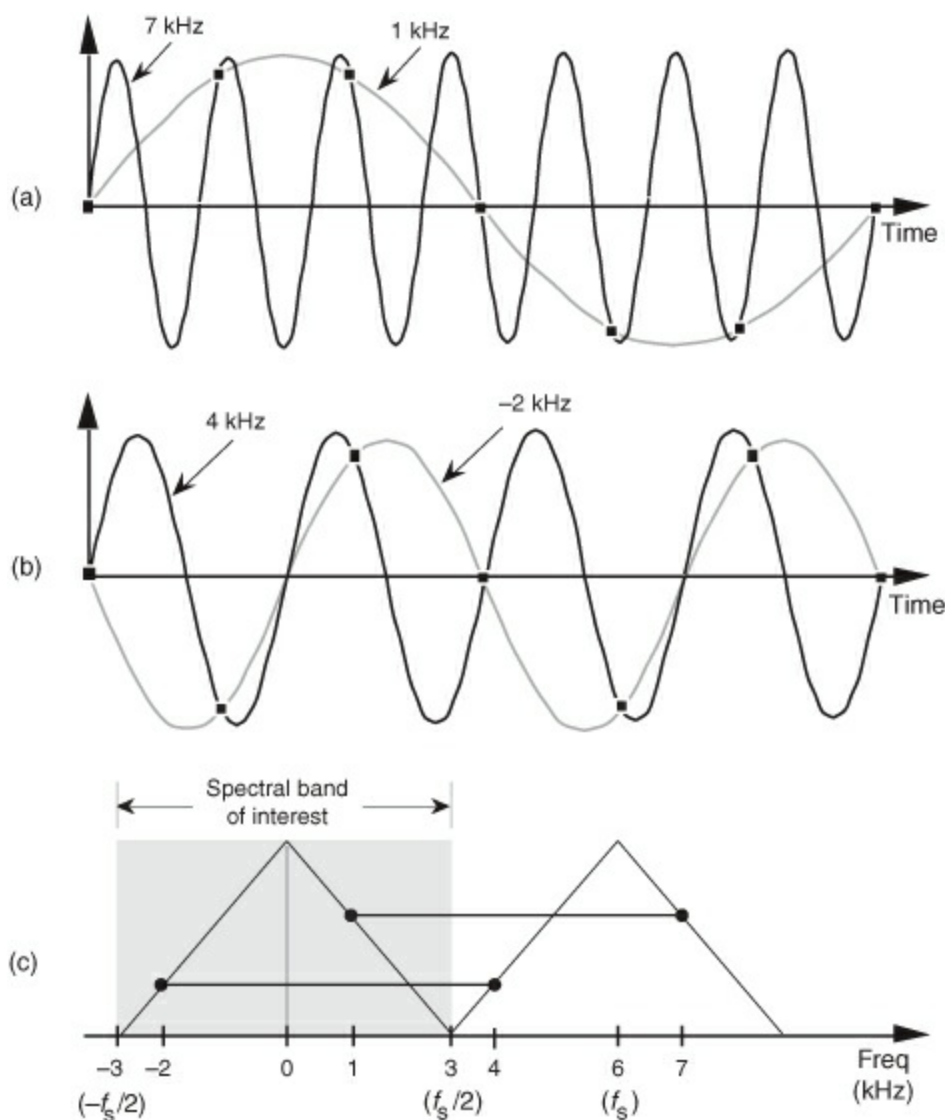
When sampling at a rate of f_s samples/second, if k is any positive or negative integer, we cannot distinguish between the sampled values of a sinewave of f_o Hz and a sinewave of $(f_o + k f_s)$ Hz.

It's true. No sequence of values stored in a computer, for example, can unambiguously represent one and only one sinusoid without additional information. This fact applies equally to A/D-converter output samples as well as signal samples generated by computer software routines. The sampled nature of any sequence of discrete values makes that sequence also represent an infinite number of different sinusoids.

[Equation \(2-5\)](#) influences all digital signal processing schemes. It's the reason that, although we've only shown it for sinewaves, we'll see in [Chapter 3](#) that the spectrum of any discrete series of sampled values contains periodic replications of the original continuous spectrum. The period between these replicated spectra in the frequency domain will always be f_s , and the spectral replications repeat all the way from *DC* to *daylight* in both directions of the frequency spectrum. That's because k in [Eq. \(2-5\)](#) can be any positive or negative integer. (In [Chapters 5](#) and [6](#), we'll learn that [Eq. \(2-5\)](#) is the reason that all digital filter frequency responses are periodic in the frequency domain and is crucial to analyzing and designing a popular type of digital filter known as the infinite impulse response filter.)

To illustrate the effects of [Eq. \(2-5\)](#), let's build on [Figure 2-1](#) and consider the sampling of a 7 kHz sinewave at a sample rate of 6 kHz. A new sample is determined every 1/6000 seconds, or once every 167 microseconds, and their values are shown as the dots in [Figure 2-2\(a\)](#).

Figure 2-2 Frequency ambiguity effects of [Eq. \(2-5\)](#): (a) sampling a 7 kHz sinewave at a sample rate of 6 kHz; (b) sampling a 4 kHz sinewave at a sample rate of 6 kHz; (c) spectral relationships showing aliasing of the 7 and 4 kHz sinewaves.



Notice that the sample values would not change at all if, instead, we were sampling a 1 kHz sinewave. In this example $f_0 = 7$ kHz, $f_s = 6$ kHz, and $k = -1$ in [Eq. \(2-5\)](#), such that $f_0 + kf_s = [7 + (-1 \cdot 6)] = 1$ kHz. Our problem is that no processing scheme can determine if the sequence of sampled values, whose amplitudes are represented by the dots, came from a 7 kHz or a 1 kHz sinusoid. If these amplitude values are applied to a digital process that detects energy at 1 kHz, the detector output would indicate energy at 1 kHz. But we know that there is no 1 kHz tone there—our input is a spectrally pure 7 kHz tone. [Equation \(2-5\)](#) is causing a sinusoid, whose name is 7 kHz, to go by the *alias* of 1 kHz. Asking someone to determine which sinewave frequency accounts for the sample values in [Figure 2-2\(a\)](#) is like asking, “When I add two numbers I get a sum of four. What are the two numbers?” The answer is that there is an infinite number of number pairs that can add up to four.

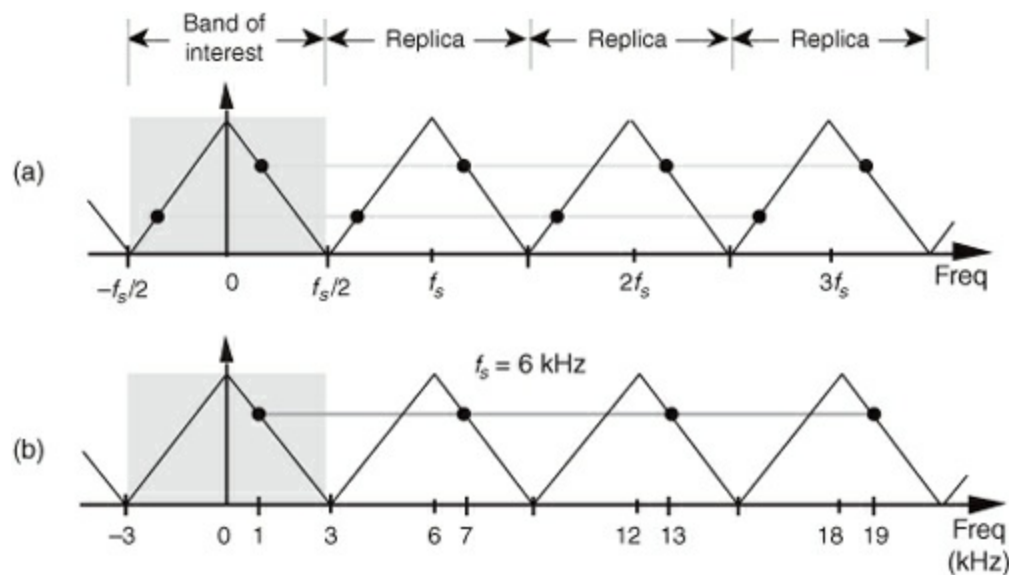
[Figure 2-2\(b\)](#) shows another example of frequency ambiguity that we’ll call *aliasing*, where a 4 kHz sinewave could be mistaken for a -2 kHz sinewave. In [Figure 2-2\(b\)](#), $f_0 = 4$ kHz, $f_s = 6$ kHz, and $k = -1$ in [Eq. \(2-5\)](#), so that $f_0 + kf_s = [4 + (-1 \cdot 6)] = -2$ kHz. Again, if we examine a sequence of numbers representing the dots in [Figure 2-2\(b\)](#), we could not determine if the sampled sinewave was a 4 kHz tone or a -2 kHz tone. (Although the concept of negative frequencies might seem a bit strange, it provides a beautifully consistent methodology for predicting the spectral effects of sampling. [Chapter 8](#) discusses negative frequencies and how they relate to real and complex signals.)

Now, if we restrict our spectral band of interest to the frequency range of $\pm f_s/2$, the previous two examples take on a special significance. The frequency $f_s/2$ is an important quantity in sampling

theory and is referred to by different names in the literature, such as critical Nyquist, half Nyquist, and folding frequency. A graphical depiction of our two frequency aliasing examples is provided in [Figure 2-2\(c\)](#). We're interested in signal components that are aliased into the frequency band between $-f_s/2$ and $+f_s/2$. Notice in [Figure 2-2\(c\)](#) that within the spectral band of interest (± 3 kHz, because $f_s = 6$ kHz), there is energy at -2 kHz and $+1$ kHz, aliased from 4 kHz and 7 kHz, respectively. Note also that the vertical positions of the dots in [Figure 2-2\(c\)](#) have no amplitude significance but that their horizontal positions indicate which frequencies are related through aliasing.

A general illustration of aliasing is provided in the *shark's tooth* pattern in [Figure 2-3\(a\)](#). Note how the peaks of the pattern are located at integer multiples of f_s Hz. The pattern shows how signals residing at the intersection of a horizontal line and a sloped line will be aliased to all of the intersections of that horizontal line and all other lines with like slopes. For example, the pattern in [Figure 2-3\(b\)](#) shows that our sampling of a 7 kHz sinewave at a sample rate of 6 kHz will provide a discrete sequence of numbers whose spectrum ambiguously represents tones at 1 kHz, 7 kHz, 13 kHz, 19 kHz, etc. Let's pause for a moment and let these very important concepts soak in a bit. Again, discrete sequence representations of a continuous signal have unavoidable ambiguities in their frequency domains. These ambiguities must be taken into account in all practical digital signal processing algorithms.

Figure 2-3 Shark's tooth pattern: (a) aliasing at multiples of the sampling frequency; (b) aliasing of the 7 kHz sinewave to 1 kHz, 13 kHz, and 19 kHz.



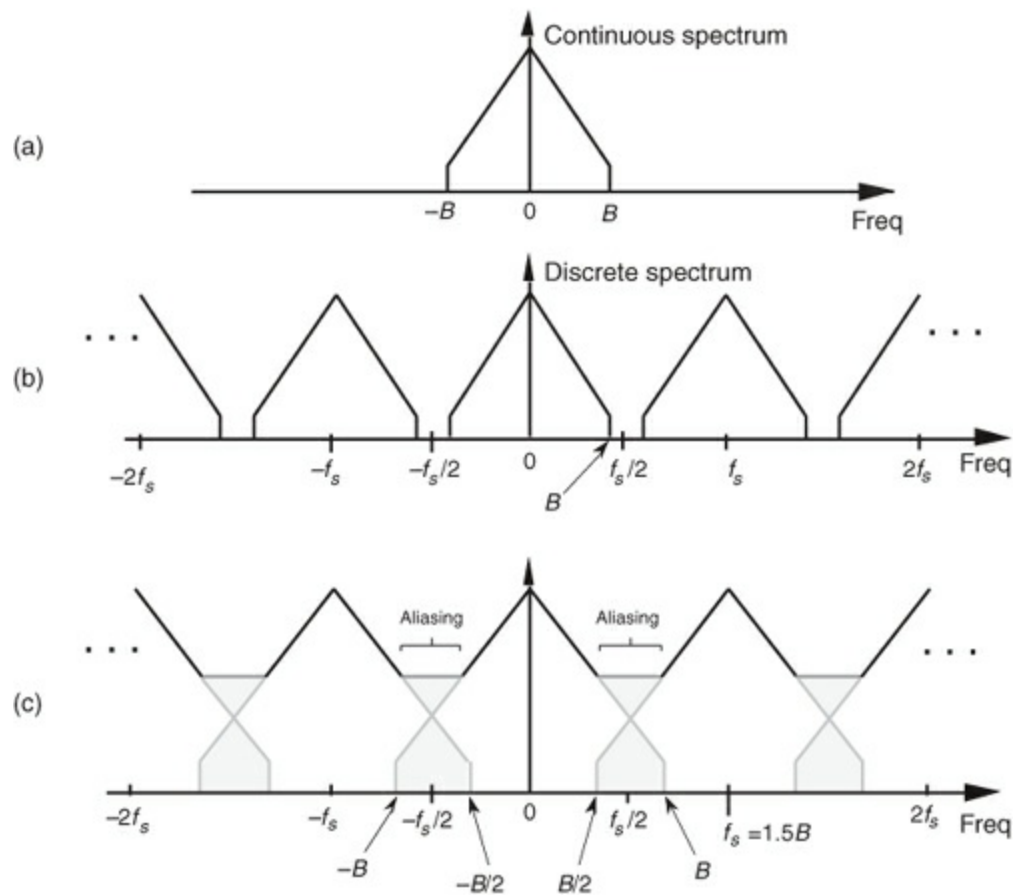
OK, let's review the effects of sampling signals that are more interesting than just simple sinusoids.

2.2 Sampling Lowpass Signals

Consider the situation of sampling a signal such as a continuous real-valued lowpass $x(t)$ signal whose spectrum is shown in [Figure 2-4\(a\)](#). Notice that the spectrum is symmetrical around zero Hz, and the spectral amplitude is zero above $+B$ Hz and below $-B$ Hz; i.e., the signal is *band-limited*. (From a practical standpoint, the term *band-limited signal* merely implies that any signal energy outside the range of $\pm B$ Hz is below the sensitivity of our system.) The $x(t)$ time signal is called a *lowpass signal* because its spectral energy is low in frequency.

Figure 2-4 Spectral replications: (a) original continuous lowpass signal spectrum; (b) spectral replications of the sampled lowpass signal when $f_s/2 > B$; (c) frequency overlap and aliasing when

the sampling rate is too low because $f_s/2 < B$.



Pausing for a moment, if the continuous $x(t)$ signal were a voltage on a coax cable applied to the input of an analog spectrum analyzer, we would only see the spectral energy over the positive-frequency range of 0 to $+B$ Hz on the analyzer's screen. However, in our world of discrete signals (DSP) we show the spectrum of real-valued signals as having both positive- and negative-frequency spectral energy. Throughout this book we'll repeatedly see why such spectral representations are often useful, and sometimes mandatory in our work. The mathematical justification for two-sided spectral diagrams is provided in both [Chapters 3](#) and [8](#). For now, we request the reader's acceptance that [Figure 2-4\(a\)](#) is a valid representation of the spectrum of the continuous $x(t)$ signal.

Given that the continuous $x(t)$ signal, whose spectrum is shown in [Figure 2-4\(a\)](#), is sampled at a rate of f_s samples/second, we can see the spectral replication effects of sampling in [Figure 2-4\(b\)](#) showing the original spectrum in addition to an infinite number of replications. The period of spectral replication is f_s Hz. [Figure 2-4\(b\)](#) is the spectrum of the sequence of $x(n)$ sampled values of the continuous $x(t)$ signal. (Although we stated in [Section 1.1](#) that frequency-domain representations of discrete time-domain sequences are themselves discrete, the replicated spectra in [Figure 2-4\(b\)](#) are shown as continuous lines, instead of discrete dots, merely to keep the figure from looking too cluttered. We'll cover the full implications of discrete frequency spectra in [Chapter 3](#).)

Let's step back a moment and understand [Figure 2-4](#) for all it's worth. [Figure 2-4\(a\)](#) is the spectrum of a continuous signal, a signal that can only exist in one of two forms. Either it's a continuous signal that can be sampled, through A/D conversion, or it is merely an abstract concept such as a mathematical expression for a signal. It *cannot* be represented in a digital machine in its current band-limited form. Once the signal is represented by a sequence of discrete sample values, its spectrum takes the replicated form of [Figure 2-4\(b\)](#).

The replicated spectra are not just figments of the mathematics; they exist and have a profound effect on subsequent digital signal processing.[‡] The replications may appear harmless, and it's natural to ask, "Why care about spectral replications? We're only interested in the frequency band within $\pm f_s/2$." Well, if we perform a frequency translation operation or induce a change in sampling rate through decimation or interpolation, the spectral replications will shift up or down right in the middle of the frequency range of interest $\pm f_s/2$ and could cause problems[1]. Let's see how we can control the locations of those spectral replications.

[‡] Toward the end of [Section 5.9](#), as an example of using the convolution theorem, another derivation of periodic sampling's replicated spectra will be presented.

In practical A/D conversion schemes, f_s is always greater than $2B$ to separate spectral replications at the *folding frequencies* of $\pm f_s/2$. This very important relationship of $f_s \geq 2B$ is known as the *Nyquist criterion*. To illustrate why the term *folding frequency* is used, let's lower our sampling frequency to $f_s = 1.5B$ Hz. The spectral result of this *undersampling* is illustrated in [Figure 2-4\(c\)](#). The spectral replications are now overlapping the original baseband spectrum centered at zero Hz. Limiting our attention to the band $\pm f_s/2$ Hz, we see two very interesting effects. First, the lower edge and upper edge of the spectral replications centered at $+f_s$ and $-f_s$ now lie in our band of interest. This situation is equivalent to the original spectrum folding to the left at $+f_s/2$ and folding to the right at $-f_s/2$. Portions of the spectral replications now combine with the original spectrum, and the result is aliasing errors. The discrete sampled values associated with the spectrum of [Figure 2-4\(c\)](#) no longer truly represent the original input signal. The spectral information in the bands of $-B$ to $-B/2$ and $B/2$ to B Hz has been corrupted. We show the amplitude of the aliased regions in [Figure 2-4\(c\)](#) as shaded lines because we don't really know what the amplitudes will be if aliasing occurs.

The second effect illustrated by [Figure 2-4\(c\)](#) is that the entire spectral content of the original continuous signal is now residing in the band of interest between $-f_s/2$ and $+f_s/2$. This key property was true in [Figure 2-4\(b\)](#) and will always be true, regardless of the original signal or the sample rate. This effect is particularly important when we're digitizing (A/D converting) continuous signals. It warns us that any signal energy located above $+B$ Hz and below $-B$ Hz in the original continuous spectrum of [Figure 2-4\(a\)](#) will always end up in the band of interest after sampling, regardless of the sample rate. For this reason, continuous (analog) *lowpass* filters are necessary in practice.

We illustrate this notion by showing a continuous signal of bandwidth B accompanied by noise energy in [Figure 2-5\(a\)](#). Sampling this composite continuous signal at a rate that's greater than $2B$ prevents replications of the signal of interest from overlapping each other, but all of the noise energy still ends up in the range between $-f_s/2$ and $+f_s/2$ of our discrete spectrum shown in [Figure 2-5\(b\)](#). This problem is solved in practice by using an analog lowpass *anti-aliasing* filter prior to A/D conversion to attenuate any unwanted signal energy above $+B$ and below $-B$ Hz as shown in [Figure 2-6](#). An example lowpass filter response shape is shown as the dotted line superimposed on the original continuous signal spectrum in [Figure 2-6](#). Notice how the output spectrum of the lowpass filter has been band-limited, and spectral aliasing is avoided at the output of the A/D converter.

Figure 2-5 Spectral replications: (a) original continuous signal-plus-noise spectrum; (b) discrete spectrum with noise contaminating the signal of interest.

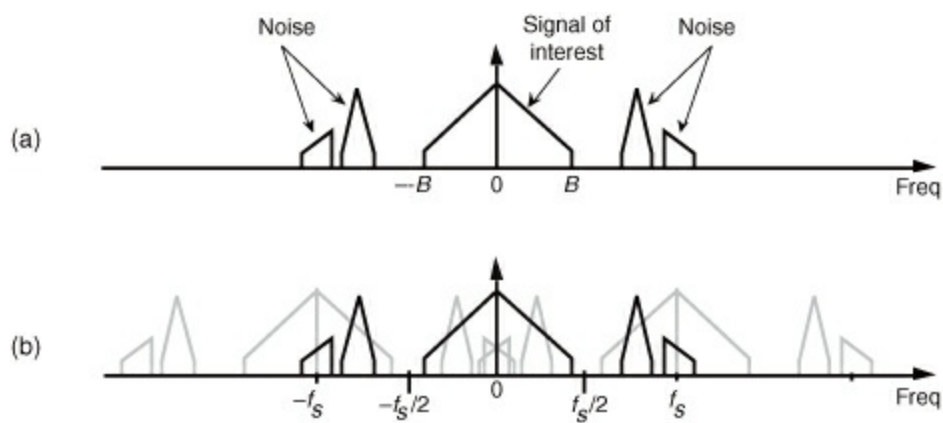
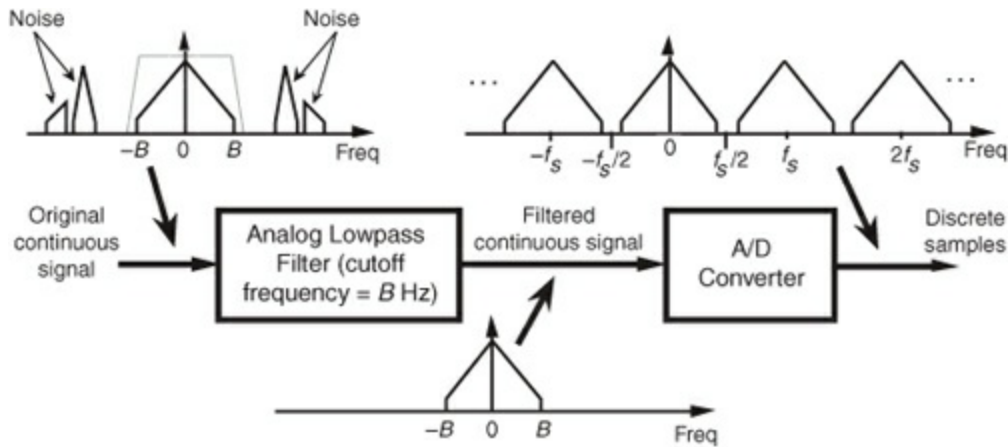


Figure 2-6 Lowpass analog filtering prior to sampling at a rate of f_s Hz.



As a historical note, the notion of periodic sampling was studied by various engineers, scientists, and mathematicians such as the Russian V. Kotelnikov, the Swedish-born H. Nyquist, the Scottish E. Whittaker, and the Japanese I. Someya[2]. But it was the American Claude Shannon, acknowledging the work of others, that formalized the concept of periodic sampling as we know it today and brought it to the broad attention of communications engineers[3]. That was in 1948—the birth year of the transistor, marshmallows, and this author.

This completes the discussion of simple lowpass sampling. Now let's go on to a more advanced sampling topic that's proven so useful in practice.

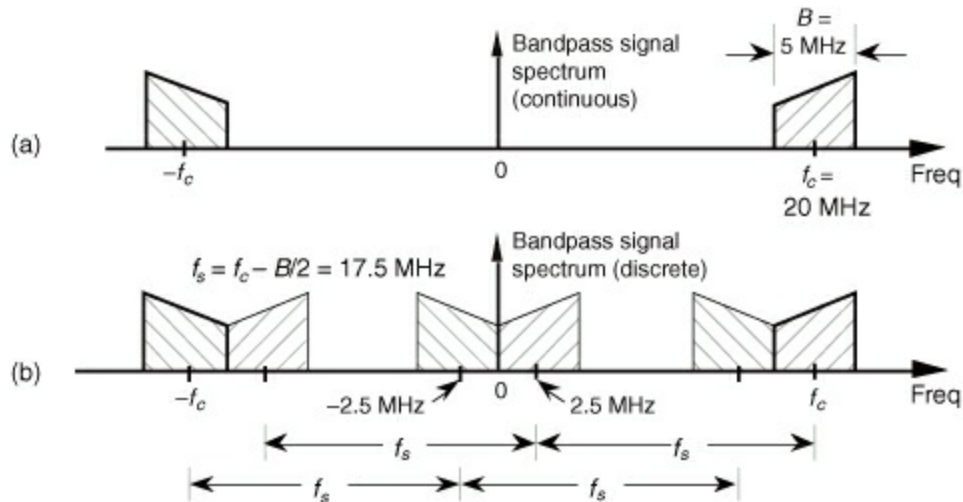
2.3 Sampling Bandpass Signals

Although satisfying the majority of sampling requirements, the sampling of lowpass signals, as in Figure 2-6, is not the only sampling scheme used in practice. We can use a technique known as *bandpass sampling* to sample a continuous bandpass signal that is centered about some frequency other than zero Hz. When a continuous input signal's bandwidth and center frequency permit us to do so, bandpass sampling not only reduces the speed requirement of A/D converters below that necessary with traditional lowpass sampling; it also reduces the amount of digital memory necessary to capture a given time interval of a continuous signal.

By way of example, consider sampling the band-limited signal shown in Figure 2-7(a) centered at $f_c = 20$ MHz, with a bandwidth $B = 5$ MHz. We use the term *bandpass sampling* for the process of sampling continuous signals whose center frequencies have been translated up from zero Hz. What we're calling bandpass sampling goes by various other names in the literature, such as IF sampling, harmonic sampling[4], sub-Nyquist sampling, and undersampling[5]. In bandpass sampling, we're more concerned with a signal's bandwidth than its highest-frequency component. Note that the

negative frequency portion of the signal, centered at $-f_c$, is the mirror image of the positive frequency portion—as it must be for real signals. Our bandpass signal’s highest-frequency component is 22.5 MHz. Conforming to the Nyquist criterion (sampling at twice the highest-frequency content of the signal) implies that the sampling frequency must be a minimum of 45 MHz. Consider the effect if the sample rate is 17.5 MHz shown in [Figure 2-7\(b\)](#). Note that the original spectral components remain located at $\pm f_c$, and spectral replications are located exactly at baseband, i.e., butting up against each other at zero Hz. [Figure 2-7\(b\)](#) shows that sampling at 45 MHz was unnecessary to avoid aliasing—instead we’ve used the spectral replicating effects of [Eq. \(2-5\)](#) to our advantage.

Figure 2-7 Bandpass signal sampling: (a) original continuous signal spectrum; (b) sampled signal spectrum replications when sample rate is 17.5 MHz.



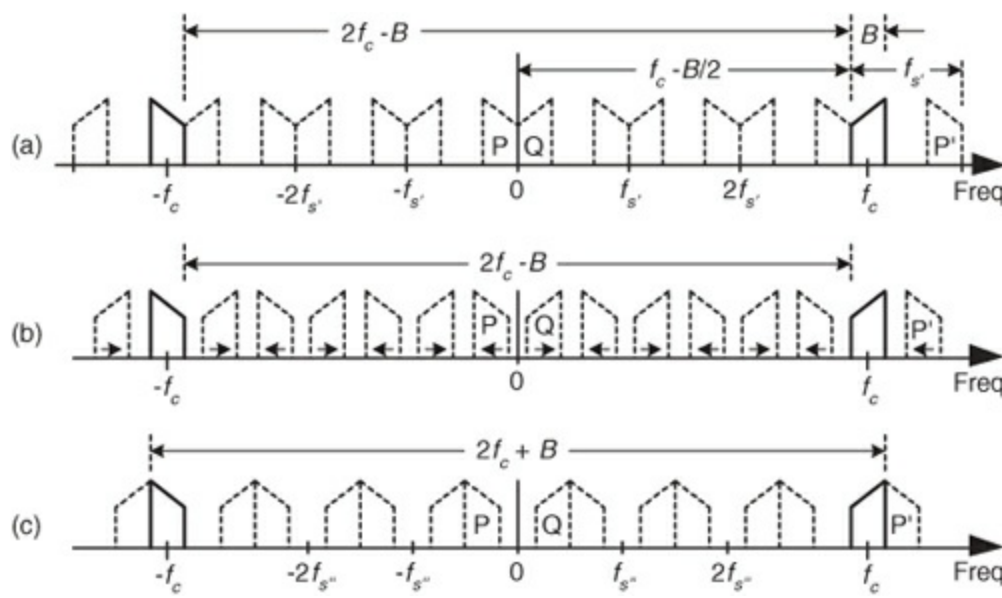
Bandpass sampling performs digitization and frequency translation in a single process, often called *sampling translation*. The processes of sampling and frequency translation are intimately bound together in the world of digital signal processing, and every sampling operation inherently results in spectral replications. The inquisitive reader may ask, “Can we sample at some still lower rate and avoid aliasing?” The answer is yes, but, to find out how, we have to grind through the derivation of an important bandpass sampling relationship. Our reward, however, will be worth the trouble because here’s where bandpass sampling really gets interesting.

Let’s assume we have a continuous input bandpass signal of bandwidth B . Its *carrier frequency* is f_c Hz, i.e., the bandpass signal is centered at f_c Hz, and its sampled value spectrum is that shown in [Figure 2-8\(a\)](#). We can sample that continuous signal at a rate, say $f_{s'}$ Hz, so the spectral replications of the positive and negative bands, Q and P, just butt up against each other exactly at zero Hz. This situation, depicted in [Figure 2-8\(a\)](#), is reminiscent of [Figure 2-7\(b\)](#). With an arbitrary number of replications, say m , in the range of $2f_c - B$, we see that

(2-6)

$$mf_{s'} = 2f_c - B \quad \text{or} \quad f_{s'} = \frac{2f_c - B}{m}.$$

Figure 2-8 Bandpass sampling frequency limits: (a) sample rate $f_{s'} = (2f_c - B)/6$; (b) sample rate is less than $f_{s'}$; (c) minimum sample rate $f_{s''} < f_{s'}$.



In [Figure 2-8\(a\)](#), $m = 6$ for illustrative purposes only. Of course m can be any positive integer so long as $f_{s'}$ is never less than $2B$. If the sample rate $f_{s'}$ is increased, the original spectra (bold) do not shift, but all the replications will shift. At zero Hz, the P band will shift to the right, and the Q band will shift to the left. These replications will overlap and aliasing occurs. Thus, from [Eq. \(2-6\)](#), for an arbitrary m , there is a frequency that the sample rate must not exceed, or

(2-7)

$$f_{s'} \leq \frac{2f_c - B}{m} \quad \text{or} \quad \frac{2f_c - B}{m} \geq f_{s'}.$$

If we reduce the sample rate below the $f_{s'}$ value shown in [Figure 2-8\(a\)](#), the spacing between replications will decrease in the direction of the arrows in [Figure 2-8\(b\)](#). Again, the original spectra do not shift when the sample rate is changed. At some new sample rate $f_{s''}$, where $f_{s''} < f_{s'}$, the replication P' will just butt up against the positive original spectrum centered at f_c as shown in [Figure 2-8\(c\)](#). In this condition, we know that

(2-8)

$$(m+1)f_{s''} = 2f_c + B \quad \text{or} \quad f_{s''} = \frac{2f_c + B}{m+1}.$$

Should $f_{s''}$ be decreased in value, P' will shift further down in frequency and start to overlap with the positive original spectrum at f_c and aliasing occurs. Therefore, from [Eq. \(2-8\)](#) and for $m+1$, there is a frequency that the sample rate must always exceed, or

(2-9)

$$f_{s''} \geq \frac{2f_c + B}{m+1}.$$

We can now combine [Eqs. \(2-7\)](#) and [\(2-9\)](#) to say that f_s may be chosen anywhere in the range between $f_{s''}$ and $f_{s'}$ to avoid aliasing, or

(2-10)

$$\frac{2f_c - B}{m} \geq f_s \geq \frac{2f_c + B}{m+1},$$

where m is an arbitrary, positive integer ensuring that $f_s \geq 2B$. (For this type of periodic sampling of

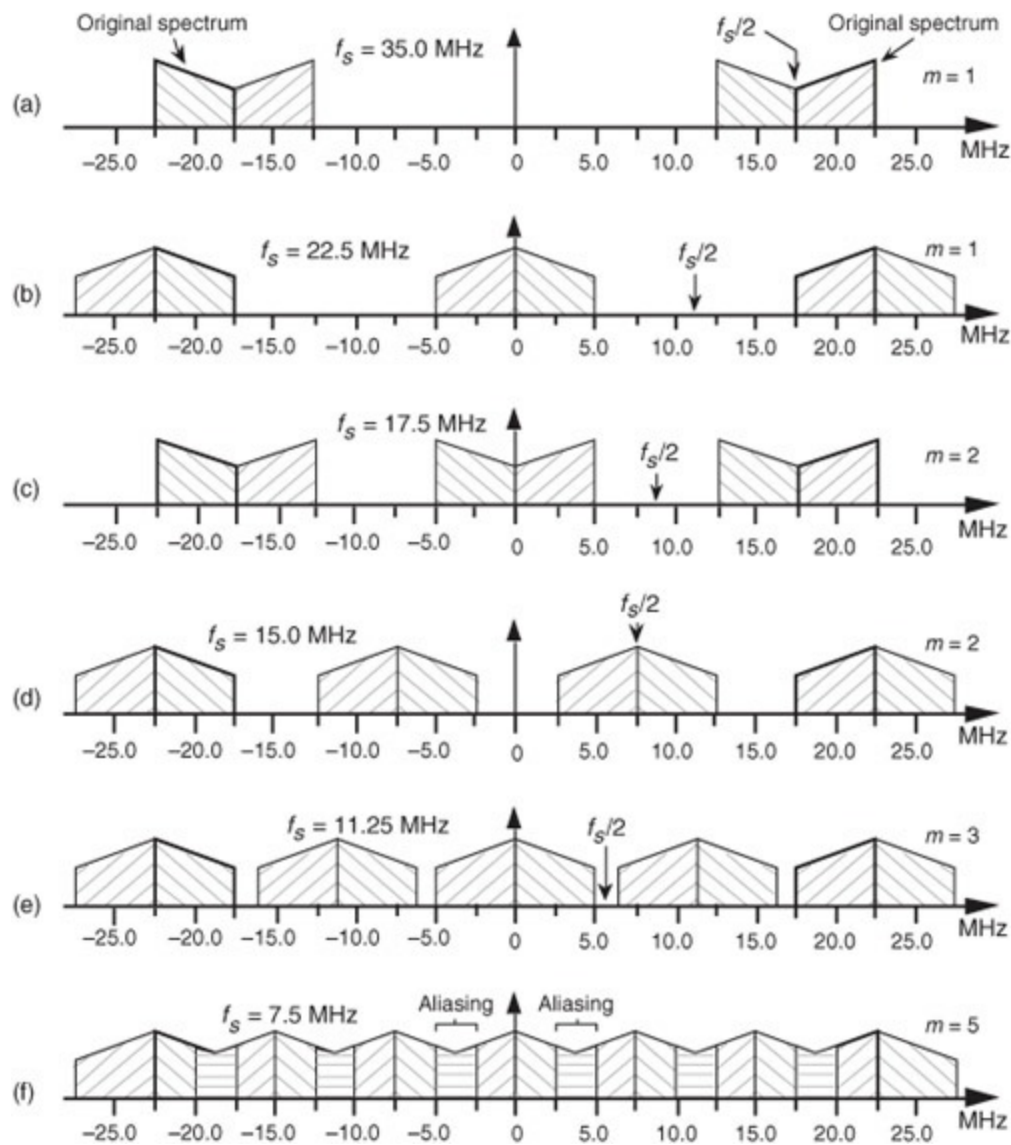
real signals, known as *real* or *1st-order sampling*, the Nyquist criterion $f_s \geq 2B$ must still be satisfied.)

To appreciate the important relationships in [Eq. \(2-10\)](#), let's return to our bandpass signal example, where [Eq. \(2-10\)](#) enables the generation of [Table 2-1](#). This table tells us that our sample rate can be anywhere in the range of 22.5 to 35 MHz, anywhere in the range of 15 to 17.5 MHz, or anywhere in the range of 11.25 to 11.66 MHz. Any sample rate below 11.25 MHz is unacceptable because it will not satisfy [Eq. \(2-10\)](#) as well as $f_s \geq 2B$. The spectra resulting from several of the sampling rates from [Table 2-1](#) are shown in [Figure 2-9](#) for our bandpass signal example. Notice in [Figure 2-9\(f\)](#) that when f_s equals 7.5 MHz ($m = 5$), we have aliasing problems because neither the greater-than relationships in [Eq. \(2-10\)](#) nor $f_s \geq 2B$ have been satisfied. The $m = 4$ condition is also unacceptable because $f_s \geq 2B$ is not satisfied. The last column in [Table 2-1](#) gives the *optimum* sampling frequency for each acceptable m value. Optimum sampling frequency is defined here as that frequency where spectral replications butt up against each other at zero Hz. For example, in the $m = 1$ range of permissible sampling frequencies, it is much easier to perform subsequent digital filtering or other processing on the signal samples whose spectrum is that of [Figure 2-9\(b\)](#), as opposed to the spectrum in [Figure 2-9\(a\)](#).

Table 2-1 [Equation \(2-10\)](#) Applied to the Bandpass Signal Example

m	$(2f_c - B)/m$	$(2f_c + B)/(m + 1)$	Optimum sampling rate
1	35.0 MHz	22.5 MHz	22.5 MHz
2	17.5 MHz	15.0 MHz	17.5 MHz
3	11.66 MHz	11.25 MHz	11.25 MHz
4	8.75 MHz	9.0 MHz	—
5	7.0 MHz	7.5 MHz	—

Figure 2-9 Various spectral replications from [Table 2-1](#): (a) $f_s = 35$ MHz; (b) $f_s = 22.5$ MHz; (c) $f_s = 17.5$ MHz; (d) $f_s = 15$ MHz; (e) $f_s = 11.25$ MHz; (f) $f_s = 7.5$ MHz.



2.4 Practical Aspects of Bandpass Sampling

Now that we're familiar with the theory of bandpass sampling, let's discuss a few aspects of bandpass sampling in practical applications.

2.4.1 Spectral Inversion in Bandpass Sampling

Some of the permissible f_s values from [Eq. \(2-10\)](#) will, although avoiding aliasing problems, provide a sampled baseband spectrum (located near zero Hz) that is inverted from the original analog signal's positive and negative spectral shapes. That is, the positive-frequency sampled baseband will have the inverted shape of the negative half from the original analog spectrum. This spectral inversion happens whenever m , in [Eq. \(2-10\)](#), is an odd integer, as illustrated in [Figures 2-9\(c\)](#) and [2-9\(d\)](#). When the original positive spectral bandpass components are symmetrical about the f_c frequency, spectral inversion presents no problem and any nonaliasing value for f_s from [Eq. \(2-10\)](#) may be chosen.

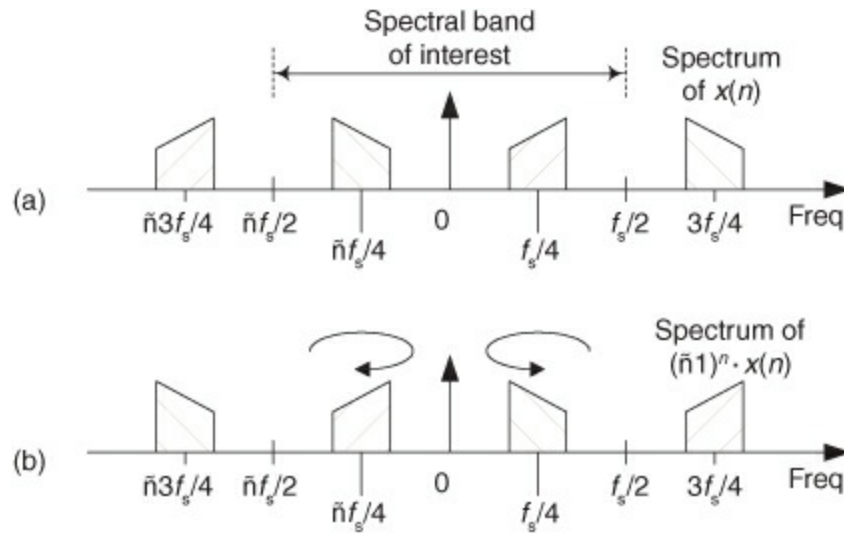
However, if spectral inversion is something to be avoided, for example, when single sideband signals are being processed, the applicable sample rates to avoid spectral inversion are defined by [Eq. \(2-10\)](#) with the restriction that m is an even integer and $f_s > 2B$ is satisfied.

Now here's some good news. With a little additional digital processing we can sample at rates defined by [Eq. \(2-10\)](#) with odd m , with their spectral inversion, and easily reinvert the spectrum back to its original orientation. The discrete spectrum of any digital signal can be inverted by multiplying

the signal's discrete-time samples by a sequence of alternating plus ones and minus ones (1, -1, 1, -1, etc.), indicated in the literature by the succinct expression $(-1)^n$.

Although multiplying time samples by $(-1)^n$ is explored in detail in [Section 13.1](#), all we need to remember at this point is the simple rule that multiplication of real signal samples by $(-1)^n$ flips the positive-frequency band of interest, from zero to $+f_s/2$ Hz, where the center of the flipping is $f_s/4$ Hz. Likewise, the multiplication flips the negative frequency band of interest, from $-f_s/2$ to zero Hz, where the center of the flipping is $-f_s/4$ Hz as shown in [Figure 2-10](#). In the literature of DSP, occasionally you'll see the $(-1)^n$ sequence expressed by the equivalent expression $\cos(\pi n)$.

Figure 2-10 Spectral inversion through multiplication by $(-1)^n$: (a) spectrum of original $x(n)$; (b) spectrum of $(-1)^n \cdot x(n)$.



2.4.2 Positioning Sampled Spectra at $f_s/4$

In many signal processing applications we'll find it useful to use an f_s bandpass sampling rate that forces the sampled spectra to be centered exactly at $\pm f_s/4$ as shown in [Figure 2-10\(a\)](#). As we'll see in later chapters, this scenario greatly simplifies certain common operations such as digital filtering, complex down-conversion, and Hilbert transformations.

To ensure that sampled spectra reside at $\pm f_s/4$, we select f_s using

$$(2-11)$$

$$f_s = \frac{4f_c}{2k-1}, \text{ where } k = 1, 2, 3, \dots$$

where f_c is the center frequency of the original analog signal's bandpass signal.

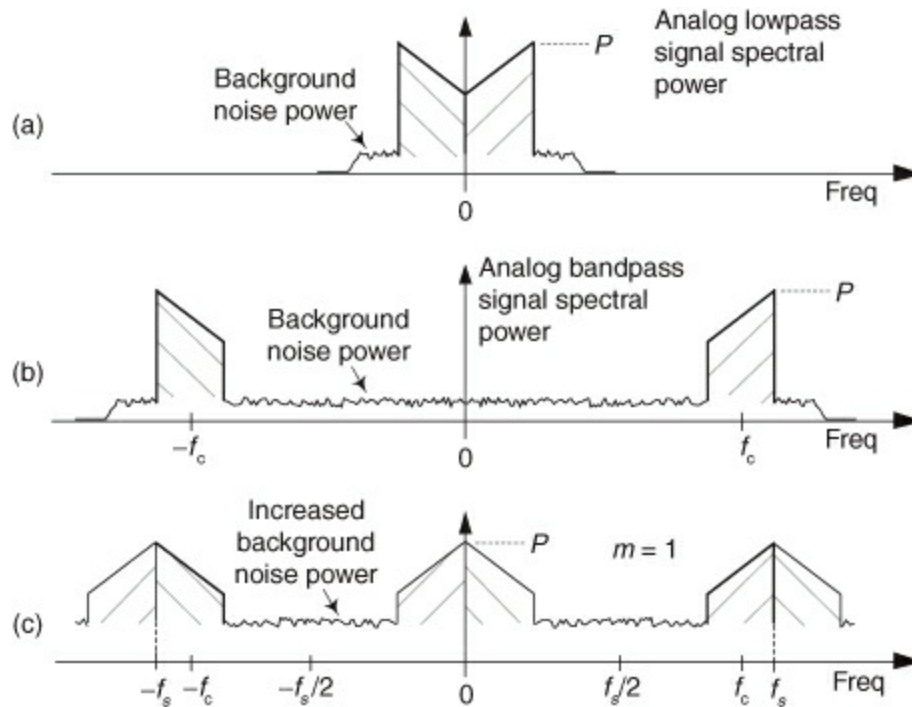
2.4.3 Noise in Bandpass-Sampled Signals

We have painted a rosy picture of bandpass sampling, with its analog signal capture capabilities at reduced sample rates. However, there is a negative aspect associated with bandpass sampling. The signal-to-noise ratio (SNR), the ratio of the power of a signal over the total background noise power, of our digitized signal is degraded when we perform bandpass sampling. (A general discussion of SNR is provided in [Appendix D](#).)

Here's the story. The spectrum of an analog lowpass signal, output from an analog anti-aliasing

lowpass filter, is that shown in [Figure 2-11\(a\)](#). That lowpass signal contains some amount of background noise power. Now if an analog bandpass signal is likewise contaminated with background noise, as shown by the spectral plot in [Figure 2-11\(b\)](#), the bandpass-sampled signal will have an increased level of background noise as shown in [Figure 2-11\(c\)](#). That's because all of the background spectral noise in [Figure 2-11\(b\)](#) must now reside in the range of $-f_s/2$ to $f_s/2$ in [Figure 2-11\(c\)](#). As such, the bandpass-sampled signal's SNR is reduced (degraded).

Figure 2-11 Sampling SNR degradation: (a) analog lowpass signal spectral power; (b) analog bandpass signal spectral power; (c) bandpass-sampled signal spectral power when $m = 1$.



As detailed in reference [\[6\]](#), if the analog bandpass signal's background noise spectral power level is relatively flat, as in [Figure 2-11\(b\)](#), the bandpass-sampled background noise power increases by a factor of $m + 1$ (the denominator of the right-side ratio in [Eq. \(2-10\)](#)) while the desired signal power P remains unchanged. As such, the bandpass-sampled signal's SNR, measured in decibels, is reduced by

(2-12)

$$D_{\text{SNR}} = 10 \cdot \log_{10}(m + 1) \text{ dB}$$

below the SNR of the original analog signal. So for the [Figure 2-11](#) example, when $m = 1$, the bandpass-sampled signal's background noise power doubles, and the total bandpass-sampled signal's SNR is $D_{\text{SNR}} = 3 \text{ dB}$ less than the analog bandpass signal's SNR.

The notion of using decibels, a very convenient method of comparing the power of two signals (the two signals, in this case, are our bandpass signal and the background noise signal), is discussed in [Appendix E](#).

References

- [1] Crochiere, R., and Rabiner, L. "Optimum FIR Digital Implementations for Decimation, Interpolation, and Narrow-band Filtering," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-23, No. 5, October 1975.
- [2] Luke, H. "The Origins of the Sampling Theorem," *IEEE Communications Magazine*, April 1999,

- [3] Shannon, C. “A Mathematical Theory of Communication,” *Bell Sys. Tech. Journal*, Vol. 27, 1948, pp. 379–423, 623–656.
- [4] Steyskal, H. “Digital Beamforming Antennas,” *Microwave Journal*, January 1987.
- [5] Hill, G. “The Benefits of Undersampling,” *Electronic Design*, July 11, 1994.
- [6] Vaughan, R., et al., “The Theory of Bandpass Sampling,” *IEEE Trans. on Signal Processing*, Vol. 39, No. 9, September 1991, pp. 1973–1984.

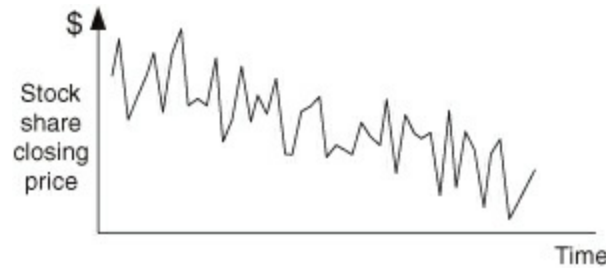
Chapter 2 Problems

- 2.1** Suppose you have a mechanical clock that has a minute hand, but no hour hand. Next, suppose you took a photograph of the clock when the minute hand was pointed at 12:00 noon and then took additional photos every 55 minutes. Upon showing those photos, in time order, to someone:
- (a) What would that person think about the direction of motion of the minute hand as time advances?
 - (b) With the idea of *lowpass sampling* in mind, how often would you need to take photos, measured in photos/hour, so that the successive photos show proper (true) clockwise minute-hand rotation?
- 2.2** Assume we sampled a continuous $x(t)$ signal and obtained 100 $x(n)$ time-domain samples. What important information (parameter that we need to know in order to analyze $x(t)$) is missing from the $x(n)$ sequence?
- 2.3** National Instruments Corporation produces an analog-to-digital (A/D) converter (Model #NI-5154) that can sample (digitize) an analog signal at a sample rate of $f_s = 2.0$ GHz (gigahertz).
- (a) What is the t_s period of the output samples of such a device?
 - (b) Each A/D output sample is an 8-bit binary word (one byte), and the converter is able to store 256 million samples. What is the maximum time interval over which the converter can continuously sample an analog signal?
- 2.4** Consider a continuous time-domain sinewave, whose cyclic frequency is 500 Hz, defined by
- $$x(t) = \cos[2\pi(500)t + \pi/7].$$
- Write the equation for the discrete $x(n)$ sinewave sequence that results from sampling $x(t)$ at an f_s sample rate of 4000 Hz.
- Note:** This problem is not “busy work.” If you ever want to model the $x(t)$ signal using software (MathCAD, MATLAB, Octave, etc.), then it is the desired $x(n)$ equation that you program into your software.
- 2.5** If we sampled a single continuous sinewave whose frequency is f_0 Hz, over what range must t_s (the time between digital samples) be to satisfy the Nyquist criterion? Express that t_s range in terms of f_0 .
- 2.6** Suppose we used the following statement to describe the Nyquist criterion for lowpass sampling: “When sampling a single continuous sinusoid (a single analog tone), we must obtain no fewer than N discrete samples per continuous sinewave cycle.” What is the value of this integer N ?

2.7 The Nyquist criterion, regarding the sampling of lowpass signals, is sometimes stated as “The sampling rate f_s must be equal to, or greater than, twice the highest spectral component of the continuous signal being sampled.” Can you think of how a continuous sinusoidal signal can be sampled in accordance with that Nyquist criterion definition to yield all zero-valued discrete samples?

2.8 Stock market analysts study time-domain *charts* (plots) of the *closing price* of stock shares. A typical plot takes the form of that in [Figure P2-8](#), where instead of plotting discrete closing price sample values as dots, they draw straight lines connecting the closing price value samples. What is the t_s period for such stock market charts?

Figure P2-8



2.9 Consider a continuous time-domain sinewave defined by

$$x(t) = \cos(4000\pi t)$$

that was sampled to produce the discrete sinewave sequence defined by

$$x(n) = \cos(n\pi/2).$$

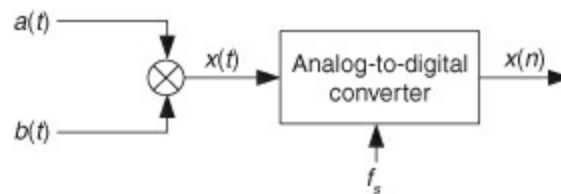
What is the f_s sample rate, measured in Hz, that would result in sequence $x(n)$?

2.10 Consider the two continuous signals defined by

$$a(t) = \cos(4000\pi t) \text{ and } b(t) = \cos(200\pi t)$$

whose product yields the $x(t)$ signal shown in [Figure P2-10](#). What is the minimum f_s sample rate, measured in Hz, that would result in a sequence $x(n)$ with no aliasing errors (no spectral replication overlap)?

Figure P2-10



2.11 Consider a discrete time-domain sinewave sequence defined by

$$x(n) = \sin(n\pi/4)$$

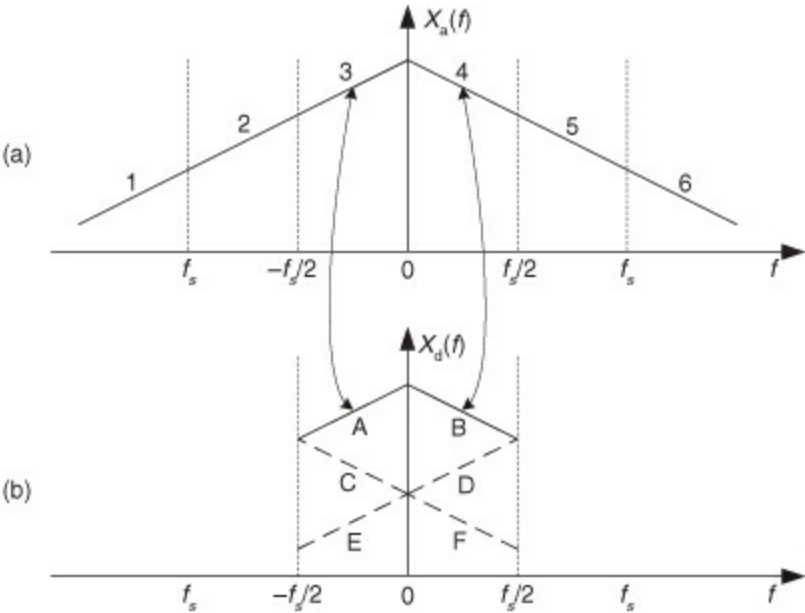
that was obtained by sampling an analog $x(t) = \sin(2\pi f_0 t)$ sinewave signal whose frequency is f_0 Hz. If the sample rate of $x(n)$ is $f_s = 160$ Hz, what are three possible positive frequency values, measured in Hz, for f_0 that would result in sequence $x(n)$?

2.12 In the text we discussed the notion of *spectral folding* that can take place when an $x_a(t)$ analog

signal is sampled to produce a discrete $x_d(n)$ sequence. We also stated that all of the analog spectral energy contained in $X_a(f)$ will reside within the frequency range of $\pm f_s/2$ of the $X_d(f)$ spectrum of the sampled $x_d(n)$ sequence. Given those concepts, consider the spectrum of an analog signal shown in [Figure P2-12\(a\)](#) whose spectrum is divided into the six segments marked as 1 to 6. Fill in the following table showing which of the A-to-F spectral segments of $X_d(f)$, shown in [Figure P2-12\(b\)](#), are aliases of the 1-to-6 spectral segments of $X_a(f)$.

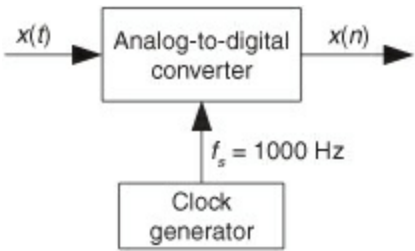
Aliased Spectral Segments	
$X_d(f)$ spectral segment	Associated $X_a(f)$ spectral segment
A	3
B	4
C	
D	
E	
F	

Figure P2-12



2.13 Consider the simple analog signal defined by $x(t) = \sin(2\pi 700t)$ shown in [Figure P2-13](#). Draw the spectrum of $x(n)$ showing all spectral components, labeling their frequency locations, in the frequency range $-2f_s$ to $+2f_s$.

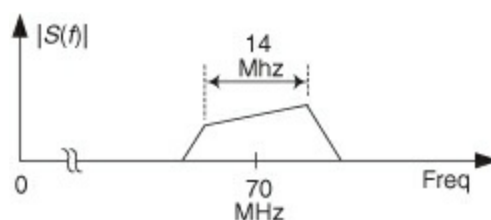
Figure P2-13



2.14 The Nançay Observatory, in France, uses a radio astronomy receiver that generates a wideband

analog $s(t)$ signal whose spectral magnitude is represented in [Figure P2-14](#). The Nançay scientists bandpass sample the analog $s(t)$ signal, using an analog-to-digital (A/D) converter to produce an $x(n)$ discrete sequence, at a sample rate of $f_s = 56$ MHz.

Figure P2-14

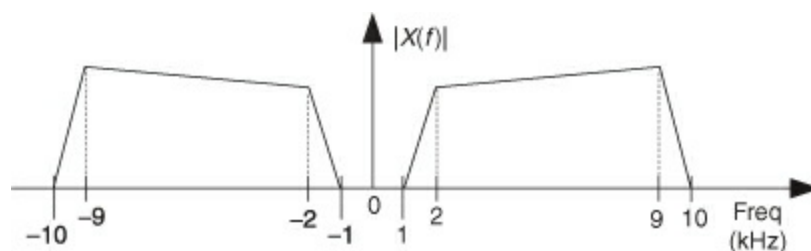


- (a) Draw the spectrum of the $x(n)$ sequence, $X(f)$, showing its spectral energy over the frequency range -70 MHz to 70 MHz.
- (b) What is the center frequency of the first positive-frequency spectral replication in $X(f)$?
- (c) How is your solution to Part (b) related to the f_s sample rate?

Hint: How is your solution to Part (b) related to $f_s/2$?

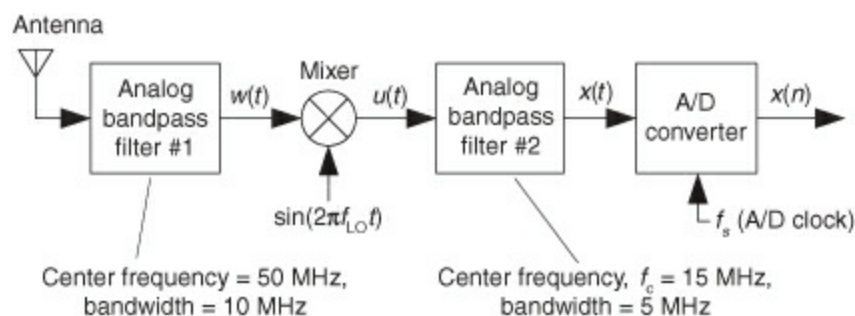
- 2.15 Think about the continuous (analog) signal $x(t)$ that has the spectral magnitude shown in [Figure P2-15](#). What is the minimum f_s sample rate for *lowpass sampling* such that no spectral overlap occurs in the frequency range of 2 to 9 kHz in the spectrum of the discrete $x(n)$ samples?

Figure P2-15



- 2.16 If a person wants to be classified as a soprano in classical opera, she must be able to sing notes in the frequency range of 247 Hz to 1175 Hz. What is the minimum f_s sampling rate allowable for *bandpass sampling* of the full audio spectrum of a singing soprano?
- 2.17 This problem requires the student to have some knowledge of electronics and how a mixer operates inside a radio. (The definition of a bandpass filter is given in [Appendix F](#).) Consider the simplified version of what is called a *superheterodyne* digital radio depicted in [Figure P2-17](#).

Figure P2-17



- (a) For what local oscillator frequency, f_{LO} , would an *image* (a copy, or duplication) of the $w(t)$ signal's spectrum be centered at 15 MHz (megahertz) in signal $u(t)$?

(b) What is the purpose of the analog bandpass filter #2?

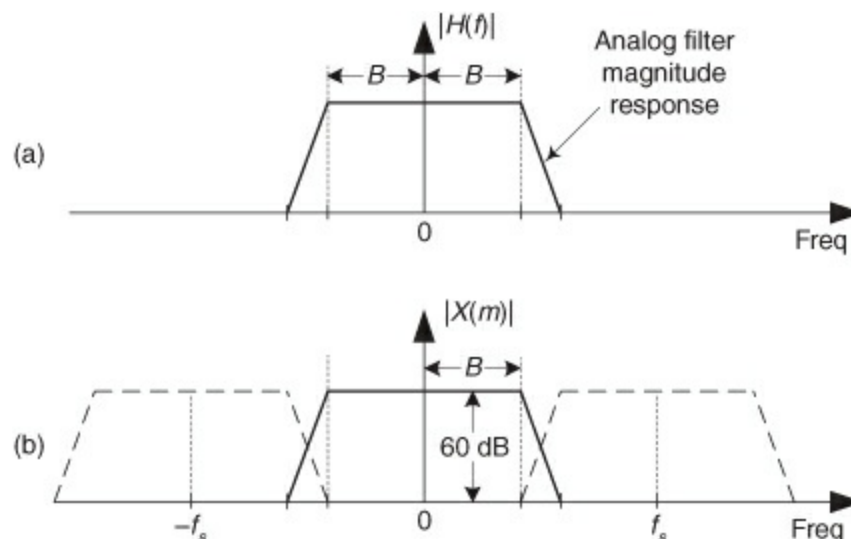
(c) Fill in the following table showing all ranges of acceptable f_s bandpass sampling rates to avoid aliasing errors in the discrete $x(n)$ sequence. Also list, in the rightmost column, for which values of m the sampled spectrum, centered at 15 MHz, will be inverted.

m	$(2f_c - B)/m$	$(2f_c + B)/(m+1)$	Positive-frequency sampled spectrum is inverted (Yes or No)
1			
2			
3			

(d) In digital receivers, to simplify AM and FM demodulation, it is advantageous to have the spectrum of the discrete $x(n)$ sequence be centered at one-quarter of the sample rate. The text's [Eq. 2-11](#) describes how to achieve this situation. If we were constrained to have f_s equal to 12 MHz, what would be the maximum f_{LO} local oscillator frequency such that the spectra of $u(t)$, $x(t)$, and $x(n)$ are centered at $f_s/4$? (Note: In this scenario, the f_c center frequency of analog bandpass filter #2 will no longer be 15 MHz.)

2.18 Think about the analog anti-aliasing filter given in [Figure P2-18\(a\)](#), having a one-sided bandwidth of B Hz. A wideband analog signal passed through that filter, and then sampled, would have an $|X(m)|$ spectrum as shown in [Figure P2-18\(b\)](#), where the dashed curves represent spectral replications.

Figure P2-18

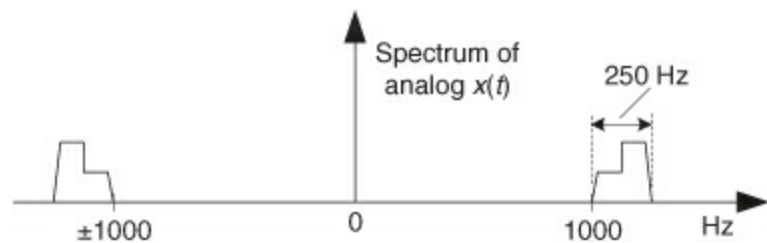


Suppose we desired that all aliased spectral components in $|X(m)|$ over our B Hz bandwidth of interest must be attenuated by at least 60 dB. Determine the equation, in terms of B and the f_s sampling rate, for the frequency at which the anti-aliasing filter must have an attenuation value of -60 dB. The solution to this problem gives us a useful *rule of thumb* we can use in specifying the desired performance of analog anti-aliasing filters.

2.19 This problem demonstrates a popular way of performing frequency down-conversion (converting a bandpass signal into a lowpass signal) by way of bandpass sampling. Consider the continuous 250-Hz-wide bandpass $x(t)$ signal whose spectral magnitude is shown in [Figure P2-19](#).

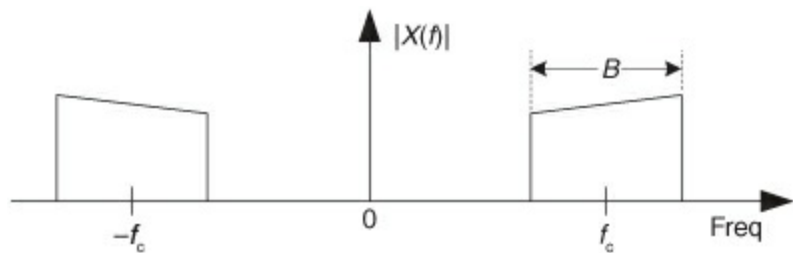
Draw the spectrum, over the frequency range of $-1.3f_s$ to $+1.3f_s$, of the $x(n)$ sampled sequence obtained when $x(t)$ is sampled at $f_s = 1000$ samples/second.

Figure P2-19



2.20 Here’s a problem to test your understanding of bandpass sampling. Think about the continuous (analog) signal $x(t)$ that has the spectral magnitude shown in [Figure P2-20](#).

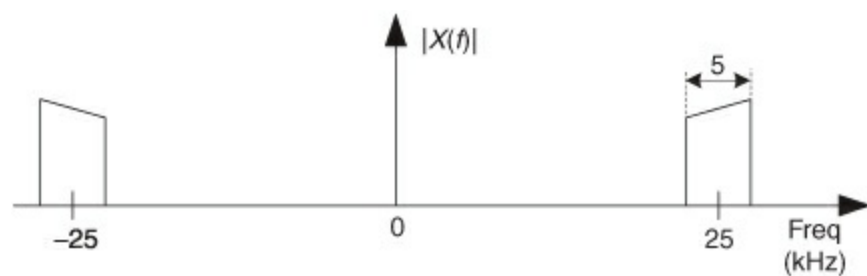
Figure P2-20



- (a) What is the minimum f_c center frequency, in terms of $x(t)$ ’s bandwidth B , that enables bandpass sampling of $x(t)$? Show your work.
- (b) Given your results in Part (a) above, determine if it is possible to perform bandpass sampling of the full spectrum of the commercial AM (amplitude modulation) broadcast radio band in North America. Explain your solution.

2.21 Suppose we want to perform bandpass sampling of a continuous 5 kHz-wide bandpass signal whose spectral magnitude is shown in [Figure P2-21](#).

Figure P2-21



Fill in the following table showing the various ranges of acceptable f_s bandpass sampling rates, similar to the text’s [Table 2-1](#), to avoid aliasing errors. Also list, in the rightmost column, for which values of m the sampled spectrum in the vicinity of zero Hz is inverted.

Acceptable Bandpass Sample Rate Ranges

m	$(2f_c - B)/m$	$(2f_c + B)/(m+1)$	Positive-frequency sampled spectrum is inverted (Yes or No)
1			
2			
3			
4			
5			

2.22 I recently encountered an Internet website that allegedly gave an algorithm for the minimum f_s bandpass sampling rate for an analog bandpass signal centered at f_c Hz, whose bandwidth is B Hz. The algorithm is

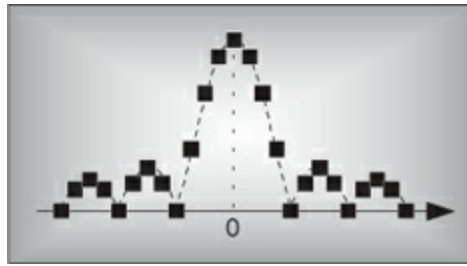
$$f_{s,\min} = \frac{4f_c}{2Z - 1}$$

where

$$Z = \left\lfloor \frac{4f_c + 2B}{4B} \right\rfloor.$$

In the above notation, $\lfloor x \rfloor$ means the integer part of x . Here's the problem: Is the above $f_{s,\min}$ algorithm correct in computing the *absolute minimum* possible nonaliasing f_s bandpass sampling rate for an analog bandpass signal centered at f_c Hz, whose bandwidth is B Hz? Verify your answer with an example.

Chapter Three. The Discrete Fourier Transform



The discrete Fourier transform (DFT) is one of the two most common, and powerful, procedures encountered in the field of digital signal processing. (Digital filtering is the other.) The DFT enables us to analyze, manipulate, and synthesize signals in ways not possible with continuous (analog) signal processing. Even though it's now used in almost every field of engineering, we'll see applications for DFT continue to flourish as its utility becomes more widely understood. Because of this, a solid understanding of the DFT is mandatory for anyone working in the field of digital signal processing.

The DFT is a mathematical procedure used to determine the harmonic, or frequency, content of a discrete signal sequence. Although, for our purposes, a discrete signal sequence is a set of values obtained by periodic sampling of a continuous signal in the time domain, we'll find that the DFT is useful in analyzing any discrete sequence regardless of what that sequence actually represents. The DFT's origin, of course, is the continuous Fourier transform $X(f)$ defined as

$$(3-1)$$
$$X(f) = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi ft} dt$$

where $x(t)$ is some continuous time-domain signal.[‡]

[‡] Fourier is pronounced 'for-yā. In engineering school, we called [Eq. \(3-1\)](#) the "four-year" transform because it took about four years to do one homework problem.

In the field of continuous signal processing, [Eq. \(3-1\)](#) is used to transform an expression of a continuous time-domain function $x(t)$ into a continuous frequency-domain function $X(f)$. Subsequent evaluation of the $X(f)$ expression enables us to determine the frequency content of any practical signal of interest and opens up a wide array of signal analysis and processing possibilities in the fields of engineering and physics. One could argue that the Fourier transform is the most dominant and widespread mathematical mechanism available for the analysis of physical systems. (A prominent quote from Lord Kelvin better states this sentiment: "Fourier's theorem is not only one of the most beautiful results of modern analysis, but it may be said to furnish an indispensable instrument in the treatment of nearly every recondite question in modern physics." By the way, the history of Fourier's original work in harmonic analysis, relating to the problem of heat conduction, is fascinating. References [\[1\]](#) and [\[2\]](#) are good places to start for those interested in the subject.)

With the advent of the digital computer, the efforts of early digital processing pioneers led to the development of the DFT defined as the discrete frequency-domain sequence $X(m)$, where

$$(3-2)$$

DFT equation
(exponential form): \rightarrow

$$X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N}.$$

For our discussion of [Eq. \(3-2\)](#), $x(n)$ is a discrete sequence of time-domain sampled values of the continuous variable $x(t)$. The “ e ” in [Eq. \(3-2\)](#) is, of course, the base of natural logarithms and $j = \sqrt{-1}$.

3.1 Understanding the DFT Equation

[Equation \(3-2\)](#) has a tangled, almost unfriendly, look about it. Not to worry. After studying this chapter, [Eq. \(3-2\)](#) will become one of our most familiar and powerful tools in understanding digital signal processing. Let’s get started by expressing [Eq. \(3-2\)](#) in a different way and examining it carefully. From *Euler’s relationship*, $e^{-j\theta} = \cos(\theta) - j\sin(\theta)$, [Eq. \(3-2\)](#) is equivalent to

(3-3)

DFT equation
(rectangular form): \rightarrow

$$X(m) = \sum_{n=0}^{N-1} x(n)[\cos(2\pi nm/N) - j\sin(2\pi nm/N)].$$

We have separated the complex exponential of [Eq. \(3-2\)](#) into its real and imaginary components where

$X(m)$ = the m th DFT output component, i.e., $X(0)$, $X(1)$, $X(2)$, $X(3)$, etc.,

m = the index of the DFT output in the frequency domain, $m = 0, 1, 2, 3, \dots, N-1$,

$x(n)$ = the sequence of input samples, $x(0)$, $x(1)$, $x(2)$, $x(3)$, etc.,

n = the time-domain index of the input samples, $n = 0, 1, 2, 3, \dots, N-1$,

$j = \sqrt{-1}$, and

N = the number of samples of the input sequence and the number of frequency points in the DFT output.

Although it looks more complicated than [Eq. \(3-2\)](#), [Eq. \(3-3\)](#) turns out to be easier to understand. (If you’re not too comfortable with it, don’t let the $j = \sqrt{-1}$ concept bother you too much. It’s merely a convenient abstraction that helps us compare the phase relationship between various sinusoidal components of a signal. [Chapter 8](#) discusses the j operator in some detail.)[‡] The indices for the input samples (n) and the DFT output samples (m) always go from 0 to $N-1$ in the standard DFT notation. This means that with N input time-domain sample values, the DFT determines the spectral content of the input at N equally spaced frequency points. The value N is an important parameter because it determines how many input samples are needed, the resolution of the frequency-domain results, and the amount of processing time necessary to calculate an N -point DFT.

[‡] Instead of the letter j , be aware that mathematicians often use the letter i to represent the $\sqrt{-1}$ operator.

It’s useful to see the structure of [Eq. \(3-3\)](#) by eliminating the summation and writing out all the terms. For example, when $N = 4$, n and m both go from 0 to 3, and [Eq. \(3-3\)](#) becomes

(3-4a)

$$X(m) = \sum_{n=0}^3 x(n)[\cos(2\pi nm/4) - j\sin(2\pi nm/4)].$$

Writing out all the terms for the first DFT output term corresponding to $m = 0$,

(3-4b)

$$\begin{aligned}
X(0) = & x(0)\cos(2\pi \cdot 0 \cdot 0 / 4) - jx(0)\sin(2\pi \cdot 0 \cdot 0 / 4) \\
& + x(1)\cos(2\pi \cdot 1 \cdot 0 / 4) - jx(1)\sin(2\pi \cdot 1 \cdot 0 / 4) \\
& + x(2)\cos(2\pi \cdot 2 \cdot 0 / 4) - jx(2)\sin(2\pi \cdot 2 \cdot 0 / 4) \\
& + x(3)\cos(2\pi \cdot 3 \cdot 0 / 4) - jx(3)\sin(2\pi \cdot 3 \cdot 0 / 4).
\end{aligned}$$

For the second DFT output term corresponding to $m = 1$, [Eq. \(3-4a\)](#) becomes

(3-4c)

$$\begin{aligned}
X(1) = & x(0)\cos(2\pi \cdot 0 \cdot 1 / 4) - jx(0)\sin(2\pi \cdot 0 \cdot 1 / 4) \\
& + x(1)\cos(2\pi \cdot 1 \cdot 1 / 4) - jx(1)\sin(2\pi \cdot 1 \cdot 1 / 4) \\
& + x(2)\cos(2\pi \cdot 2 \cdot 1 / 4) - jx(2)\sin(2\pi \cdot 2 \cdot 1 / 4) \\
& + x(3)\cos(2\pi \cdot 3 \cdot 1 / 4) - jx(3)\sin(2\pi \cdot 3 \cdot 1 / 4).
\end{aligned}$$

For the third output term corresponding to $m = 2$, [Eq. \(3-4a\)](#) becomes

(3-4d)

$$\begin{aligned}
X(2) = & x(0)\cos(2\pi \cdot 0 \cdot 2 / 4) - jx(0)\sin(2\pi \cdot 0 \cdot 2 / 4) \\
& + x(1)\cos(2\pi \cdot 1 \cdot 2 / 4) - jx(1)\sin(2\pi \cdot 1 \cdot 2 / 4) \\
& + x(2)\cos(2\pi \cdot 2 \cdot 2 / 4) - jx(2)\sin(2\pi \cdot 2 \cdot 2 / 4) \\
& + x(3)\cos(2\pi \cdot 3 \cdot 2 / 4) - jx(3)\sin(2\pi \cdot 3 \cdot 2 / 4).
\end{aligned}$$

Finally, for the fourth and last output term corresponding to $m = 3$, [Eq. \(3-4a\)](#) becomes

(3-4e)

$$\begin{aligned}
X(3) = & x(0)\cos(2\pi \cdot 0 \cdot 3 / 4) - jx(0)\sin(2\pi \cdot 0 \cdot 3 / 4) \\
& + x(1)\cos(2\pi \cdot 1 \cdot 3 / 4) - jx(1)\sin(2\pi \cdot 1 \cdot 3 / 4) \\
& + x(2)\cos(2\pi \cdot 2 \cdot 3 / 4) - jx(2)\sin(2\pi \cdot 2 \cdot 3 / 4) \\
& + x(3)\cos(2\pi \cdot 3 \cdot 3 / 4) - jx(3)\sin(2\pi \cdot 3 \cdot 3 / 4).
\end{aligned}$$

The above multiplication symbol “.” in [Eq. \(3-4\)](#) is used merely to separate the factors in the sine and cosine terms. The pattern in [Eqs. \(3-4b\)](#) through [\(3-4e\)](#) is apparent now, and we can certainly see why it’s convenient to use the summation sign in [Eq. \(3-3\)](#). Each $X(m)$ DFT output term is the sum of the *point-for-point* product between an input sequence of signal values and a complex sinusoid of the form $\cos(\phi) - j\sin(\phi)$. The exact frequencies of the different sinusoids depend on both the sampling rate f_s at which the original signal was sampled, and the number of samples N . For example, if we are sampling a continuous signal at a rate of 500 samples/second and, then, perform a 16-point DFT on the sampled data, the fundamental frequency of the sinusoids is $f_s/N = 500/16$ or 31.25 Hz. The other $X(m)$ analysis frequencies are integral multiples of the fundamental frequency, i.e.,

$X(0)$ = 1st frequency term, with analysis frequency = $0 \cdot 31.25 = 0$ Hz,
 $X(1)$ = 2nd frequency term, with analysis frequency = $1 \cdot 31.25 = 31.25$ Hz,
 $X(2)$ = 3rd frequency term, with analysis frequency = $2 \cdot 31.25 = 62.5$ Hz,
 $X(3)$ = 4th frequency term, with analysis frequency = $3 \cdot 31.25 = 93.75$ Hz,

...

...

$X(15)$ = 16th frequency term, with analysis frequency = $15 \cdot 31.25 = 468.75$ Hz.

The N separate DFT analysis frequencies are

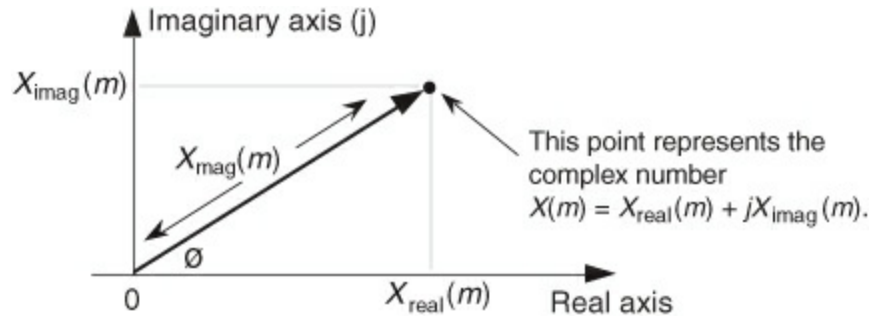
(3-5)

$$f_{\text{analysis}}(m) = \frac{mf_s}{N}.$$

So, in this example, the $X(0)$ DFT term tells us the magnitude of any 0 Hz DC (direct current) component contained in the input signal, the $X(1)$ term specifies the magnitude of any 31.25 Hz component in the input signal, and the $X(2)$ term indicates the magnitude of any 62.5 Hz component in the input signal, etc. Moreover, as we'll soon show by example, the DFT output terms also determine the phase relationship between the various analysis frequencies contained in an input signal.

Quite often we're interested in both the magnitude and the power (magnitude squared) contained in each $X(m)$ term, and the standard definitions for right triangles apply here as depicted in [Figure 3-1](#).

Figure 3-1 Trigonometric relationships of an individual DFT $X(m)$ complex output value.



If we represent an arbitrary DFT output value, $X(m)$, by its real and imaginary parts

$$(3-6)$$

$$X(m) = X_{\text{real}}(m) + jX_{\text{imag}}(m) = X_{\text{mag}}(m) \text{ at an angle of } X_{\theta}(m),$$

the magnitude of $X(m)$ is

$$(3-7)$$

$$X_{\text{mag}}(m) = |X(m)| = \sqrt{X_{\text{real}}(m)^2 + X_{\text{imag}}(m)^2}.$$

By definition, the phase angle of $X(m)$, $X_{\theta}(m)$, is

$$(3-8)$$

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

The power of $X(m)$, referred to as the power spectrum, is the magnitude squared where

$$(3-9)$$

$$X_{\text{PS}}(m) = X_{\text{mag}}(m)^2 = X_{\text{real}}(m)^2 + X_{\text{imag}}(m)^2.$$

3.1.1 DFT Example 1

The above [Eqs. \(3-2\)](#) and [\(3-3\)](#) will become more meaningful by way of an example, so let's go through a simple one step by step. Let's say we want to sample and perform an 8-point DFT on a continuous input signal containing components at 1 kHz and 2 kHz, expressed as

$$(3-10)$$

$$x_{\text{in}}(t) = \sin(2\pi \cdot 1000 \cdot t) + 0.5\sin(2\pi \cdot 2000 \cdot t + 3\pi/4).$$

To make our example input signal $x_{\text{in}}(t)$ a little more interesting, we have the 2 kHz term shifted in

phase by 135° ($3\pi/4$ radians) relative to the 1 kHz sinewave. With a sample rate of f_s , we sample the input every $1/f_s = t_s$ seconds. Because $N = 8$, we need 8 input sample values on which to perform the DFT. So the 8-element sequence $x(n)$ is equal to $x_{\text{in}}(t)$ sampled at the nt_s instants in time so that

$$(3-11)$$

$$x(n) = x_{\text{in}}(nt_s) = \sin(2\pi \cdot 1000 \cdot nt_s) + 0.5\sin(2\pi \cdot 2000 \cdot nt_s + 3\pi/4).$$

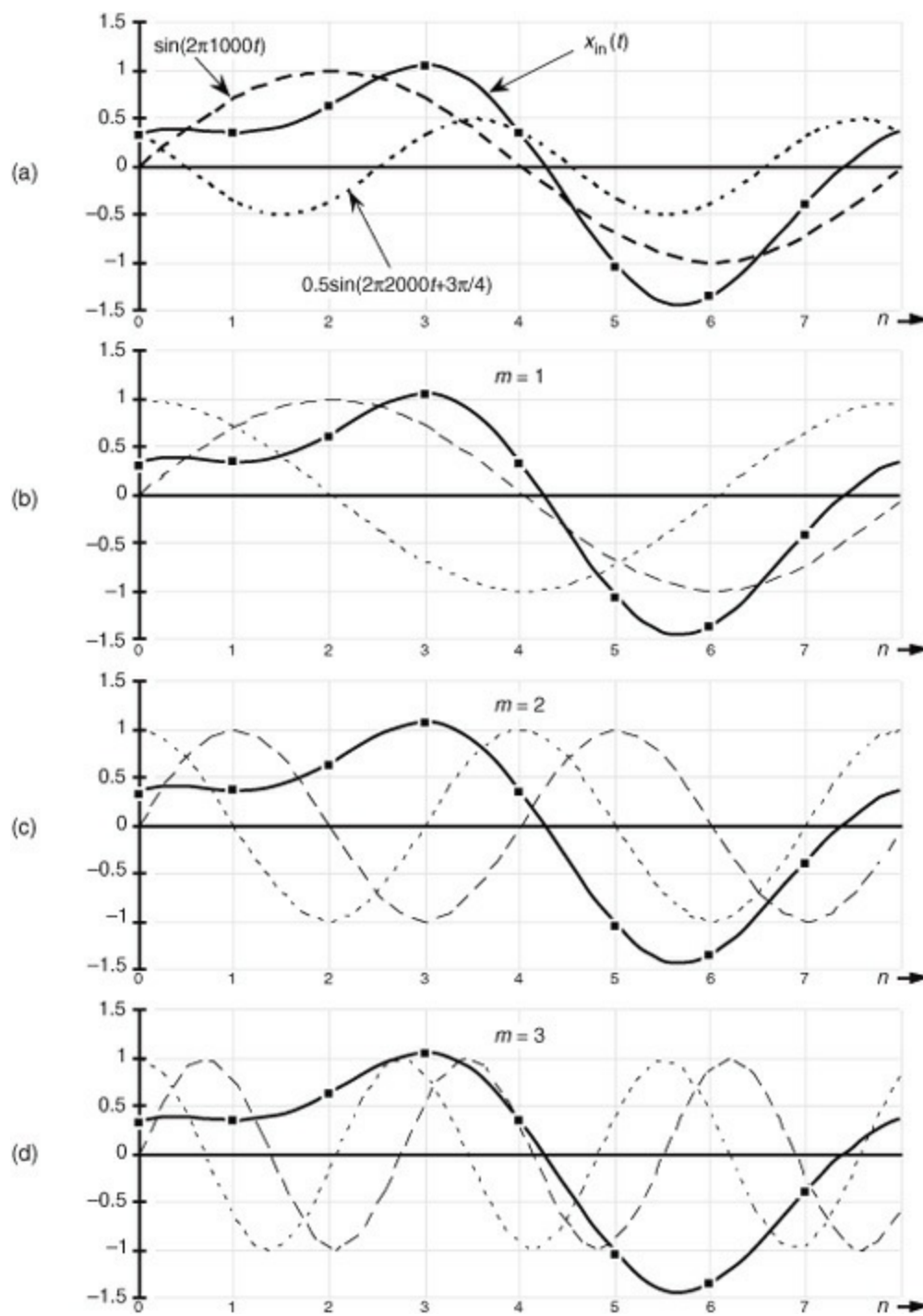
If we choose to sample $x_{\text{in}}(t)$ at a rate of $f_s = 8000$ samples/second from [Eq. \(3-5\)](#), our DFT results will indicate what signal amplitude exists in $x(n)$ at the analysis frequencies of mf_s/N , or 0 kHz, 1 kHz, 2 kHz, . . . , 7 kHz. With $f_s = 8000$ samples/second, our eight $x(n)$ samples are

$$(3-11')$$

$$\begin{aligned} x(0) &= 0.3535, & x(1) &= 0.3535, \\ x(2) &= 0.6464, & x(3) &= 1.0607, \\ x(4) &= 0.3535, & x(5) &= -1.0607, \\ x(6) &= -1.3535, & x(7) &= -0.3535. \end{aligned}$$

These $x(n)$ sample values are the dots plotted on the solid continuous $x_{\text{in}}(t)$ curve in [Figure 3-2\(a\)](#). (Note that the sum of the sinusoidal terms in [Eq. \(3-10\)](#), shown as the dashed curves in [Figure 3-2\(a\)](#), is equal to $x_{\text{in}}(t)$.)

Figure 3-2 DFT Example 1: (a) the input signal; (b) the input signal and the $m = 1$ sinusoids; (c) the input signal and the $m = 2$ sinusoids; (d) the input signal and the $m = 3$ sinusoids.



Now we're ready to apply [Eq. \(3-3\)](#) to determine the DFT of our $x(n)$ input. We'll start with $m = 1$ because the $m = 0$ case leads to a special result that we'll discuss shortly. So, for $m = 1$, or the 1 kHz ($mf_s/N = 1 \cdot 8000/8$) DFT frequency term, [Eq. \(3-3\)](#) for this example becomes

$$(3-12)$$

$$X(1) = \sum_{n=0}^7 x(n) \cos(2\pi n / 8) - jx(n) \sin(2\pi n / 8).$$

Next we multiply $x(n)$ by successive points on the cosine and sine curves of the first analysis frequency that have a single cycle over our eight input samples. In our example, for $m = 1$, we'll sum the products of the $x(n)$ sequence with a 1 kHz cosine wave and a 1 kHz sinewave evaluated at the angular values of $2\pi n/8$. Those analysis sinusoids are shown as the dashed curves in [Figure 3-2\(b\)](#). Notice how the cosine and sinewaves have $m = 1$ complete cycles in our sample interval.

Substituting our $x(n)$ sample values into [Eq. \(3-12\)](#) and listing the cosine terms in the left column and the sine terms in the right column, we have

$$\begin{array}{lll}
X(1) = 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) & \leftarrow \text{this is the } n = 0 \text{ term} \\
+ 0.3535 \cdot 0.707 & -j(0.3535 \cdot 0.707) & \leftarrow \text{this is the } n = 1 \text{ term} \\
+ 0.6464 \cdot 0.0 & -j(0.6464 \cdot 1.0) & \leftarrow \text{this is the } n = 2 \text{ term} \\
+ 1.0607 \cdot -0.707 & -j(1.0607 \cdot 0.707) & \dots \\
+ 0.3535 \cdot -1.0 & -j(0.3535 \cdot 0.0) & \dots \\
- 1.0607 \cdot -0.707 & -j(-1.0607 \cdot -0.707) & \dots \\
- 1.3535 \cdot 0.0 & -j(-1.3535 \cdot -1.0) & \dots \\
- 0.3535 \cdot 0.707 & -j(-0.3535 \cdot -0.707) & \leftarrow \text{this is the } n = 7 \text{ term} \\
\\
= 0.3535 & + j0.0 & \\
+ 0.250 & -j0.250 & \\
+ 0.0 & -j0.6464 & \\
- 0.750 & -j0.750 & \\
- 0.3535 & -j0.0 & \\
+ 0.750 & -j0.750 & \\
+ 0.0 & -j1.3535 & \\
- 0.250 & -j0.250 & \\
\\
= 0.0 - j4.0 = 4 \angle -90^\circ.
\end{array}$$

So we now see that the input $x(n)$ contains a signal component at a frequency of 1 kHz. Using [Eqs. \(3-7\)](#), [\(3-8\)](#), and [\(3-9\)](#) for our $X(1)$ result, $X_{\text{mag}}(1) = 4$, $X_{\text{PS}}(1) = 16$, and $X(1)$'s phase angle relative to a 1 kHz cosine is $X_\phi(1) = -90^\circ$.

For the $m = 2$ frequency term, we correlate $x(n)$ with a 2 kHz cosine wave and a 2 kHz sinewave. These waves are the dashed curves in [Figure 3-2\(c\)](#). Notice here that the cosine and sinewaves have $m = 2$ complete cycles in our sample interval in [Figure 3-2\(c\)](#). Substituting our $x(n)$ sample values in [Eq. \(3-3\)](#) for $m = 2$ gives

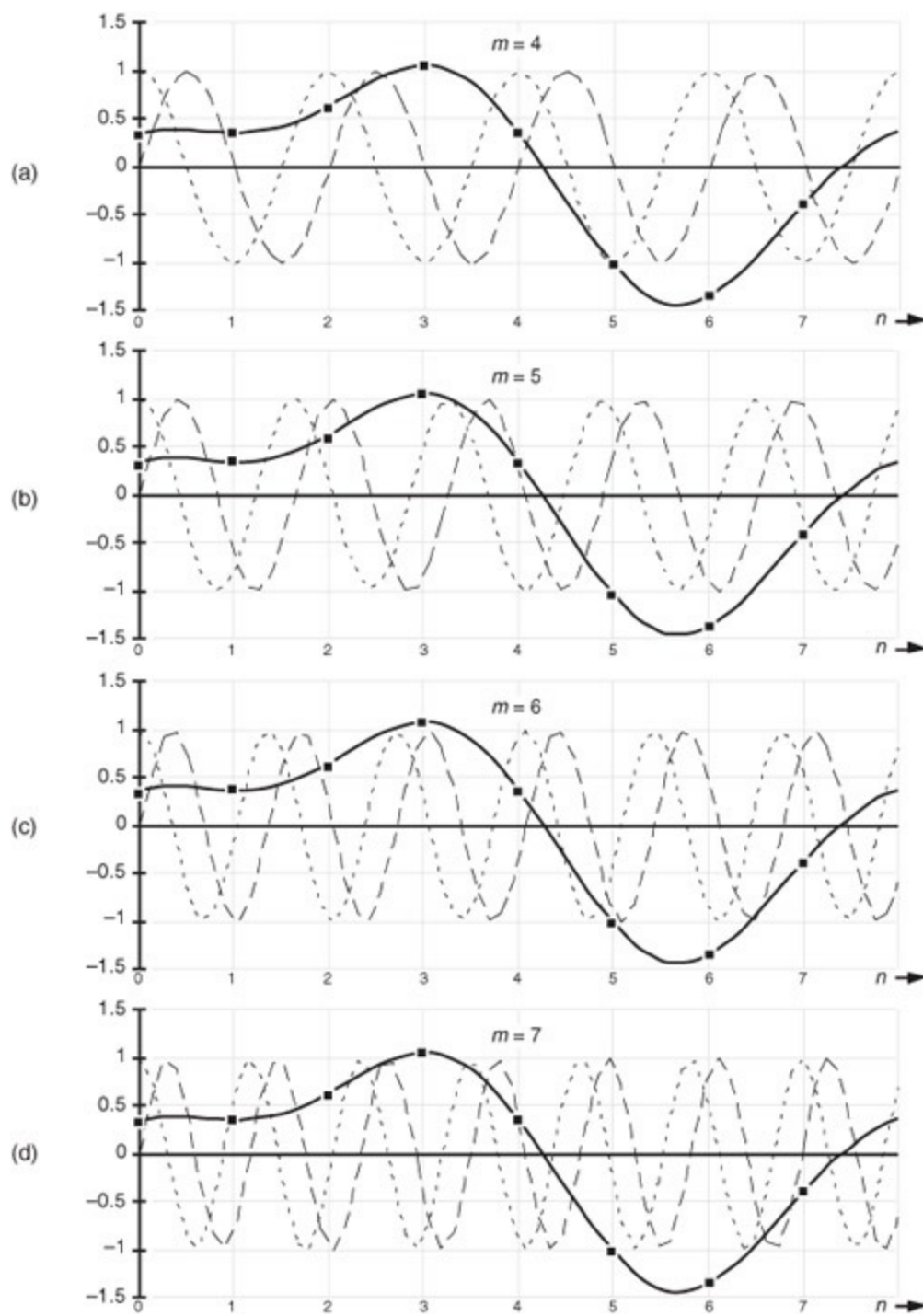
$$\begin{array}{ll}
X(2) = 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
+ 0.3535 \cdot 0.0 & -j(0.3535 \cdot 1.0) \\
+ 0.6464 \cdot -1.0 & -j(0.6464 \cdot 0.0) \\
+ 1.0607 \cdot 0.0 & -j(1.0607 \cdot -1.0) \\
+ 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
- 1.0607 \cdot 0.0 & -j(-1.0607 \cdot 1.0) \\
- 1.3535 \cdot -1.0 & -j(-1.3535 \cdot 0.0) \\
- 0.3535 \cdot 0.0 & -j(-0.3535 \cdot -1.0) \\
\\
= 0.3535 & + j0.0 \\
+ 0.0 & -j0.3535 \\
- 0.6464 & -j0.0 \\
- 0.0 & + j1.0607 \\
+ 0.3535 & -j0.0 \\
+ 0.0 & + j1.0607 \\
+ 1.3535 & -j0.0 \\
- 0.0 & -j0.3535 \\
\\
= 1.414 + j1.414 = 2 \angle 45^\circ.
\end{array}$$

Here our input $x(n)$ contains a signal at a frequency of 2 kHz whose relative amplitude is 2, and whose phase angle relative to a 2 kHz cosine is 45° . For the $m = 3$ frequency term, we correlate $x(n)$ with a 3 kHz cosine wave and a 3 kHz sinewave. These waves are the dashed curves in [Figure 3-2\(d\)](#). Again, see how the cosine and sinewaves have $m = 3$ complete cycles in our sample interval in [Figure 3-2\(d\)](#). Substituting our $x(n)$ sample values in [Eq. \(3-3\)](#) for $m = 3$ gives

$$\begin{aligned}
X(3) &= 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
&+ 0.3535 \cdot -0.707 & -j(0.3535 \cdot 0.707) \\
&+ 0.6464 \cdot 0.0 & -j(0.6464 \cdot -1.0) \\
&+ 1.0607 \cdot 0.707 & -j(1.0607 \cdot 0.707) \\
&+ 0.3535 \cdot -1.0 & -j(0.3535 \cdot 0.0) \\
&- 1.0607 \cdot 0.707 & -j(-1.0607 \cdot -0.707) \\
&- 1.3535 \cdot 0.0 & -j(-1.3535 \cdot 1.0) \\
&- 0.3535 \cdot -0.707 & -j(-0.3535 \cdot -0.707) \\
\\
&= 0.3535 & +j0.0 \\
&- 0.250 & -j0.250 \\
&+ 0.0 & +j0.6464 \\
&+ 0.750 & -j0.750 \\
&- 0.3535 & -j0.0 \\
&- 0.750 & -j0.750 \\
&+ 0.0 & +j1.3535 \\
&+ 0.250 & -j0.250 \\
\\
&= 0.0 - j0.0 = 0 \angle 0^\circ.
\end{aligned}$$

Our DFT indicates that $x(n)$ contained no signal at a frequency of 3 kHz. Let's continue our DFT for the $m = 4$ frequency term using the sinusoids in [Figure 3-3\(a\)](#).

Figure 3-3 DFT Example 1: (a) the input signal and the $m = 4$ sinusoids; (b) the input and the $m = 5$ sinusoids; (c) the input and the $m = 6$ sinusoids; (d) the input and the $m = 7$ sinusoids.



So [Eq. \(3-3\)](#) is

$$\begin{aligned}
X(4) &= 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
&+ 0.3535 \cdot -1.0 & -j(0.3535 \cdot 0.0) \\
&+ 0.6464 \cdot 1.0 & -j(0.6464 \cdot 0.0) \\
&+ 1.0607 \cdot -1.0 & -j(1.0607 \cdot 0.0) \\
&+ 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
&- 1.0607 \cdot -1.0 & -j(-1.0607 \cdot 0.0) \\
&- 1.3535 \cdot 1.0 & -j(-1.3535 \cdot 0.0) \\
&- 0.3535 \cdot -1.0 & -j(-0.3535 \cdot 0.0) \\
\\
&= 0.3535 & -j0.0 \\
&- 0.3535 & -j0.0 \\
&+ 0.6464 & -j0.0 \\
&- 1.0607 & -j0.0 \\
&+ 0.3535 & -j0.0 \\
&+ 1.0607 & -j0.0 \\
&- 1.3535 & -j0.0 \\
&+ 0.3535 & -j0.0 \\
\\
&= 0.0 - j0.0 = 0 \angle 0^\circ.
\end{aligned}$$

Our DFT for the $m = 5$ frequency term using the sinusoids in [Figure 3-3\(b\)](#) yields

$$\begin{aligned}
X(5) &= 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
&+ 0.3535 \cdot -0.707 & -j(0.3535 \cdot -0.707) \\
&+ 0.6464 \cdot 0.0 & -j(0.6464 \cdot 1.0) \\
&+ 1.0607 \cdot 0.707 & -j(1.0607 \cdot -0.707) \\
&+ 0.3535 \cdot -1.0 & -j(0.3535 \cdot 0.0) \\
&- 1.0607 \cdot 0.707 & -j(-1.0607 \cdot 0.707) \\
&- 1.3535 \cdot 0.0 & -j(-1.3535 \cdot -1.0) \\
&- 0.3535 \cdot -0.707 & -j(-0.3535 \cdot 0.707) \\
\\
&= 0.3535 & -j0.0 \\
&- 0.250 & +j0.250 \\
&+ 0.0 & -j0.6464 \\
&+ 0.750 & +j0.750 \\
&- 0.3535 & -j0.0 \\
&- 0.750 & +j0.750 \\
&+ 0.0 & -j1.3535 \\
&+ 0.250 & +j0.250 \\
\\
&= 0.0 - j.0 = 0 \angle 0^\circ.
\end{aligned}$$

For the $m = 6$ frequency term using the sinusoids in [Figure 3-3\(c\)](#), [Eq. \(3-3\)](#) is

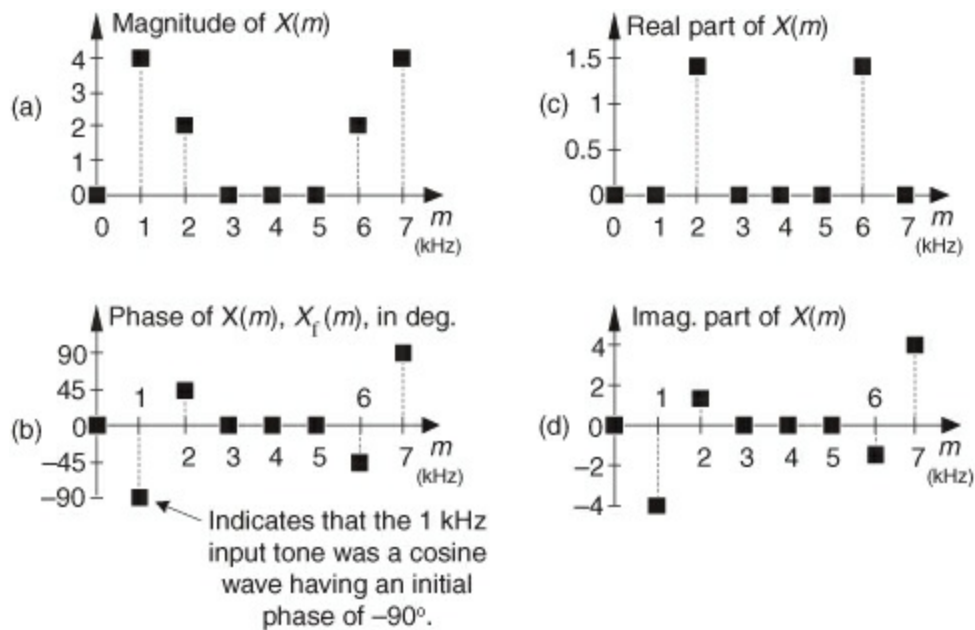
$$\begin{aligned}
X(6) &= 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
&+ 0.3535 \cdot 0.0 & -j(0.3535 \cdot -1.0) \\
&+ 0.6464 \cdot -1.0 & -j(0.6464 \cdot 0.0) \\
&+ 1.0607 \cdot 0.0 & -j(1.0607 \cdot 1.0) \\
&+ 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
&- 1.0607 \cdot 0.0 & -j(-1.0607 \cdot -1.0) \\
&- 1.3535 \cdot -1.0 & -j(-1.3535 \cdot 0.0) \\
&- 0.3535 \cdot 0.0 & -j(-0.3535 \cdot 1.0) \\
\\
&= 0.3535 & -j0.0 \\
&+ 0.0 & +j0.3535 \\
&- 0.6464 & -j0.0 \\
&+ 0.0 & -j1.0607 \\
&+ 0.3535 & -j0.0 \\
&+ 0.0 & -j1.0607 \\
&+ 1.3535 & -j0.0 \\
&+ 0.0 & +j0.3535 \\
\\
&= 1.414 - j1.414 = 2 \angle -45^\circ.
\end{aligned}$$

For the $m = 7$ frequency term using the sinusoids in [Figure 3-3\(d\)](#), [Eq. \(3-3\)](#) is

$$\begin{aligned}
X(7) &= 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
&+ 0.3535 \cdot 0.707 & -j(0.3535 \cdot -0.707) \\
&+ 0.6464 \cdot 0.0 & -j(0.6464 \cdot -1.0) \\
&+ 1.0607 \cdot -0.707 & -j(1.0607 \cdot -0.707) \\
&+ 0.3535 \cdot -1.0 & -j(0.3535 \cdot 0.0) \\
&- 1.0607 \cdot -0.707 & -j(-1.0607 \cdot 0.707) \\
&- 1.3535 \cdot 0.0 & -j(-1.3535 \cdot 1.0) \\
&- 0.3535 \cdot 0.707 & -j(-0.3535 \cdot 0.707) \\
\\
&= 0.3535 & + j0.0 \\
&+ 0.250 & + j0.250 \\
&+ 0.0 & + j0.6464 \\
&- 0.750 & + j0.750 \\
&- 0.3535 & - j0.0 \\
&+ 0.750 & + j0.750 \\
&+ 0.0 & + j1.3535 \\
&- 0.250 & + j0.250 \\
\\
&= 0.0 + j4.0 = 4 \angle 90^\circ.
\end{aligned}$$

If we plot the $X(m)$ output magnitudes as a function of frequency, we produce the magnitude *spectrum* of the $x(n)$ input sequence, shown in [Figure 3-4\(a\)](#). The phase angles of the $X(m)$ output terms are depicted in [Figure 3-4\(b\)](#).

Figure 3-4 DFT results from Example 1: (a) magnitude of $X(m)$; (b) phase of $X(m)$; (c) real part of $X(m)$; (d) imaginary part of $X(m)$.



Hang in there; we're almost finished with our example. We've saved the calculation of the $m = 0$ frequency term to the end because it has a special significance. When $m = 0$, we correlate $x(n)$ with $\cos(0) - j\sin(0)$ so that [Eq. \(3-3\)](#) becomes

(3-13)

$$X(0) = \sum_{n=0}^{N-1} x(n) [\cos(0) - j\sin(0)].$$

Because $\cos(0) = 1$, and $\sin(0) = 0$,

(3-13')

$$X(0) = \sum_{n=0}^{N-1} x(n).$$

We can see that [Eq. \(3-13'\)](#) is the sum of the $x(n)$ samples. This sum is, of course, proportional to the average of $x(n)$. (Specifically, $X(0)$ is equal to N times $x(n)$'s average value.) This makes sense because the $X(0)$ frequency term is the non-time-varying (DC) component of $x(n)$. If $X(0)$ were nonzero, this would tell us that the $x(n)$ sequence is riding on a DC bias and has some nonzero average value. For our specific example input from [Eq. \(3-10\)](#), the sum, however, is zero. The input sequence has no DC component, so we know that $X(0)$ will be zero. But let's not be lazy—we'll calculate $X(0)$ anyway just to be sure. Evaluating [Eq. \(3-3\)](#) or [Eq. \(3-13'\)](#) for $m = 0$, we see that

$$\begin{array}{rcl}
 X(0) & = & 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
 & + & 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
 & + & 0.6464 \cdot 1.0 & -j(0.6464 \cdot 0.0) \\
 & + & 1.0607 \cdot 1.0 & -j(1.0607 \cdot 0.0) \\
 & + & 0.3535 \cdot 1.0 & -j(0.3535 \cdot 0.0) \\
 & - & 1.0607 \cdot 1.0 & -j(-1.0607 \cdot 0.0) \\
 & - & 1.3535 \cdot 1.0 & -j(-1.3535 \cdot 0.0) \\
 & - & 0.3535 \cdot 1.0 & -j(-0.3535 \cdot 0.0) \\
 \\
 & & X(0) = 0.3535 & -j0.0 \\
 & & + 0.3535 & -j0.0 \\
 & & + 0.6464 & -j0.0 \\
 & & + 1.0607 & -j0.0 \\
 & & + 0.3535 & -j0.0 \\
 & & - 1.0607 & -j0.0 \\
 & & - 1.3535 & -j0.0 \\
 & & - 0.3535 & -j0.0 \\
 \\
 & & = 0.0 - j0.0 = 0 \angle 0^\circ.
 \end{array}$$

So our $x(n)$ had no DC component, and, thus, its average value is zero. Notice that [Figure 3-4](#) indicates that $x_{\text{in}}(t)$, from [Eq. \(3-10\)](#), has signal components at 1 kHz ($m = 1$) and 2 kHz ($m = 2$). Moreover, the 1 kHz tone has a magnitude twice that of the 2 kHz tone. The DFT results depicted in [Figure 3-4](#) tell us exactly the spectral content of the signal defined by [Eqs. \(3-10\)](#) and [\(3-11\)](#).

While looking at [Figure 3-4\(b\)](#), we might notice that the phase of $X(1)$ is -90 degrees and ask, “This -90 degrees phase is relative to what?” The answer is: The DFT phase at the frequency mf_s/N is relative to a cosine wave at that same frequency of mf_s/N Hz where $m = 1, 2, 3, \dots, N-1$. For example, the phase of $X(1)$ is -90 degrees, so the input sinusoid whose frequency is $1 \cdot f_s/N = 1000$ Hz was a cosine wave having an initial phase shift of -90 degrees. From the trigonometric identity $\cos(\alpha - 90^\circ) = \sin(\alpha)$, we see that the 1000 Hz input tone was a sinewave having an initial phase of zero. This agrees with our [Eq. \(3-11\)](#). The phase of $X(2)$ is 45 degrees so the 2000 Hz input tone was a cosine wave having an initial phase of 45 degrees, which is equivalent to a sinewave having an initial phase of 135 degrees ($3\pi/4$ radians from [Eq. \(3-11\)](#)).

When the DFT input signals are real-valued, the DFT phase at 0 Hz ($m = 0$, DC) is always zero because $X(0)$ is always real-only as shown by [Eq. \(3-13'\)](#).

The perceptive reader should be asking two questions at this point. First, what do those nonzero magnitude values at $m = 6$ and $m = 7$ in [Figure 3-4\(a\)](#) mean? Also, why do the magnitudes seem four times larger than we would expect? We'll answer those good questions shortly. The above 8-point DFT example, although admittedly simple, illustrates two very important characteristics of the DFT that we should never forget. First, any individual $X(m)$ output value is nothing more than the sum of the term-by-term products, a correlation, of an input signal sample sequence with a cosine and a sinewave whose frequencies are m complete cycles in the total sample interval of N samples. This is

true no matter what the f_s sample rate is and no matter how large N is in an N -point DFT. The second important characteristic of the DFT of real input samples is the symmetry of the DFT output terms.

3.2 DFT Symmetry

Looking at [Figure 3-4\(a\)](#) again, we see that there is an obvious symmetry in the DFT results. Although the standard DFT is designed to accept complex input sequences, most physical DFT inputs (such as digitized values of some continuous signal) are referred to as *real*; that is, real inputs have nonzero real sample values, and the imaginary sample values are assumed to be zero. When the input sequence $x(n)$ is real, as it will be for all of our examples, the complex DFT outputs for $m = 1$ to $m = (N/2) - 1$ are redundant with frequency output values for $m > (N/2)$. The m th DFT output will have the same magnitude as the $(N-m)$ th DFT output. The phase angle of the DFT's m th output is the negative of the phase angle of the $(N-m)$ th DFT output. So the m th and $(N-m)$ th outputs are related by the following

(3-14)

$$\begin{aligned} X(m) &= |X(m)| \text{ at } X_\phi(m) \text{ degrees} \\ &= |X(N-m)| \text{ at } -X_\phi(N-m) \text{ degrees} \end{aligned}$$

for $1 \leq m \leq (N/2)-1$. We can state that when the DFT input sequence is real, $X(m)$ is the complex conjugate of $X(N-m)$, or

(3-14')

$$X(m) = X^*(N-m),^\dagger$$

[†] Using our notation, the complex conjugate of $x = a + jb$ is defined as $x^* = a - jb$; that is, we merely change the sign of the imaginary part of x . In an equivalent form, if $x = e^{j\theta}$, then $x^* = e^{-j\theta}$.

where the superscript “*” symbol denotes conjugation, and $m = 1, 2, 3, \dots, N-1$.

In our example above, notice in [Figures 3-4\(b\)](#) and [3-4\(d\)](#) that $X(5)$, $X(6)$, and $X(7)$ are the complex conjugates of $X(3)$, $X(2)$, and $X(1)$, respectively. Like the DFT's magnitude symmetry, the real part of $X(m)$ has what is called *even symmetry*, as shown in [Figure 3-4\(c\)](#), while the DFT's imaginary part has *odd symmetry*, as shown in [Figure 3-4\(d\)](#). This relationship is what is meant when the DFT is called conjugate symmetric in the literature. It means that if we perform an N -point DFT on a real input sequence, we'll get N separate complex DFT output terms, but only the first $N/2+1$ terms are independent. So to obtain the DFT of $x(n)$, we need only compute the first $N/2+1$ values of $X(m)$ where $0 \leq m \leq (N/2)$; the $X(N/2+1)$ to $X(N-1)$ DFT output terms provide no additional information about the spectrum of the real sequence $x(n)$.

The above N -point DFT symmetry discussion applies to DFTs, whose inputs are real-valued, where N is an even number. If N happens to be an odd number, then only the first $(N+1)/2$ samples of the DFT are independent. For example, with a 9-point DFT only the first five DFT samples are independent.

Although [Eqs. \(3-2\)](#) and [\(3-3\)](#) are equivalent, expressing the DFT in the exponential form of [Eq. \(3-2\)](#) has a terrific advantage over the form of [Eq. \(3-3\)](#). Not only does [Eq. \(3-2\)](#) save pen and paper, but [Eq. \(3-2\)](#)'s exponentials are much easier to manipulate when we're trying to analyze DFT relationships. Using [Eq. \(3-2\)](#), products of terms become the addition of exponents and, with due respect to Euler, we don't have all those trigonometric relationships to memorize. Let's demonstrate this by proving [Eq. \(3-14\)](#) to show the symmetry of the DFT of real input sequences. Substituting $N-m$

for m in [Eq. \(3-2\)](#), we get the expression for the $(N-m)$ th component of the DFT:

$$(3-15)$$

$$\begin{aligned} X(N-m) &= \sum_{n=0}^{N-1} x(n) e^{-j2\pi n(N-m)/N} = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nN/N} e^{-j2\pi n(-m)/N} \\ &= \sum_{n=0}^{N-1} x(n) e^{-j2\pi n} e^{j2\pi nm/N}. \end{aligned}$$

Because $e^{-j2\pi n} = \cos(2\pi n) - j\sin(2\pi n) = 1$ for all integer values of n ,

$$(3-15')$$

$$X(N-m) = \sum_{n=0}^{N-1} x(n) e^{j2\pi nm/N}.$$

We see that $X(N-m)$ in [Eq. \(3-15'\)](#) is merely $X(m)$ in [Eq. \(3-2\)](#) with the sign reversed on $X(m)$'s exponent—and that's the definition of the complex conjugate. This is illustrated by the DFT output phase-angle plot in [Figure 3-4\(b\)](#) for our DFT Example 1. Try deriving [Eq. \(3-15'\)](#) using the cosines and sines of [Eq. \(3-3\)](#), and you'll see why the exponential form of the DFT is so convenient for analytical purposes.

There's an additional symmetry property of the DFT that deserves mention at this point. In practice, we're occasionally required to determine the DFT of real input functions where the input index n is defined over both positive and negative values. If that real input function is even, then $X(m)$ is always real and even; that is, if the real $x(n) = x(-n)$, then, $X_{\text{real}}(m)$ is in general nonzero and $X_{\text{imag}}(m)$ is zero. Conversely, if the real input function is odd, $x(n) = -x(-n)$, then $X_{\text{real}}(m)$ is always zero and $X_{\text{imag}}(m)$ is, in general, nonzero. This characteristic of input function symmetry is a property that the DFT shares with the continuous Fourier transform, and (don't worry) we'll cover specific examples of it later in [Section 3.13](#) and in [Chapter 5](#).

3.3 DFT Linearity

The DFT has a very important property known as *linearity*. This property states that the DFT of the sum of two signals is equal to the sum of the transforms of each signal; that is, if an input sequence $x_1(n)$ has a DFT $X_1(m)$ and another input sequence $x_2(n)$ has a DFT $X_2(m)$, then the DFT of the sum of these sequences $x_{\text{sum}}(n) = x_1(n) + x_2(n)$ is

$$(3-16)$$

$$X_{\text{sum}}(m) = X_1(m) + X_2(m).$$

This is certainly easy enough to prove. If we plug $x_{\text{sum}}(n)$ into [Eq. \(3-2\)](#) to get $X_{\text{sum}}(m)$, then

$$\begin{aligned} X_{\text{sum}}(m) &= \sum_{n=0}^{N-1} [x_1(n) + x_2(n)] e^{-j2\pi nm/N} \\ &= \sum_{n=0}^{N-1} x_1(n) e^{-j2\pi nm/N} + \sum_{n=0}^{N-1} x_2(n) e^{-j2\pi nm/N} = X_1(m) + X_2(m). \end{aligned}$$

Without this property of linearity, the DFT would be useless as an analytical tool because we could

transform only those input signals that contain a single sinewave. The real-world signals that we want to analyze are much more complicated than a single sinewave.

3.4 DFT Magnitudes

The DFT Example 1 results of $|X(1)| = 4$ and $|X(2)| = 2$ may puzzle the reader because our input $x(n)$ signal, from [Eq. \(3-11\)](#), had peak amplitudes of 1.0 and 0.5, respectively. There's an important point to keep in mind regarding DFTs defined by [Eq. \(3-2\)](#). When a real input signal contains a sinewave component, whose frequency is less than half the f_s sample rate, of peak amplitude A_0 with an integral number of cycles over N input samples, the output magnitude of the DFT for that particular sinewave is M_r where

$$(3-17)$$

$$M_r = A_0 N / 2.$$

If the DFT input is a complex sinusoid of magnitude A_0 (i.e., $A_0 e^{j2\pi f n t_s}$) with an integer number of cycles over N samples, the M_c output magnitude of the DFT for that particular sinewave is

$$(3-17')$$

$$M_c = A_0 N.$$

As stated in relation to [Eq. \(3-13'\)](#), if the DFT input was riding on a DC bias value equal to D_0 , the magnitude of the DFT's $X(0)$ output will be $D_0 N$.

Looking at the real input case for the 1000 Hz component of [Eq. \(3-11\)](#), $A_0 = 1$ and $N = 8$, so that $M_{\text{real}} = 1 \cdot 8/2 = 4$, as our example shows. [Equation \(3-17\)](#) may not be so important when we're using software or floating-point hardware to perform DFTs, but if we're implementing the DFT with fixed-point hardware, we have to be aware that the output can be as large as $N/2$ times the peak value of the input. This means that, for real inputs, hardware memory registers must be able to hold values as large as $N/2$ times the maximum amplitude of the input sample values. We discuss DFT output magnitudes in further detail later in this chapter. The DFT magnitude expressions in [Eqs. \(3-17\)](#) and [\(3-17'\)](#) are why we occasionally see the DFT defined in the literature as

$$(3-18)$$

$$X'(m) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm / N}.$$

The $1/N$ scale factor in [Eq. \(3-18\)](#) makes the amplitudes of $X'(m)$ equal to half the time-domain input sinusoid's peak value at the expense of the additional division by N computation. Thus, hardware or software implementations of the DFT typically use [Eq. \(3-2\)](#) as opposed to [Eq. \(3-18\)](#). Of course, there are always exceptions. There are commercial software packages using

$$(3-18')$$

$$X''(m) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm / N}$$

and

$$x(n) = \frac{1}{\sqrt{N}} \sum_{m=0}^{N-1} X''(m) e^{j2\pi nm / N}$$

for the forward and inverse DFTs. (In [Section 3.7](#), we discuss the meaning and significance of the inverse DFT.) The $1/\sqrt{N}$ scale factors in [Eqs. \(3-18'\)](#) seem a little strange, but they're used so that there's no scale change when transforming in either direction. When analyzing signal spectra in practice, we're normally more interested in the relative magnitudes rather than absolute magnitudes of the individual DFT outputs, so scaling factors aren't usually that important to us.

3.5 DFT Frequency Axis

The frequency axis m of the DFT result in [Figure 3-4](#) deserves our attention once again. Suppose we hadn't previously seen our DFT Example 1, were given the eight input sample values, from [Eq. \(3-11\)](#), and were asked to perform an 8-point DFT on them. We'd grind through [Eq. \(3-2\)](#) and obtain the $X(m)$ values shown in [Figure 3-4](#). Next we ask, "What's the frequency of the highest magnitude component in $X(m)$ in Hz?" The answer is not "1" kHz. The answer depends on the original sample rate f_s . Without prior knowledge, we have no idea over what time interval the samples were taken, so we don't know the absolute scale of the $X(m)$ frequency axis. The correct answer to the question is to take f_s and plug it into [Eq. \(3-5\)](#) with $m = 1$. Thus, if $f_s = 8000$ samples/second, then the frequency associated with the largest DFT magnitude term is

$$f_{\text{analysis}}(m) = \frac{mf_s}{N} = f_{\text{analysis}}(1) = \frac{1 \cdot 8000}{8} = 1000 \text{ Hz.}$$

If we said the sample rate f_s was 75 samples/second, we'd know, from [Eq. \(3-5\)](#), that the frequency associated with the largest magnitude term is now

$$f_{\text{analysis}}(1) = \frac{1 \cdot 75}{8} = 9.375 \text{ Hz.}$$

OK, enough of this—just remember that the DFT's frequency spacing (resolution) is f_s/N .

To recap what we've learned so far:

- Each DFT output term is the sum of the term-by-term products of an input time-domain sequence with sequences representing a sine and a cosine wave.
- For real inputs, an N -point DFT's output provides only $N/2+1$ independent terms.
- The DFT is a linear operation.
- The magnitude of the DFT results is directly proportional to N .
- The DFT's frequency resolution is f_s/N .

It's also important to realize, from [Eq. \(3-5\)](#), that $X(N/2)$, when $m = N/2$, corresponds to half the sample rate, i.e., the folding (Nyquist) frequency $f_s/2$.

3.6 DFT Shifting Theorem

There's an important property of the DFT known as the *shifting theorem*. It states that a shift in time of a periodic $x(n)$ input sequence manifests itself as a constant phase shift in the angles associated with the DFT results. (We won't derive the shifting theorem equation here because its derivation is included in just about every digital signal processing textbook in print.) If we decide to sample $x(n)$ starting at n equals some integer k , as opposed to $n = 0$, the DFT of those time-shifted sample values is $X_{\text{shifted}}(m)$ where

$$(3-19)$$

$$X_{\text{shifted}}(m) = e^{j2\pi km/N} X(m).$$

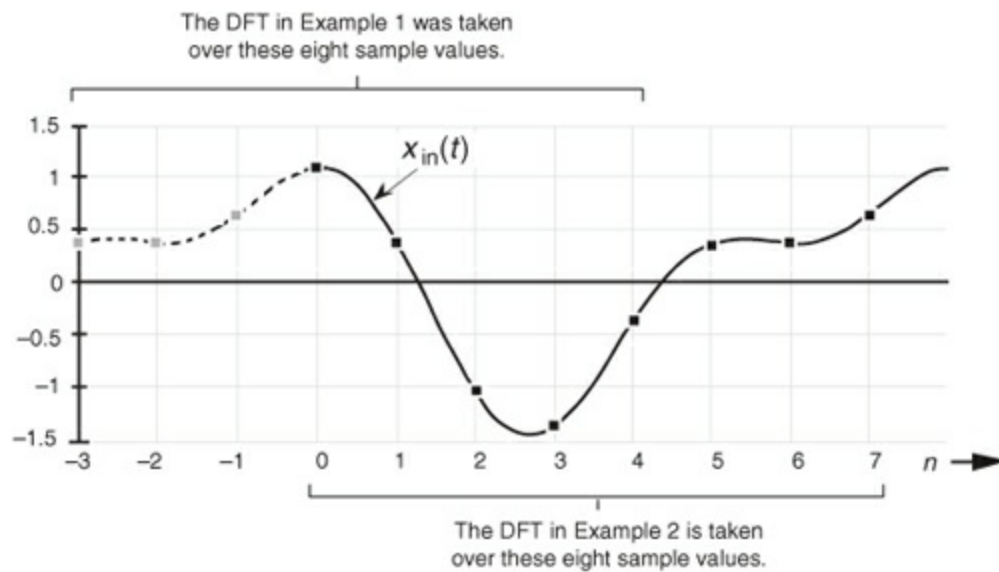
[Equation \(3-19\)](#) tells us that if the point where we start sampling $x(n)$ is shifted to the right by k samples, the DFT output spectrum of $X_{\text{shifted}}(m)$ is $X(m)$ with each of $X(m)$'s complex terms multiplied by the linear phase shift $e^{j2\pi km/N}$, which is merely a phase shift of $2\pi km/N$ radians or $360km/N$ degrees. Conversely, if the point where we start sampling $x(n)$ is shifted to the left by k samples, the spectrum of $X_{\text{shifted}}(m)$ is $X(m)$ multiplied by $e^{-j2\pi km/N}$. Let's illustrate [Eq. \(3-19\)](#) with an example.

3.6.1 DFT Example 2

Suppose we sampled our DFT Example 1 input sequence later in time by $k = 3$ samples. [Figure 3-5](#) shows the original input time function,

$$x_{\text{in}}(t) = \sin(2\pi 1000t) + 0.5\sin(2\pi 2000t + 3\pi/4).$$

Figure 3-5 Comparison of sampling times between DFT Example 1 and DFT Example 2.



We can see that [Figure 3-5](#) is a continuation of [Figure 3-2\(a\)](#). Our new $x(n)$ sequence becomes the values represented by the solid black dots in [Figure 3-5](#) whose values are

$$(3-20)$$

$$\begin{array}{ll} x(0) = 1.0607, & x(1) = 0.3535, \\ x(2) = -1.0607, & x(3) = -1.3535, \\ x(4) = -0.3535, & x(5) = 0.3535, \\ x(6) = 0.3535, & x(7) = 0.6464. \end{array}$$

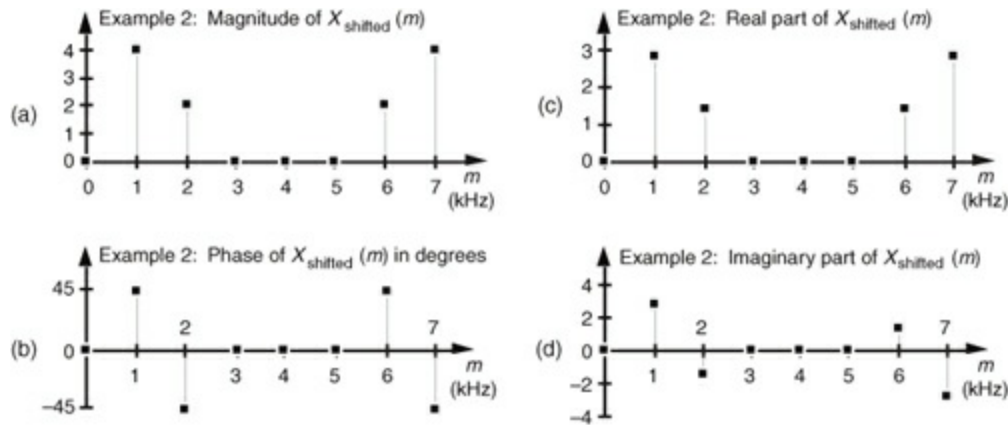
Performing the DFT on [Eq. \(3-20\)](#), $X_{\text{shifted}}(m)$ is

$$(3-21)$$

$X_{\text{shifted}}(m)$	m	$X_{\text{shifted}}(m)$'s magnitude	$X_{\text{shifted}}(m)$'s phase	$X_{\text{shifted}}(m)$'s real part	$X_{\text{shifted}}(m)$'s imaginary part
	0	0	0	0	0
	1	4	+45	2.8284	2.8284
	2	2	-45	1.4142	-1.414
	3	0	0	0	0
	4	0	0	0	0
	5	0	0	0	0
	6	2	+45	1.4142	1.4142
	7	4	-45	2.8284	-2.828

The values in [Eq. \(3-21\)](#) are illustrated as the dots in [Figure 3-6](#). Notice that [Figure 3-6\(a\)](#) is identical to [Figure 3-4\(a\)](#). [Equation \(3-19\)](#) told us that the magnitude of $X_{\text{shifted}}(m)$ should be unchanged from that of $X(m)$. That's a comforting thought, isn't it? We wouldn't expect the DFT magnitude of our original periodic $x_{\text{in}}(t)$ to change just because we sampled it over a different time interval. The phase of the DFT result does, however, change depending on the instant at which we started to sample $x_{\text{in}}(t)$.

Figure 3-6 DFT results from Example 2: (a) magnitude of $X_{\text{shifted}}(m)$; (b) phase of $X_{\text{shifted}}(m)$; (c) real part of $X_{\text{shifted}}(m)$; (d) imaginary part of $X_{\text{shifted}}(m)$.



By looking at the $m = 1$ component of $X_{\text{shifted}}(m)$, for example, we can double-check to see that phase values in [Figure 3-6\(b\)](#) are correct. Using [Eq. \(3-19\)](#) and remembering that $X(1)$ from DFT Example 1 had a magnitude of 4 at a phase angle of -90° (or $-\pi/2$ radians), $k = 3$ and $N = 8$ so that

$$(3-22)$$

$$X_{\text{shifted}}(1) = e^{j2\pi km/N} \cdot X(1) = e^{j2\pi 3 \cdot 1/8} \cdot 4e^{-j\pi/2} = 4e^{j(6\pi/8 - 4\pi/8)} = 4e^{j\pi/4}.$$

So $X_{\text{shifted}}(1)$ has a magnitude of 4 and a phase angle of $\pi/4$ or $+45^\circ$, which is what we set out to prove using [Eq. \(3-19\)](#).

3.7 Inverse DFT

Although the DFT is the major topic of this chapter, it's appropriate, now, to introduce the inverse discrete Fourier transform (IDFT). Typically we think of the DFT as transforming time-domain data into a frequency-domain representation. Well, we can reverse this process and obtain the original time-domain signal by performing the IDFT on the $X(m)$ frequency-domain values. The standard expressions for the IDFT are

(3-23)

$$x(n) = \frac{1}{N} \sum_{m=0}^{N-1} X(m) e^{j2\pi mn / N}$$

and equally,

(3-23')

$$x(n) = \frac{1}{N} \sum_{m=0}^{N-1} X(m) [\cos(2\pi mn / N) + j \sin(2\pi mn / N)].$$

Remember the statement we made in [Section 3.1](#) that a discrete time-domain signal can be considered the sum of various sinusoidal analytical frequencies and that the $X(m)$ outputs of the DFT are a set of N complex values indicating the magnitude and phase of each analysis frequency comprising that sum. [Equations \(3-23\)](#) and [\(3-23'\)](#) are the mathematical expressions of that statement. It's very important for the reader to understand this concept. If we perform the IDFT by plugging our results from DFT Example 1 into [Eq. \(3-23\)](#), we'll go from the frequency domain back to the time domain and get our original real [Eq. \(3-11'\)](#) $x(n)$ sample values of

$x(0) = 0.3535 + j0.0$	$x(1) = 0.3535 + j0.0$
$x(2) = 0.6464 + j0.0$	$x(3) = 1.0607 + j0.0$
$x(4) = 0.3535 + j0.0$	$x(5) = -1.0607 + j0.0$
$x(6) = -1.3535 + j0.0$	$x(7) = -0.3535 + j0.0$

Notice that [Eq. \(3-23\)](#)'s IDFT expression differs from the DFT's [Eq. \(3-2\)](#) only by a $1/N$ scale factor and a change in the sign of the exponent. Other than the magnitude of the results, every characteristic that we've covered thus far regarding the DFT also applies to the IDFT.

3.8 DFT Leakage

Hold on to your seat now. Here's where the DFT starts to get really interesting. The two previous DFT examples gave us correct results because the input $x(n)$ sequences were very carefully chosen sinusoids. As it turns out, the DFT of sampled real-world signals provides frequency-domain results that can be misleading. A characteristic known as *leakage* causes our DFT results to be only an approximation of the true spectra of the original input signals prior to digital sampling. Although there are ways to minimize leakage, we can't eliminate it entirely. Thus, we need to understand exactly what effect it has on our DFT results.

Let's start from the beginning. DFTs are constrained to operate on a finite set of N input values, sampled at a sample rate of f_s , to produce an N -point transform whose discrete outputs are associated with the individual analytical frequencies $f_{\text{analysis}}(m)$, with

(3-24)

$$f_{\text{analysis}}(m) = \frac{mf_s}{N}, \text{ where } m = 0, 1, 2, \dots, N-1.$$

[Equation \(3-24\)](#), illustrated in DFT Example 1, may not seem like a problem, but it is. The DFT produces correct results only when the input data sequence contains energy precisely at the analysis frequencies given in [Eq. \(3-24\)](#), at integral multiples of our fundamental frequency f_s/N . If the input has a signal component at some intermediate frequency between our analytical frequencies of mf_s/N , say $1.5f_s/N$, this input signal will show up to some degree in *all* of the N output analysis frequencies of our DFT! (We typically say that input signal energy shows up in all of the DFT's output *bins*, and

we'll see, in a moment, why the phrase "output bins" is appropriate. Engineers often refer to DFT samples as "bins." So when you see, or hear, the word *bin* it merely means a frequency-domain sample.) Let's understand the significance of this problem with another DFT example.

Assume we're taking a 64-point DFT of the sequence indicated by the dots in [Figure 3-7\(a\)](#). The sequence is a sinewave with exactly three cycles contained in our $N = 64$ samples. [Figure 3-7\(b\)](#) shows the first half of the DFT of the input sequence and indicates that the sequence has an average value of zero ($X(0) = 0$) and no signal components at any frequency other than the $m = 3$ frequency. No surprises so far. [Figure 3-7\(a\)](#) also shows, for example, the $m = 4$ sinewave analysis frequency, superimposed over the input sequence, to remind us that the analytical frequencies always have an integral number of cycles over our total sample interval of 64 points. The sum of the products of the input sequence and the $m = 4$ analysis frequency is zero. (Or we can say, the correlation of the input sequence and the $m = 4$ analysis frequency is zero.) The sum of the products of this particular three-cycle input sequence and any analysis frequency other than $m = 3$ is zero. Continuing with our leakage example, the dots in [Figure 3-8\(a\)](#) show an input sequence having 3.4 cycles over our $N = 64$ samples. Because the input sequence does not have an integral number of cycles over our 64-sample interval, input energy has leaked into all the other DFT output bins as shown in [Figure 3-8\(b\)](#). The $m = 4$ bin, for example, is not zero because the sum of the products of the input sequence and the $m = 4$ analysis frequency is no longer zero. This is leakage—it causes any input signal whose frequency is not exactly at a DFT bin center to leak into all of the other DFT output bins. Moreover, leakage is an unavoidable fact of life when we perform the DFT on real-world finite-length time sequences.

Figure 3-7 Sixty-four-point DFT: (a) input sequence of three cycles and the $m = 4$ analysis frequency sinusoid; (b) DFT output magnitude.

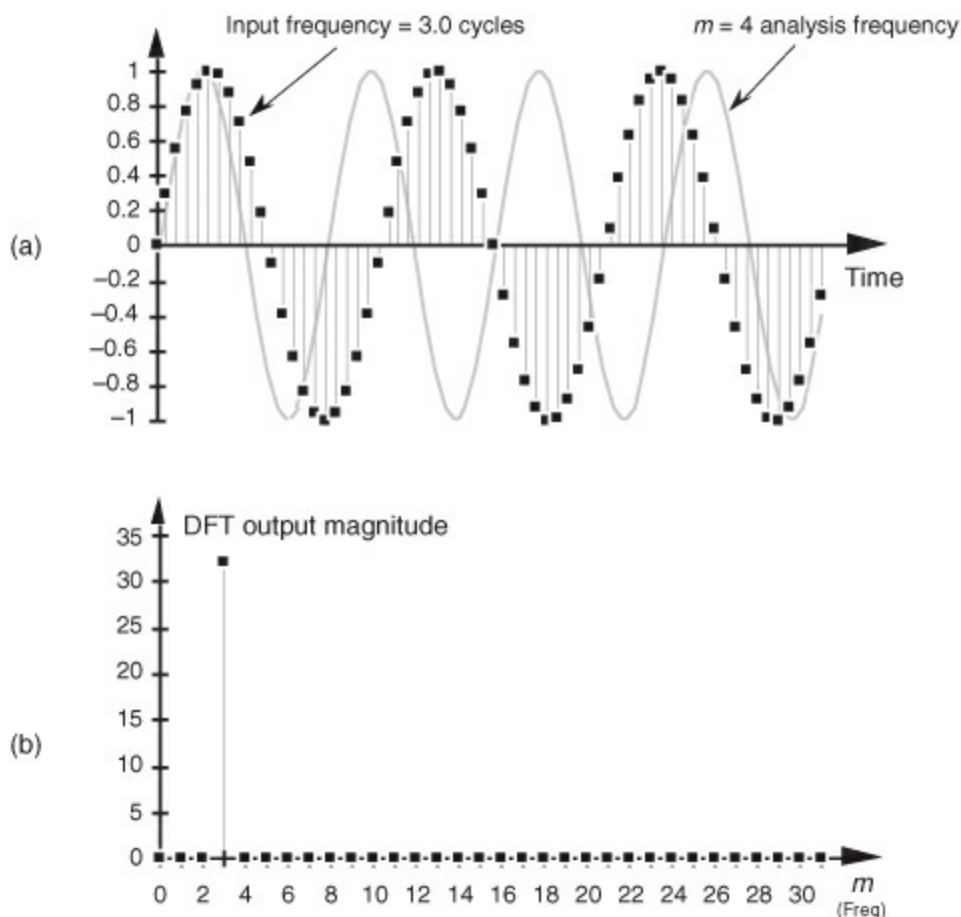
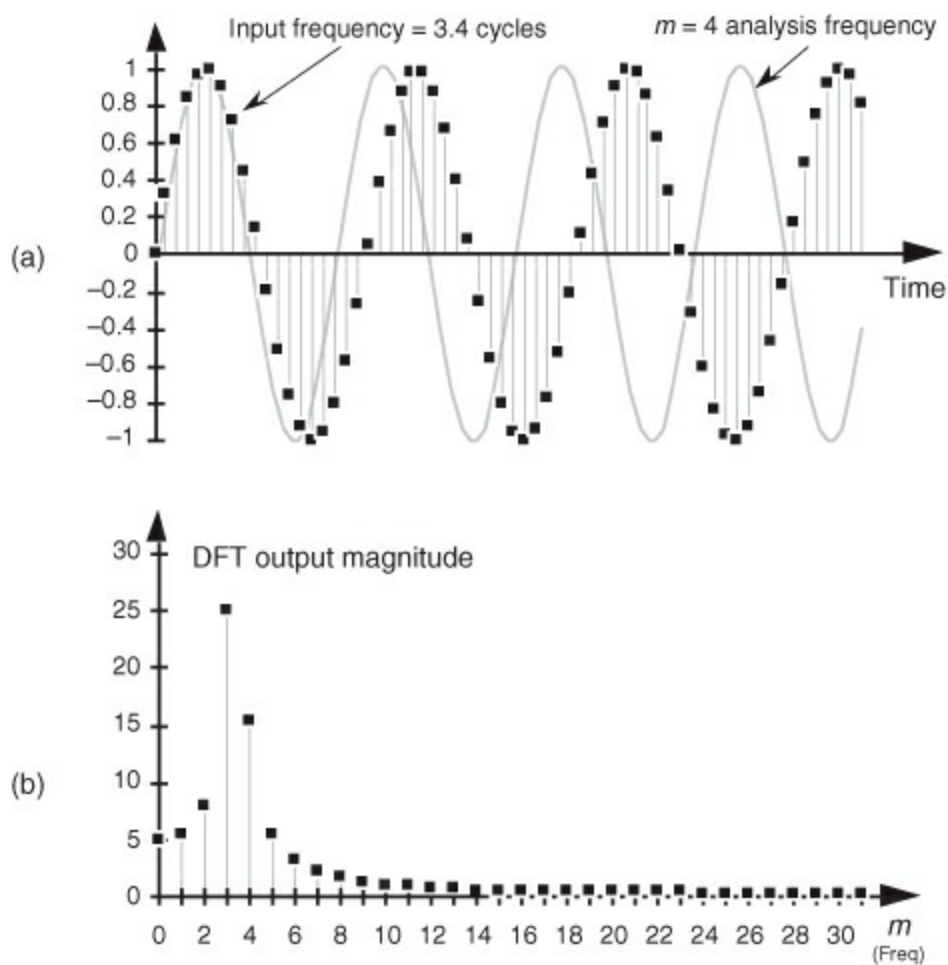


Figure 3-8 Sixty-four-point DFT: (a) 3.4 cycles input sequence and the $m = 4$ analysis frequency sinusoid; (b) DFT output magnitude.



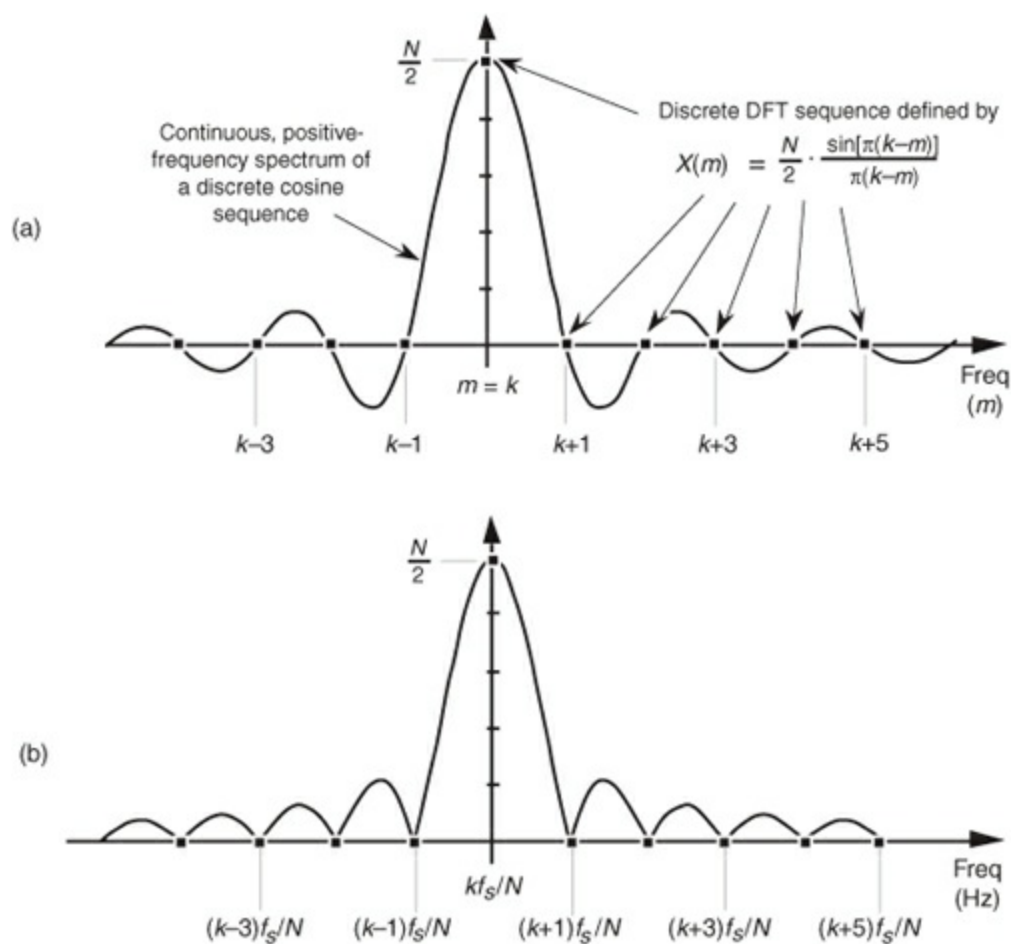
Now, as the English philosopher Douglas Adams would say, “Don’t panic.” Let’s take a quick look at the cause of leakage to learn how to predict and minimize its unpleasant effects. To understand the effects of leakage, we need to know the amplitude response of a DFT when the DFT’s input is an arbitrary, real sinusoid. Although [Sections 3.13](#) discusses this issue in detail, for our purposes, here, we’ll just say that for a real cosine input having k cycles (k need not be an integer) in the N -point input time sequence, the amplitude response of an N -point DFT bin in terms of the bin index m is approximated by the sinc function

(3-25)

$$X(m) = \frac{A_o N}{2} \cdot \frac{\sin[\pi(k - m)]}{\pi(k - m)}$$

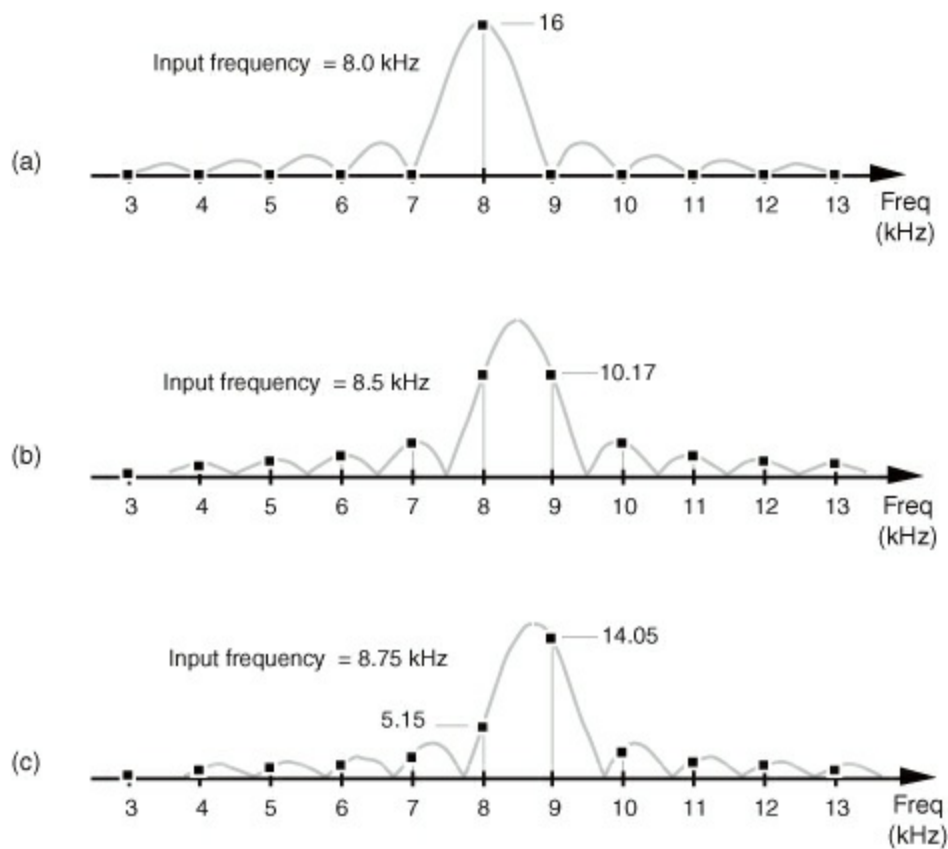
where A_o is the peak value of the DFT’s input sinusoid. For our examples here, A_o is unity. We’ll use [Eq. \(3-25\)](#), illustrated in [Figure 3-9\(a\)](#), to help us determine how much leakage occurs in DFTs. We can think of the curve in [Figure 3-9\(a\)](#), comprising a main lobe and periodic peaks and valleys known as *sidelobes*, as the continuous positive spectrum of an N -point, real cosine time sequence having k cycles in the N -point input time interval. The DFT’s outputs are discrete samples that reside on the curves in [Figure 3-9](#); that is, our DFT output will be a sampled version of the continuous spectrum. (We show the DFT’s magnitude response to a real input in terms of frequency (Hz) in [Figure 3-9\(b\)](#).) When the DFT’s input sequence has exactly an integral k number of cycles (centered exactly in the $m = k$ bin), no leakage occurs, as in [Figure 3-9](#), because when the angle in the numerator of [Eq. \(3-25\)](#) is a nonzero integral multiple of π , the sine of that angle is zero.

Figure 3-9 DFT positive-frequency response due to an N -point input sequence containing k cycles of a real cosine: (a) amplitude response as a function of bin index m ; (b) magnitude response as a function of frequency in Hz.



By way of example, we can illustrate again what happens when the input frequency k is not located at a bin center. Assume that a real 8 kHz sinusoid, having unity amplitude, has been sampled at a rate of $f_s = 32000$ samples/second. If we take a 32-point DFT of the samples, the DFT's frequency resolution, or bin spacing, is $f_s/N = 32000/32$ Hz = 1.0 kHz. We can predict the DFT's magnitude response by centering the input sinusoid's spectral curve at the positive frequency of 8 kHz, as shown in [Figure 3-10\(a\)](#). The dots show the DFT's output bin magnitudes.

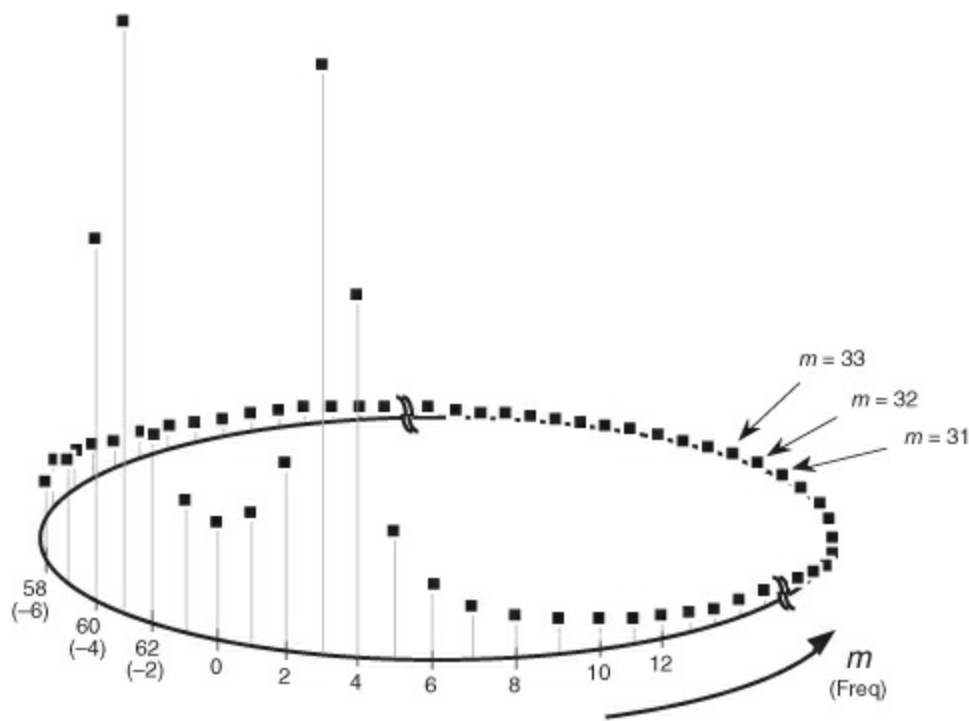
Figure 3-10 DFT bin positive-frequency responses: (a) DFT input frequency = 8.0 kHz; (b) DFT input frequency = 8.5 kHz; (c) DFT input frequency = 8.75 kHz.



Again, here's the important point to remember: the DFT output is a sampled version of the continuous spectral curve in [Figure 3-10\(a\)](#). Those sampled values in the frequency domain, located at mf_s/N , are the dots in [Figure 3-10\(a\)](#). Because the input signal frequency is exactly at a DFT bin center, the DFT results have only one nonzero value. Stated in another way, when an input sinusoid has an integral number of cycles over N time-domain input sample values, the DFT outputs reside on the continuous spectrum at its peak and exactly at the curve's zero crossing points. From [Eq. \(3-25\)](#) we know the peak output magnitude is $32/2 = 16$. (If the real input sinusoid had an amplitude of 2, the peak of the response curve would be $2 \cdot 32/2$, or 32.) [Figure 3-10\(b\)](#) illustrates DFT leakage where the input frequency is 8.5 kHz, and we see that the frequency-domain sampling results in nonzero magnitudes for all DFT output bins. An 8.75 kHz input sinusoid would result in the leaky DFT output shown in [Figure 3-10\(c\)](#). If we're sitting at a computer studying leakage by plotting the magnitude of DFT output values, of course, we'll get the dots in [Figure 3-10](#) and won't see the continuous spectral curves.

At this point, the attentive reader should be thinking: "If the continuous spectra that we're sampling are symmetrical, why does the DFT output in [Figure 3-8\(b\)](#) look so asymmetrical?" In [Figure 3-8\(b\)](#), the bins to the right of the third bin are decreasing in amplitude faster than the bins to the left of the third bin. "And another thing, with $k = 3.4$ and $m = 3$, from [Eq. \(3-25\)](#) the $X(3)$ bin's magnitude should be approximately equal to 24.2—but [Figure 3-8\(b\)](#) shows the $X(3)$ bin magnitude to be slightly greater than 25. What's going on here?" We answer this by remembering what [Figure 3-8\(b\)](#) really represents. When examining a DFT output, we're normally interested only in the $m = 0$ to $m = (N/2 - 1)$ bins. Thus, for our 3.4 cycles per sample interval example in [Figure 3-8\(b\)](#), only the first 32 bins are shown. Well, the DFT is periodic in the frequency domain as illustrated in [Figure 3-11](#). (We address this periodicity issue in [Section 3.14](#).) Upon examining the DFT's output for higher and higher frequencies, we end up going in circles, and the spectrum repeats itself forever.

Figure 3-11 Cyclic representation of the DFT's spectral replication when the DFT input is 3.4 cycles per sample interval.



The more conventional way to view a DFT output is to *unwrap* the spectrum in [Figure 3-11](#) to get the spectrum in [Figure 3-12](#). [Figure 3-12](#) shows some of the additional replications in the spectrum for the 3.4 cycles per sample interval example. Concerning our DFT output asymmetry problem, as some of the input 3.4-cycle signal amplitude leaks into the 2nd bin, the 1st bin, and the 0th bin, leakage continues into the -1st bin, the -2nd bin, the -3rd bin, etc. Remember, the 63rd bin is the -1st bin, the 62nd bin is the -2nd bin, and so on. These bin equivalencies allow us to view the DFT output bins as if they extend into the negative-frequency range, as shown in [Figure 3-13\(a\)](#). The result is that the leakage *wraps around* the $m = 0$ frequency bin, as well as around the $m = N$ frequency bin. This is not surprising, because the $m = 0$ frequency *is* the $m = N$ frequency. The leakage wraparound at the $m = 0$ frequency accounts for the asymmetry around the DFT's $m = 3$ bin in [Figure 3-8\(b\)](#).

Figure 3-12 Spectral replication when the DFT input is 3.4 cycles per sample interval.

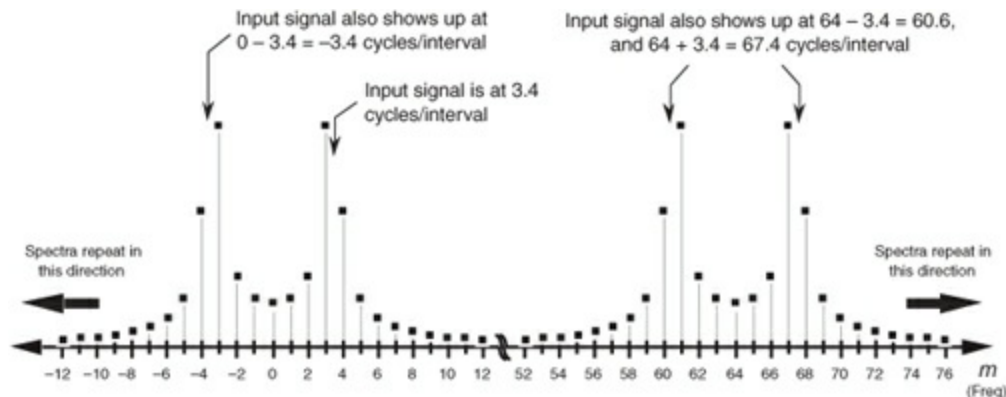
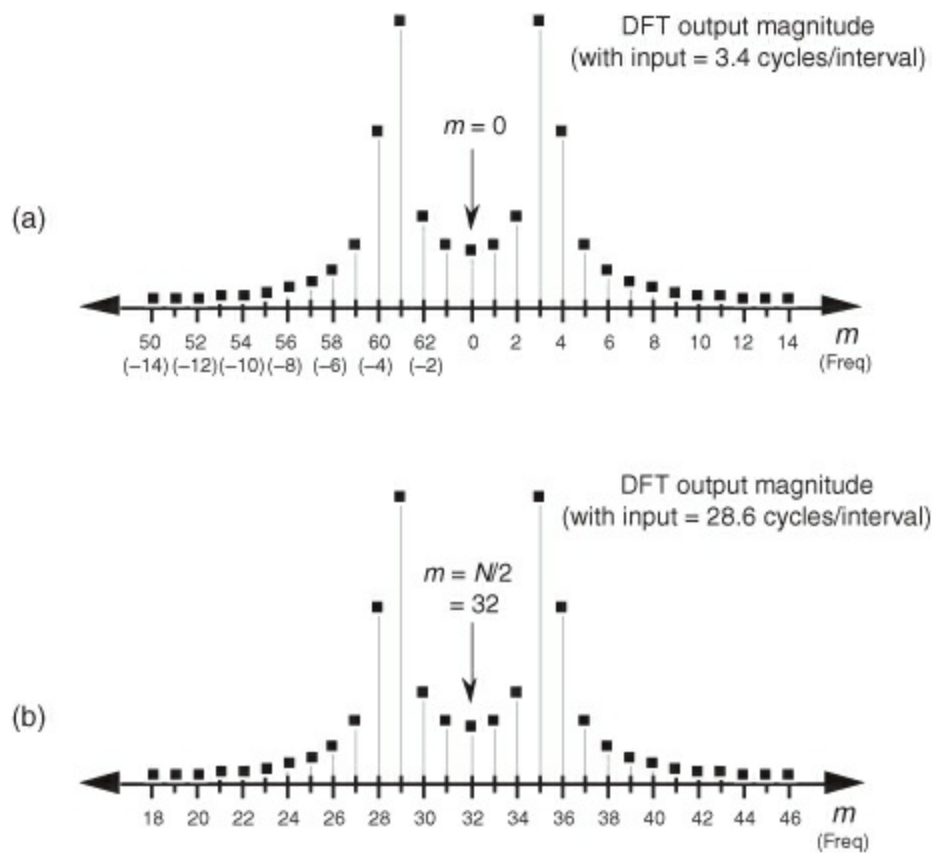
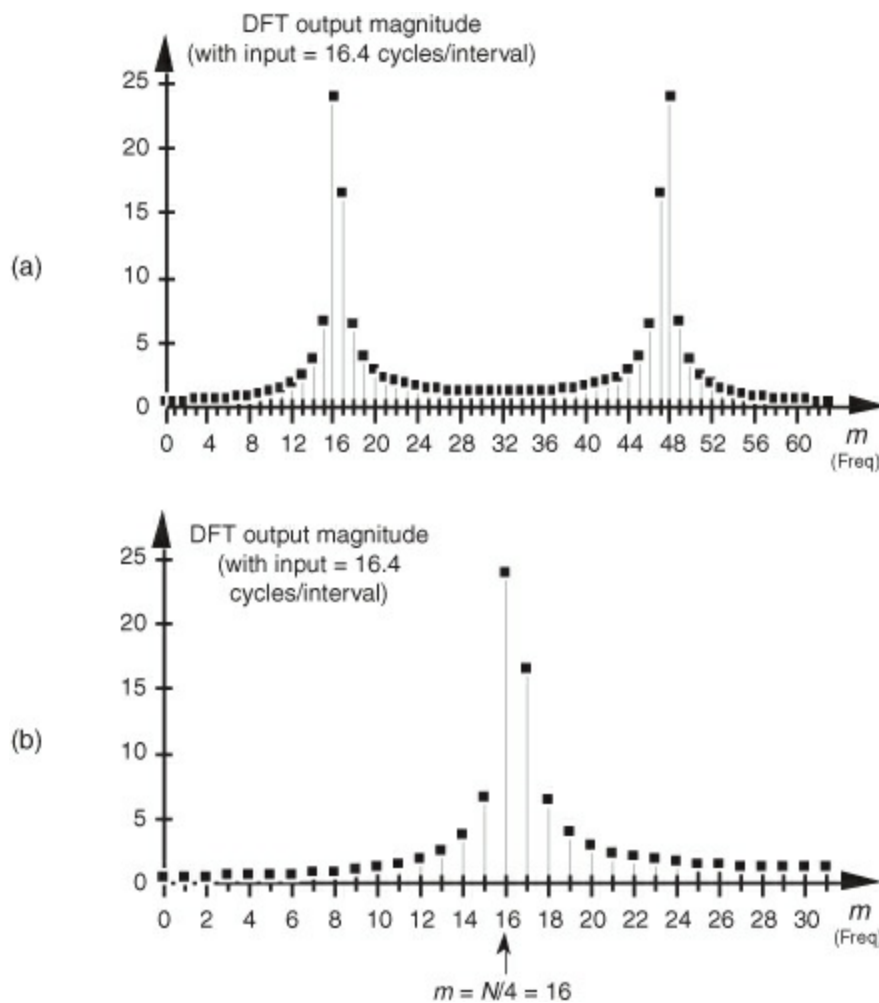


Figure 3-13 DFT output magnitude: (a) when the DFT input is 3.4 cycles per sample interval; (b) when the DFT input is 28.6 cycles per sample interval.



Recall from the DFT symmetry discussion that when a DFT input sequence $x(n)$ is real, the DFT outputs from $m = 0$ to $m = (N/2 - 1)$ are redundant with frequency bin values for $m > (N/2)$, where N is the DFT size. The m th DFT output will have the same magnitude as the $(N - m)$ th DFT output. That is, $|X(m)| = |X(N - m)|$. What this means is that leakage wraparound also occurs around the $m = N/2$ bin. This can be illustrated using an input of 28.6 cycles per sample interval ($32 - 3.4$) whose spectrum is shown in [Figure 3-13\(b\)](#). Notice the similarity between [Figures 3-13\(a\)](#) and [3-13\(b\)](#). So the DFT exhibits leakage wraparound about the $m = 0$ and $m = N/2$ bins. Minimum leakage asymmetry will occur near the $N/4$ th bin as shown in [Figure 3-14\(a\)](#) where the full spectrum of a 16.4 cycles per sample interval input is provided. [Figure 3-14\(b\)](#) shows a close-up view of the first 32 bins of the 16.4 cycles per sample interval spectrum.

Figure 3-14 DFT output magnitude when the DFT input is 16.4 cycles per sample interval: (a) full output spectrum view; (b) close-up view showing minimized leakage asymmetry at frequency $m = N/4$.



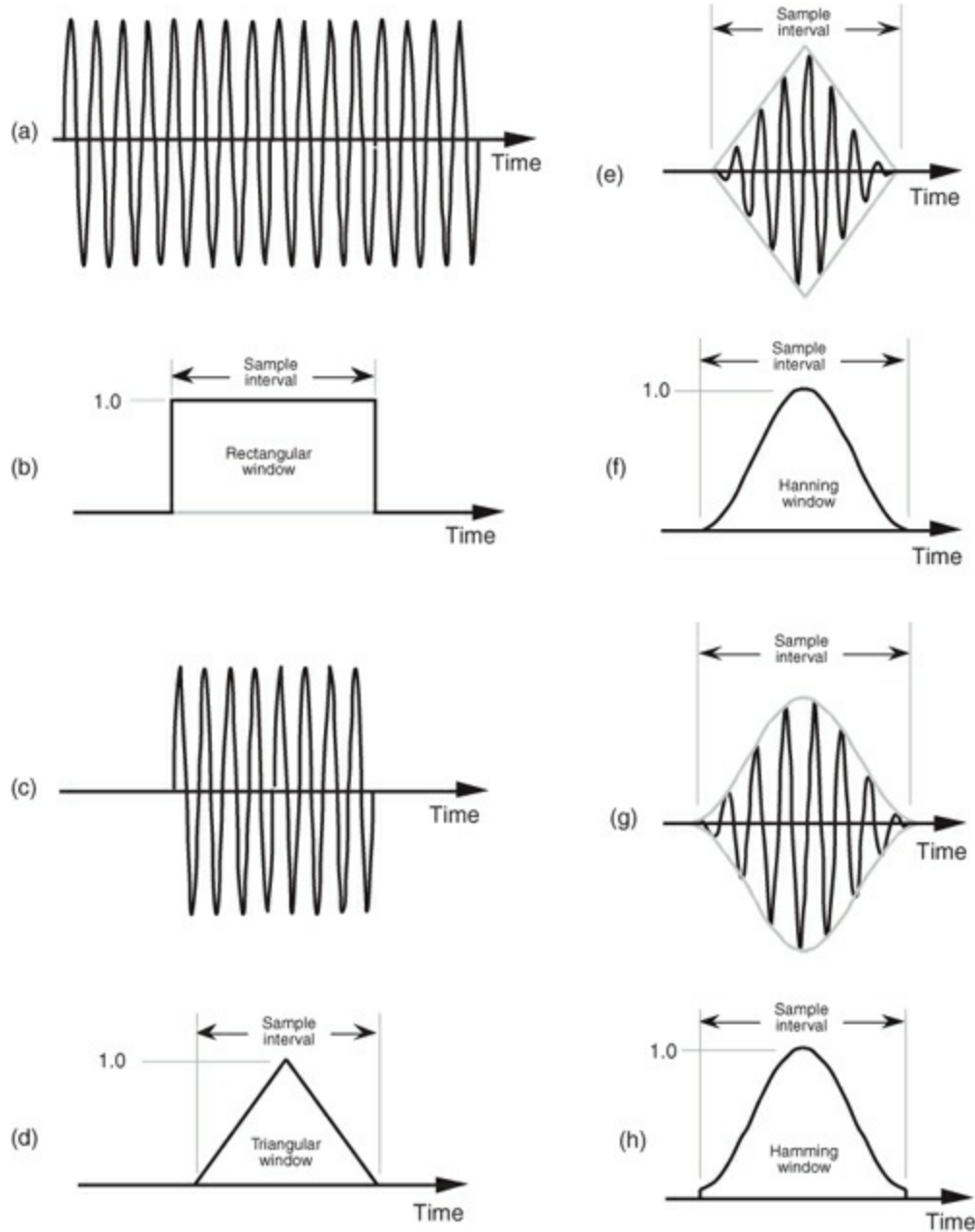
You could read about leakage all day. However, the best way to appreciate its effects is to sit down at a computer and use a software program to take DFTs, in the form of fast Fourier transforms (FFTs), of your personally generated test signals like those in [Figures 3-7](#) and [3-8](#). You can then experiment with different combinations of input frequencies and various DFT sizes. You'll be able to demonstrate that the DFT leakage effect is troublesome because the bins containing low-level signals are corrupted by the sidelobe levels from neighboring bins containing high-amplitude signals.

Although there's no way to eliminate leakage completely, an important technique known as *windowing* is the most common remedy to reduce its unpleasant effects. Let's look at a few DFT window examples.

3.9 Windows

Windowing reduces DFT leakage by minimizing the magnitude of [Eq. \(3-25\)](#)'s sinc function's $\sin(x)/x$ sidelobes shown in [Figure 3-9](#). We do this by forcing the amplitude of the input time sequence at both the beginning and the end of the sample interval to go smoothly toward a single common amplitude value. [Figure 3-15](#) shows how this process works. If we consider the infinite-duration time signal shown in [Figure 3-15\(a\)](#), a DFT can only be performed over a finite-time sample interval like that shown in [Figure 3-15\(c\)](#). We can think of the DFT input signal in [Figure 3-15\(c\)](#) as the product of an input signal existing for all time, [Figure 3-15\(a\)](#), and the rectangular window whose magnitude is 1 over the sample interval shown in [Figure 3-15\(b\)](#). Anytime we take the DFT of a finite-extent input sequence, we are, by default, multiplying that sequence by a window of all ones and effectively multiplying the input values outside that window by zeros. As it turns out, [Eq. \(3-25\)](#)'s sinc function's $\sin(x)/x$ shape, shown in [Figure 3-9](#), is caused by this rectangular window because the continuous Fourier transform of the rectangular window in [Figure 3-15\(b\)](#) is the sinc function.

Figure 3-15 Minimizing sample interval end-point discontinuities: (a) infinite-duration input sinusoid; (b) rectangular window due to finite-time sample interval; (c) product of rectangular window and infinite-duration input sinusoid; (d) triangular window function; (e) product of triangular window and infinite-duration input sinusoid; (f) Hanning window function; (g) product of Hanning window and infinite-duration input sinusoid; (h) Hamming window function.



As we'll soon see, it's the rectangular window's abrupt changes between one and zero that are the cause of the sidelobes in the $\sin(x)/x$ sinc function. To minimize the spectral leakage caused by those sidelobes, we have to reduce the sidelobe amplitudes by using window functions other than the rectangular window. Imagine if we multiplied our DFT input, [Figure 3-15\(c\)](#), by the triangular window function shown in [Figure 3-15\(d\)](#) to obtain the windowed input signal shown in [Figure 3-15\(e\)](#). Notice that the values of our final input signal appear to be the same at the beginning and end of the sample interval in [Figure 3-15\(e\)](#). The reduced discontinuity decreases the level of relatively high-frequency components in our overall DFT output; that is, our DFT bin sidelobe levels are reduced in magnitude using a triangular window. There are other window functions that reduce leakage even more than the triangular window, such as the Hanning window in [Figure 3-15\(f\)](#). The product of the window in [Figure 3-15\(f\)](#) and the input sequence provides the signal shown in [Figure](#)

[3-15\(g\)](#) as the input to the DFT. Another common window function is the Hamming window shown in [Figure 3-15\(h\)](#). It's much like the Hanning window, but it's raised on a *pedestal*.

Before we see exactly how well these windows minimize DFT leakage, let's define them mathematically. Assuming that our original N input signal samples are indexed by n , where $0 \leq n \leq N-1$, we'll call the N time-domain window coefficients $w(n)$; that is, an input sequence $x(n)$ is multiplied by the corresponding window $w(n)$ coefficients before the DFT is performed. So the DFT of the windowed $x(n)$ input sequence, $X_w(m)$, takes the form of

(3-26)

$$X_w(m) = \sum_{n=0}^{N-1} w(n) \cdot x(n) e^{-j2\pi nm/N}.$$

To use window functions, we need mathematical expressions of them in terms of n . The following expressions define our window function coefficients:

(3-27)

Rectangular window:
(also called the uniform,
or boxcar, window)

$$w(n) = 1, \text{ for } n = 0, 1, 2, \dots, N-1.$$

(3-28)

Triangular window:
(very similar to the
Bartlett[3], and
Parzen[4,5] windows)

$$w = \begin{cases} \frac{n}{N/2}, & \text{for } n = 0, 1, 2, \dots, N/2 \\ 2 - \frac{n}{N/2}, & \text{for } n = N/2 + 1, N/2 + 2, \dots, N-1. \end{cases}$$

(3-29)

Hanning window:
(also called the raised
cosine, Hann, or
von Hann window)

$$w(n) = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N}\right),$$

for $n = 0, 1, 2, \dots, N-1$.

(3-30)

Hamming window:

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right),$$

for $n = 0, 1, 2, \dots, N-1$.

If we plot the $w(n)$ values from [Eqs. \(3-27\)](#) through [\(3-30\)](#), we'd get the corresponding window functions like those in [Figures 3-15\(b\)](#), [3-15\(d\)](#), [3-15\(f\)](#), and [3-15\(h\)](#).[‡]

[‡] In the literature, the equations for window functions depend on the range of the sample index n . We define n to be in the range $0 \leq n \leq N-1$. Some authors define n to be in the range $-N/2 \leq n \leq N/2-1$, in which case, for example, the expression for the Hanning window would have a sign change and be $w(n) = 0.5 + 0.5 \cos(2\pi n/N)$.

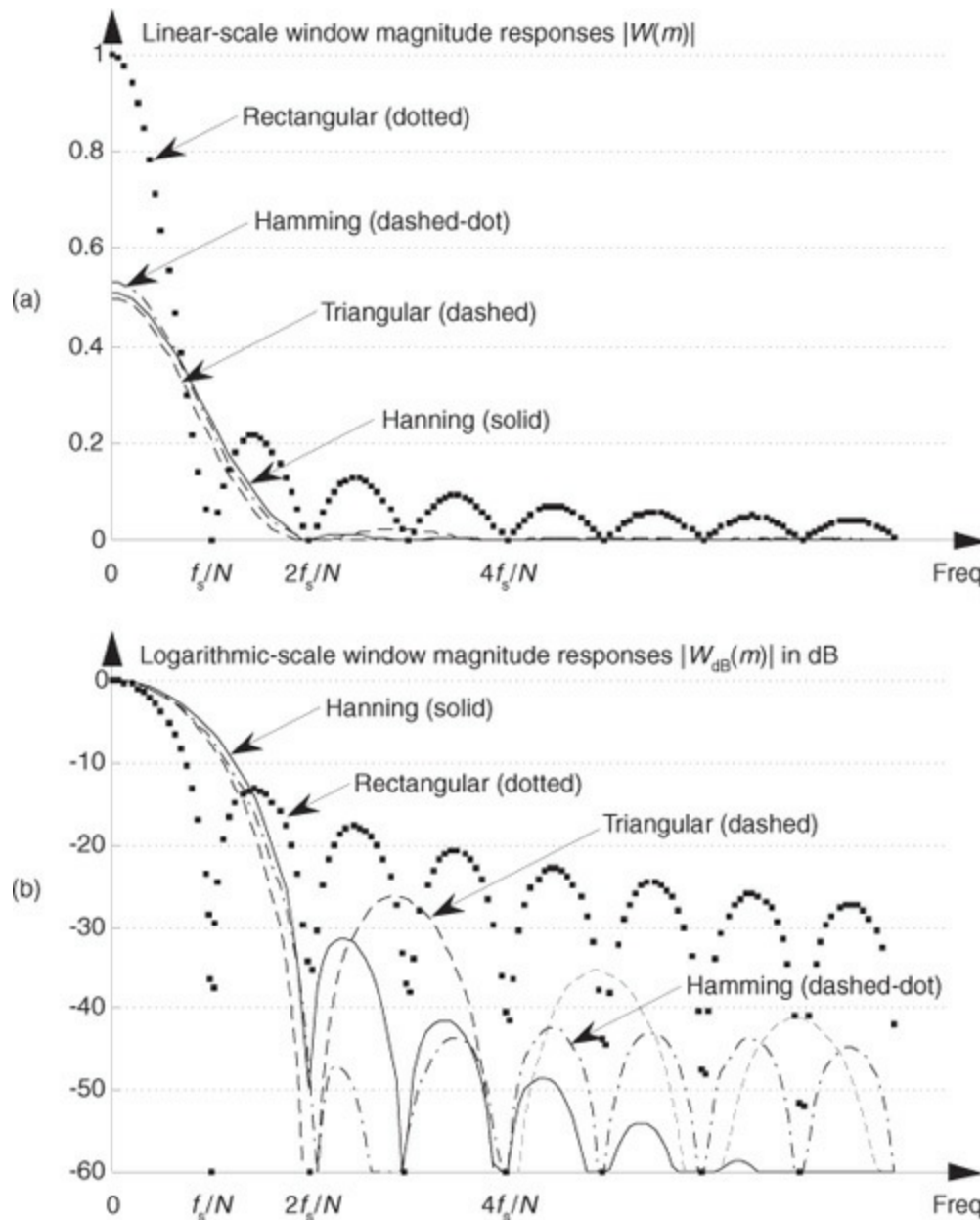
The rectangular window's amplitude response is the yardstick we normally use to evaluate another window function's amplitude response; that is, we typically get an appreciation for a window's response by comparing it to the rectangular window that exhibits the magnitude response shown in [Figure 3-9\(b\)](#). The rectangular window's $\sin(x)/x$ magnitude response, $|W(m)|$, is repeated in [Figure 3-16\(a\)](#). Also included in [Figure 3-16\(a\)](#) are the Hamming, Hanning, and triangular window magnitude responses. (The frequency axis in [Figure 3-16](#) is such that the curves show the response of a single N -point DFT bin when the various window functions are used.) We can see that the last three

windows give reduced sidelobe levels relative to the rectangular window. Because the Hamming, Hanning, and triangular windows reduce the time-domain signal levels applied to the DFT, their main lobe peak values are reduced relative to the rectangular window. (Because of the near-zero $w(n)$ coefficients at the beginning and end of the sample interval, this signal level loss is called the processing gain, or loss, of a window.) Be that as it may, we're primarily interested in the windows' sidelobe levels, which are difficult to see in [Figure 3-16\(a\)](#)'s linear scale. We will avoid this difficulty by plotting the windows' magnitude responses on a logarithmic decibel scale, and normalize each plot so its main lobe peak values are zero dB. ([Appendix E](#) provides a discussion of the origin and utility of measuring frequency-domain responses on a logarithmic scale using decibels.) Defining the log magnitude response to be $|W_{\text{dB}}(m)|$, we get $|W_{\text{dB}}(m)|$ by using the expression

$$(3-31)$$

$$|W_{\text{dB}}(m)| = 20 \cdot \log_{10} \left(\frac{|W(m)|}{|W(0)|} \right).$$

Figure 3-16 Window magnitude responses: (a) $|W(m)|$ on a linear scale; (b) $|W_{\text{dB}}(m)|$ on a normalized logarithmic scale.



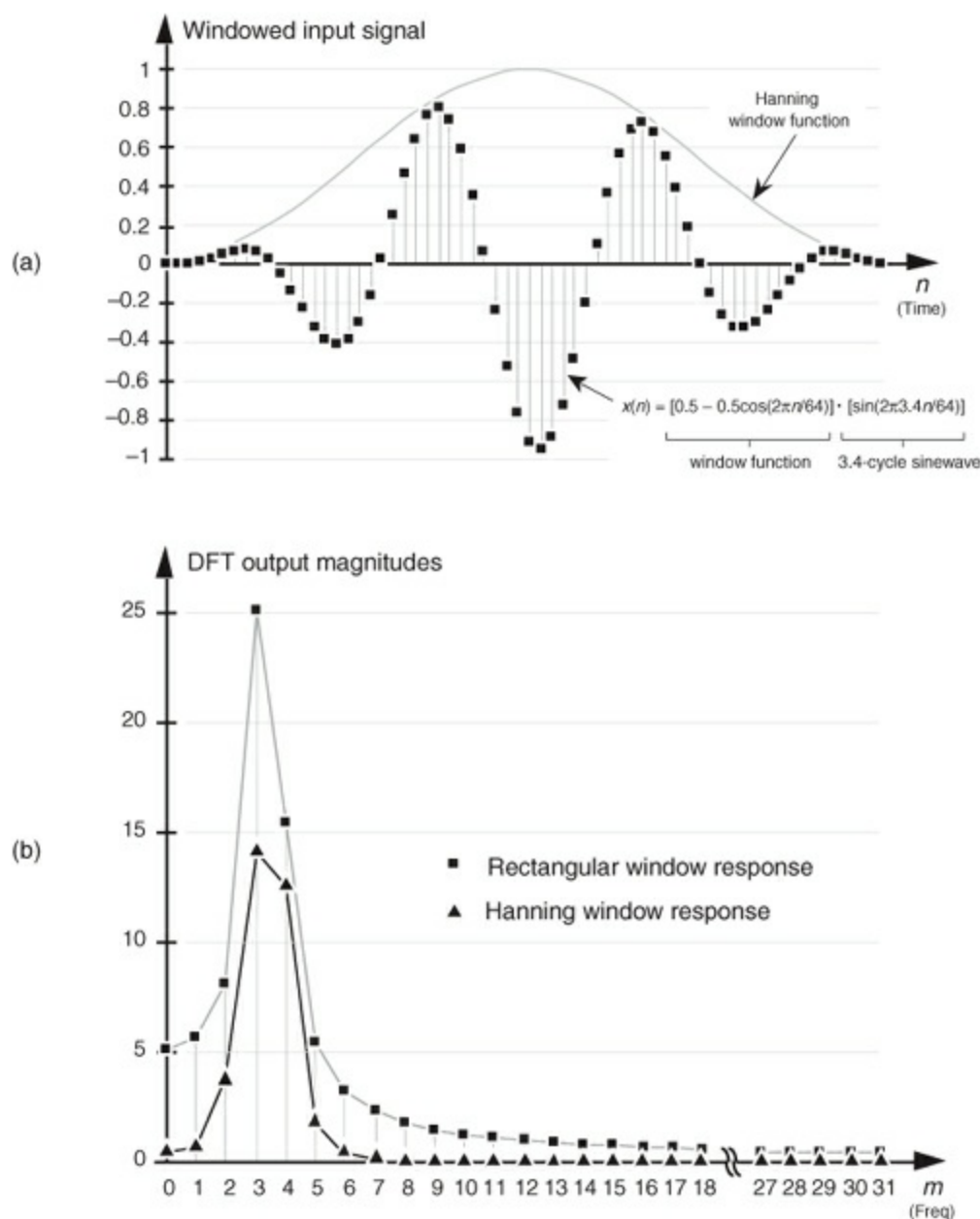
(The $|W(0)|$ term in the denominator of [Eq. \(3-31\)](#) is the value of $W(m)$ at the peak of the main lobe when $m = 0$.) The $|W_{\text{dB}}(m)|$ curves for the various window functions are shown in [Figure 3-16\(b\)](#). Now we can really see how the various window sidelobe responses compare to each other.

Looking at the rectangular window's magnitude response, we see that its main lobe is the most narrow, f_s/N . However, unfortunately, its first sidelobe level is only -13 dB below the main lobe peak, which is not so good. (Notice that we're only showing the positive-frequency portion of the window responses in [Figure 3-16](#).) The triangular window has reduced sidelobe levels, but the price we've paid is that the triangular window's main lobe width is twice as wide as that of the rectangular window's. The various nonrectangular windows' wide main lobes degrade the windowed DFT's frequency resolution by almost a factor of two. However, as we'll see, the important benefits of leakage reduction usually outweigh the loss in DFT frequency resolution.

Notice the further reduction of the first sidelobe level, and the rapid sidelobe roll-off of the Hanning window. The Hamming window has even lower first sidelobe levels, but this window's sidelobes roll off slowly relative to the Hanning window. This means that leakage three or four bins away from the center bin is lower for the Hamming window than for the Hanning window, and leakage a half-dozen or so bins away from the center bin is lower for the Hanning window than for the Hamming window.

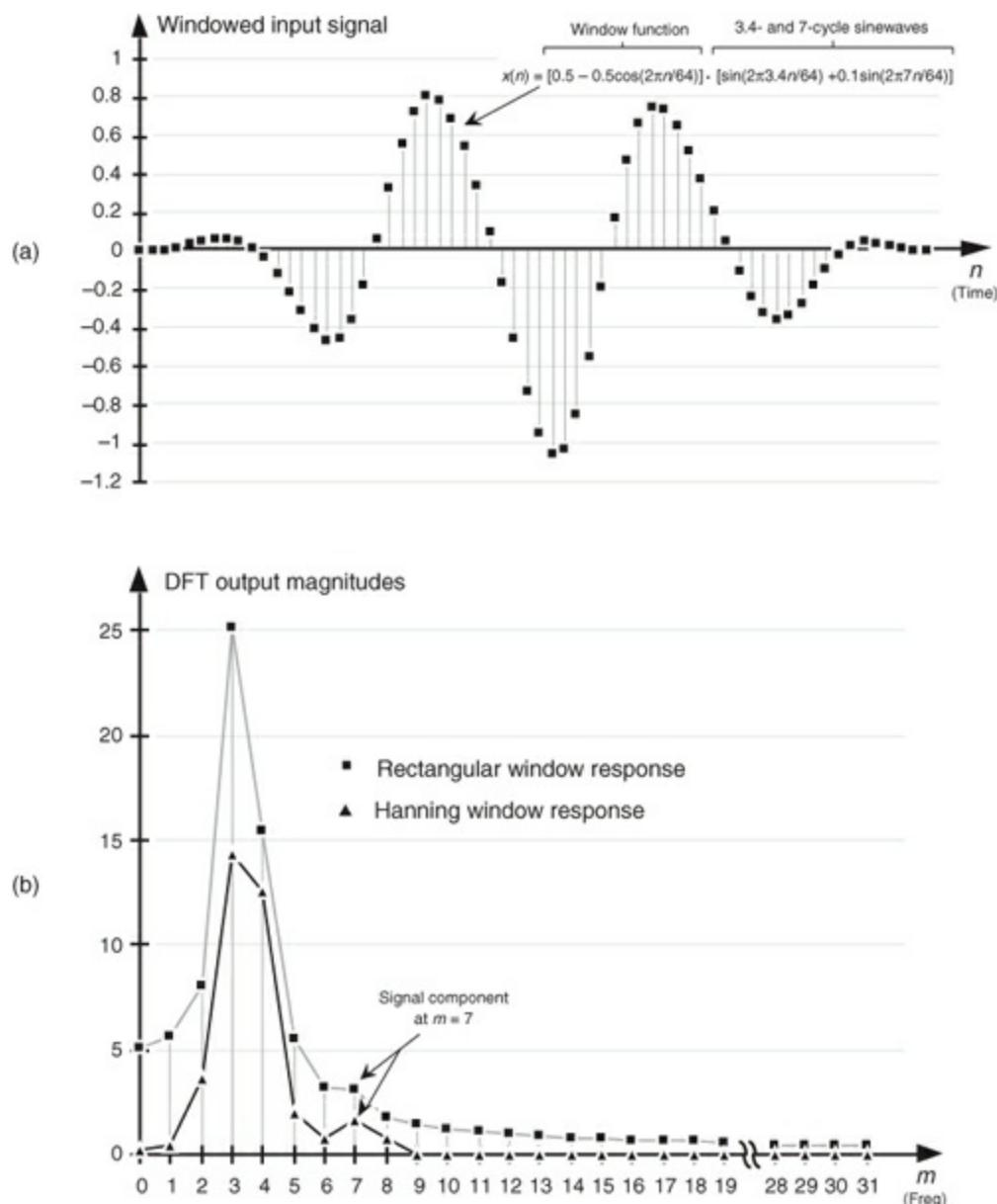
When we apply the Hanning window to [Figure 3-8\(a\)](#)'s 3.4 cycles per sample interval example, we end up with the DFT input shown in [Figure 3-17\(a\)](#) under the Hanning window envelope. The DFT outputs for the windowed waveform are shown in [Figure 3-17\(b\)](#) along with the DFT results with no windowing, i.e., the rectangular window. As we expected, the shape of the Hanning window's response looks broader and has a lower peak amplitude, but its sidelobe leakage is noticeably reduced from that of the rectangular window.

Figure 3-17 Hanning window: (a) 64-sample product of a Hanning window and a 3.4 cycles per sample interval input sinewave; (b) Hanning DFT output response versus rectangular window DFT output response.



We can demonstrate the benefit of using a window function to help us detect a low-level signal in the presence of a nearby high-level signal. Let's add 64 samples of a 7 cycles per sample interval sinewave, with a peak amplitude of only 0.1, to [Figure 3-8\(a\)](#)'s unity-amplitude 3.4 cycles per sample sinewave. When we apply a Hanning window to the sum of these sinewaves, we get the time-domain input shown in [Figure 3-18\(a\)](#). Had we not windowed the input data, our DFT output would be the squares in [Figure 3-18\(b\)](#) where DFT leakage causes the input signal component at $m = 7$ to be barely discernible. However, the DFT of the windowed data shown as the triangles in [Figure 3-18\(b\)](#) makes it easier for us to detect the presence of the $m = 7$ signal component. From a practical standpoint, people who use the DFT to perform real-world signal detection have learned that their overall frequency resolution and signal sensitivity are affected much more by the size and shape of their window function than the mere size of their DFTs.

Figure 3-18 Increased signal detection sensitivity afforded using windowing: (a) 64-sample product of a Hanning window and the sum of a 3.4 cycles and a 7 cycles per sample interval sinewaves; (b) reduced leakage Hanning DFT output response versus rectangular window DFT output response.



As we become more experienced using window functions on our DFT input data, we'll see how different window functions have their own individual advantages and disadvantages. Furthermore, regardless of the window function used, we've decreased the leakage in our DFT output from that of the rectangular window. There are many different window functions described in the literature of digital signal processing—so many, in fact, that they've been named after just about everyone in the digital signal processing business. It's not that clear that there's a great deal of difference among many of these window functions. What we find is that window selection is a trade-off between main lobe widening, first sidelobe levels, and how fast the sidelobes decrease with increased frequency. The use of any particular window depends on the application[5], and there are many applications.

Windows are used to improve DFT spectrum analysis accuracy[6], to design digital filters[7,8], to simulate antenna radiation patterns, and even in the hardware world to improve the performance of certain mechanical force to voltage conversion devices[9]. So there's plenty of window information available for those readers seeking further knowledge. (The mother of all technical papers on windows is that by Harris[10]. A useful paper by Nuttall corrected and extended some portions of Harris's paper[11].) Again, the best way to appreciate windowing effects is to have access to a computer software package that contains DFT, or FFT, routines and start analyzing windowed signals. (By the way, while we delayed their discussion until [Section 5.3](#), there are two other commonly used window functions that can be used to reduce DFT leakage. They're the Chebyshev and Kaiser

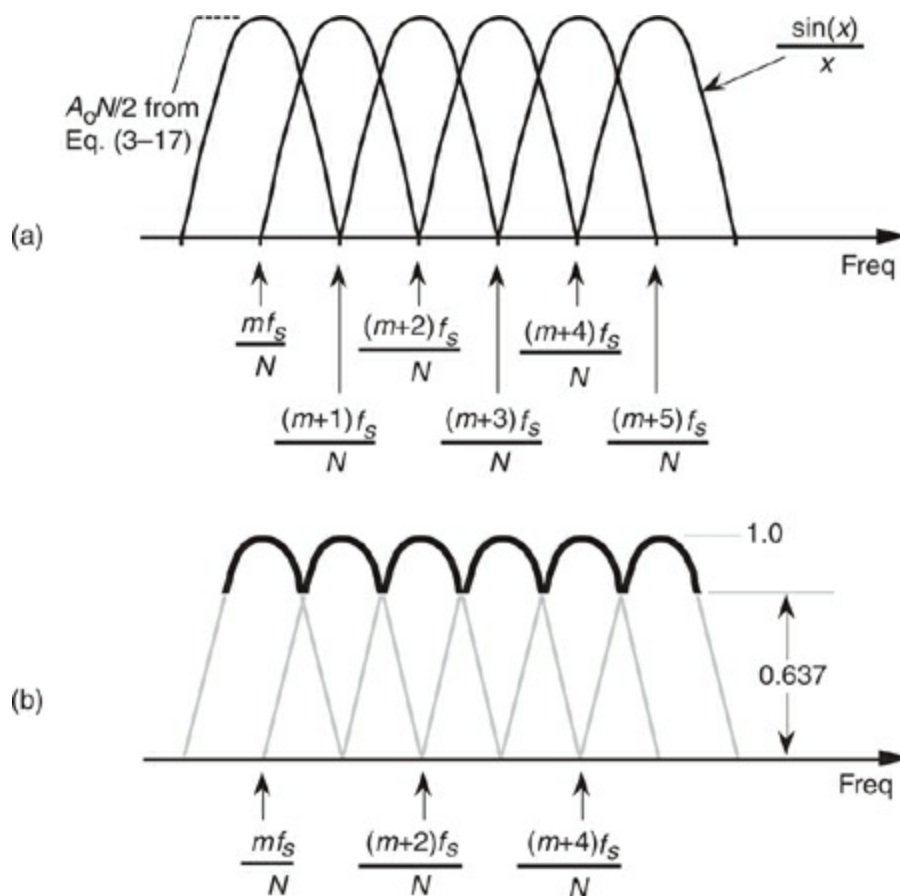
window functions, which have adjustable parameters, enabling us to strike a compromise between widening main lobe width and reducing sidelobe levels.)

3.10 DFT Scalping Loss

Scalping is the name used to describe fluctuations in the overall magnitude response of an N -point DFT. Although we derive this fact in Section 3.16, for now we'll just say that when no input windowing function is used, the $\sin(x)/x$ shape of the sinc function's magnitude response applies to each DFT output bin. [Figure 3-19\(a\)](#) shows a DFT's aggregate magnitude response by superimposing several $\sin(x)/x$ bin magnitude responses.[‡] (Because the sinc function's sidelobes are not key to this discussion, we don't show them in [Figure 3-19\(a\)](#).) Notice from [Figure 3-19\(b\)](#) that the overall DFT frequency-domain response is indicated by the bold envelope curve. This rippled curve, also called the picket fence effect, illustrates the processing loss for input frequencies between the bin centers.

[‡] Perhaps [Figure 3-19\(a\)](#) is why individual DFT outputs are called "bins." Any signal energy under a $\sin(x)/x$ curve will show up in the *enclosed storage compartment* of that DFT's output sample.

Figure 3-19 DFT bin magnitude response curves: (a) individual $\sin(x)/x$ responses for each DFT bin; (b) equivalent overall DFT magnitude response.



From [Figure 3-19\(b\)](#), we can determine that the magnitude of the DFT response fluctuates from 1.0, at bin center, to 0.637 halfway between bin centers. If we're interested in DFT output power levels, this envelope ripple exhibits a scalping loss of almost -4 dB halfway between bin centers. [Figure 3-19](#) illustrates a DFT output when no window (i.e., a rectangular window) is used. Because nonrectangular window functions broaden the DFT's main lobe, their use results in a scalping loss that will not be as severe as with the rectangular window[10,12]. That is, their wider main lobes overlap more and fill in the valleys of the envelope curve in [Figure 3-19\(b\)](#). For example, the scalping loss of a Hanning window is approximately 0.82, or -1.45 dB, halfway between bin

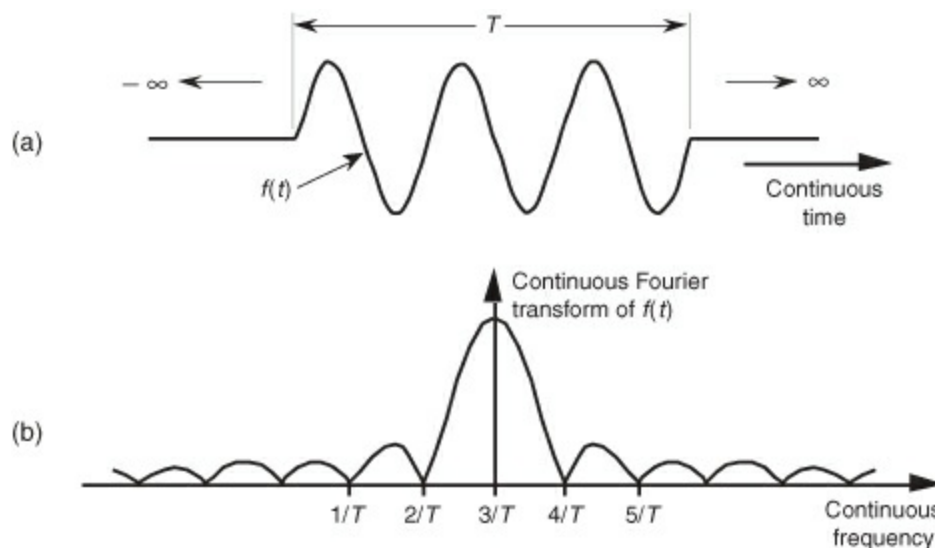
centers. Scalloping loss is not, however, a severe problem in practice. Real-world signals normally have bandwidths that span many frequency bins so that DFT magnitude response ripples can go almost unnoticed. Let's look at a scheme called *zero padding* that's used to both alleviate scalloping loss effects and to improve the DFT's frequency granularity.

3.11 DFT Resolution, Zero Padding, and Frequency-Domain Sampling

One popular method used to improve DFT spectral estimation is known as *zero padding*. This process involves the addition of zero-valued data samples to an original DFT input sequence to increase the total number of input data samples. Investigating this zero-padding technique illustrates the DFT's important property of frequency-domain sampling alluded to in the discussion on leakage. When we sample a continuous time-domain function, having a continuous Fourier transform (CFT), and take the DFT of those samples, the DFT results in a frequency-domain sampled approximation of the CFT. The more points in our DFT, the better our DFT output approximates the CFT.

To illustrate this idea, suppose we want to approximate the CFT of the continuous $f(t)$ function in [Figure 3-20\(a\)](#). This $f(t)$ waveform extends to infinity in both directions but is nonzero only over the time interval of T seconds. If the nonzero portion of the time function is a sinewave of three cycles in T seconds, the magnitude of its CFT is shown in [Figure 3-20\(b\)](#). (Because the CFT is taken over an infinitely wide time interval, the CFT has infinitesimally small frequency resolution, resolution so fine-grained that it's continuous.) It's this CFT that we'll approximate with a DFT.

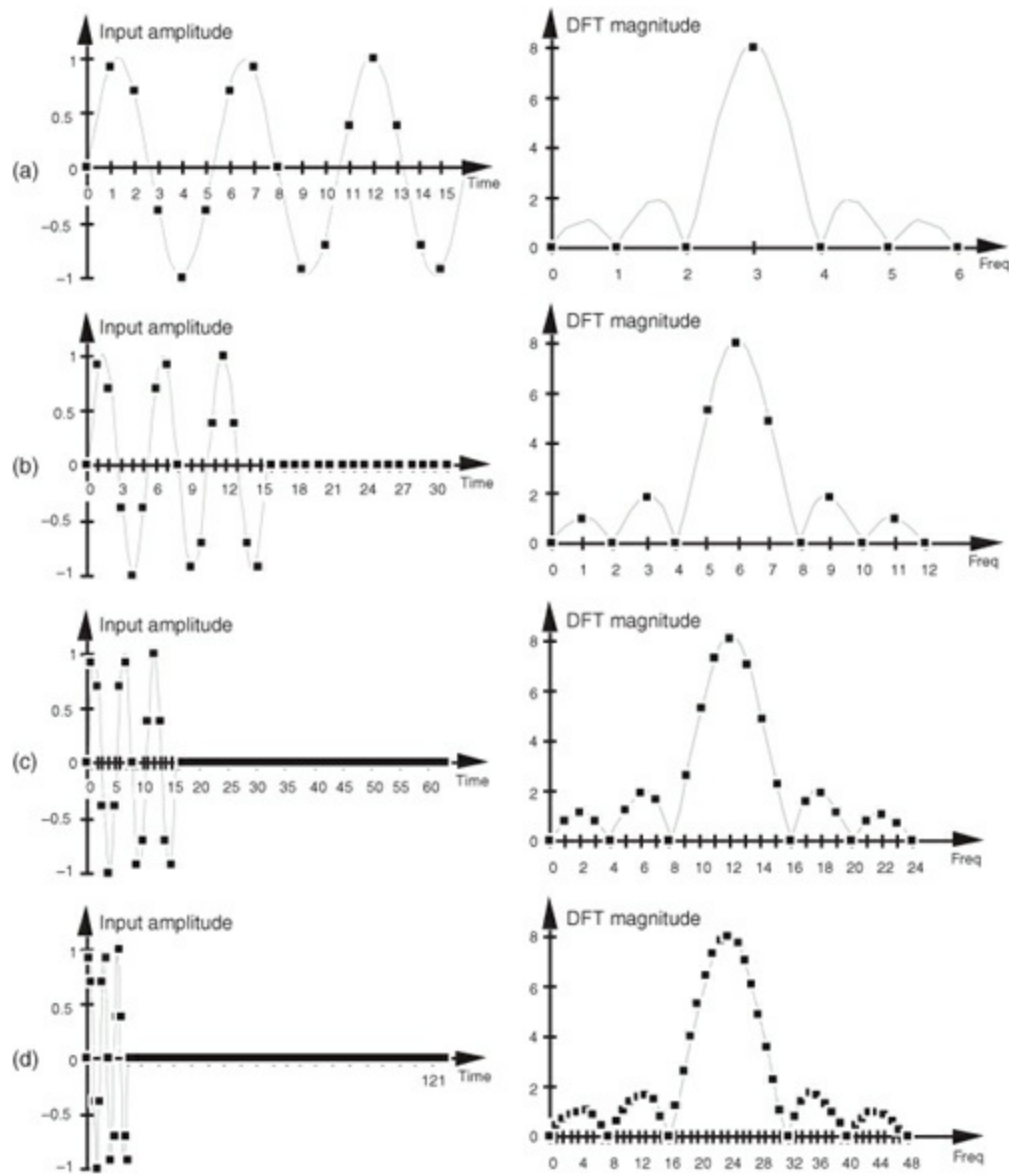
Figure 3-20 Continuous Fourier transform: (a) continuous time-domain $f(t)$ of a truncated sinusoid of frequency $3/T$; (b) continuous Fourier transform of $f(t)$.



Suppose we want to use a 16-point DFT to approximate the CFT of $f(t)$ in [Figure 3-20\(a\)](#). The 16 discrete samples of $f(t)$, spanning the three periods of $f(t)$'s sinusoid, are those shown on the left side of [Figure 3-21\(a\)](#). Applying those time samples to a 16-point DFT results in discrete frequency-domain samples, the positive frequencies of which are represented by the dots on the right side of [Figure 3-21\(a\)](#). We can see that the DFT output comprises samples of [Figure 3-20\(b\)](#)'s CFT. If we append (or zero-pad) 16 zeros to the input sequence and take a 32-point DFT, we get the output shown on the right side of [Figure 3-21\(b\)](#), where we've increased our DFT frequency sampling by a factor of two. Our DFT is sampling the input function's CFT more often now. Adding 32 more zeros and taking a 64-point DFT, we get the output shown on the right side of [Figure 3-21\(c\)](#). The 64-point DFT output now begins to show the true shape of the CFT. Adding 64 more zeros and taking a 128-point DFT, we get the output shown on the right side of [Figure 3-21\(d\)](#). The DFT frequency-domain

sampling characteristic is obvious now, but notice that the bin index for the center of the main lobe is different for each of the DFT outputs in [Figure 3-21](#).

Figure 3-21 DFT frequency-domain sampling: (a) 16 input data samples and $N = 16$; (b) 16 input data samples, 16 padded zeros, and $N = 32$; (c) 16 input data samples, 48 padded zeros, and $N = 64$; (d) 16 input data samples, 112 padded zeros, and $N = 128$.



Does this mean we have to redefine the DFT's frequency axis when using the zero-padding technique? Not really. If we perform zero padding on L nonzero input samples to get a total of N time samples for an N -point DFT, the zero-padded DFT output bin center frequencies are related to the original f_s by our old friend [Eq. \(3-5\)](#), or

$$(3-32)$$

$$\text{center frequency of the } m\text{th bin} = \frac{mf_s}{N}.$$

So in our [Figure 3-21\(a\)](#) example, we use [Eq. \(3-32\)](#) to show that although the zero-padded DFT output bin index of the main lobe changes as N increases, the zero-padded DFT output frequency associated with the main lobe remains the same. The following list shows how this works:

Figure no.	Main lobe peak located at $m =$	$L =$	$N =$	Frequency of main lobe peak relative to $f_s =$
Figure 3-21(a)	3	16	16	$3f_s/16$
Figure 3-21(b)	6	16	32	$6 \cdot f_s/32 = 3f_s/16$
Figure 3-21(c)	12	16	64	$12 \cdot f_s/64 = 3f_s/16$
Figure 3-21(d)	24	16	128	$24 \cdot f_s/128 = 3f_s/16$

Do we gain anything by appending more zeros to the input sequence and taking larger DFTs? Not really, because our 128-point DFT is sampling the input's CFT sufficiently now in [Figure 3-21\(d\)](#). Sampling it more often with a larger DFT won't improve our understanding of the input's frequency content. The issue here is that adding zeros to an input sequence will improve our DFT's output resolution, but there's a practical limit on how much we gain by adding more zeros. For our example here, a 128-point DFT shows us the detailed content of the input spectrum. We've hit a *law of diminishing returns* here. Performing a 256-point or 512-point DFT, in our case, would serve little purpose.[‡] There's no reason to *oversample* this particular input sequence's CFT. Of course, there's nothing sacred about stopping at a 128-point DFT. Depending on the number of samples in some arbitrary input sequence and the sample rate, we might, in practice, need to append any number of zeros to get some desired DFT frequency resolution.

[‡] Notice that the DFT sizes (N) we've discussed are powers of 2 (64, 128, 256, 512). That's because we actually perform DFTs using a special algorithm known as the *fast Fourier transform* (FFT). As we'll see in [Chapter 4](#), the typical implementation of the FFT requires that N be a power of two.

There are two final points to be made concerning zero padding. First, the DFT magnitude expressions in [Eqs. \(3-17\)](#) and [\(3-17'\)](#) don't apply if zero padding is being used. If we perform zero padding on L nonzero samples of a sinusoid whose frequency is located at a bin center to get a total of N input samples for an N -point DFT, we must replace the N with L in [Eqs. \(3-17\)](#) and [\(3-17'\)](#) to predict the DFT's output magnitude for that particular sinewave. Second, in practical situations, if we want to perform both zero padding and windowing on a sequence of input data samples, we must be careful not to apply the window to the entire input including the appended zero-valued samples. The window function must be applied only to the original nonzero time samples; otherwise the padded zeros will *zero out* and distort part of the window function, leading to erroneous results. ([Section 4.2](#) gives additional practical pointers on performing the DFT using the FFT algorithm to analyze real-world signals.)

To digress slightly, now's a good time to define the term *discrete-time Fourier transform* (DTFT) which the reader may encounter in the literature. The DTFT is the continuous Fourier transform of an L -point discrete time-domain sequence, and some authors use the DTFT to describe many of the digital signal processing concepts we've covered in this chapter. On a computer we can't perform the DTFT because it has an infinitely fine frequency resolution—but we can approximate the DTFT by performing an N -point DFT on an L -point discrete time sequence where $N > L$. That is, in fact, what we did in [Figure 3-21](#) when we zero-padded the original 16-point time sequence. (When $N = L$, the DTFT approximation is identical to the DFT.)

To make the connection between the DTFT and the DFT, know that the infinite-resolution DTFT magnitude (i.e., continuous Fourier transform magnitude) of the 16 nonzero time samples in [Figure 3-21\(a\)](#) is the shaded $\sin(x)/x$ -like spectral function in [Figure 3-21](#). Our DFTs approximate (sample) that function. Increased zero padding of the 16 nonzero time samples merely interpolates our DFT's

sampled version of the DTFT function with smaller and smaller frequency-domain sample spacing.

Please keep in mind, however, that zero padding does not improve our ability to resolve, to distinguish between, two closely spaced signals in the frequency domain. (For example, the main lobes of the various spectra in [Figure 3-21](#) do *not* change in width, if measured in Hz, with increased zero padding.) To improve our true spectral resolution of two signals, we need more nonzero time samples. The rule by which we must live is: To realize F_{res} Hz spectral resolution, we must collect $1/F_{\text{res}}$ seconds, worth of nonzero time samples for our DFT processing.

We'll discuss applications of time-domain zero padding in [Section 13.15](#), revisit the DTFT in [Section 3.14](#), and frequency-domain zero padding in [Section 13.28](#).

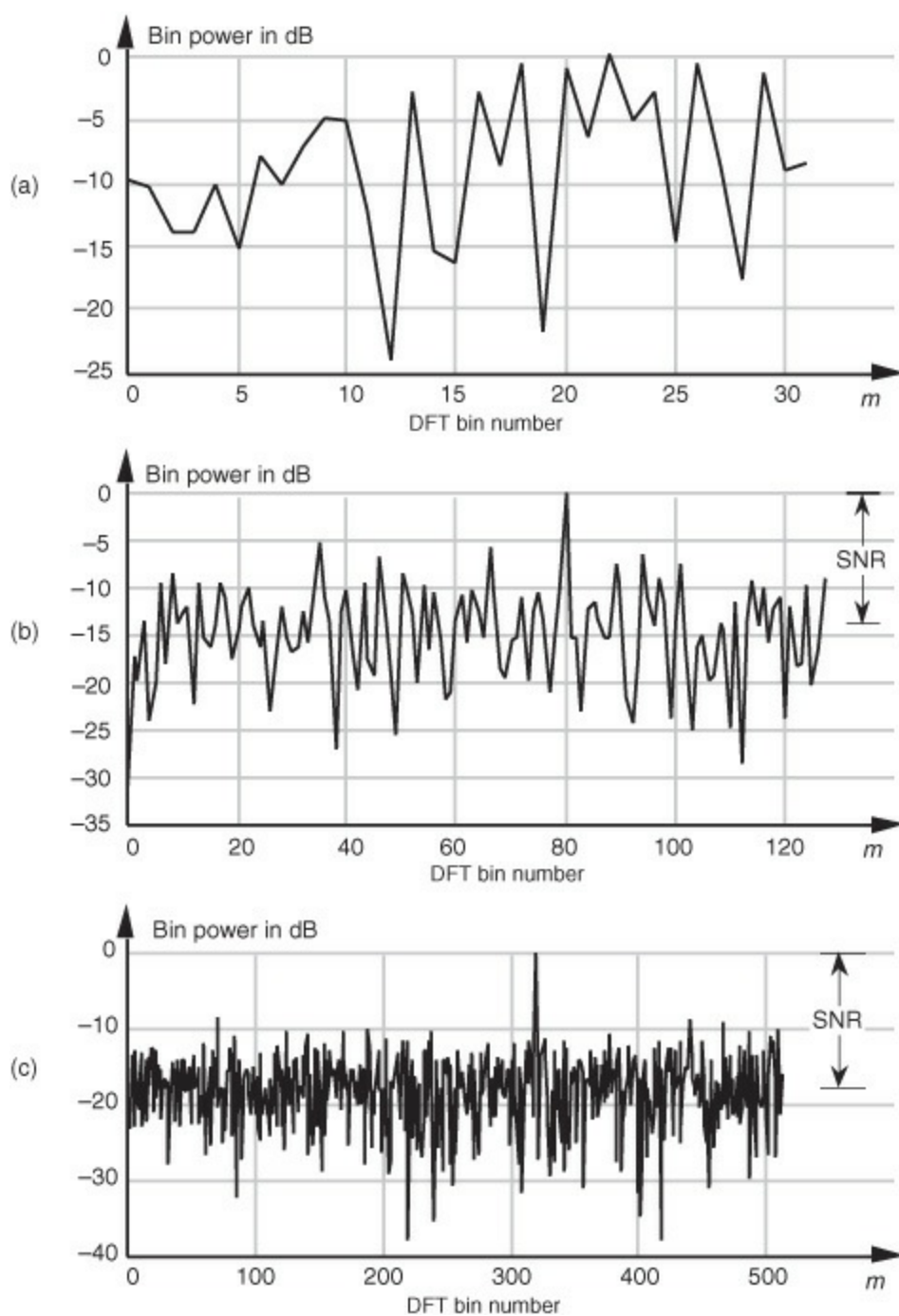
3.12 DFT Processing Gain

There are two types of processing gain associated with DFTs. People who use the DFT to detect signal energy embedded in noise often speak of the DFT's *processing gain* because the DFT can *pull* signals out of background noise. This is due to the inherent correlation gain that takes place in any N -point DFT. Beyond this natural processing gain, additional *integration gain* is possible when multiple DFT outputs are averaged. Let's look at the DFT's inherent processing gain first.

3.12.1 Processing Gain of a Single DFT

The concept of the DFT having processing gain is straightforward if we think of a particular DFT bin output as the output of a narrowband filter. Because a DFT output bin has the amplitude response of the $\sin(x)/x$ function, that bin's output is primarily due to input energy residing under, or very near, the bin's main lobe. It's valid to think of a DFT bin as a kind of *bandpass* filter whose band center is located at mf_s/N . We know from [Eq. \(3-17\)](#) that the maximum possible DFT output magnitude increases as the number of points (N) in a DFT increases. Also, as N increases, the DFT output bin main lobes become narrower. So a DFT output bin can be treated as a bandpass filter whose gain can be increased and whose bandwidth can be reduced by increasing the value of N . Decreasing a bandpass filter's bandwidth is useful in energy detection because the frequency resolution improves in addition to the filter's ability to minimize the amount of background noise that resides within its passband. We can demonstrate this by looking at the DFT of a spectral tone (a constant-frequency sinewave) added to random noise. [Figure 3-22\(a\)](#) is a logarithmic plot showing the first 32 outputs of a 64-point DFT when the input tone is at the center of the DFT's $m = 20$ th bin. The output power levels (DFT magnitude squared) in [Figure 3-22\(a\)](#) are normalized so that the highest bin output power is set to 0 dB. Because the tone's original signal power is below the average noise power level, the tone is a bit difficult to detect when $N = 64$. (The time-domain noise, used to generate [Figure 3-22\(a\)](#), has an average value of zero, i.e., no DC bias or amplitude offset.) If we quadruple the number of input samples and increase the DFT size to $N = 256$, we can now see the tone power raised above the average background noise power as shown for $m = 80$ in [Figure 3-22\(b\)](#). Increasing the DFT's size to $N = 1024$ provides additional processing gain to pull the tone further up out of the noise as shown in [Figure 3-22\(c\)](#).

Figure 3-22 Single DFT processing gain: (a) $N = 64$; (b) $N = 256$; (c) $N = 1024$.



To quantify the idea of DFT processing gain, we can define a signal-to-noise ratio (SNR) as the DFT's *output signal-power level* over the *average output noise-power level*. (In practice, of course, we like to have this ratio as large as possible.) For several reasons, it's hard to say what any given single DFT output SNR will be. That's because we can't exactly predict the energy in any given N samples of random noise. Also, if the input signal frequency is not at bin center, leakage will raise the effective background noise and reduce the DFT's output SNR. In addition, any window being used will have some effect on the leakage and, thus, on the output SNR. What we'll see is that the DFT's output SNR increases as N gets larger because a DFT bin's output noise standard deviation (*rms*) value is proportional to \sqrt{N} , and the DFT's output magnitude for the bin containing the signal tone is proportional to N .[‡] More generally for real inputs, if $N > N'$, an N -point DFT's output SNR_N increases over the N' -point DFT $SNR_{N'}$ by the following relationship:

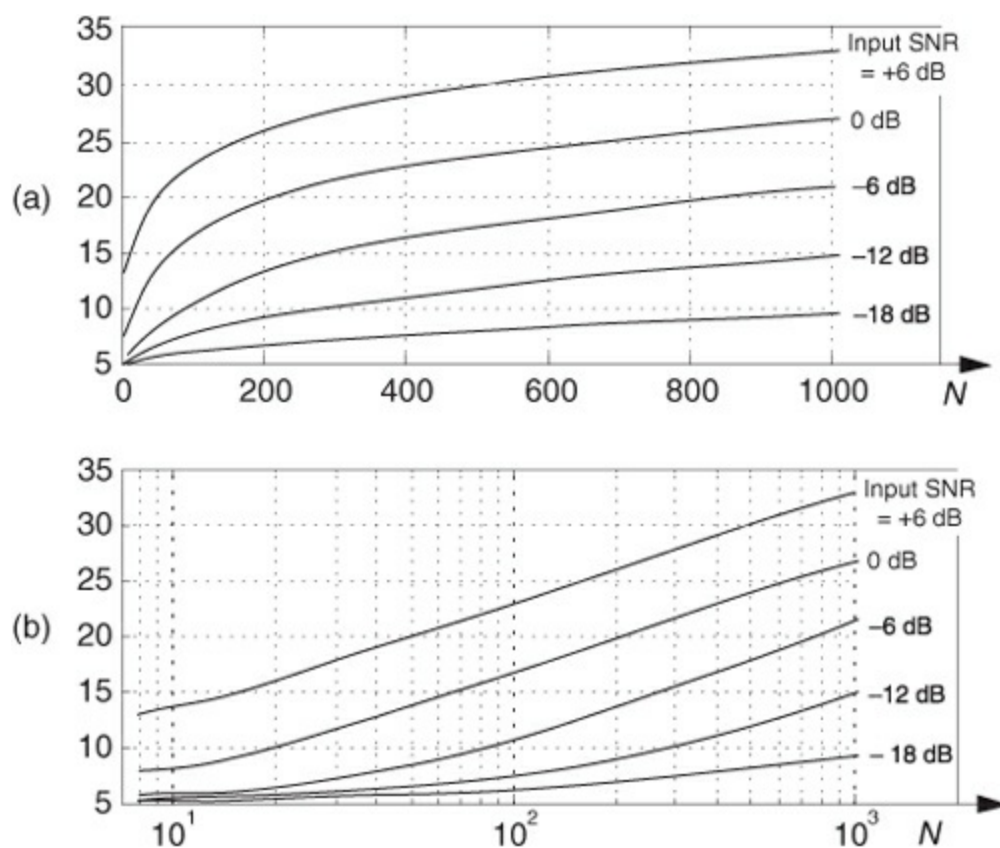
[‡] *rms* = root mean square.

$$SNR_N = SNR_{N'} + 10 \log_{10} \left(\frac{N}{N'} \right).$$

If we increase a DFT's size from N' to $N = 2N'$, from [Eq. \(3-33\)](#), the DFT's output SNR increases by 3 dB. So we say that a DFT's processing gain increases by 3 dB whenever N is doubled. Be aware that we may double a DFT's size and get a resultant processing gain of less than 3 dB in the presence of random noise; then again, we may gain slightly more than 3 dB. That's the nature of random noise. If we perform many DFTs, we'll see an average processing gain, shown in [Figure 3-23\(a\)](#), for various input signal SNRs. Because we're interested in the slope of the curves in [Figure 3-23\(a\)](#), we plot those curves on a logarithmic scale for N in [Figure 3-23\(b\)](#) where the curves straighten out and become linear. Looking at the slope of the curves in [Figure 3-23\(b\)](#), we can now see the 3 dB increase in processing gain as N doubles so long as N is greater than 20 or 30 and the signal is not overwhelmed by noise. There's nothing sacred about the absolute values of the curves in [Figures 3-23\(a\)](#) and [3-23\(b\)](#). They were generated through a simulation of noise and a tone whose frequency was at a DFT bin center. Had the tone's frequency been between bin centers, the processing gain curves would have been shifted downward, but their shapes would still be the same;[‡] that is, [Eq. \(3-33\)](#) is still valid regardless of the input tone's frequency.

[‡] The curves would be shifted downward, indicating a lower SNR, because leakage would raise the average noise-power level, and scalloping loss would reduce the DFT bin's output power level.

Figure 3-23 DFT processing gain versus number of DFT points N for various input signal-to-noise ratios: (a) linear N axis; (b) logarithmic N axis.



3.12.2 Integration Gain Due to Averaging Multiple DFTs

Theoretically, we could get very large DFT processing gains by increasing the DFT size arbitrarily. The problem is that the number of necessary DFT multiplications increases proportionally to N^2 , and larger DFTs become very computationally intensive. Because addition is easier and faster to perform than multiplication, we can average the outputs of multiple DFTs to obtain further processing gain and

signal detection sensitivity. The subject of averaging multiple DFT outputs is covered in [Section 11.3](#).

3.13 The DFT of Rectangular Functions

We continue this chapter by providing the mathematical details of two important aspects of the DFT. First, we obtain the expressions for the DFT of a rectangular function (rectangular window), and then we'll use these results to illustrate the magnitude response of the DFT. We're interested in the DFT's magnitude response because it provides an alternate viewpoint to understand the leakage that occurs when we use the DFT as a signal analysis tool.

One of the most prevalent and important computations encountered in digital signal processing is the DFT of a rectangular function. We see it in sampling theory, window functions, discussions of convolution, spectral analysis, and in the design of digital filters. As common as it is, however, the literature covering the DFT of rectangular functions can be confusing to the digital signal processing beginner for several reasons. The standard mathematical notation is a bit hard to follow at first, and sometimes the equations are presented with too little explanation. Compounding the problem, for the beginner, are the various expressions of this particular DFT. In the literature, we're likely to find any one of the following forms for the DFT of a rectangular function:

(3-34)

$$\text{DFT}_{\text{rect.function}} = \frac{\sin(x)}{\sin(x/N)}, \text{ or } \frac{\sin(x)}{x}, \text{ or } \frac{\sin(Nx/2)}{\sin(x/2)}.$$

In this section we'll show how the forms in [Eq. \(3-34\)](#) were obtained, see how they're related, and create a kind of *Rosetta Stone* table allowing us to move back and forth between the various DFT expressions. Take a deep breath and let's begin our discussion with the definition of a rectangular function.

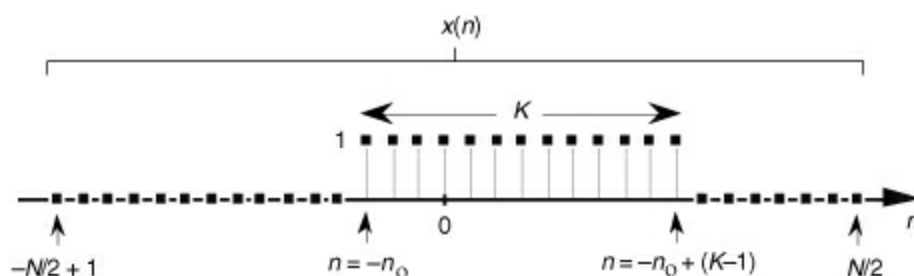
3.13.1 DFT of a General Rectangular Function

A general rectangular function $x(n)$ can be defined as N samples containing K unity-valued samples as shown in [Figure 3-24](#). The full N -point sequence, $x(n)$, is the rectangular function that we want to transform. We call this the general form of a rectangular function because the K unity samples begin at an arbitrary index value of $-n_0$. Let's take the DFT of $x(n)$ in [Figure 3-24](#) to get our desired $X(m)$. Using m as our frequency-domain sample index, the expression for an N -point DFT is

(3-35)

$$X(m) = \sum_{n=-(N/2)+1}^{N/2} x(n) e^{-j2\pi nm/N}.$$

Figure 3-24 Rectangular function of width K samples defined over N samples where $K < N$.



With $x(n)$ being nonzero only over the range of $-n_0 \leq n \leq -n_0 + (K-1)$, we can modify the summation

limits of [Eq. \(3-35\)](#) to express $X(m)$ as

(3-36)

$$X(m) = \sum_{n=-n_o}^{-n_o+(K-1)} 1 \cdot e^{-j2\pi nm/N},$$

because only the K samples contribute to $X(m)$. That last step is important because it allows us to eliminate the $x(n)$ terms and make [Eq. \(3-36\)](#) easier to handle. To keep the following equations from being too messy, let's use the dummy variable $q = 2\pi m/N$.

OK, here's where the algebra comes in. Over our new limits of summation, we eliminate the factor of one and [Eq. \(3-36\)](#) becomes

(3-37)

$$\begin{aligned} X(q) &= \sum_{n=-n_o}^{-n_o+(K-1)} e^{-jqn} \\ &= e^{-jq(-n_o)} + e^{-jq(-n_o+1)} + e^{-jq(-n_o+2)} + \dots + e^{-jq(-n_o+(K-1))} \\ &= e^{-jq(-n_o)} e^{-j0q} + e^{-jq(-n_o)} e^{-j1q} + e^{-jq(-n_o)} e^{-j2q} + \dots + e^{-jq(-n_o)} e^{-jq(K-1)} \\ &= e^{jq(n_o)} \cdot [e^{-j0q} + e^{-j1q} + e^{-j2q} + \dots + e^{-jq(K-1)}]. \end{aligned}$$

The series inside the brackets of [Eq. \(3-37\)](#) allows the use of a summation, such as

(3-38)

$$X(q) = e^{jq(n_o)} \sum_{p=0}^{K-1} e^{-j pq}.$$

[Equation \(3-38\)](#) certainly doesn't look any simpler than [Eq. \(3-36\)](#), but it is. [Equation \(3-38\)](#) is a *geometric series* and, from the discussion in [Appendix B](#), it can be evaluated to the closed form of

(3-39)

$$\sum_{p=0}^{K-1} e^{-j pq} = \frac{1 - e^{-jqK}}{1 - e^{-jq}}.$$

We can now simplify [Eq. \(3-39\)](#)—here's the clever step. If we multiply and divide the numerator and denominator of [Eq. \(3-39\)](#)'s right-hand side by the appropriate half-angled exponentials, we break the exponentials into two parts and get

(3-40)

$$\begin{aligned}\sum_{p=0}^{K-1} e^{-j p q} &= \frac{e^{-j q K / 2} (e^{j q K / 2} - e^{-j q K / 2})}{e^{-j q / 2} (e^{j q / 2} - e^{-j q / 2})} \\ &= e^{-j q (K-1) / 2} \cdot \frac{(e^{j q K / 2} - e^{-j q K / 2})}{(e^{j q / 2} - e^{-j q / 2})}.\end{aligned}$$

Let's pause for a moment here to remind ourselves where we're going. We're trying to get [Eq. \(3-40\)](#) into a usable form because it's part of [Eq. \(3-38\)](#) that we're using to evaluate $X(m)$ in [Eq. \(3-36\)](#) in our quest for an understandable expression for the DFT of a rectangular function.

[Equation \(3-40\)](#) looks even more complicated than [Eq. \(3-39\)](#), but things can be simplified inside the parentheses. From Euler's equation, $\sin(\vartheta) = (e^{j\vartheta} - e^{-j\vartheta})/2j$, [Eq. \(3-40\)](#) becomes

(3-41)

$$\begin{aligned}\sum_{p=0}^{K-1} e^{-j p q} &= e^{-j q (K-1) / 2} \cdot \frac{2j \sin(qK / 2)}{2j \sin(q / 2)} \\ &= e^{-j q (K-1) / 2} \cdot \frac{\sin(qK / 2)}{\sin(q / 2)}.\end{aligned}$$

Substituting [Eq. \(3-41\)](#) for the summation in [Eq. \(3-38\)](#), our expression for $X(q)$ becomes

(3-42)

$$\begin{aligned}X(q) &= e^{j q (n_o)} \cdot e^{-j q (K-1) / 2} \cdot \frac{\sin(qK / 2)}{\sin(q / 2)} \\ &= e^{j q (n_o - (K-1) / 2)} \cdot \frac{\sin(qK / 2)}{\sin(q / 2)}.\end{aligned}$$

Returning our dummy variable q to its original value of $2\pi m/N$,

$$\begin{aligned}X(m) &= e^{j(2\pi m / N)(n_o - (K-1) / 2)} \cdot \frac{\sin(2\pi m K / 2N)}{\sin(2\pi m / 2N)}, \text{ or} \\ &\quad (3-43)\end{aligned}$$

General form of the

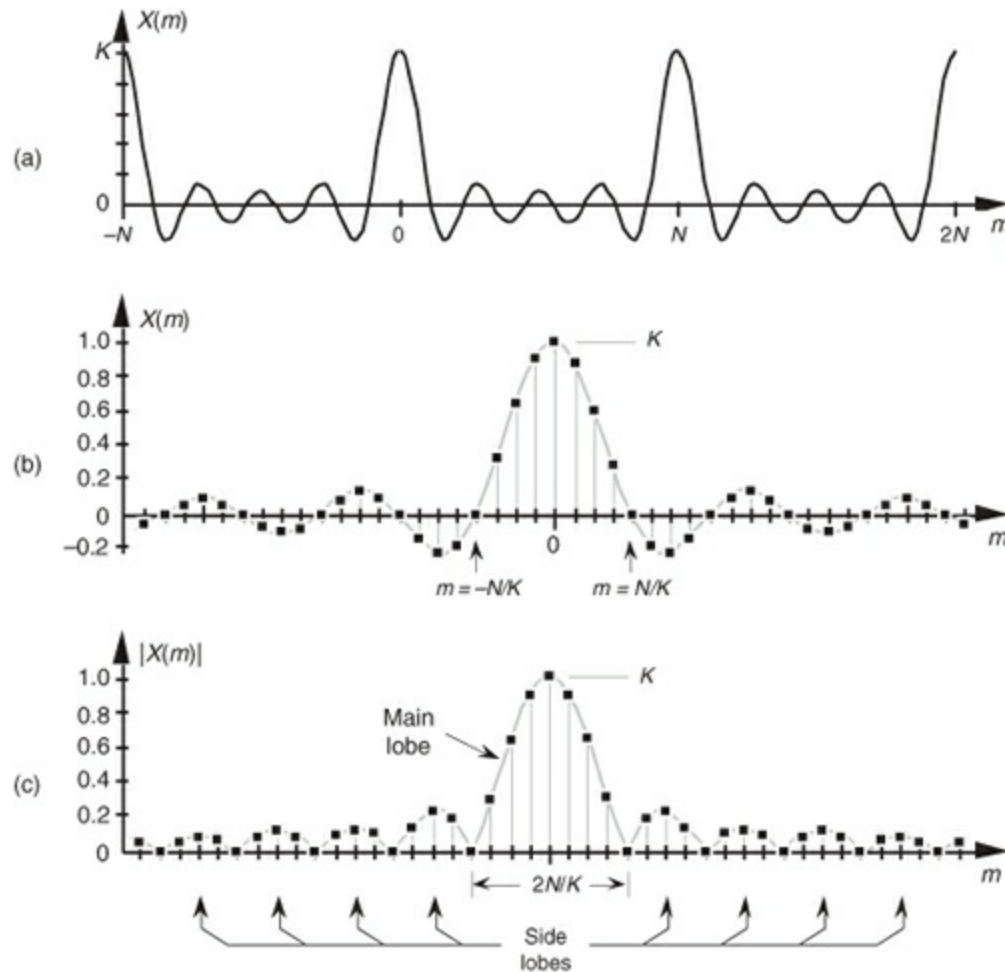
Dirichlet kernel:→

$$X(m) = e^{j(2\pi m / N)(n_o - (K-1) / 2)} \cdot \frac{\sin(\pi m K / N)}{\sin(\pi m / N)}.$$

So there it is (whew!). [Equation \(3-43\)](#) is the general expression for the DFT of the rectangular function as shown in [Figure 3-24](#). Our $X(m)$ is a complex expression (pun intended) where a ratio of sine terms is the amplitude of $X(m)$ and the exponential term is the phase angle of $X(m)$.[‡] The ratio of sines factor in [Eq. \(3-43\)](#) lies on the periodic curve shown in [Figure 3-25\(a\)](#), and like all N -point DFT representations, the periodicity of $X(m)$ is N . This curve is known as the *Dirichlet kernel* (or the *aliased sinc function*) and has been thoroughly described in the literature[[10,13,14](#)]. (It's named after the nineteenth-century German mathematician Peter Dirichlet [pronounced dee-ree-'klay], who studied the convergence of trigonometric series used to represent arbitrary functions.)

[‡] N was an even number in [Figure 3-24](#) depicting the $x(n)$. Had N been an odd number, the limits on the summation in [Eq. \(3-35\)](#) would have been $-(N-1)/2 \leq n \leq (N-1)/2$. Using these alternate limits would have led us to exactly the same $X(m)$ as in [Eq. \(3-43\)](#).

Figure 3-25 The Dirichlet kernel of $X(m)$: (a) periodic continuous curve on which the $X(m)$ samples lie; (b) $X(m)$ amplitudes about the $m = 0$ sample; (c) $|X(m)|$ magnitudes about the $m = 0$ sample.



We can zoom in on the curve at the $m = 0$ point and see more detail in [Figure 3-25\(b\)](#). The dots are shown in [Figure 3-25\(b\)](#) to remind us that the DFT of our rectangular function results in discrete amplitude values that lie on the curve. So when we perform DFTs, our discrete results are sampled values of the continuous sinc function's curve in [Figure 3-25\(a\)](#). As we'll show later, we're primarily interested in the absolute value, or magnitude, of the Dirichlet kernel in [Eq. \(3-43\)](#). That magnitude, $|X(m)|$, is shown in [Figure 3-25\(c\)](#). Although we first saw the sinc function's curve in [Figure 3-9](#) in [Section 3.8](#), where we introduced the topic of DFT leakage, we'll encounter this curve often in our study of digital signal processing.

For now, there are just a few things we need to keep in mind concerning the Dirichlet kernel. First, the DFT of a rectangular function has a main lobe, centered about the $m = 0$ point. The peak amplitude of the main lobe is K . This peak value makes sense, right? The $m = 0$ sample of a DFT $X(0)$ is the sum of the original samples, and the sum of K unity-valued samples is K . We can show this in a more substantial way by evaluating [Eq. \(3-43\)](#) for $m = 0$. A difficulty arises when we plug $m = 0$ into [Eq. \(3-43\)](#) because we end up with $\sin(0)/\sin(0)$, which is the indeterminate ratio $0/0$. Well, hardcore mathematics to the rescue here. We can use L'Hopital's Rule to take the derivative of the numerator and the denominator of [Eq. \(3-43\)](#), and *then* set $m = 0$ to determine the peak value of the magnitude of the Dirichlet kernel.[‡] We proceed as

[‡] L'Hopital is pronounced 'lō-pē-tōl', like baby doll.

(3-44)

$$\begin{aligned}
 |X(m)|_{m \rightarrow 0} &= \frac{d}{dm} X(m) = \frac{d[\sin(\pi m K / N)] / dm}{d[\sin(\pi m / N)] / dm} \\
 &= \frac{\cos(\pi m K / N)}{\cos(\pi m / N)} \cdot \frac{d(\pi m K / N) / dm}{d(\pi m / N) / dm} \\
 &= \frac{\cos(0)}{\cos(0)} \cdot \frac{\pi K / N}{\pi / N} = 1 \cdot K = K,
 \end{aligned}$$

which is what we set out to show. (We could have been clever and evaluated [Eq. \(3-35\)](#) with $m = 0$ to get the result of [Eq. \(3-44\)](#). Try it, and keep in mind that $e^{j0} = 1$.) Had the amplitudes of the nonzero samples of $x(n)$ been other than unity, say some amplitude A_0 , then, of course, the peak value of the Dirichlet kernel would be $A_0 K$ instead of just K . The next important thing to notice about the Dirichlet kernel is the main lobe's width. The first zero crossing of [Eq. \(3-43\)](#) occurs when the numerator's argument is equal to π , that is, when $\pi m K / N = \pi$. So the value of m at the first zero crossing is given by

(3-45)

$$m_{\text{first zero crossing}} = \frac{\pi N}{\pi K} = \frac{N}{K}$$

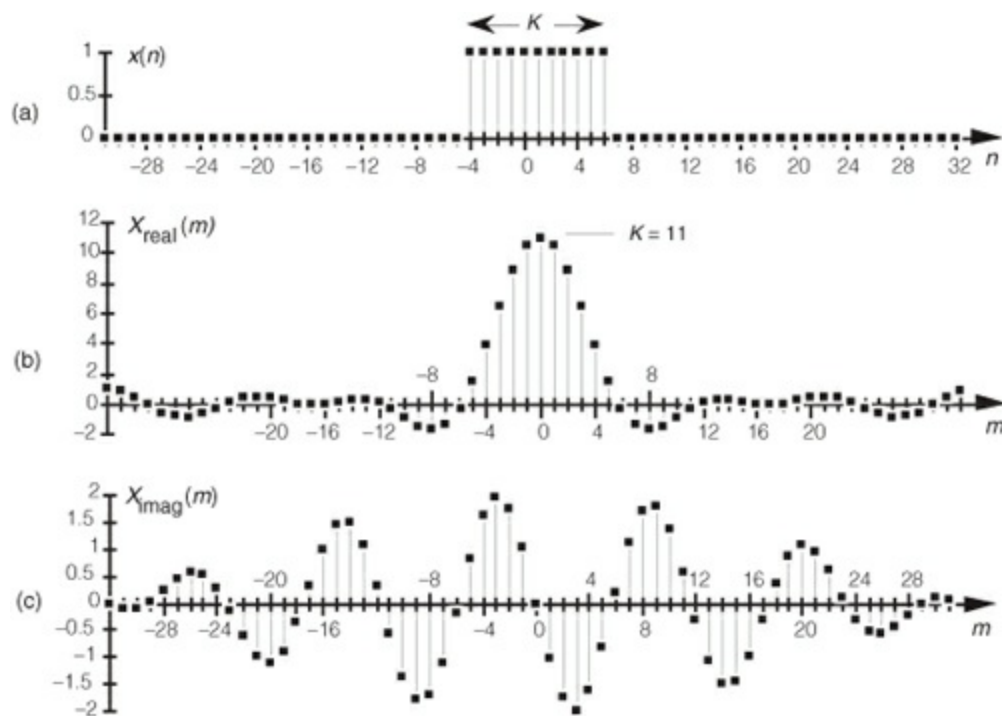
as shown in [Figure 3-25\(b\)](#). Thus the main lobe width $2N/K$, as shown in [Figure 3-25\(c\)](#), is inversely proportional to K .^{††}

^{††} This is a fundamental characteristic of Fourier transforms. The narrower the function in one domain, the wider its transform will be in the other domain.

Notice that the main lobe in [Figure 3-25\(a\)](#) is surrounded by a series of oscillations, called *sidelobes*, as in [Figure 3-25\(c\)](#). These sidelobe magnitudes decrease the farther they're separated from the main lobe. However, no matter how far we look away from the main lobe, these sidelobes never reach zero magnitude—and they cause a great deal of heartache for practitioners in digital signal processing. These sidelobes cause high-amplitude signals to overwhelm and hide neighboring low-amplitude signals in spectral analysis, and they complicate the design of digital filters. As we'll see in [Chapter 5](#), the unwanted ripple in the passband and the poor stopband attenuation in simple digital filters are caused by the rectangular function's DFT sidelobes. (The development, analysis, and application of window functions came about to minimize the ill effects of those sidelobes in [Figure 3-25](#).)

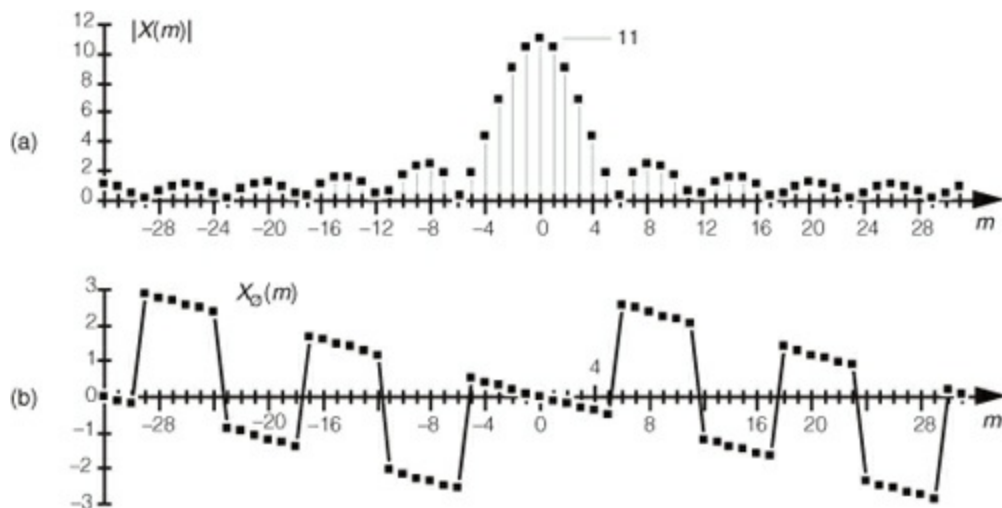
Let's demonstrate the relationship in [Eq. \(3-45\)](#) by way of a simple but concrete example. Assume that we're taking a 64-point DFT of the 64-sample rectangular function, with 11 unity values, shown in [Figure 3-26\(a\)](#). In this example, $N = 64$ and $K = 11$. Taking the 64-point DFT of the sequence in [Figure 3-26\(a\)](#) results in an $X(m)$ whose real and imaginary parts, $X_{\text{real}}(m)$ and $X_{\text{imag}}(m)$, are shown in [Figures 3-26\(b\)](#) and [3-26\(c\)](#) respectively. [Figure 3-26\(b\)](#) is a good illustration of how the real part of the DFT of a real input sequence has even symmetry, and [Figure 3-26\(c\)](#) confirms that the imaginary part of the DFT of a real input sequence has odd symmetry.

Figure 3-26 DFT of a rectangular function: (a) original function $x(n)$; (b) real part of the DFT of $x(n)$, $X_{\text{real}}(m)$; (c) imaginary part of the DFT of $x(n)$, $X_{\text{imag}}(m)$.



Although $X_{\text{real}}(m)$ and $X_{\text{imag}}(m)$ tell us everything there is to know about the DFT of $x(n)$, it's a bit easier to comprehend the true spectral nature of $X(m)$ by viewing its absolute magnitude. This magnitude, from [Eq. \(3-7\)](#), is provided in [Figure 3-27\(a\)](#) where the main and sidelobes are clearly evident now. As we expected, the peak value of the main lobe is 11 because we had $K = 11$ samples in $x(n)$. The width of the main lobe from [Eq. \(3-45\)](#) is $64/11$, or 5.82. Thus, the first positive-frequency zero-crossing location lies just below the $m = 6$ sample of our discrete $|X(m)|$ represented by the squares in [Figure 3-27\(a\)](#). The phase angles associated with $|X(m)|$, first introduced in [Eqs. \(3-6\)](#) and [\(3-8\)](#), are shown in [Figure 3-27\(b\)](#).

Figure 3-27 DFT of a generalized rectangular function: (a) magnitude $|X(m)|$; (b) phase angle in radians.



To understand the nature of the DFT of rectangular functions more fully, let's discuss a few more examples using less general rectangular functions that are more common in digital signal processing than the $x(n)$ in [Figure 3-24](#).

3.13.2 DFT of a Symmetrical Rectangular Function

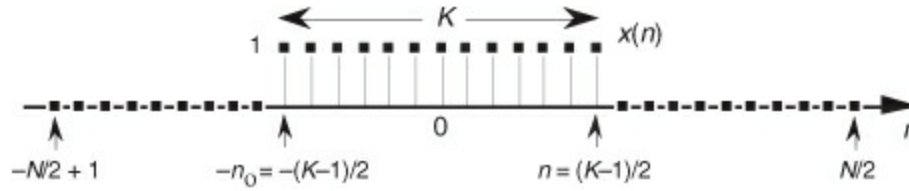
[Equation \(3-43\)](#) is a bit complicated because our original function $x(n)$ was so general. In practice, special cases of rectangular functions lead to simpler versions of [Eq. \(3-43\)](#). Consider the

symmetrical $x(n)$ rectangular function in [Figure 3-28](#). As shown in [Figure 3-28](#), we often need to determine the DFT of a rectangular function that's centered about the $n = 0$ index point. In this case, the K unity-valued samples begin at $n = -n_0 = -(K-1)/2$. So substituting $(K-1)/2$ for n_0 in [Eq. \(3-43\)](#) yields

(3-46)

$$\begin{aligned} X(m) &= e^{j(2\pi m / N)((K-1)/2 - (K-1)/2)} \cdot \frac{\sin(\pi m K / N)}{\sin(\pi m / N)} \\ &= e^{j(2\pi m / N)(0)} \cdot \frac{\sin(\pi m K / N)}{\sin(\pi m / N)}. \end{aligned}$$

Figure 3-28 Rectangular $x(n)$ with K samples centered about $n = 0$.



Because $e^{j0} = 1$, [Eq. \(3-46\)](#) becomes

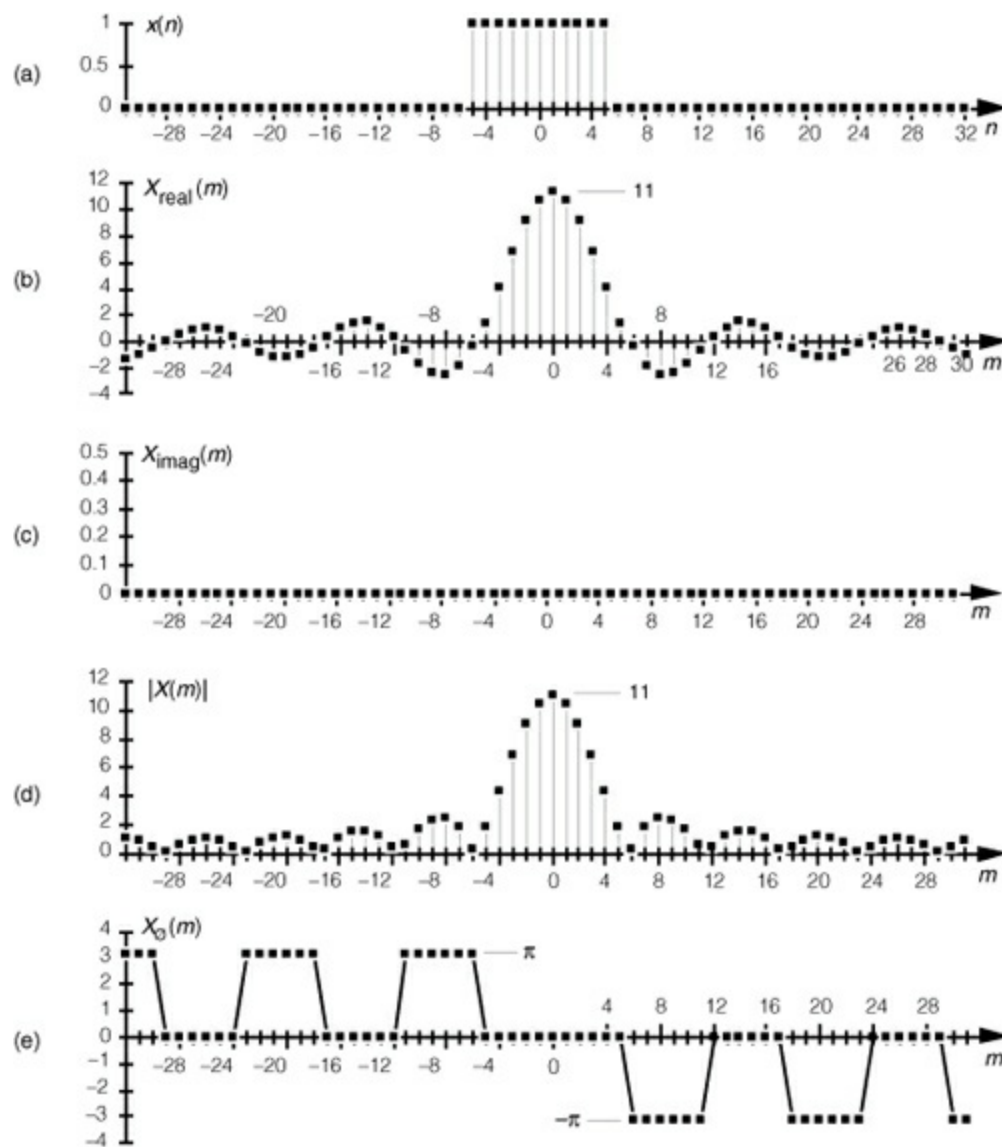
(3-47)

Symmetrical form of the
Dirichlet kernel: \rightarrow

$$X(m) = \frac{\sin(\pi m K / N)}{\sin(\pi m / N)}.$$

[Equation \(3-47\)](#) indicates that the DFT of the symmetrical rectangular function in [Figure 3-28](#) is itself a real function; that is, there's no complex exponential in [Eq. \(3-47\)](#), so this particular DFT contains no imaginary part or phase term. As we stated in [Section 3.2](#), if $x(n)$ is real and even, $x(n) = x(-n)$, then $X_{\text{real}}(m)$ is nonzero and $X_{\text{imag}}(m)$ is always zero. We demonstrate this by taking the 64-point DFT of the sequence in [Figure 3-29\(a\)](#). Our $x(n)$ is 11 unity-valued samples centered about the $n = 0$ index. Here the DFT results in an $X(m)$ whose real and imaginary parts are shown in [Figures 3-29\(b\)](#) and [3-29\(c\)](#), respectively. As [Eq. \(3-47\)](#) predicted, $X_{\text{real}}(m)$ is nonzero and $X_{\text{imag}}(m)$ is zero. The magnitude and phase of $X(m)$ are depicted in [Figures 3-29\(d\)](#) and [3-29\(e\)](#).

Figure 3-29 DFT of a rectangular function centered about $n = 0$: (a) original $x(n)$; (b) $X_{\text{real}}(m)$; (c) $X_{\text{imag}}(m)$; (d) magnitude of $X(m)$; (e) phase angle of $X(m)$ in radians.

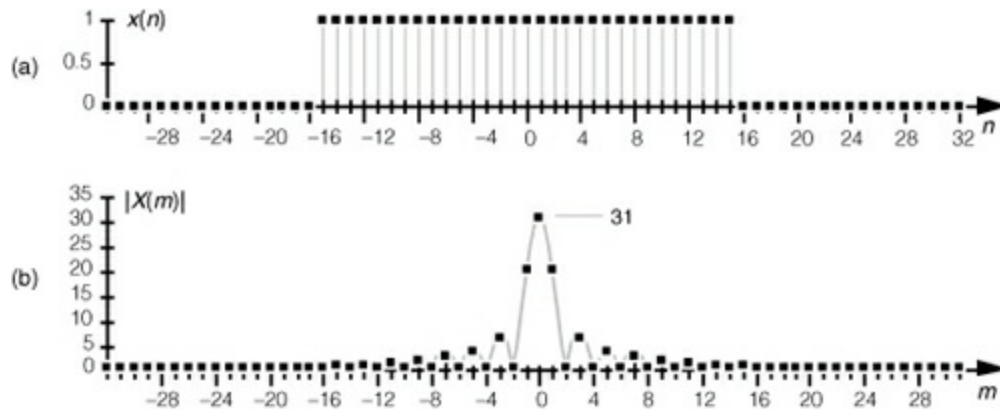


Notice that the magnitudes in [Figures 3-27\(a\)](#) and [3-29\(d\)](#) are identical. This verifies the very important shifting theorem of the DFT; that is, the magnitude $|X(m)|$ depends only on the number of nonzero samples in $x(n)$, K , and *not* on their position relative to the $n = 0$ index value. Shifting the K unity-valued samples to center them about the $n = 0$ index merely affects the phase angle of $X(m)$, not its magnitude.

Speaking of phase angles, it's interesting to realize here that even though $X_{\text{imag}}(m)$ is zero in [Figure 3-29\(c\)](#), the phase angle of $X(m)$ is not always zero. In this case, $X(m)$'s individual phase angles in [Figure 3-29\(e\)](#) are either $+\pi$, zero, or $-\pi$ radians. With $e^{j\pi}$ and $e^{j(-\pi)}$ both being equal to -1 , we could easily reconstruct $X_{\text{real}}(m)$ from $|X(m)|$ and the phase angle $X_{\phi}(m)$ if we must. $X_{\text{real}}(m)$ is equal to $|X(m)|$ with the signs of $|X(m)|$'s alternate sidelobes reversed.[‡] To gain some further appreciation of how the DFT of a rectangular function is a sampled version of the Dirichlet kernel, let's increase the number of our nonzero $x(n)$ samples. [Figure 3-30\(a\)](#) shows a 64-point $x(n)$ where 31 unity-valued samples are centered about the $n = 0$ index location. The magnitude of $X(m)$ is provided in [Figure 3-30\(b\)](#). By broadening the $x(n)$ function, i.e., increasing K , we've narrowed the Dirichlet kernel of $X(m)$. This follows from [Eq. \(3-45\)](#), right? The kernel's first zero crossing is inversely proportional to K , so, as we extend the width of K , we squeeze $|X(m)|$ in toward $m = 0$. In this example, $N = 64$ and $K = 31$. From [Eq. \(3-45\)](#) the first positive zero crossing of $X(m)$ occurs at $64/31$, or just slightly to the right of the $m = 2$ sample in [Figure 3-30\(b\)](#). Also notice that the peak value of $|X(m)| = K = 31$, as mandated by [Eq. \(3-44\)](#).

† The particular pattern of $+\pi$ and $-\pi$ values in [Figure 3-29\(e\)](#) is an artifact of the software used to generate that figure. A different software package may show a different pattern, but as long as the nonzero phase samples are either $+\pi$ or $-\pi$, the phase results will be correct.

Figure 3-30 DFT of a symmetrical rectangular function with 31 unity values: (a) original $x(n)$; (b) magnitude of $X(m)$.



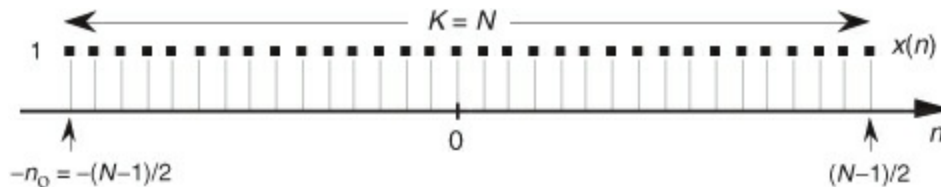
3.13.3 DFT of an All-Ones Rectangular Function

The DFT of a special form of $x(n)$ is routinely called for, leading to yet another simplified form of [Eq. \(3-43\)](#). In the literature, we often encounter a rectangular function where $K = N$; that is, all N samples of $x(n)$ are nonzero, as shown in [Figure 3-31](#). In this case, the N unity-valued samples begin at $n = -n_0 = -(N-1)/2$. We obtain the expression for the DFT of the function in [Figure 3-31](#) by substituting $K = N$ and $n_0 = (N-1)/2$ in [Eq. \(3-43\)](#) to get

$$\begin{aligned} X(m) &= e^{j(2\pi m/N)[(N-1)/2 - (N-1)/2]} \cdot \frac{\sin(\pi m N / N)}{\sin(\pi m / N)} \\ &= e^{j(2\pi m/N)(0)} \cdot \frac{\sin(\pi m)}{\sin(\pi m / N)}, \text{ or} \\ (3-48) \end{aligned}$$

All-ones form of
the Dirichlet
kernel (Type 1): $\rightarrow X(m) = \frac{\sin(\pi m)}{\sin(\pi m / N)}.$

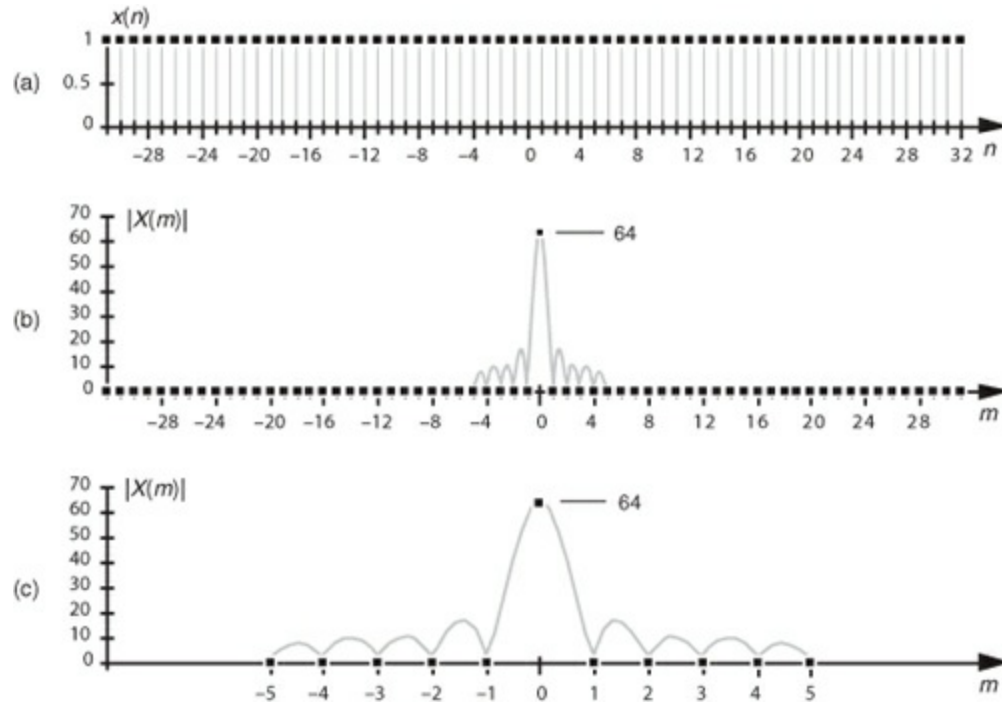
Figure 3-31 Rectangular function with N unity-valued samples.



[Equation \(3-48\)](#) takes the first form of [Eq. \(3-34\)](#) that we alluded to at the beginning of [Section 3.13](#).† [Figure 3-32](#) demonstrates the meaning of [Eq. \(3-48\)](#). The DFT magnitude of the all-ones function, $x(n)$ in [Figure 3-32\(a\)](#), is shown in [Figures 3-32\(b\)](#) and [3-32\(c\)](#). Take note that if m is continuous, [Eq. \(3-48\)](#) describes the shaded curves in [Figure 3-32\(b\)](#) and [Figure 3-32\(c\)](#). If m is restricted to being integers, then [Eq. \(3-48\)](#) represents the dots in those figures.

† By the way, there's nothing *official* about calling [Eq. \(3-48\)](#) a Type 1 Dirichlet kernel. We're using the phrase *Type 1* merely to distinguish [Eq. \(3-48\)](#) from other mathematical expressions for the Dirichlet kernel that we're about to encounter.

Figure 3-32 All-ones function: (a) rectangular function with $N = 64$ unity-valued samples; (b) DFT magnitude of the all-ones time function; (c) close-up view of the DFT magnitude of an all-ones time function.



The Dirichlet kernel of $X(m)$ in [Figure 3-32\(b\)](#) is now as narrow as it can get. The main lobe's first positive zero crossing occurs at the $m = 64/64 = 1$ sample in [Figure 3-32\(b\)](#) and the peak value of $|X(m)| = N = 64$. With $x(n)$ being all ones, $|X(m)|$ is zero for all $m \neq 0$. The sinc function in [Eq. \(3-48\)](#) is of utmost importance—as we'll see at the end of this chapter, it defines the overall DFT frequency response to an input sinusoidal sequence, and it's also the amplitude response of a single DFT bin.

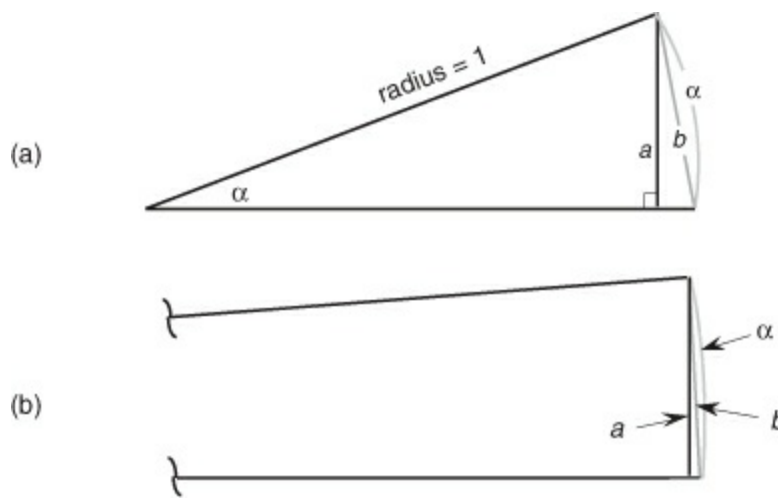
The form of [Eq. \(3-48\)](#) allows us to go one step further to identify the most common expression for the DFT of an all-ones rectangular function found in the literature. To do this, we have to use an approximation principle found in the mathematics of trigonometry that you may have heard before. It states that when α is small, then $\sin(\alpha)$ is approximately equal to α , i.e., $\sin(\alpha) \approx \alpha$. This idea comes about when we consider a pie-shaped section of a circle whose radius is 1 as shown in [Figure 3-33\(a\)](#). That section is defined by the length of the arc α measured in radians and α 's chord b . If we draw a right triangle inside the section, we can say that $a = \sin(\alpha)$. As α gets smaller, the long sides of our triangle become almost parallel, the length of chord b approaches the length of arc α , and the length of line a approaches the length of b . So, as depicted in [Figure 3-33\(b\)](#), when α is small, $\alpha \approx b \approx a = \sin(\alpha)$. We use this $\sin(\alpha) \approx \alpha$ approximation when we look at the denominator of [Eq. \(3-48\)](#). When $\pi m/N$ is small, then $\sin(\pi m/N)$ is approximately equal to $\pi m/N$. So we can, when N is large, state

(3-49)

All-ones form of the Dirichlet
kernel (Type 2): \rightarrow

$$X(m) \approx \frac{\sin(\pi m)}{\pi m / N} = N \cdot \frac{\sin(\pi m)}{\pi m}.$$

Figure 3-33 Relationships between an angle α , line $a = \sin(\alpha)$, and α 's chord b : (a) large angle α ; (b) small angle α .



It has been shown that when N is larger than, say, 10 in [Eq. \(3-48\)](#), [Eq. \(3-49\)](#) accurately describes the DFT's output.[‡] [Equation \(3-49\)](#) is often normalized by dividing it by N , so we can express the normalized DFT of an all-ones rectangular function as

[‡] We can be comfortable with this result because, if we let $K = N$, we'll see that the peak value of $X(m)$ in [Eq. \(3-49\)](#), for $m = 0$, is equal to N , which agrees with [Eq. \(3-44\)](#).

(3-50)

All-ones form of the
Dirichlet kernel (Type 3): \rightarrow

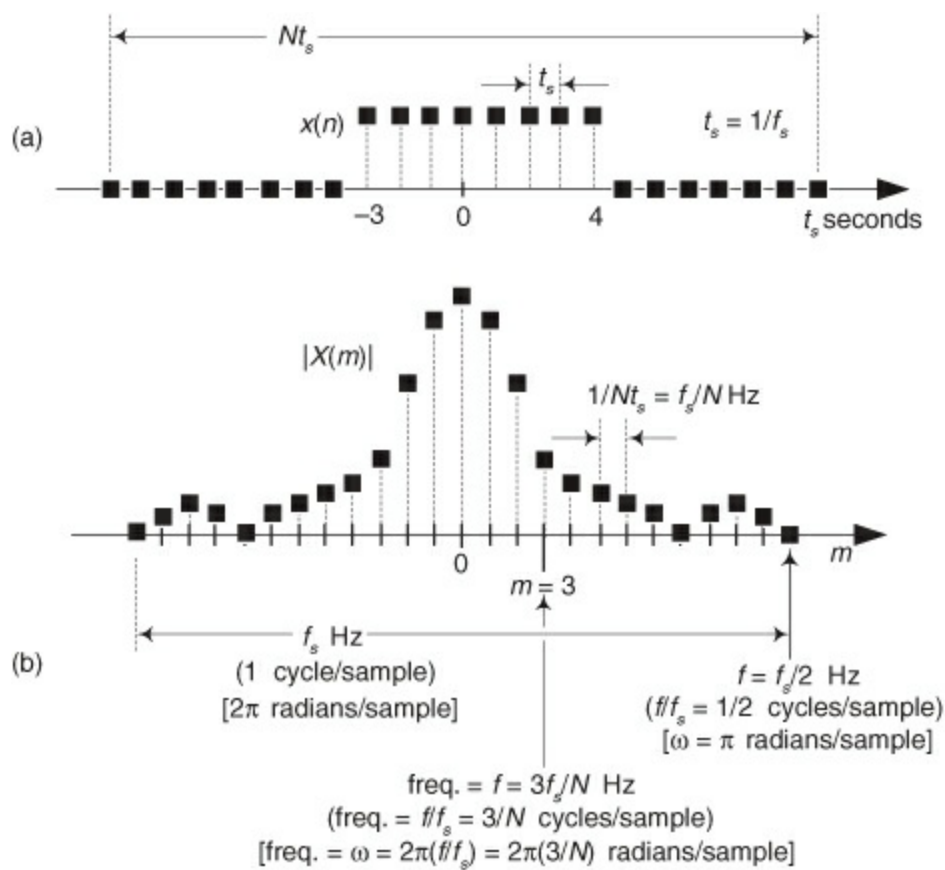
$$X(m) \approx \frac{\sin(\pi m)}{\pi m}.$$

[Equation \(3-50\)](#), taking the second form of [Eq. \(3-34\)](#) that is so often seen in the literature, also has the DFT magnitude shown in [Figures 3-32\(b\)](#) and [3-32\(c\)](#).

3.13.4 Time and Frequency Axes Associated with the DFT

Let's establish the physical dimensions associated with the n and m index values. So far in our discussion, the n index was merely an integer enabling us to keep track of individual $x(n)$ sample values. If the n index represents instants in time, we can identify the time period separating adjacent $x(n)$ samples to establish the time scale for the $x(n)$ axis and the frequency scale for the $X(m)$ axis. Consider the time-domain rectangular function given in [Figure 3-34\(a\)](#). That function comprises N time samples obtained t_s seconds apart, and the full sample interval is Nt_s seconds. Each $x(n)$ sample occurs at nt_s seconds for some value of n . For example, the $n = 9$ sample value, $x(9) = 0$, occurs at $9t_s$ seconds.

Figure 3-34 DFT time and frequency axis dimensions: (a) time-domain axis uses time index n ; (b) various representations of the DFT's frequency axis.



The frequency axis of $X(m)$ can be represented in a number of different ways. Three popular types of DFT frequency axis labeling are shown in [Figure 3-34\(b\)](#) and listed in [Table 3-1](#). Let's consider each representation individually.

Table 3-1 Characteristics of Various DFT Frequency Axis Representations

DFT frequency axis representation	Frequency variable	Resolution of $X(m)$	Repetition interval of $X(m)$	Frequency axis range
Frequency in Hz	f	f_s/N	f_s	$-f_s/2$ to $f_s/2$
Frequency in cycles/sample	f/f_s	$1/N$	1	$-1/2$ to $1/2$
Frequency in radians/sample	ω	$2\pi/N$	2π	$-\pi$ to π

3.13.4.1 DFT Frequency Axis in Hz

If we decide to relate the frequencies of $X(m)$ to the time sample period t_s , or the sample rate $f_s = 1/t_s$, then the frequency axis variable is $f = m/Nt_s = mf_s/N$ Hz. So each $X(m)$ DFT sample is associated with a frequency of mf_s/N Hz. In this case, the sample spacing of $X(m)$ is f_s/N Hz. The DFT repetition period, or periodicity, is f_s Hz as shown in [Figure 3-34\(b\)](#). The first row of [Table 3-1](#) illustrates the characteristics of labeling the frequency axis in Hz.

3.13.4.2 DFT Frequency Axis Normalized by f_s

If we think of some frequency f , in Hz, we can divide that frequency by the sampling frequency f_s to create a normalized frequency variable f/f_s . The dimensions of such a normalized frequency are cycles/sample. Using this notation, each $X(m)$ DFT sample is associated with a normalized frequency of m/N cycles/sample, and our highest frequency are $1/2$ cycles/sample as shown in [Figure 3-34\(b\)](#).

In this scenario the sample spacing of $X(m)$ is $1/N$ cycles/sample, and the DFT repetition period is one cycle/sample as shown by the expressions in parentheses in [Figure 3-34\(b\)](#). This normalized f/f_s frequency variable only has meaning in sampled-data systems. That is, this type of frequency notation has no meaning in the world of continuous (analog) systems.

It may seem strange to use such a normalized f/f_s frequency variable, but sometimes it's convenient for us to do so. Furthermore, the built-in plotting functions of MATLAB (a popular signal processing software package) often label the frequency axis in terms of the normalized f/f_s variable.

3.13.4.3 DFT Frequency Axis Using a Normalized Angle

We can multiply the above normalized f/f_s frequency variable by 2π to create a normalized angular notation representing frequency. Doing so would result in a frequency variable expressed as $\omega = 2\pi(f/f_s)$ radians/sample. Using this notation, each $X(m)$ DFT sample is associated with a normalized frequency of $2\pi m/N$ radians/sample, and our highest frequency is π radians/sample as shown in [Figure 3-34\(b\)](#). In this scenario the sample spacing of $X(m)$ is $2\pi/N$ radians/sample, and the DFT repetition period is one radian/sample as shown by the expressions in brackets in [Figure 3-34\(b\)](#). Using the normalized angular ω frequency variable is very popular in the literature of DSP, and its characteristics are described in the last row of [Table 3-1](#).

Unfortunately having three different representations of the DFT's frequency axis may initially seem a bit puzzling to a DSP beginner, but don't worry. You'll soon become fluent in all three frequency notations. When reviewing the literature, the reader can learn to convert between these frequency axis notation schemes by reviewing [Figure 3-34](#) and [Table 3-1](#).

3.13.5 Alternate Form of the DFT of an All-Ones Rectangular Function

Using the radians/sample frequency notation for the DFT axis from the bottom row of [Table 3-1](#) leads to another prevalent form of the DFT of the all-ones rectangular function in [Figure 3-31](#). Letting our normalized discrete frequency axis variable be $\omega = 2\pi m/N$, then $\pi m = N\omega/2$. Substituting the term $N\omega/2$ for πm in [Eq. \(3-48\)](#), we obtain

(3-51)

All-ones form of the
Dirichlet kernel (Type 4): \rightarrow

$$\sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}$$

[Equation \(3-51\)](#), taking the third form of [Eq. \(3-34\)](#) sometimes seen in the literature, also has the DFT magnitude shown in [Figures 3-32\(b\)](#) and [3-32\(c\)](#).

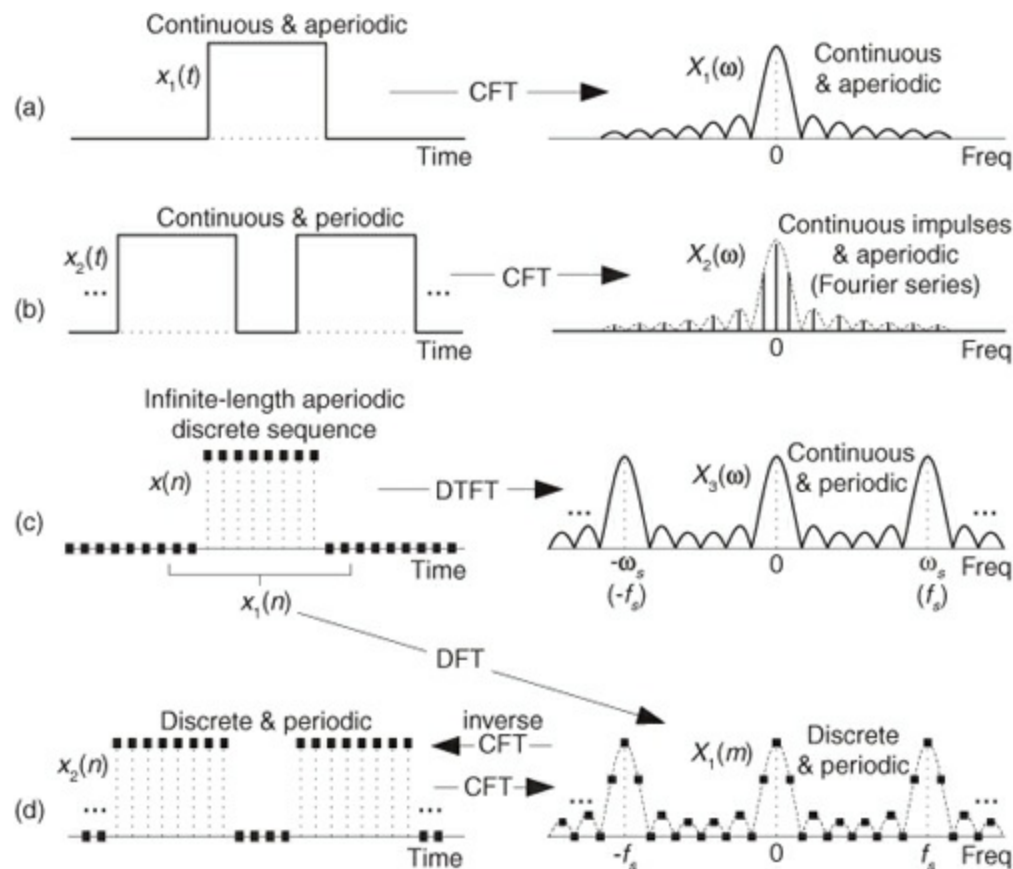
3.14 Interpreting the DFT Using the Discrete-Time Fourier Transform

Now that we've learned about the DFT, it's appropriate to ensure we understand what the DFT actually represents and avoid a common misconception regarding its behavior. In the literature of DSP you'll encounter the topics of *continuous Fourier transform*, *Fourier series*, *discrete-time Fourier transform*, *discrete Fourier transform*, and *periodic spectra*. It takes effort to keep all those notions clear in your mind, especially when you read or hear someone say something like "the DFT assumes its input sequence is periodic in time." (You wonder how this can be true because it's easy to take the DFT of an aperiodic time sequence.) That remark is misleading at best because DFTs don't make assumptions. What follows is how I keep the time and frequency periodicity nature of discrete

sequences straight in my mind.

Consider an infinite-length continuous-time signal containing a single finite-width pulse shown in [Figure 3-35\(a\)](#). The magnitude of its continuous Fourier transform (CFT) is the continuous frequency-domain function $X_1(\omega)$. If the single pulse can be described algebraically (with an equation), then the CFT function $X_1(\omega)$, also an equation, can be found using Fourier transform calculus. (Chances are very good that you actually did this as a homework, or test, problem sometime in the past.) The continuous frequency variable ω is radians per second. If the CFT was performed on the infinite-length signal of periodic pulses in [Figure 3-35\(b\)](#), the result would be the line spectra known as the *Fourier series* $X_2(\omega)$. Those spectral lines (impulses) are infinitely narrow and $X_2(\omega)$ is well defined in between those lines, because $X_2(\omega)$ is continuous. (A well-known example of this concept is the CFT of a continuous squarewave, which yields a Fourier series whose frequencies are all the odd multiples of the squarewave's fundamental frequency.)

Figure 3-35 Time-domain signals and sequences, and the magnitudes of their transforms in the frequency domain.



[Figure 3-35\(b\)](#) is an example of a continuous periodic function (in time) having a spectrum that's a series of individual spectral components. You're welcome to think of the $X_2(\omega)$ Fourier series as a sampled version of the continuous spectrum in [Figure 3-35\(a\)](#). The time-frequency relationship between $x_2(t)$ and $X_2(\omega)$ shows how a periodic function in one domain (time) leads to a function in the other domain (frequency) that is a series of discrete samples.

Next, consider the infinite-length discrete time sequence $x(n)$, containing several nonzero samples, in [Figure 3-35\(c\)](#). We can perform a CFT of $x(n)$ describing its spectrum as a continuous frequency-domain function $X_3(\omega)$. This continuous spectrum is called a discrete-time Fourier transform (DTFT) defined by (see page 48 of reference [\[15\]](#))

(3-52)

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}$$

where the ω frequency variable is measured in radians/sample.

To illustrate the notion of the DTFT, let's say we had a time sequence defined as $x_o(n) = (0.75)^n$ for $n \geq 0$. Its DTFT would be

(3-53)

$$X_o(\omega) = \sum_{n=0}^{\infty} 0.75^n e^{-j\omega n} = \sum_{n=0}^{\infty} (0.75e^{-j\omega})^n.$$

[Equation \(3-53\)](#) is a geometric series (see [Appendix B](#)) and can be evaluated as

(3-53')

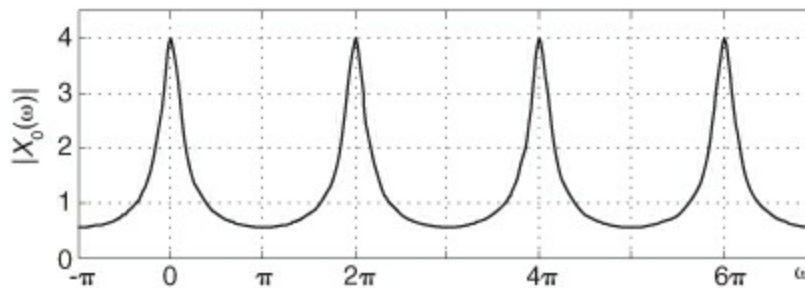
$$X_o(\omega) = \frac{1}{1 - 0.75e^{-j\omega}} = \frac{e^{j\omega}}{e^{j\omega} - 0.75}.$$

$X_o(\omega)$ is continuous and periodic with a period of 2π , whose magnitude is shown in [Figure 3-36](#). This is an example of a sampled (or discrete) time-domain sequence having a periodic spectrum. For the curious reader, we can verify the 2π periodicity of the DTFT using an integer k in the following

(3-54)

$$\begin{aligned} X(\omega + 2\pi k) &= \sum_{n=-\infty}^{\infty} x(n)e^{-j(\omega + 2\pi k)n} = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} e^{-j2\pi kn} = \\ &\sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} = X(\omega) \end{aligned}$$

Figure 3-36 DTFT magnitude $|X_o(\omega)|$.



because $e^{-j2\pi kn} = 1$ for integer values of k .

$X_3(\omega)$ in [Figure 3-35\(c\)](#) also has a 2π periodicity represented by $\omega_s = 2\pi f_s$, where the frequency f_s is the reciprocal of the time period between the $x(n)$ samples. The continuous periodic spectral function $X_3(\omega)$ is what we'd like to be able to compute in our world of DSP, but we can't. We're using computers and, sadly, we can't perform continuous signal analysis with the discrete (binary number) nature of computers. All of our processing comprises discrete numbers stored in our computer's memory and, as such, all of our time-domain signals and all of our frequency-domain spectra are discrete sampled sequences. Consequently the CFT, or inverse CFT, of the sequences with which we

work will all be periodic.

The transforms indicated in [Figures 3-35\(a\)](#) through [3-35\(c\)](#) are pencil-and-paper mathematics of calculus. In a computer, using only finite-length discrete sequences, we can only approximate the CFT (the DTFT) of the infinite-length $x(n)$ time sequence in [Figure 3-35\(c\)](#). That approximation is called the *discrete Fourier transform* (DFT), and it's the only DSP Fourier transform tool we have available to us. Taking the DFT of $x_1(n)$, where $x_1(n)$ is a finite-length portion of $x(n)$, we obtain the discrete periodic $X_1(m)$ spectral samples in [Figure 3-35\(d\)](#).

Notice how $X_1(m)$ is a sampled version of the continuous periodic $X_3(\omega)$. That sampling is represented by

(3-55)

$$X_1(m) = X_3(\omega) \big|_{\omega=2\pi m/N} = \sum_{n=0}^{N-1} x_1(n) e^{-j2\pi mn/N}.$$

We interpret [Eq. \(3-55\)](#) as follows: $X_3(\omega)$ is the continuous DTFT of the N -sample time sequence $x_1(n)$. We can evaluate $X_3(\omega)$ at the N frequencies of $\omega = 2\pi m/N$, where integer m is $0 \leq m \leq N-1$, covering a full period of $X_3(\omega)$. The result of those N evaluated values is a sequence equal to the $X_1(m)$ DFT of $x_1(n)$.

However, and here's the crucial point, $X_1(m)$ is *also* exactly equal to the CFT of the periodic time sequence $x_2(n)$ in [Figure 3-35\(d\)](#). So when people say "the DFT assumes its input sequence is periodic in time," what they really mean is *the DFT is equal to the continuous Fourier transform (the DTFT) of a periodic time-domain discrete sequence*. After all this rigmarole, the end of the story is this: if a function is periodic, its forward/inverse DTFT will be discrete; if a function is discrete, its forward/inverse DTFT will be periodic.

In concluding this discussion of the DTFT, we mention that in the literature of DSP the reader may encounter the following expression

(3-56)

$$X(F) = \sum_{n=-\infty}^{+\infty} x(n) e^{-j2\pi nF}$$

as an alternate definition of the DTFT. [Eq. \(3-56\)](#) can be used to evaluate a full period of the DTFT of an $x(n)$ sequence by letting the frequency variable F , whose dimensions are cycles/sample, be in either of the ranges of $0 \leq F \leq 1$ or $-0.5 \leq F \leq 0.5$.

References

- [1] Bracewell, R. "The Fourier Transform," *Scientific American*, June 1989.
- [2] Struik, D. *A Concise History of Mathematics*, Dover Publications, New York, 1967, p. 142.
- [3] Williams, C. S. *Designing Digital Filters*, Prentice Hall, Englewood Cliffs, New Jersey, 1986, [Section 8.6](#), p. 122.
- [4] Press, W., et al. *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, New York, 1989, p. 426.
- [5] Geckinli, N. C., and Yavuz, D. "Some Novel Windows and a Concise Tutorial Comparison of

Window Families,” *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-26, No. 6, December 1978. (By the way, on page 505 of this paper, the phrase “such that $W(f) \geq 0 \quad \forall f$ ” indicates that $W(f)$ is never negative. The symbol \forall means “for all.”)

- [6] O’Donnell, J. “Looking Through the Right Window Improves Spectral Analysis,” *EDN*, November 1984.
- [7] Kaiser, J. F. “Digital Filters,” in *System Analysis by Digital Computer*, ed. by F. F. Kuo and J. F. Kaiser, John Wiley and Sons, New York, 1966, pp. 218–277.
- [8] Rabiner, L. R., and Gold, B. *The Theory and Application of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1975, p. 88.
- [9] Schoenwald, J. “The Surface Acoustic Wave Filter: Window Functions,” *RF Design*, March 1986.
- [10] Harris, F. “On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform,” *Proceedings of the IEEE*, Vol. 66, No. 1, January 1978.
- [11] Nuttall, A. H. “Some Windows with Very Good Sidelobe Behavior,” *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-29, No. 1, February 1981.
- [12] Yanagimoto, Y. “Receiver Design for a Combined RF Network and Spectrum Analyzer,” *Hewlett-Packard Journal*, October 1993.
- [13] Gullemin, E. A. *The Mathematics of Circuit Analysis*, John Wiley and Sons, New York, 1949, p. 511.
- [14] Lanczos, C. *Discourse on Fourier Series*, Hafner Publishing Co., New York, 1966, [Chapter 1](#), pp. 7–47.
- [15] Oppenheim, A., et al. *Discrete-Time Signal Processing*, 2nd ed., Prentice Hall, Upper Saddle River, New Jersey, 1999, pp. 48–51.

Chapter 3 Problems

- 3.1** Let’s assume that we have performed a 20-point DFT on a sequence of real-valued time-domain samples, and we want to send our $X(m)$ DFT results to a colleague using e-mail. What is the absolute minimum number of (complex) frequency-domain sample values we will need to type in our e-mail so that our colleague has complete information regarding our DFT results?
- 3.2** Assume a systems engineer directs you to start designing a system that performs spectrum analysis using DFTs. The systems engineer states that the spectrum analysis system’s input data sample rate, f_s , is 1000 Hz and specifies that the DFT’s frequency-domain sample spacing must be exactly 45 Hz.
 - (a) What is the number of necessary input time samples, N , for a single DFT operation?
 - (b) What do you tell the systems engineer regarding the spectrum analysis system’s specifications?
- 3.3** We want to compute an N -point DFT of a one-second-duration compact disc (CD) audio signal $x(n)$, whose sample rate is $f_s = 44.1$ kHz, with a DFT sample spacing of 1 Hz.
 - (a) What is the number of necessary $x(n)$ time samples, N ?
 - (b) What is the time duration of the $x(n)$ sequence measured in seconds?

Hint: This Part (b) of the problem is trickier than it first appears. Think carefully.

3.4 Assume we have a discrete $x(n)$ time-domain sequence of samples obtained from *lowpass sampling* of an analog signal, $x(t)$. If $x(n)$ contains $N = 500$ samples, and it was obtained at a sample rate of $f_s = 3000$ Hz:

- (a) What is the frequency spacing of $x(n)$'s DFT samples, $X(m)$, measured in Hz?
- (b) What is the highest-frequency spectral component that can be present in the analog $x(t)$ signal where no aliasing errors occur in $x(n)$?
- (c) If you drew the full $X(m)$ spectrum and several of its spectral replications, what is the spacing between the spectral replications measured in Hz?

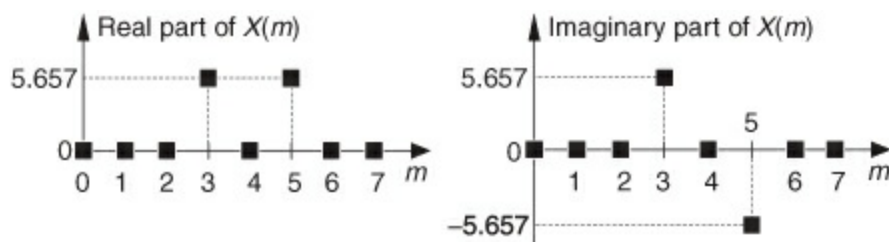
3.5 What are the magnitudes of the 8-point DFT samples of

- (a) the $x_1(n) = 9, 9, 9, 9, 9, 9, 9, 9$ sequence (explain how you arrived at your solution)?
- (b) the $x_2(n) = 1, 0, 0, 0, 0, 0, 0, 0$ sequence?
- (c) the $x_3(n) = 0, 1, 0, 0, 0, 0, 0, 0$ sequence?

Because the $x_3(n)$ sequence in Part (c) is merely a time-shifted version of the $x_2(n)$ sequence in Part (b), comment on the relationship of the $|X_2(m)|$ and $|X_3(m)|$ DFT samples.

3.6 Consider sampling exactly three cycles of a continuous $x(t)$ sinusoid resulting in an 8-point $x(n)$ time sequence whose 8-point DFT is the $X(m)$ shown in [Figure P3-6](#). If the sample rate used to obtain $x(n)$ was 4000 Hz, write the time-domain equation for the discrete $x(n)$ sinusoid in trigonometric form. Show how you arrived at your answer.

Figure P3-6



3.7 In the text's [Section 3.1](#) we discussed the computations necessary to compute the $X(0)$ sample of an N -point DFT. That $X(0)$ output sample represents the zero Hz (DC) spectral component of an $x(n)$ input sequence. Because it is the DC component, $X(0)$ is real-only and we're free to say that an $X(0)$ sample *always* has zero phase. With that said, here are two interesting DFT problems:

- (a) Given that an N -point DFT's input sequence $x(n)$ is real-only, and N is an even number, is there any value for m (other than $m = 0$) for which an $X(m)$ DFT output sample is always real-only?
- (b) Given that N is an odd number, is there any value for m (other than $m = 0$) where an $X(m)$ DFT output sample is always real-only?

3.8 Using the following rectangular form for the DFT equation:

$$X(m) = \sum_{n=0}^{N-1} x(n) \cdot [\cos(2\pi nm / N) - j \sin(2\pi nm / N)]$$

(a) Prove that the $f_s/2$ spectral sample is $X(N/2) = N \cdot \sin(\theta)$ when the $x(n)$ input is a sinusoidal sequence defined by

$$x(n) = \sin[2\pi(f_s/2)nt_s + \theta].$$

N is an even number, frequency f_s is the $x(n)$ sequence's sample rate in Hz, time index $n = 0, 1, 2, \dots, N-1$, and θ is an initial phase angle measured in radians.

Hint: Recall the trigonometric identity $\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)$.

(b) What is $X(N/2)$ when $x(n) = \sin[2\pi(f_s/2)nt_s]$?

(c) What is $X(N/2)$ when $x(n) = \cos[2\pi(f_s/2)nt_s]$?

3.9 To gain some practice in using the algebra of discrete signals and the geometric series identities in [Appendix B](#), and to reinforce our understanding of the output magnitude properties of a DFT when its input is an exact integer number of sinusoidal cycles:

(a) Prove that when a DFT's input is a complex sinusoid of magnitude A_0 (i.e., $x(n) = A_0 e^{j2\pi f n t_s}$) with exactly three cycles over N samples, the output magnitude of the DFT's $m = 3$ bin will be $|X(3)| = A_0 N$.

Hint: The first step is to redefine $x(n)$'s f and t_s variables in terms of a sample rate f_s and N so that $x(n)$ has exactly three cycles over N samples. The redefined $x(n)$ is then applied to the standard DFT equation.

(b) Prove that when a DFT's input is a real-only sinewave of peak amplitude A_0 (i.e., $x(n) = A_0 \sin(2\pi f n t_s)$) with exactly three cycles over N samples, the output magnitude of the DFT's $m = 3$ bin will be $|X(3)| = A_0 N/2$.

Hint: Once you redefine $x(n)$'s f and t_s variables in terms of a sample rate f_s and N so that $x(n)$ has exactly three cycles over N samples, you must convert that real sinewave to complex exponential form so that you can evaluate its DFT for $m = 3$.

The purpose of this problem is to remind us that DFT output magnitudes are proportional to the size, N , of the DFT. That fact is important in a great many DSP analysis activities and applications.

3.10 Consider performing the 5-point DFT on the following $x_1(n)$ time-domain samples

$$x_1(n) = [1, 2.2, -4, 17, 21],$$

and the DFT's first sample is $X_1(0) = 37.2$. Next, consider performing the 5-point DFT on the following $x_2(n)$ time samples

$$x_2(n) = [1, 2.2, -4, 17, Q],$$

and that DFT's first sample is $X_2(0) = 57.2$. What is the value of Q in the $x_2(n)$ time sequence? Justify your answer.

3.11 Derive the equation describing $X(m)$, the N -point DFT of the following $x(n)$ sequence:

$$x(n) = a^n, \quad \text{for } 0 \leq n \leq N-1.$$

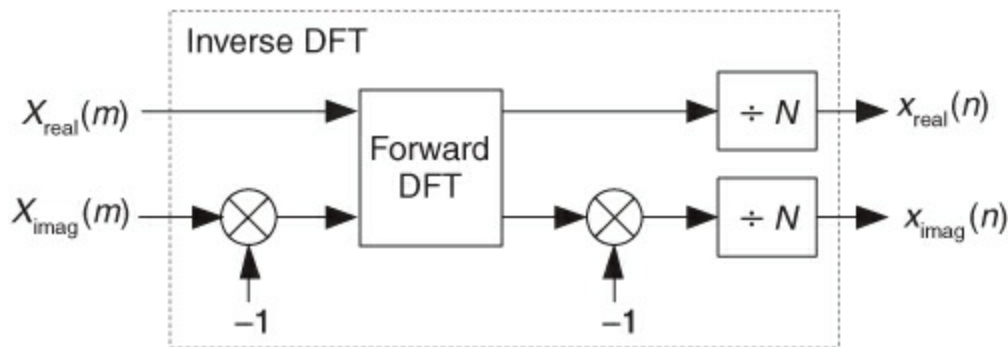
Hint: Recall one of the laws of exponents, $p^b q^{bc} = (pq^c)^b$, and the geometric series identities in [Appendix B](#).

3.12 Consider an N -sample $x(n)$ time sequence whose DFT is represented by $X(m)$, where $0 \leq m \leq N-1$. Given this situation, an Internet website once stated, “The sum of the $X(m)$ samples is equal to N times the first $x(n)$ sample.” Being suspicious of anything we read on the Internet, show whether or not that statement is true.

Hint: Use the inverse DFT process to determine the appropriate $x(n)$ time sample of interest in terms of $X(m)$.

3.13 Here is a problem whose solution may be useful to you in the future. On the Internet you will find information suggesting that an inverse DFT can be computed using a forward DFT software routine in the process shown in [Figure P3-13](#).

Figure P3-13



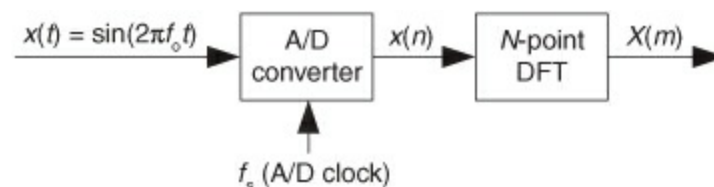
(a) Using the forward and inverse DFT equations, and the material in [Appendix A](#), show why the process in [Figure P3-13](#) computes correct inverse DFTs.

Hint: Begin your solution by writing the inverse DFT equation and conjugating both sides of that equation.

(b) Comment on how the process in [Figure P3-13](#) changes if the original frequency-domain $X(m)$ sequence is conjugate symmetric.

3.14 One useful way to test the performance of commercial analog-to-digital (A/D) converters is to digitize an f_0 Hz analog sinewave, apply the N -sample $x(n)$ sequence to a DFT, and examine the DFT's $X(m)$ results. The process is depicted in [Figure P3-14](#). An ideal (A/D) converter will produce $X(m)$ results showing spectral energy at f_0 Hz and no spectral energy at any other frequency. As such, nonzero spectral energy in $X(m)$ at frequencies other than f_0 Hz indicates real-world A/D converter performance. However, the DFT's inherent property of leakage “smears” spectral energy over multiple $X(m)$ samples, as was shown in the text's [Figure 3-8\(b\)](#), which degrades the effectiveness of this A/D converter test method. What can we do to minimize the DFT's inherent spectral leakage as much as possible for this type of converter testing?

Figure P3-14

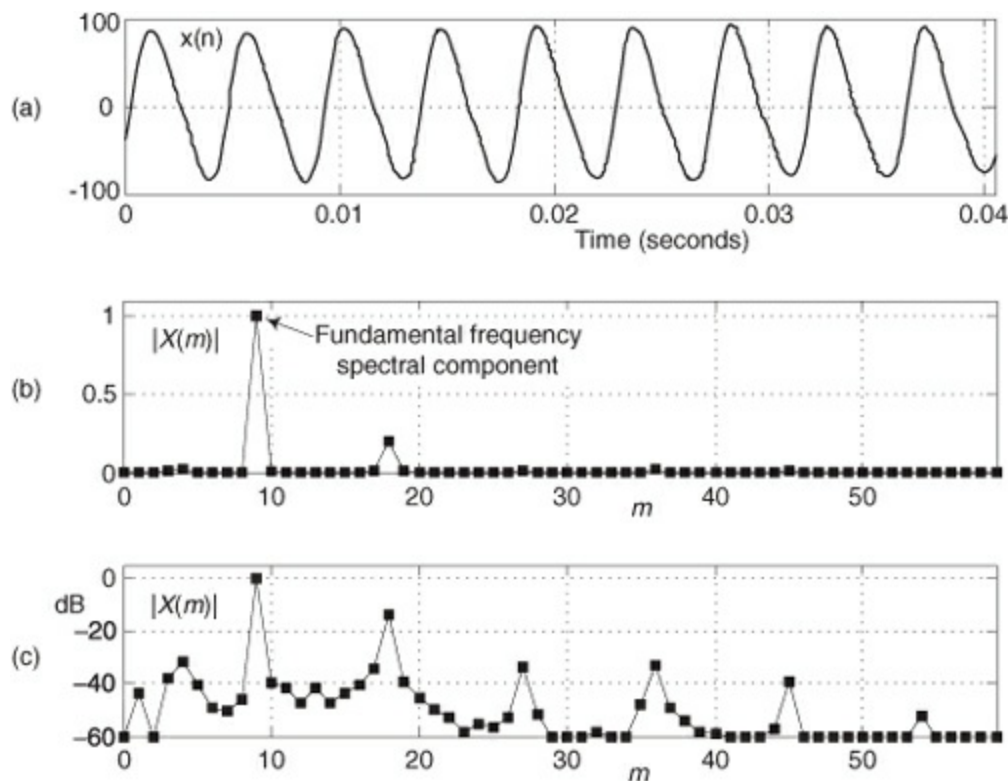


3.15 Here is a real-world spectrum analysis problem. [Figure P3-15\(a\)](#) shows 902 samples of an $x(n)$ time sequence. (For clarity, we do not show the $x(n)$ samples as individual dots.) That sequence is the sound of the “A3” note (“A” below middle “C”) from an acoustic guitar, sampled at $f_s = 22.255$ kHz. [Figure P3-15\(b\)](#) shows the $X(m)$ spectral magnitude samples, the DFT of $x(n)$, on a linear scale for the frequency index range of $0 \leq m \leq 59$.

(a) Based on the $X(m)$ samples, what is the *fundamental* frequency, in Hz, of the guitar’s “A3” note?

(b) When we plot the DFT magnitude samples on a logarithmic scale, as in [Figure P3-15\(c\)](#), we see spectral *harmonics* and learn that the guitar note is rich in spectral content. (The harmonics are integer multiples of the fundamental frequency.) That’s why guitars have their pleasing sound, depending on the guitarist’s skill, of course. What is the frequency of the highest nonzero spectral component of the guitar’s “A3” note?

Figure P3-15



3.16 [Figure P3-16\(a\)](#) shows a 16-point Hanning window sequence, $h_1(n)$, defined by

$$h_1(n) = 0.5 - 0.5 \cos\left(\frac{2\pi n}{16}\right), \text{ for } n = 0, 1, 2, \dots, 15.$$

The magnitude of its DFT samples, $|H_1(m)|$, is shown in [Figure P3-16\(b\)](#). (For simplicity, we show only the positive-frequency range of the $|H_1(m)|$ samples.) Notice that only the $|H_1(0)|$ and the $|H_1(1)|$ frequency-domain samples are nonzero.

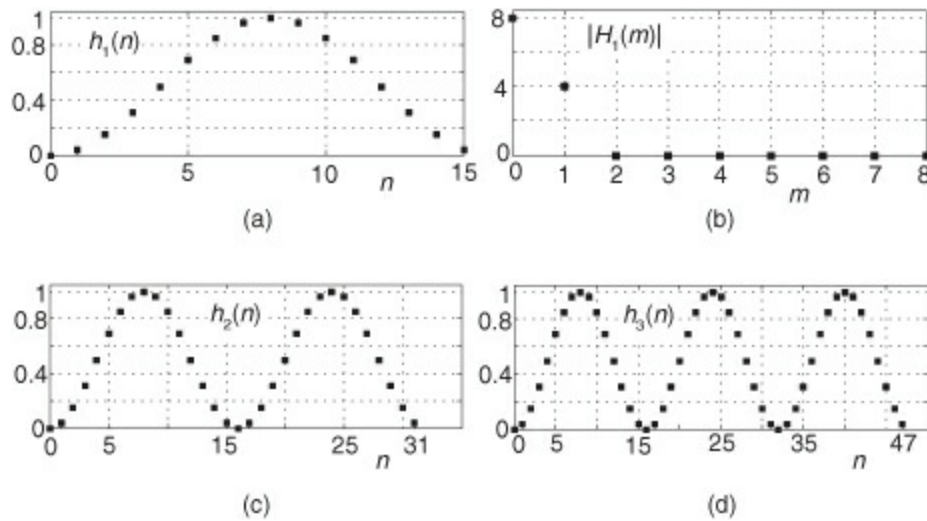
(a) Sequence $h_1(n)$ comprises two signals. Looking carefully at $h_1(n)$, describe what those two signals are and justify why $|H_1(m)|$ looks the way it does.

(b) Given your understanding of the relationship between $h_1(n)$ and $|H_1(m)|$, look at $h_2(n)$, in [Figure P3-16\(c\)](#), which is two repetitions of the original $h_1(n)$ sequence. Draw a rough sketch of the

spectral magnitude sequence $|H_2(m)|$ over its positive-frequency range.

- (c) Given that the $h_3(n)$ in [Figure P3-16\(d\)](#) is three repetitions of the original $h_1(n)$ sequence, draw the spectral magnitude sequence $|H_3(m)|$ over its positive-frequency range.

Figure P3-16



- (d) Considering the $h_1(n)$, $h_2(n)$, and $h_3(n)$ sequences, and their $|H_1(m)|$, $|H_2(m)|$, and $|H_3(m)|$ spectral magnitude samples, complete the following *important* statement: “ K repetitions of an $h_1(n)$ sequence result in an extended-length time sequence whose spectral magnitudes have $K-1$...”

3.17 In the literature of DSP, you may see an *alternate* expression for an N -point Hanning window defined by

$$w_{\text{han,alt}}(n) = \sin^2\left(\frac{\pi n}{N}\right), \text{ for } n = 0, 1, 2, \dots, N-1.$$

Prove that the above alternate expression for a Hanning window is equivalent to the [Section 3.9](#) text’s definition of a Hanning window.

3.18 Considering the DFT of an N -point $x(n)$ sequence, what is the spectral effect of zero-padding the $x(n)$ sequence to a length of Q samples (with Q being an integer power of two, and $Q > N$) and performing a Q -point DFT on the zero-padded sequence?

3.19 Assume that an N -point DFT, performed on an N -sample $x(n)$ time-domain sequence, results in a DFT frequency-domain sample spacing of 100 Hz. What would be the DFT frequency-domain sample spacing in Hz if the N -sample $x(n)$ time sequence was padded with $4N$ zero-valued samples and we performed a DFT on that extended-time sequence?

3.20 There is a program, in the U.S. and other countries, called “Search for Extraterrestrial Intelligence” (SETI). These folk point radio antennas around in the night sky searching for “nonrandom radio” signals, hoping to find evidence of “little green men.” They search for radio-frequency (RF) signal energy that significantly exceeds the background RF noise energy in the sky. Their primary method for detecting low-level RF energy is to tune a narrowband receiver to some RF frequency and collect millions of time-domain samples, and then perform million-point DFTs in the hope of finding spectral magnitude components that significantly exceed the background spectral noise. High-level spectral components would indicate the existence of intelligent life

that's broadcasting radio signals of some sort.

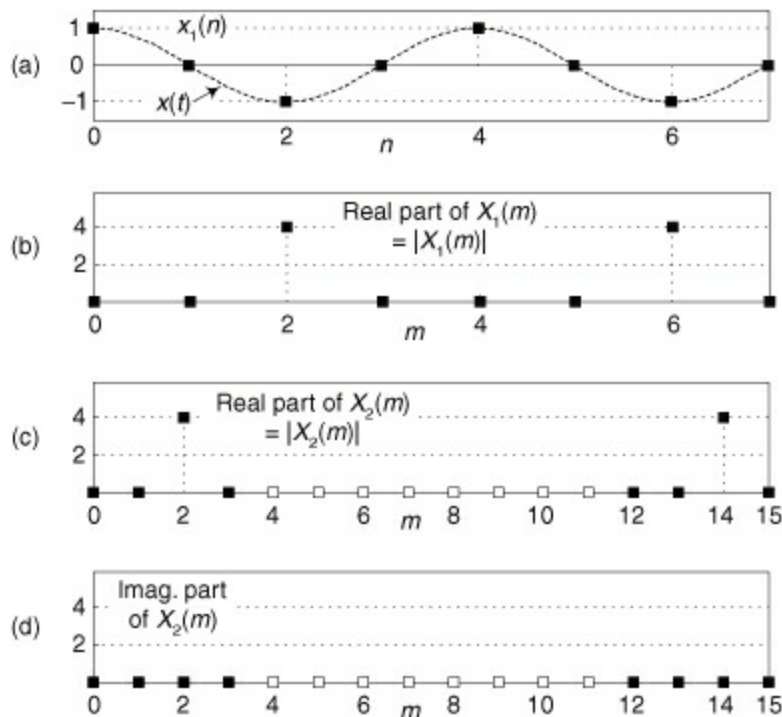
Here's the question: If a SETI researcher collects one million time samples and performs a one-million-point DFT, roughly what DFT processing gain (in dB) improvement can that person expect to achieve in *pulling* a weak spectral component up above the background galactic spectral noise in comparison to using a 100-point DFT?

3.21 This problem tests your understanding of the DFT's frequency-domain axis. Consider sampling exactly two cycles of an analog $x(t)$ cosine wave resulting in the 8-point $x_1(n)$ time sequence in [Figure P3-21\(a\)](#). The real part of the DFT of $x_1(n)$ is the sequence shown in [Figure P3-21\(b\)](#). Because $x_1(n)$ is exactly two cycles of a cosine sequence, the imaginary parts of $X_1(m)$ are all zero-valued samples, making $|X_1(m)|$ equal to the real part of $X_1(m)$. (Note that no leakage is apparent in $|X_1(m)|$.) Think, now, of a new frequency-domain sequence $X_2(m)$ that is equal to $X_1(m)$ with eight zero-valued samples, the white squares in [Figures P3-21\(c\)](#) and [P3-21\(d\)](#), inserted in the center of the real and imaginary parts of $X_1(m)$.

(a) Draw the $x_2(n)$ time sequence that is the inverse DFT of $X_2(m)$.

(b) Comment on how the $x_2(n)$ time sequence is related to the original analog $x(t)$ signal and the $x_1(n)$ sequence.

Figure P3-21



3.22 There is a useful spectrum analysis process, discussed in [Chapter 13](#), that uses the results of an N -point DFT, $X(m)$, and requires us to compute

$$S = P \cdot X(0) - Q \cdot X(N-1) - Q \cdot X(1)$$

where P and Q are scalar constants. Value S is the sum of three complex numbers. If we represent the three DFT samples in rectangular form, we can write

$$S = P \cdot [a + jb] - Q \cdot [c + jd] - Q \cdot [e + jg].$$

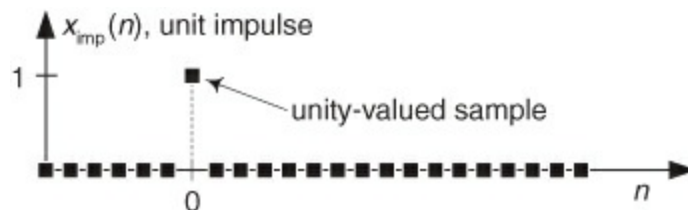
In the general case, the above expression for S requires six real multiply operations. If the DFT's

$x(n)$ input sequence is real-only, what is the equation for S that requires fewer than six real multiplies? Show your work.

3.23 For an N -length time-domain sequence $x(n)$, why is the DFT useful in plotting $x(n)$'s discrete-time Fourier transform (DTFT) which is a function of the continuous frequency variable ω ?

3.24 In [Chapter 1](#) we mentioned a special time-domain sequence called a *unit impulse*. We'll be using that sequence, the $x_{\text{imp}}(n)$ shown in [Figure P3-24](#), in later chapters to test digital filters. As such, it's useful to know the spectral content of this unit impulse.

Figure P3-24

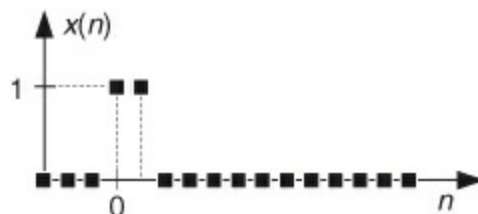


(a) Draw the continuous $X_{\text{imp}}(\omega)$ discrete-time Fourier transform (DTFT), over the frequency range of $0 \leq \omega \leq 2\pi$, of the $x_{\text{imp}}(n)$ unit impulse sequence.

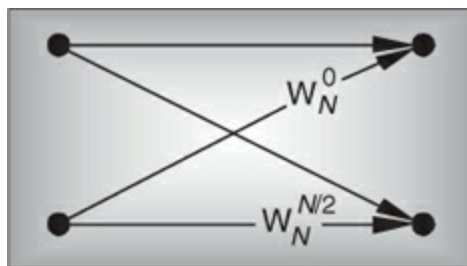
(b) With your $X_{\text{imp}}(\omega)$ solution in mind, assume a person is listening to an AM (amplitude modulation) radio station centered at 640 kHz in the North American AM Broadcast band and a neighbor is listening to an international shortwave AM signal on a radio receiver tuned to 5.2 MHz. Can you explain why, when lightning strikes, both people hear the static noise from the lightning on their radios even though the radios are tuned to very different center frequencies?

3.25 Draw a rough sketch of the magnitude of the discrete-time Fourier transform (DTFT), over the frequency range of $-\pi \leq \omega \leq \pi$, of the $x(n)$ sequence in [Figure P3-25](#).

Figure P3-25



Chapter Four. The Fast Fourier Transform



Although the DFT is the most straightforward mathematical procedure for determining the frequency content of a time-domain sequence, it's terribly inefficient. As the number of points in the DFT is increased to hundreds, or thousands, the amount of necessary number crunching becomes excessive. In 1965 a paper was published by Cooley and Tukey describing a very efficient algorithm to implement the DFT[\[1\]](#). That algorithm is now known as the *fast Fourier transform* (FFT).[‡] Before the advent of the FFT, thousand-point DFTs took so long to perform that their use was restricted to the larger research and university computer centers. Thanks to Cooley, Tukey, and the semiconductor industry, 1024-point DFTs can now be performed in a few seconds on home computers.

[‡] Actually, the FFT has an interesting history. While analyzing X-ray scattering data, a couple of physicists in the 1940s were taking advantage of the symmetries of sines and cosines using a mathematical method based on a technique published in the early 1900s. Remarkably, over 20 years passed before the FFT was (re)discovered. Reference [\[2\]](#) tells the full story.

Volumes have been written about the FFT, and, as for no other innovation, the development of this algorithm transformed the discipline of digital signal processing by making the power of Fourier analysis affordable. In this chapter, we'll show why the most popular FFT algorithm (called the *radix-2* FFT) is superior to the classical DFT algorithm, present a series of recommendations to enhance our use of the FFT in practice, and provide a list of sources for FFT routines in various software languages. We conclude this chapter, for those readers wanting to know the internal details, with a derivation of the radix-2 FFT and introduce several different ways in which this FFT is implemented.

4.1 Relationship of the FFT to the DFT

Although many different FFT algorithms have been developed, in this section we'll see why the radix-2 FFT algorithm is so popular and learn how it's related to the classical DFT algorithm. The radix-2 FFT algorithm is a very efficient process for performing DFTs under the constraint that the DFT size be an integral power of two. (That is, the number of points in the transform is $N = 2^k$, where k is some positive integer.) Let's see just why the radix-2 FFT is the favorite spectral analysis technique used by signal processing practitioners.

Recall that our DFT Example 1 in [Section 3.1](#) illustrated the number of redundant arithmetic operations necessary for a simple 8-point DFT. (For example, we ended up calculating the product of $1.0607 \cdot 0.707$ four separate times.) On the other hand, the radix-2 FFT eliminates these redundancies and greatly reduces the number of necessary arithmetic operations. To appreciate the FFT's efficiency, let's consider the number of complex multiplications necessary for our old friend, the expression for an N -point DFT,

$$(4-1)$$

$$X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi mn/N}.$$

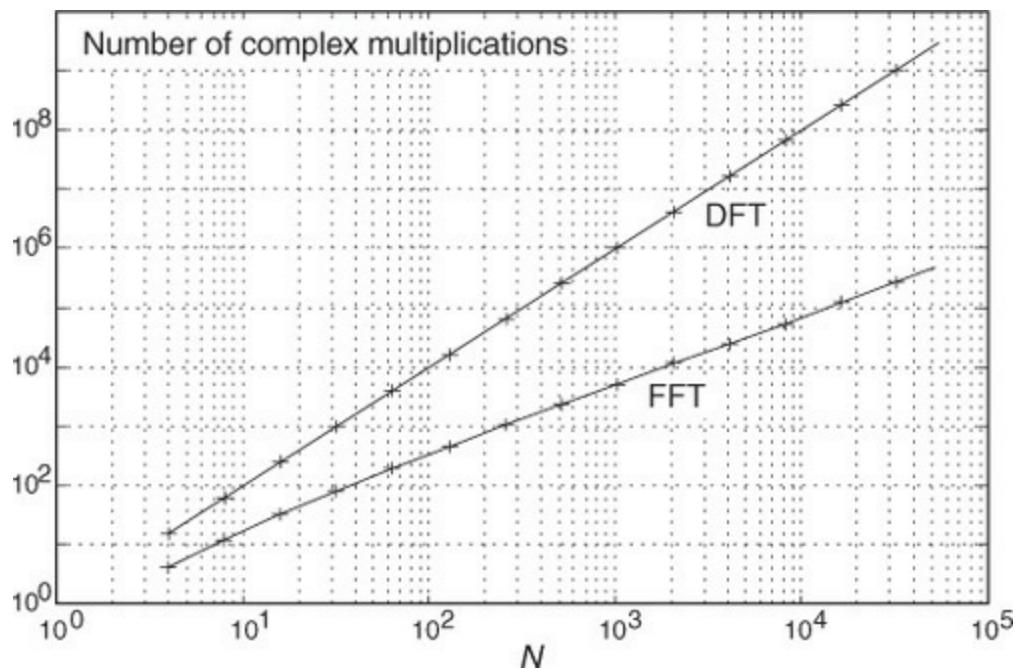
For an 8-point DFT, [Eq. \(4-1\)](#) tells us that we'd have to perform N^2 or 64 complex multiplications. (That's because we assume, in the general case, that $x(n)$ are complex-valued samples and for each of the eight $X(m)$ s we have to sum eight complex products as n goes from 0 to 7.) As we'll verify in later sections of this chapter, the number of complex multiplications, for an N -point FFT, is approximately

$$(4-2)$$

$$\frac{N}{2} \cdot \log_2 N.$$

(We say approximately because some multiplications turn out to be multiplications by $+1$ or -1 , which amount to mere sign changes.) Well, this $(N/2)\log_2 N$ value is a significant reduction from the N^2 complex multiplications required by [Eq. \(4-1\)](#), particularly for large N . To show just how significant, [Figure 4-1](#) compares the number of complex multiplications required by DFTs and radix-2 FFTs as a function of the number of input data points N . When $N = 512$, for example, the DFT requires 114 times the number of complex multiplications than needed by the FFT. When $N = 8192$, the DFT must calculate 1260 complex multiplications for *each* complex multiplication in the FFT!

Figure 4-1 Number of complex multiplications in the DFT and the radix-2 FFT as a function of N .



Here's my favorite example of the efficiency of the radix-2 FFT. Say you perform a two-million-point FFT ($N = 2,097,152$) on your desktop computer and it takes 10 seconds. A two-million-point DFT, on the other hand, using your computer, will take more than three weeks! The publication and dissemination of the radix-2 FFT algorithm was, arguably, the most important event in digital signal processing.

It's appropriate now to make clear that the FFT is not an approximation of the DFT. It's exactly equal to the DFT; it *is* the DFT. Moreover, all of the performance characteristics of the DFT described in the previous chapter, output symmetry, linearity, output magnitudes, leakage, scalloping loss, etc., also describe the behavior of the FFT.

4.2 Hints on Using FFTs in Practice

Based on how useful FFTs are, here's a list of practical pointers, or tips, on acquiring input data samples and using the radix-2 FFT to analyze real-world signals or data.

4.2.1 Sample Fast Enough and Long Enough

When digitizing continuous signals with an A/D converter, for example, we know, from [Chapter 2](#), that our sampling rate must be greater than twice the bandwidth of the continuous A/D input signal to prevent frequency-domain aliasing. Depending on the application, practitioners typically sample at 2.5 to 4 times the signal bandwidth. If we know that the bandwidth of the continuous signal is not too large relative to the maximum sample rate of our A/D converter, it's easy to avoid aliasing. If we don't know the continuous A/D input signal's bandwidth, how do we tell if we're having aliasing problems? Well, we should mistrust any FFT results that have significant spectral components at frequencies near half the sample rate. Ideally, we'd like to work with signals whose spectral amplitudes decrease with increasing frequency. Be very suspicious of aliasing if there are any spectral components whose frequencies appear to depend on the sample rate. If we suspect that aliasing is occurring or that the continuous signal contains broadband noise, we'll have to use an analog lowpass filter prior to A/D conversion. The cutoff frequency of the lowpass filter must, of course, be greater than the frequency band of interest but less than half the sample rate.

Although we know that an N -point radix-2 FFT requires $N = 2^k$ input samples, just how many samples must we collect before we perform our FFT? The answer is that the data collection time interval must be long enough to satisfy our desired FFT frequency resolution for the given sample rate f_s . The data collection time interval is the reciprocal of the desired FFT frequency resolution, and the longer we sample at a fixed f_s sample rate, the finer our frequency resolution will be; that is, the total data collection time interval is N/f_s seconds, and our N -point FFT bin-to-bin (sample-to-sample) frequency resolution is f_s/N Hz. So, for example, if we need a spectral resolution of 5 Hz, then $f_s/N = 5$ Hz, and

(4-3)

$$N = \frac{f_s}{\text{desired resolution}} = \frac{f_s}{5} = 0.2f_s.$$

In this case, if f_s is, say, 10 kHz, then N must be at least 2000, and we'd choose N equal to 2048 because this number is a power of two.

4.2.2 Manipulating the Time Data Prior to Transformation

When using the radix-2 FFT, if we don't have control over the length of our time-domain data sequence, and that sequence length is not an integral power of two, we have two options. We could discard enough data samples so that the remaining FFT input sequence length is some integral power of two. This scheme is not recommended because ignoring data samples degrades our resultant frequency-domain resolution. (The larger N is, the better our frequency resolution, right?) A better approach is to append enough zero-valued samples to the end of the time data sequence to match the number of points of the next largest radix-2 FFT. For example, if we have 1000 time samples to transform, rather than analyzing only 512 of them with a 512-point FFT, we should add 24 trailing zero-valued samples to the original sequence and use a 1024-point FFT. (This *zero-padding*

technique is discussed in more detail in [Section 3.11](#).)

FFTs suffer the same ill effects of spectral leakage that we discussed for the DFT in [Section 3.8](#). We can multiply the time data by a window function to alleviate this leakage problem. Be prepared, though, for the frequency resolution degradation inherent when windows are used. By the way, if appending zeros is necessary to extend a time sequence, we have to make sure that we append the zeros *after* multiplying the original time data sequence by a window function. Applying a window function to the appended zeros will distort the resultant window and worsen our FFT leakage problems.

Although windowing will reduce leakage problems, it will not eliminate them altogether. Even when windowing is employed, high-level spectral components can obscure nearby low-level spectral components. This is especially evident when the original time data has a nonzero average, i.e., it's riding on a DC bias. When the FFT is performed in this case, a large-amplitude DC spectral component at 0 Hz will overshadow its spectral neighbors. We can eliminate this problem by calculating the average of the time sequence and subtracting that average value from each sample in the original sequence. (The averaging and subtraction process must be performed before windowing.) This technique makes the new time sequence's average (mean) value equal to zero and eliminates any high-level, zero Hz component in the FFT results.

4.2.3 Enhancing FFT Results

If we're using the FFT to detect signal energy in the presence of noise and enough time-domain data is available, we can improve the sensitivity of our processing by averaging multiple FFTs. This technique, discussed in [Section 11.3](#), can be implemented to detect signal energy that's actually below the average noise level; that is, given enough time-domain data, we can detect signal components that have negative signal-to-noise ratios.

If our original time-domain data is real-valued only, we can take advantage of the $2N$ -Point Real FFT technique in [Section 13.5](#) to speed up our processing; that is, a $2N$ -point real sequence can be transformed with a single N -point complex radix-2 FFT. Thus we can get the frequency resolution of a $2N$ -point FFT for just about the computational price of performing a standard N -point FFT. Another FFT speed enhancement is the possible use of the frequency-domain windowing technique discussed in [Section 13.3](#). If we need the FFT of unwindowed time-domain data and, at the same time, we also want the FFT of that same time data with a window function applied, we don't have to perform two separate FFTs. We can perform the FFT of the unwindowed data, and then we can perform frequency-domain windowing to reduce spectral leakage on any, or all, of the FFT bin outputs.

4.2.4 Interpreting FFT Results

The first step in interpreting FFT results is to compute the absolute frequency of the individual FFT bin centers. Like the DFT, the FFT bin spacing is the ratio of the sampling rate (f_s) over the number of points in the FFT, or f_s/N . With our FFT output designated by $X(m)$, where $m = 0, 1, 2, 3, \dots, N-1$, the absolute frequency of the m th bin center is mf_s/N . If the FFT's input time samples are real, only the $X(m)$ outputs from $m = 0$ to $m = N/2$ are independent. So, in this case, we need determine only the absolute FFT bin frequencies for m over the range of $0 \leq m \leq N/2$. If the FFT input samples are complex, all N of the FFT outputs are independent, and we should compute the absolute FFT bin frequencies for m over the full range of $0 \leq m \leq N-1$.

If necessary, we can determine the true amplitude of time-domain signals from their FFT spectral results. To do so, we have to keep in mind that radix-2 FFT outputs are complex and of the form

(4-4)

$$X(m) = X_{\text{real}}(m) + jX_{\text{imag}}(m).$$

Also, the FFT output magnitude samples,

(4-5)

$$X_{\text{mag}}(m) = |X(m)| = \sqrt{X_{\text{real}}(m)^2 + X_{\text{imag}}(m)^2},$$

are all inherently multiplied by the factor $N/2$, as described in [Section 3.4](#), when the input samples are real. If the FFT input samples are complex, the scaling factor is N . So to determine the correct amplitudes of the time-domain sinusoidal components, we'd have to divide the FFT magnitudes by the appropriate scale factor, $N/2$ for real inputs and N for complex inputs.

If a window function was used on the original time-domain data, some of the FFT input samples will be attenuated. This reduces the resultant FFT output magnitudes from their true unwindowed values. To calculate the correct amplitudes of various time-domain sinusoidal components, then, we'd have to further divide the FFT magnitudes by the appropriate processing loss factor associated with the window function used. Processing loss factors for the most popular window functions are listed in reference [\[3\]](#).

Should we want to determine the power spectrum $X_{\text{PS}}(m)$ of an FFT result, we'd calculate the magnitude-squared values using

(4-6)

$$X_{\text{PS}}(m) = |X(m)|^2 = X_{\text{real}}(m)^2 + X_{\text{imag}}(m)^2.$$

Doing so would allow us to compute the power spectrum in dB with

(4-7)

$$X_{\text{dB}}(m) = 10 \cdot \log_{10}(|X(m)|^2) \text{ dB}.$$

The normalized power spectrum in decibels can be calculated using

(4-8)

$$\text{normalized } X_{\text{dB}}(m) = 10 \cdot \log_{10}\left(\frac{|X(m)|^2}{(|X(m)|_{\text{max}})^2}\right),$$

or

(4-9)

$$\text{normalized } X_{\text{dB}}(m) = 20 \cdot \log_{10}\left(\frac{|X(m)|}{|X(m)|_{\text{max}}}\right).$$

In [Eqs. \(4-8\)](#) and [\(4-9\)](#), the term $|X(m)|_{\text{max}}$ is the largest FFT output magnitude sample. In practice, we find that plotting $X_{\text{dB}}(m)$ is very informative because of the enhanced low-magnitude resolution afforded by the logarithmic decibel scale, as described in [Appendix E](#). If either [Eq. \(4-8\)](#) or [Eq. \(4-9\)](#) is used, no compensation need be performed for the above-mentioned N or $N/2$ FFT scale or window processing loss factors. Normalization through division by $(|X(m)|_{\text{max}})^2$ or $|X(m)|_{\text{max}}$ eliminates the

effect of any absolute FFT or window scale factors.

Knowing that the phase angles $X_{\theta}(m)$ of the individual FFT outputs are given by

(4-10)

$$X_{\theta}(m) = \tan^{-1} \left(\frac{X_{\text{imag}}(m)}{X_{\text{real}}(m)} \right),$$

it's important to watch out for $X_{\text{real}}(m)$ values that are equal to zero. That would invalidate our phase-angle calculations in [Eq. \(4-10\)](#) due to division by a zero condition. In practice, we want to make sure that our calculations (or software compiler) detect occurrences of $X_{\text{real}}(m) = 0$ and set the corresponding $X_{\theta}(m)$ to 90° if $X_{\text{imag}}(m)$ is positive, set $X_{\theta}(m)$ to 0° if $X_{\text{imag}}(m)$ is zero, and set $X_{\theta}(m)$ to -90° if $X_{\text{imag}}(m)$ is negative. While we're on the subject of FFT output phase angles, be aware that FFT outputs containing significant noise components can cause large fluctuations in the computed $X_{\theta}(m)$ phase angles. This means that the $X_{\theta}(m)$ samples are only meaningful when the corresponding $|X(m)|$ is well above the average FFT output noise level.

4.3 Derivation of the Radix-2 FFT Algorithm

This section and those that follow provide a detailed description of the internal data structures and operations of the radix-2 FFT for those readers interested in developing software FFT routines or designing FFT hardware. To see just exactly how the FFT evolved from the DFT, we return to the equation for an N -point DFT,

(4-11)

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm / N}.$$

A straightforward derivation of the FFT proceeds with the separation of the input data sequence $x(n)$ into two parts. When $x(n)$ is segmented into its even and odd indexed elements, we can, then, break [Eq. \(4-11\)](#) into two parts as

(4-12)

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n) e^{-j2\pi(2n)m / N} + \sum_{n=0}^{(N/2)-1} x(2n+1) e^{-j2\pi(2n+1)m / N}.$$

Pulling the constant phase angle outside the second summation,

(4-13)

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n) e^{-j2\pi(2n)m / N} + e^{-j2\pi m / N} \sum_{n=0}^{(N/2)-1} x(2n+1) e^{-j2\pi(2n)m / N}.$$

Well, here the equations become so long and drawn out that we'll use a popular notation to simplify things. We'll define

(4-13')

$$W_N = e^{-j2\pi / N}$$

to represent the complex phase-angle factor that is constant with N . So, [Eq. \(4-13\)](#) becomes

(4-14)

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)W_N^{2nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)W_N^{2nm}.$$

Because $W_N^2 = e^{-j2\pi 2/(N)} = e^{-j2\pi/(N/2)}$, we can substitute $W_{N/2}$ for W_N^2 in [Eq. \(4-14\)](#), as

(4-15)

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)W_{N/2}^{nm}$$

where m is in the range 0 to $N/2-1$. Index m has that reduced range because each of the two $N/2$ -point DFTs on the right side of [Eq. \(4-15\)](#) are periodic in m with period $N/2$.

So we now have two $N/2$ summations whose results can be combined to give us the first $N/2$ samples of an N -point DFT. We've reduced some of the necessary number crunching in [Eq. \(4-15\)](#) relative to [Eq. \(4-11\)](#) because the W terms in the two summations of [Eq. \(4-15\)](#) are identical. There's a further benefit in breaking the N -point DFT into two parts because the upper half of the DFT outputs is easy to calculate. Consider the $X(m+N/2)$ output. If we plug $m+N/2$ in for m in [Eq. \(4-15\)](#), then

(4-16)

$$\begin{aligned} X(m+N/2) &= \sum_{n=0}^{(N/2)-1} x(2n)W_{N/2}^{n(m+N/2)} \\ &\quad + W_N^{(m+N/2)} \sum_{n=0}^{(N/2)-1} x(2n+1)W_{N/2}^{n(m+N/2)}. \end{aligned}$$

It looks like we're complicating things, right? Well, just hang in there for a moment. We can now simplify the phase-angle terms inside the summations because

(4-17)

$$\begin{aligned} W_{N/2}^{n(m+N/2)} &= W_{N/2}^{nm} W_{N/2}^{nN/2} = W_{N/2}^{nm} (e^{-j2\pi n 2N/2N}) \\ &= W_{N/2}^{nm}(1) = W_{N/2}^{nm}, \end{aligned}$$

for any integer n . Looking at the so-called *twiddle factor* in front of the second summation in [Eq. \(4-16\)](#), we can simplify it as

(4-18)

$$W_N^{(m+N/2)} = W_N^m W_N^{N/2} = W_N^m (e^{-j2\pi N/2N}) = W_N^m (-1) = -W_N^m.$$

OK, using [Eqs. \(4-17\)](#) and [\(4-18\)](#), we represent [Eq. \(4-16\)](#)'s $X(m+N/2)$ as

(4-19)

$$X(m+N/2) = \sum_{n=0}^{(N/2)-1} x(2n)W_{N/2}^{nm} - W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)W_{N/2}^{nm}.$$

Now, let's repeat [Eqs. \(4-15\)](#) and [\(4-19\)](#) to see the similarity:

(4-20)

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)W_{N/2}^{nm},$$

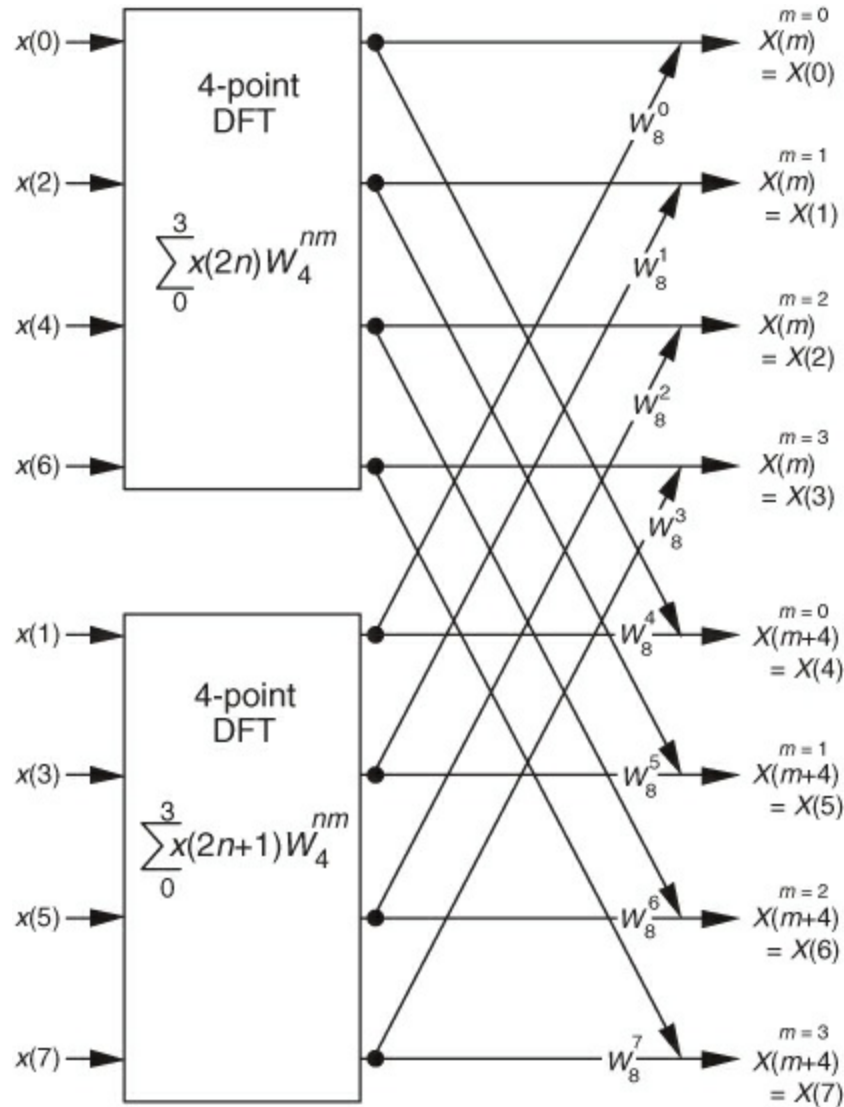
and

$$(4-20')$$

$$X(m+N/2) = \sum_{n=0}^{(N/2)-1} x(2n)W_{N/2}^{nm} - W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)W_{N/2}^{nm}.$$

So here we are. We need not perform any sine or cosine multiplications to get $X(m+N/2)$. We just change the sign of the twiddle factor W_N^m and use the results of the two summations from $X(m)$ to get $X(m+N/2)$. Of course, m goes from 0 to $(N/2)-1$ in [Eq. \(4-20\)](#), which means to compute an N -point DFT, we actually perform two $N/2$ -point DFTs—one $N/2$ -point DFT on the even-indexed $x(n)$ samples and one $N/2$ -point DFT on the odd-indexed $x(n)$ samples. For $N = 8$, [Eqs. \(4-20\)](#) and [\(4-20'\)](#) are implemented as shown in [Figure 4-2](#).

Figure 4-2 FFT implementation of an 8-point DFT using two 4-point DFTs.



Because $-e^{-j2\pi m/N} = e^{-j2\pi(m+N/2)/N}$, the negative W twiddle factors before the second summation in [Eq. \(4-20'\)](#) are implemented with positive W twiddle factors that follow the lower DFT in [Figure 4-2](#).

If we simplify [Eqs. \(4-20\)](#) and [\(4-20'\)](#) to the form

$$(4-21)$$

$$X(m) = A(m) + W_N^m B(m)$$

and

$$(4-21')$$

$$X(m+N/2) = A(m) - W_N^m B(m),$$

we can go further and think about breaking the two 4-point DFTs into four 2-point DFTs. Let's see how we can subdivide the upper 4-point DFT in [Figure 4-2](#) whose four outputs are $A(m)$ in [Eqs. \(4-21\)](#) and [\(4-21'\)](#). We segment the inputs to the upper 4-point DFT into their odd and even components:

$$(4-22)$$

$$\begin{aligned} A(m) &= \sum_{n=0}^{(N/2)-1} x(2n)W_{N/2}^{nm} = \\ &\sum_{n=0}^{(N/4)-1} x(4n)W_{N/2}^{2nm} + \sum_{n=0}^{(N/4)-1} x(4n+2)W_{N/2}^{(2n+1)m}. \end{aligned}$$

Because $W_{N/2}^{2nm} = W_{N/4}^{nm}$, we can express $A(m)$ in the form of two $N/4$ -point DFTs, as

$$(4-23)$$

$$A(m) = \sum_{n=0}^{(N/4)-1} x(4n)W_{N/4}^{nm} + W_{N/2}^m \sum_{n=0}^{(N/4)-1} x(4n+2)W_{N/4}^{nm}.$$

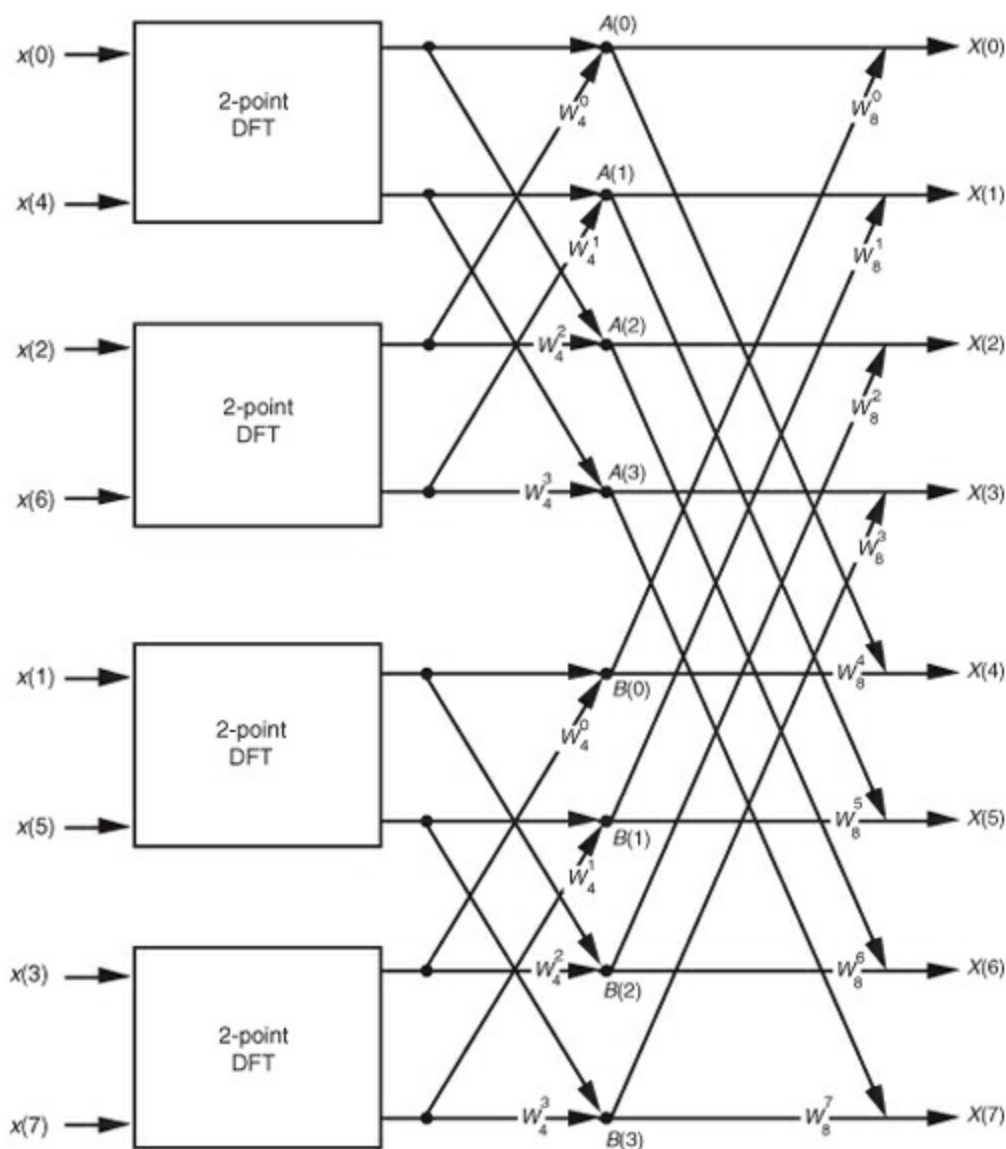
Notice the similarity between [Eqs. \(4-23\)](#) and [\(4-20\)](#). This capability to subdivide an $N/2$ -point DFT into two $N/4$ -point DFTs gives the FFT its capacity to greatly reduce the number of necessary multiplications to implement DFTs. (We're going to demonstrate this shortly.) Following the same steps we used to obtain $A(m)$, we can show that [Eq.\(4-21\)](#)'s $B(m)$ is

$$(4-24)$$

$$B(m) = \sum_{n=0}^{(N/4)-1} x(4n+1)W_{N/4}^{nm} + W_{N/2}^m \sum_{n=0}^{(N/4)-1} x(4n+3)W_{N/4}^{nm}.$$

For our $N = 8$ example, [Eqs. \(4-23\)](#) and [\(4-24\)](#) are implemented as shown in [Figure 4-3](#). The FFT's well-known *butterfly* pattern of signal flows is certainly evident, and we see the further shuffling of the input data in [Figure 4-3](#). The twiddle factor $W_{N/2}^m$ in [Eqs. \(4-23\)](#) and [\(4-24\)](#), for our $N = 8$ example, ranges from W_4^0 to W_4^3 because the m index, for $A(m)$ and $B(m)$, goes from 0 to 3. For any N -point DFT, we can break each of the $N/2$ -point DFTs into two $N/4$ -point DFTs to further reduce the number of sine and cosine multiplications. Eventually, we would arrive at an array of 2-point DFTs where no further computational savings could be realized. This is why the number of points in our FFTs is constrained to be some power of two and why this FFT algorithm is referred to as the radix-2 FFT.

Figure 4-3 FFT implementation of an 8-point DFT as two 4-point DFTs and four 2-point DFTs.



Moving right along, let's go one step further, and then we'll be finished with our $N = 8$ -point FFT derivation. The 2-point DFT functions in [Figure 4-3](#) cannot be partitioned into smaller parts—we've reached the end of our DFT reduction process, arriving at the butterfly of a single 2-point DFT as shown in [Figure 4-4](#). From the definition of W_N , $W_N^0 = e^{-j2\pi 0/N} = 1$ and $W_N^{N/2} = e^{-j2\pi N/2N} = e^{-j\pi} = -1$. So the 2-point DFT blocks in [Figure 4-3](#) can be replaced by the butterfly in [Figure 4-4](#) to give us a full 8-point FFT implementation of the DFT as shown in [Figure 4-5](#).

Figure 4-4 Single 2-point DFT butterfly.

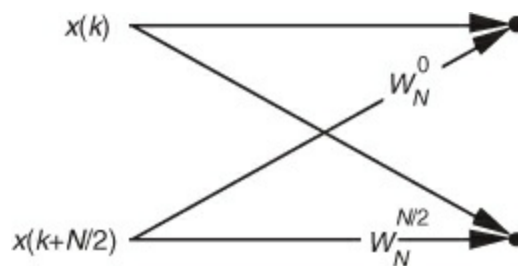
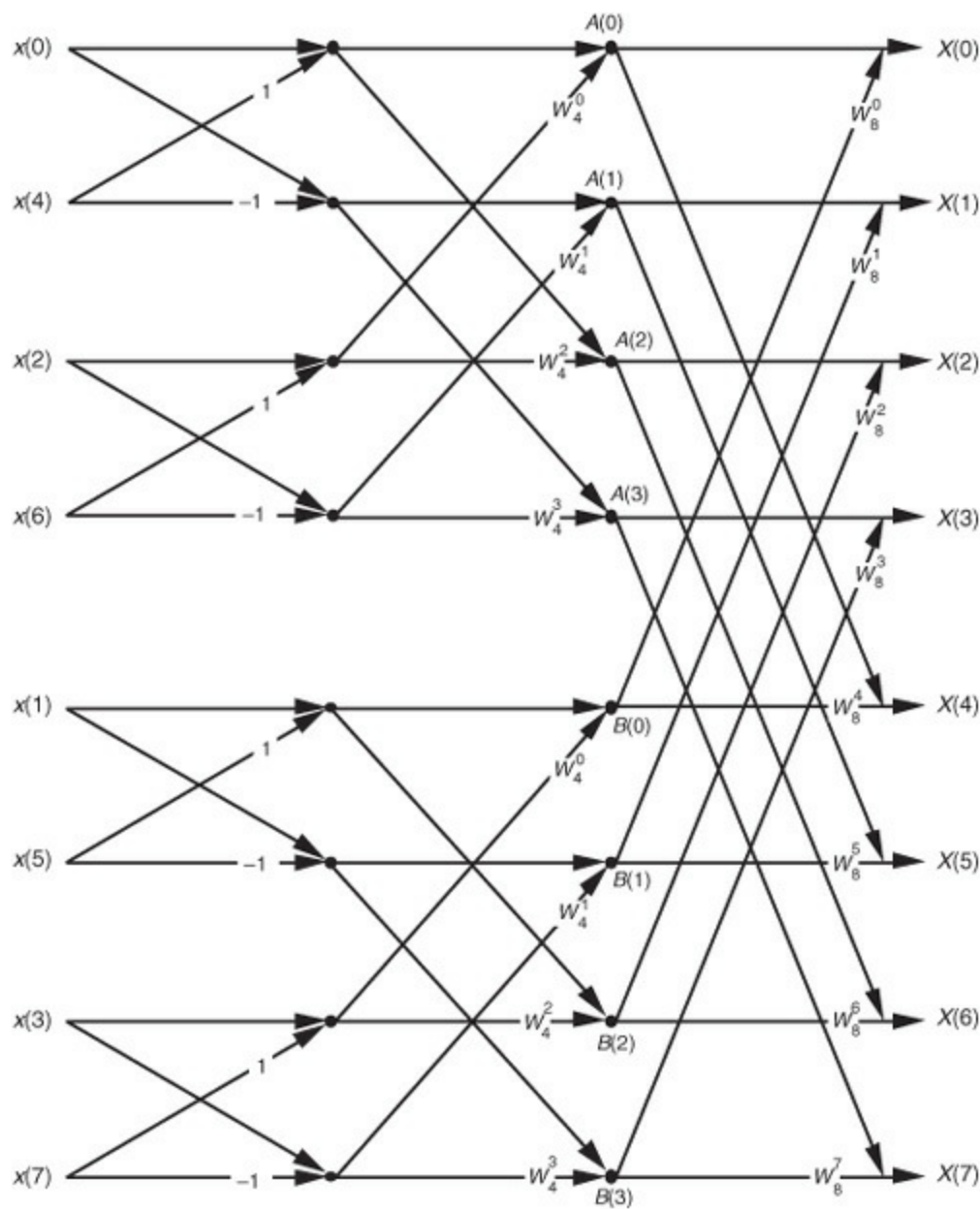


Figure 4-5 Full decimation-in-time FFT implementation of an 8-point DFT.



OK, we've gone through a fair amount of algebraic foot shuffling here. To verify that the derivation of the FFT is valid, we can apply the 8-point data sequence of [Chapter 3](#)'s DFT Example 1 to the 8-point FFT represented by [Figure 4-5](#). The data sequence representing $x(n) = \sin(2\pi 1000nt_s) + 0.5\sin(2\pi 2000nt_s + 3\pi/4)$ is

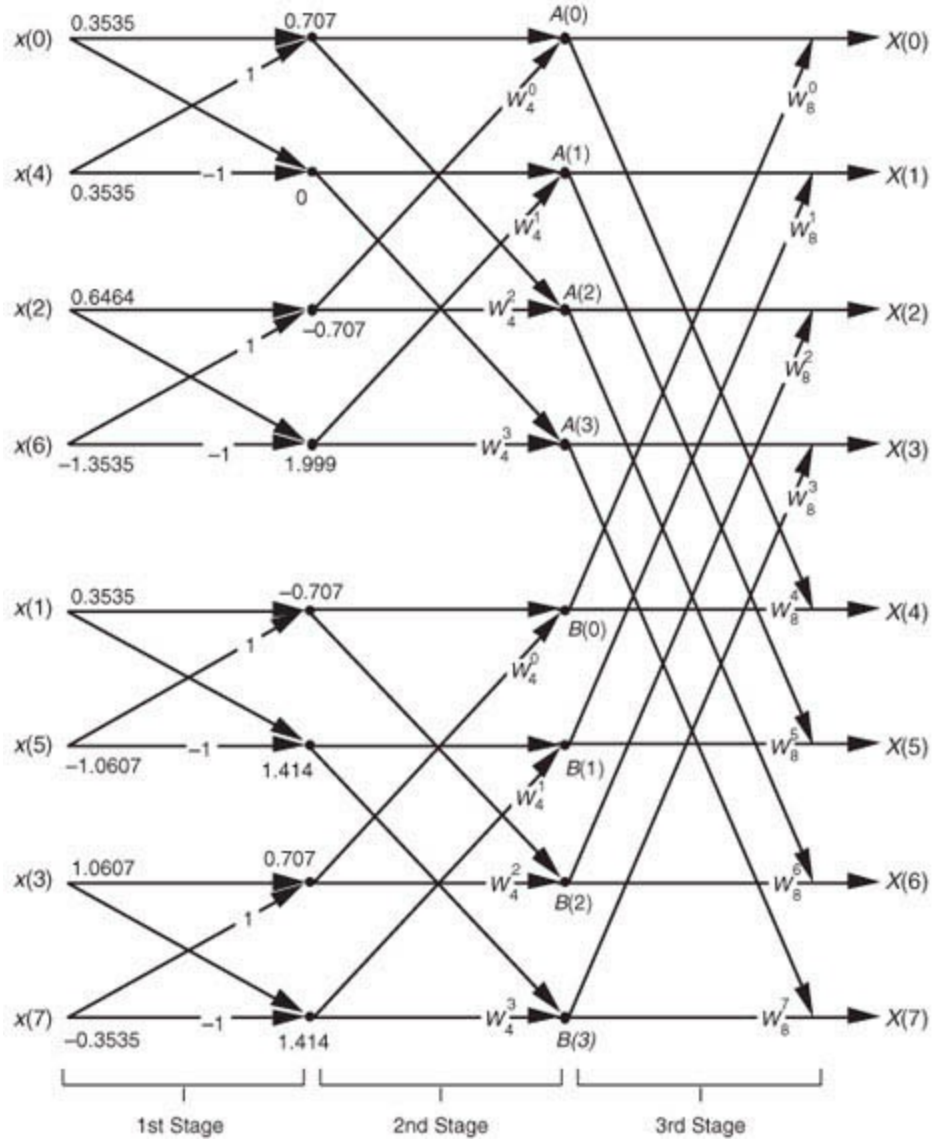
(4-25)

$$\begin{array}{ll} x(0) = 0.3535, & x(1) = 0.3535, \\ x(2) = 0.6464, & x(3) = 1.0607, \\ x(4) = 0.3535, & x(5) = -1.0607, \\ x(6) = -1.3535, & x(7) = -0.3535. \end{array}$$

We begin grinding through this example by applying the input values from [Eq. \(4-25\)](#) to [Figure 4-5](#), giving the data values shown on left side of [Figure 4-6](#). The outputs of the second stage of the FFT are

Figure 4-6 Eight-point FFT of Example 1 from [Section 3.1](#).

$$\begin{aligned}
A(0) &= 0.707 + W_4^0(-0.707) = 0.707 + (1 + j0)(-0.707) = 0 + j0, \\
A(1) &= 0.0 + W_4^1(1.999) = 0.0 + (0 - j1)(1.999) = 0 - j1.999, \\
A(2) &= 0.707 + W_4^2(-0.707) = 0.707 + (-1 + j0)(-0.707) = 1.414 + j0, \\
A(3) &= 0.0 + W_4^3(1.999) = 0.0 + (0 + j1)(1.999) = 0 + j1.999, \\
B(0) &= 0.707 + W_4^0(0.707) = -0.707 + (1 + j0)(0.707) = 0 + j0, \\
B(1) &= 1.414 + W_4^1(1.414) = 1.414 + (0 - j1)(1.414) = 1.414 - j1.414, \\
B(2) &= -0.707 + W_4^2(0.707) = -0.707 + (-1 + j0)(0.707) = -1.414 + j0, \text{ and} \\
B(3) &= 1.414 + W_4^3(1.414) = 1.414 + (0 + j1)(1.414) = 1.414 + j1.414.
\end{aligned}$$



Calculating the outputs of the third stage of the FFT to arrive at our final answer:

$$X(0) = A(0) + W_8^0 B(0) = 0 + j0 + (1 + j0)(0 + j0) = 0 + j0 + 0 + j0 = 0 \angle 0^\circ,$$

$$X(1) = A(1) + W_8^1 B(1) = 0 - j1.999 + (0.707 - j0.707)(1.414 - j1.414) \\ = 0 - j1.999 + 0 - j1.999 = 0 - j4 = 4 \angle -90^\circ,$$

$$X(2) = A(2) + W_8^2 B(2) = 1.414 + j0 + (0 - j1)(-1.414 + j0) \\ = 1.414 + j0 + 0 + j1.4242 = 1.414 + j1.414 = 2 \angle 45^\circ,$$

$$X(3) = A(3) + W_8^3 B(3) = 0 + j1.999 + (-0.707 - j0.707)(1.414 + j1.414) \\ = 0 + j1.999 + 0 - j1.999 = 0 \angle 0^\circ,$$

$$X(4) = A(0) - W_8^0 B(0) = A(0) + W_8^4 B(0) = 0 + j0 + (-1 + j0)(0 + j0) \\ = 0 + j0 + 0 + j0 = 0 \angle 0^\circ,$$

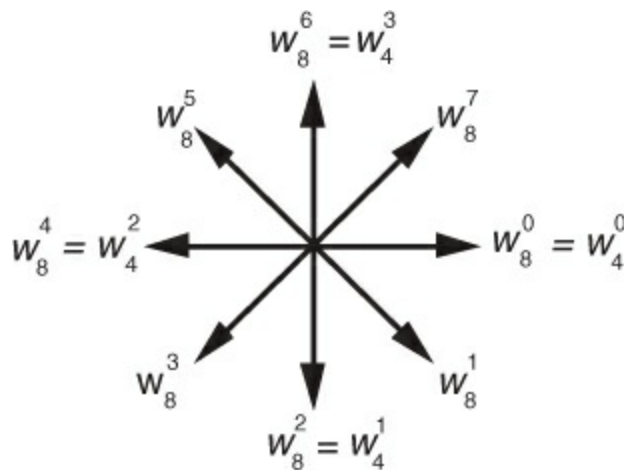
$$X(5) = A(1) - W_8^1 B(1) \\ = A(1) + W_8^5 B(1) = 0 - j1.999 + (-0.707 + j0.707)(1.414 - j1.414) \\ = 0 - j1.999 + 0 + j1.999 = 0 \angle 0^\circ,$$

$$X(6) = A(2) - W_8^2 B(2) \\ = A(2) + W_8^6 B(2) = 1.414 + j0 + (0 + j1)(-1.414 + j0) \\ = 1.414 + j0 + 0 - j1.414 = 1.414 - j1.414 = 2 \angle -45^\circ, \text{ and}$$

$$X(7) = A(3) - W_8^3 B(3) \\ = A(3) + W_8^7 B(3) = 0 + j1.999 + (0.707 + j0.707)(1.414 + j1.414) \\ = 0 + j1.999 + 0 + j1.999 = 0 + j4 = 4 \angle 90^\circ.$$

So, happily, the FFT gives us the correct results, and again we remind the reader that the FFT is not an approximation to a DFT; it is the DFT with a reduced number of necessary arithmetic operations. You've seen from the above example that the 8-point FFT example required less effort than the 8-point DFT Example 1 in [Section 3.1](#). Some authors like to explain this arithmetic reduction by the redundancies inherent in the twiddle factors W_N^m . They illustrate this with the *starburst* pattern in [Figure 4-7](#) showing the equivalencies of some of the twiddle factors in an 8-point DFT.

Figure 4-7 Cyclic redundancies in the twiddle factors of an 8-point FFT.



4.4 FFT Input/Output Data Index Bit Reversal

OK, let's look into some of the special properties of the FFT that are important to FFT software developers and FFT hardware designers. Notice that [Figure 4-5](#) was titled "Full decimation-in-time FFT implementation of an 8-point DFT." The *decimation-in-time* phrase refers to how we broke the DFT input samples into odd and even parts in the derivation of [Eqs. \(4-20\)](#), [\(4-23\)](#), and [\(4-24\)](#). This time decimation leads to the scrambled order of the input data's index n in [Figure 4-5](#). The pattern of this shuffled order can be understood with the help of [Table 4-1](#). The shuffling of the input data is known as *bit reversal* because the scrambled order of the input data index can be obtained by

reversing the bits of the binary representation of the normal input data index order. Sounds confusing, but it's really not—[Table 4-1](#) illustrates the input index bit reversal for our 8-point FFT example. Notice the normal index order in the left column of [Table 4-1](#) and the scrambled order in the right column that corresponds to the final decimated input index order in [Figure 4-5](#). We've transposed the original binary bits representing the normal index order by reversing their positions. The most significant bit becomes the least significant bit and the least significant bit becomes the most significant bit, the next to the most significant bit becomes the next to the least significant bit, and the next to the least significant bit becomes the next to the most significant bit, and so on.[‡]

[‡] Many that are first shall be last; and the last first. [Mark 10:31]

Table 4-1 Input Index Bit Reversal for an 8-Point FFT

Normal order of index n	Binary bits of index n	Reversed bits of index n	Bit-reversed order of index n
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

4.5 Radix-2 FFT Butterfly Structures

Let's explore the butterfly signal flows of the decimation-in-time FFT a bit further. To simplify the signal flows, let's replace the twiddle factors in [Figure 4-5](#) with their equivalent values referenced to W_N^m , where $N = 8$. We can show just the exponents m of W_N^m , to get the FFT structure shown in [Figure 4-8](#). That is, W_4^1 from [Figure 4-5](#) is equal to W_8^2 and is shown as a 2 in [Figure 4-8](#), W_4^2 from [Figure 4-5](#) is equal to W_8^4 and is shown as a 4 in [Figure 4-8](#), etc. The 1s and -1 s in the first stage of [Figure 4-5](#) are replaced in [Figure 4-8](#) by 0s and 4s, respectively. Other than the twiddle factor notation, [Figure 4-8](#) is identical to [Figure 4-5](#). We can shift around the signal nodes in [Figure 4-5](#) and arrive at an 8-point decimation-in-time FFT as shown in [Figure 4-9](#). Notice that the input data in [Figure 4-9](#) is in its normal order and the output data indices are bit-reversed. In this case, a bit-reversal operation needs to be performed at the output of the FFT to unscramble the frequency-domain results.

Figure 4-8 Eight-point decimation-in-time FFT with bit-reversed inputs.

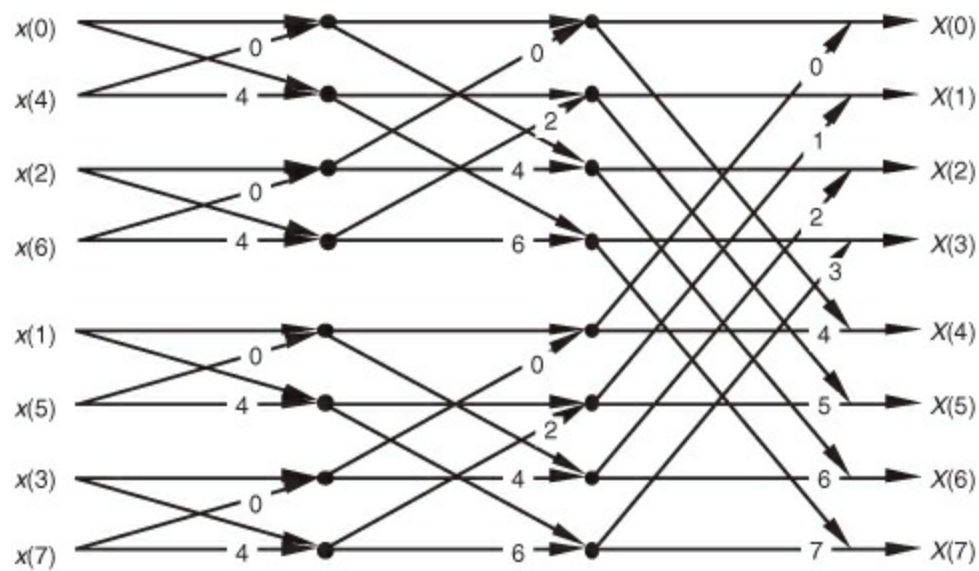
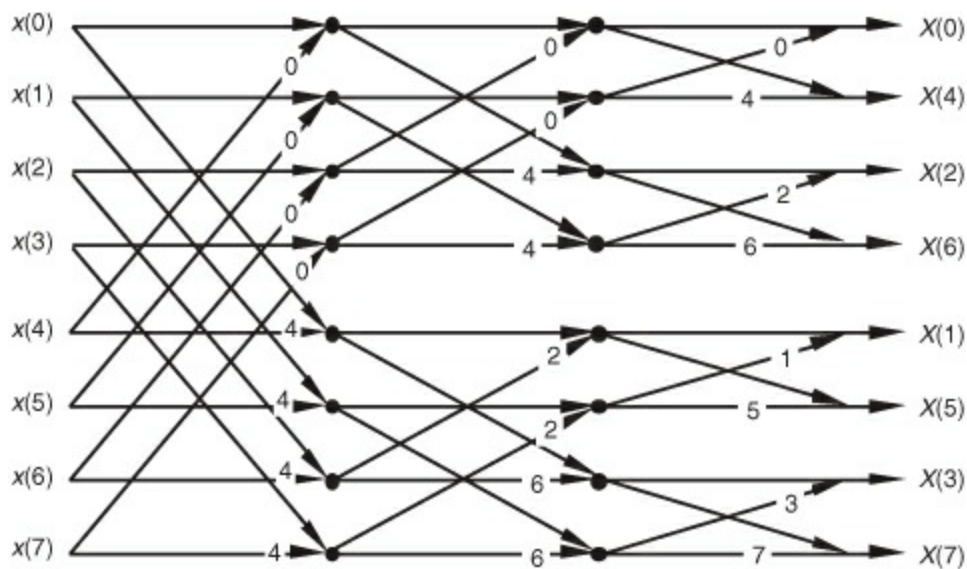
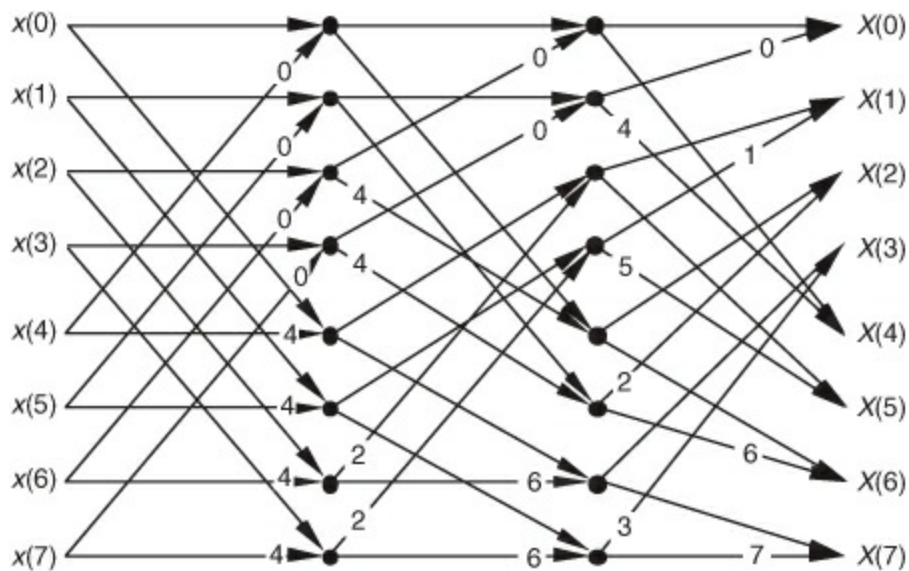


Figure 4-9 Eight-point decimation-in-time FFT with bit-reversed outputs.



[Figure 4-10](#) shows an FFT signal-flow structure that avoids the bit-reversal problem altogether, and the graceful weave of the traditional FFT butterflies is replaced with a tangled, but effective, configuration.

Figure 4-10 Eight-point decimation-in-time FFT with inputs and outputs in normal order.



Not too long ago, hardware implementations of the FFT spent most of their time (clock cycles)

performing multiplications, and the bit-reversal process necessary to access data in memory wasn't a significant portion of the overall FFT computational problem. Now that high-speed multiplier/accumulator integrated circuits can multiply two numbers in a single clock cycle, FFT data multiplexing and memory addressing have become much more important. This has led to the development of efficient algorithms to perform bit reversal[7–10].

There's another derivation for the FFT that leads to butterfly structures looking like those we've already covered, but the twiddle factors in the butterflies are different. This alternate FFT technique is known as the decimation-in-frequency algorithm. Where the decimation-in-time FFT algorithm is based on subdividing the input data into its odd and even components, the decimation-in-frequency FFT algorithm is founded upon calculating the odd and even output frequency samples separately. The derivation of the decimation-in-frequency algorithm is straightforward and included in many tutorial papers and textbooks, so we won't go through the derivation here[4,5,15,16]. We will, however, illustrate decimation-in-frequency butterfly structures (analogous to the structures in [Figures 4-8 through 4-10](#)) in [Figures 4-11 through 4-13](#).

Figure 4-11 Eight-point decimation-in-frequency FFT with bit-reversed inputs.

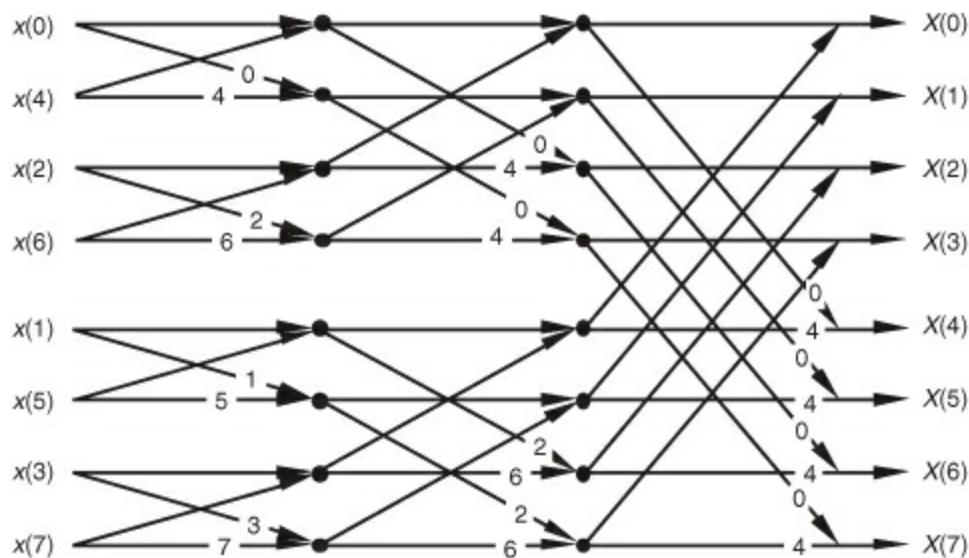


Figure 4-12 Eight-point decimation-in-frequency FFT with bit-reversed outputs.

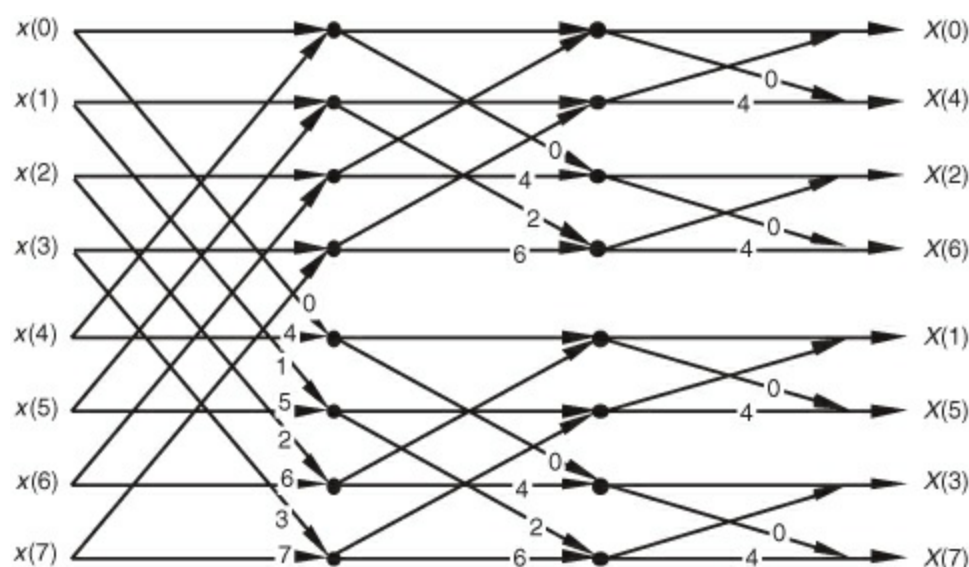
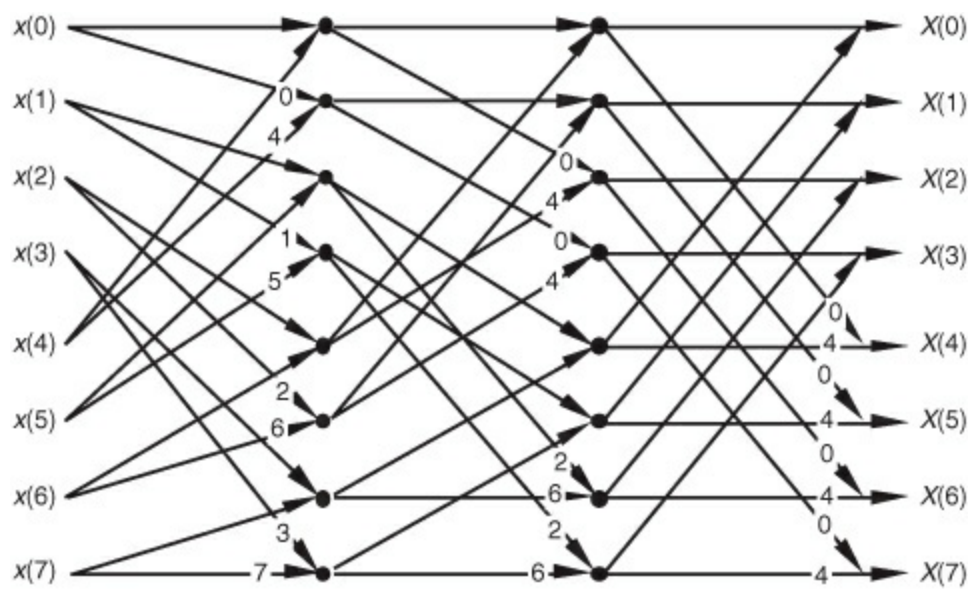


Figure 4-13 Eight-point decimation-in-frequency FFT with inputs and outputs in normal order.



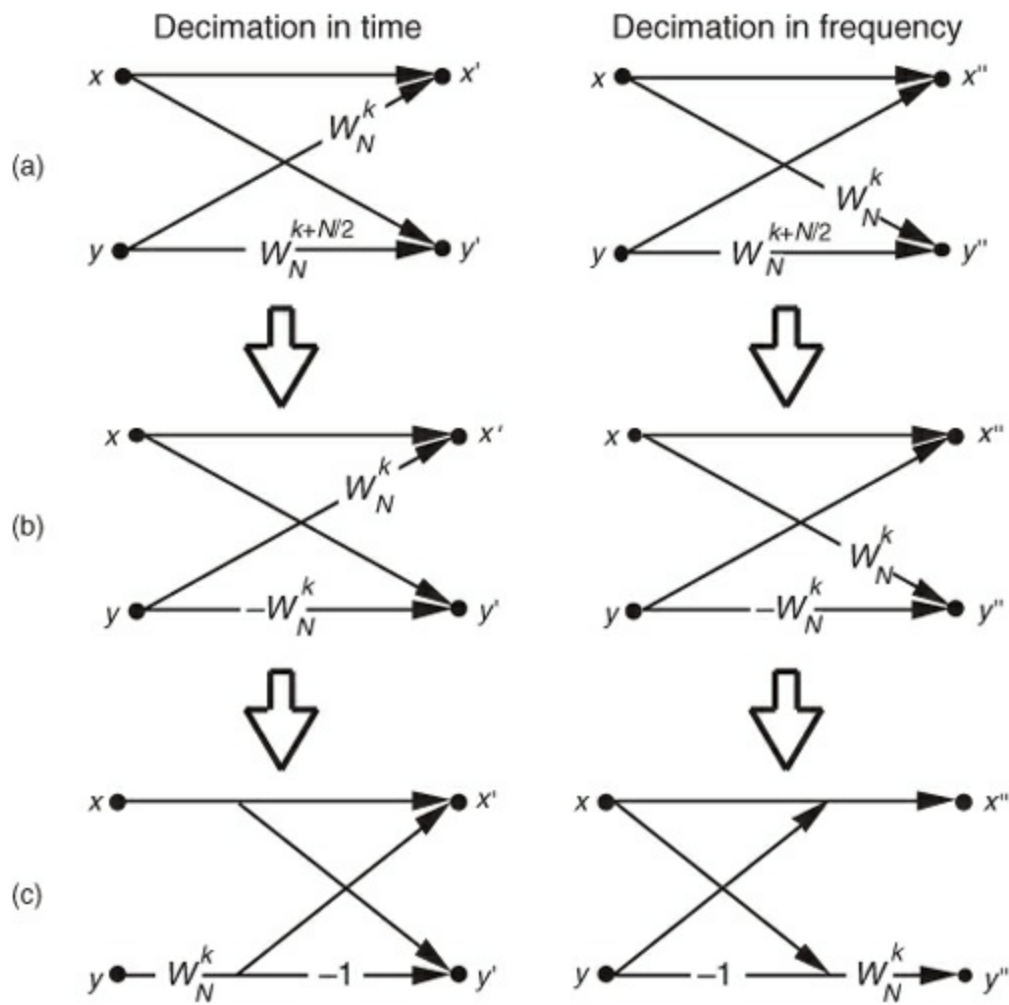
So an equivalent decimation-in-frequency FFT structure exists for each decimation-in-time FFT structure. It's important to note that the number of necessary multiplications to implement the decimation-in-frequency FFT algorithms is the same as the number necessary for the decimation-in-time FFT algorithms. There are so many different FFT butterfly structures described in the literature that it's easy to become confused about which structures are decimation-in-time and which are decimation-in-frequency. Depending on how the material is presented, it's easy for a beginner to fall into the trap of believing that decimation-in-time FFTs always have their inputs bit-reversed and decimation-in-frequency FFTs always have their outputs bit-reversed. This is not true, as the above figures show. Decimation-in-time or -frequency is determined by whether the DFT inputs or outputs are partitioned when deriving a particular FFT butterfly structure from the DFT equations.

4.6 Alternate Single-Butterfly Structures

Let's take one more look at a single butterfly. The FFT butterfly structures in [Figures 4-8](#), [4-9](#), [4-11](#), and [4-12](#) are the direct result of the derivations of the decimation-in-time and decimation-in-frequency algorithms. Although it's not very obvious at first, the twiddle factor exponents shown in these structures do have a consistent pattern. Notice how they always take the general forms shown in [Figure 4-14\(a\)](#).[‡] To implement the decimation-in-time butterfly of [Figure 4-14\(a\)](#), we'd have to perform two complex multiplications and two complex additions. Well, there's a better way. Consider the decimation-in-time butterfly in [Figure 4-14\(a\)](#). If the top input is x and the bottom input is y , the top butterfly output would be

[‡] Remember, for simplicity the butterfly structures in [Figures 4-8](#) through [4-13](#) show only the twiddle factor exponents, k and $k+N/2$, and not the entire complex twiddle factors.

Figure 4-14 Decimation-in-time and decimation-in-frequency butterfly structures: (a) original form; (b) simplified form; (c) optimized form.



$$x' = x + W_N^k y,$$

and the bottom butterfly output would be

$$(4-27)$$

$$y' = x + W_N^{k+N/2} y.$$

Fortunately, the operations in [Eqs. \(4-26\)](#) and [\(4-27\)](#) can be simplified because the two twiddle factors are related by

$$(4-28)$$

$$W_N^{k+N/2} = W_N^k W_N^{N/2} = W_N^k (e^{-j2\pi N/2N}) = W_N^k (-1) = -W_N^k.$$

So we can replace the $W_N^{k+N/2}$ twiddle factors in [Figure 4-14\(a\)](#) with $-W_N^k$ to give us the simplified butterflies shown in [Figure 4-14\(b\)](#). Because the twiddle factors in [Figure 4-14\(b\)](#) differ only by their signs, the optimized butterflies in [Figure 4-14\(c\)](#) can be used. Notice that these *optimized* butterflies require two complex additions but only one complex multiplication, thus reducing our computational workload.[‡]

[‡] It's because there are $(N/2)\log_2 N$ butterflies in an N -point FFT that we said the number of complex multiplications performed by an FFT is $(N/2)\log_2 N$ in [Eq. \(4-2\)](#).

We'll often see the optimized butterfly structures of [Figure 4-14\(c\)](#) in the literature instead of those in [Figure 4-14\(a\)](#). These optimized butterflies give us an easy way to recognize decimation-in-time and decimation-in-frequency algorithms. When we do come across the optimized butterflies from [Figure](#)

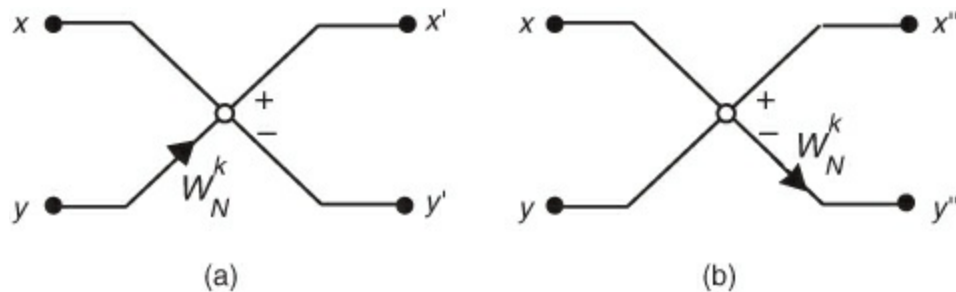
[4-14\(c\)](#), we'll know that the algorithm is decimation-in-time if the twiddle factor precedes the -1 , or else the algorithm is decimation-in-frequency if the twiddle factor follows the -1 .

Sometimes we'll encounter FFT structures in the literature that use the notation shown in [Figure 4-15](#)[\[5, 12\]](#). These wingless butterflies are equivalent to those shown in [Figure 4-14\(c\)](#). The signal-flow convention in [Figure 4-15](#) is such that the plus output of a circle is the sum of the two samples that enter the circle from the left, and the minus output of a circle is the difference of the samples that enter the circle. So the outputs of the decimation-in-time butterflies in [Figures 4-14\(c\)](#) and [4-15\(a\)](#) are given by

$$(4-29)$$

$$x' = x + W_N^k y, \text{ and } y' = x - W_N^k y.$$

Figure 4-15 Alternate FFT butterfly notation: (a) decimation in time; (b) decimation in frequency.



The outputs of the decimation-in-frequency butterflies in [Figures 4-14\(c\)](#) and [4-15\(b\)](#) are

$$(4-30)$$

$$x'' = x + y, \text{ and } y'' = W_N^k(x - y) = W_N^k x - W_N^k y.$$

So which FFT structure is the best one to use? It depends on the application, the hardware implementation, and convenience. If we're using a software routine to perform FFTs on a general-purpose computer, we usually don't have a lot of choices. Most folks just use whatever existing FFT routines happen to be included in their commercial software package. Their code may be optimized for speed, but you never know. Examination of the software code may be necessary to see just how the FFT is implemented. If we *feel the need for speed*, we should check to see if the software calculates the sines and cosines each time it needs a twiddle factor. Trigonometric calculations normally take many machine cycles. It may be possible to speed up the algorithm by calculating the twiddle factors ahead of time and storing them in a table. That way, they can be *looked up*, instead of being calculated each time they're needed in a butterfly. If we're writing our own software routine, checking for butterfly output data overflow and careful magnitude scaling may allow our FFT to be performed using integer arithmetic that can be faster on some machines.[‡] Care must be taken, however, when using integer arithmetic; some Reduced Instruction Set Computer (RISC) processors actually take longer to perform integer calculations because they're specifically designed to operate on floating-point numbers.

[‡] Overflow is what happens when the result of an arithmetic operation has too many bits, or digits, to be represented in the hardware registers designed to contain that result. FFT data overflow is described in [Section 12.3](#).

If we're using commercial array processor hardware for our calculations, the code in these processors is *always* optimized because their purpose in life is high speed. Array processor manufacturers typically publicize their products by specifying the speed at which their machines perform a 1024-point FFT. Let's look at some of our options in selecting a particular FFT structure in

case we're designing special-purpose hardware to implement an FFT.

The FFT butterfly structures previously discussed typically fall into one of two categories: in-place FFT algorithms and double-memory FFT algorithms. An in-place algorithm is depicted in [Figure 4-5](#). The output of a butterfly operation can be stored in the same hardware memory locations that previously held the butterfly's input data. No intermediate storage is necessary. This way, for an N -point FFT, only $2N$ memory locations are needed. (The 2 comes from the fact that each butterfly node represents a data value that has both a real and an imaginary part.) The rub with the in-place algorithms is that data routing and memory addressing are rather complicated. A double-memory FFT structure is that depicted in [Figure 4-10](#). With this structure, intermediate storage is necessary because we no longer have the standard butterflies, and $4N$ memory locations are needed. However, data routing and memory address control are much simpler in double-memory FFT structures than the in-place technique. The use of high-speed, floating-point integrated circuits to implement pipelined FFT architectures takes better advantage of their pipelined structure when the double-memory algorithm is used[\[13\]](#).

There's another class of FFT structures, known as constant-geometry algorithms, that make the addressing of memory both simple and constant for each stage of the FFT. These structures are of interest to those folks who build special-purpose FFT hardware devices[\[4,14\]](#). From the standpoint of general hardware the decimation-in-time algorithms are optimum for real input data sequences, and decimation-in-frequency is appropriate when the input is complex[\[6\]](#). When the FFT input data is symmetrical in time, special FFT structures exist to eliminate unnecessary calculations. These special butterfly structures based on input data symmetry are described in the literature[\[15\]](#).

For two-dimensional FFT applications, such as processing photographic images, the decimation-in-frequency algorithms appear to be the optimum choice[\[16\]](#). Your application may be such that FFT input and output bit reversal is not an important factor. Some FFT applications allow manipulating a bit-reversed FFT output sequence in the frequency domain without having to unscramble the FFT's output data. Then an inverse transform that's expecting bit-reversed inputs will give a time-domain output whose data sequence is correct. This situation avoids the need to perform any bit reversals at all. Multiplying two FFT outputs to implement convolution or correlation are examples of this possibility.[‡] As we can see, finding the optimum FFT algorithm and hardware architecture for an FFT is a fairly complex problem to solve, but the literature provides guidance[\[4,17,18\]](#).

[‡] See [Section 13.10](#) for an example of using the FFT to perform convolution.

References

- [1] Cooley, J., and Tukey, J. "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Comput.*, Vol. 19, No. 90, April 1965, pp. 297–301.
- [2] Cooley, J., Lewis, P., and Welch, P. "Historical Notes on the Fast Fourier Transform," *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-15, No. 2, June 1967.
- [3] Harris, F. J. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proceedings of the IEEE*, Vol. 66, No. 1, January 1978, p. 54.
- [4] Oppenheim, A. V., and Schaffer, R. W. *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1989, p. 608.
- [5] Rabiner, L. R., and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice

Hall, Englewood Cliffs, New Jersey, 1975, p. 367.

- [6] Sorenson, H. V., Jones, D. L., Heideman, M. T., and Burrus, C. S. “Real-Valued Fast Fourier Transform Algorithms,” *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-35, No. 6, June 1987.
- [7] Evans, D. “An Improved Digit-Reversal Permutation Algorithm for the Fast Fourier and Hartley Transforms,” *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-35, No. 8, August 1987.
- [8] Burrus, C. S. “Unscrambling for Fast DFT Algorithms,” *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. 36, No. 7, July 1988.
- [9] Rodriguez, J. J. “An Improved FFT Digit-Reversal Algorithm,” *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-37, No. 8, August 1989.
- [10] Land, A. “Bit Reverser Scrambles Data for FFT,” *EDN*, March 2, 1995.
- [11] JG-AE Subcommittee on Measurement Concepts, “What Is the Fast Fourier Transform?,” *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-15, No. 2, June 1967.
- [12] Cohen, R., and Perlman, R. “500 kHz Single Board FFT System Incorporates DSP Optimized Chips,” *EDN*, October 31, 1984.
- [13] Eldon, J., and Winter, G. E. “Floating-Point Chips Carve Out FFT Systems,” *Electronic Design*, August 4, 1983.
- [14] Lamb, K. “CMOS Building Blocks Shrink and Speed Up FFT Systems,” *Electronic Design*, August 6, 1987.
- [15] Markel, J. D. “FFT Pruning,” *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-19, No. 4, December 1971.
- [16] Wu, H. R., and Paoloni, F. J. “The Structure of Vector Radix Fast Fourier Transforms,” *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-37, No. 8, August 1989.
- [17] Ali, Z. M. “High Speed FFT Processor,” *IEEE Trans. on Communications*, Vol. COM-26, No. 5, May 1978.
- [18] Bergland, G. “Fast Fourier Transform Hardware Implementations—An Overview,” *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-17, June 1969.

Chapter 4 Problems

4.1 Thinking about the FFT:

- (a) How do the results differ between performing an N -point FFT and performing an N -point discrete Fourier transform (DFT) on the same set of time samples?
- (b) What is the restriction on the number of time samples, N , in performing an N -point radix-2 FFT?

4.2 Assume we want to compute an N -point FFT of an $x(n)$ audio signal from a compact disc (CD), with the FFT’s output frequency-domain sample spacing no greater than 1 Hz. If $x(n)$ ’s sample rate is $f_s = 44.1$ kHz, what is the number of necessary time samples, N , applied to the FFT?

4.3 Assume we have an $x(n)$ time-domain sequence, whose length is 3800 samples, on which we

want to perform an FFT. The 3800 time samples represent a total signal collection-interval duration of 2 seconds.

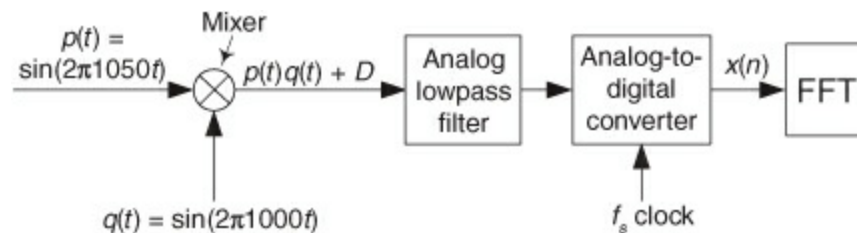
- (a) How many zero-valued samples must be appended (*zero padding*) to $x(n)$ in order to implement an FFT?
- (b) After the FFT is performed, what is the spacing, measured in Hz, between the frequency-domain FFT samples?
- (c) In the case of lowpass sampling, what is the highest-frequency spectral component permitted in the original analog $x(t)$ signal such that no aliasing errors occur in $x(n)$?

4.4 This problem illustrates the computational savings afforded by the FFT over that of the discrete Fourier transform (DFT). Suppose we wanted to perform a spectrum analysis on a time-domain sequence whose length is 32768 (2^{15}) samples. Estimate the ratio of the number of complex multiplications needed by a 32768-point DFT over the number of complex multiplies needed by a 32768-point FFT. (Assume that one of the text's *optimized* [Figure 4-14\(c\)](#) butterflies, requiring one complex multiply per butterfly operation, is used to implement the FFT.)

4.5 Think about the system in [Figure P4-5](#) using an FFT to measure the amplitude of the $p(t)$ signal. The output of the mixer, the product $p(t)q(t)$, contains the sum of two sinusoids whose amplitudes are proportional to the peak value of $p(t)$. The frequencies of those sinusoids are 50 Hz and 2050 Hz. The lowpass filter rejects the 2050 Hz signal. Due to imperfections in the mixer, signal $p(t)q(t)$ is riding on a constant DC (zero Hz) bias represented as value D . This scenario results in an $x(n)$ time sequence whose average value is 17.

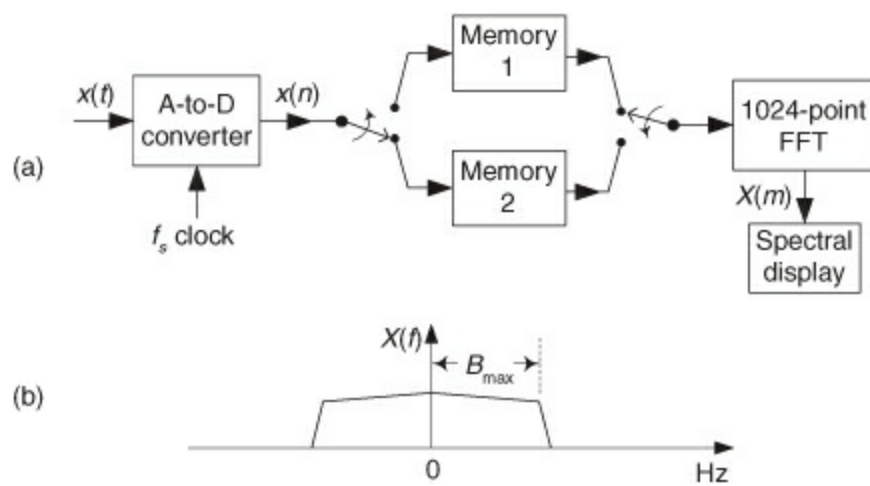
- (a) What is the minimum value for the analog-to-digital converter's f_s sample rate to satisfy the Nyquist criterion?
- (b) If we collect 2048 filter output samples and perform a 2048-point FFT, what will be the magnitude of the FFT's $X(0)$ sample?

Figure P4-5



4.6 Assume you've purchased a high-performance commercial *real-time* spectrum analyzer that contains an analog-to-digital converter so that the analyzer can accept analog (continuous) $x(t)$ input signals. The analyzer can perform a 1024-point FFT in 50 microseconds and has two banks of memory in which the analog-to-digital converter samples are stored as shown in [Figure P4-6\(a\)](#). An FFT is performed on 1024 $x(n)$ signal samples stored in Memory Bank 1 while 1024 new $x(n)$ time samples are being loaded into Memory Bank 2.

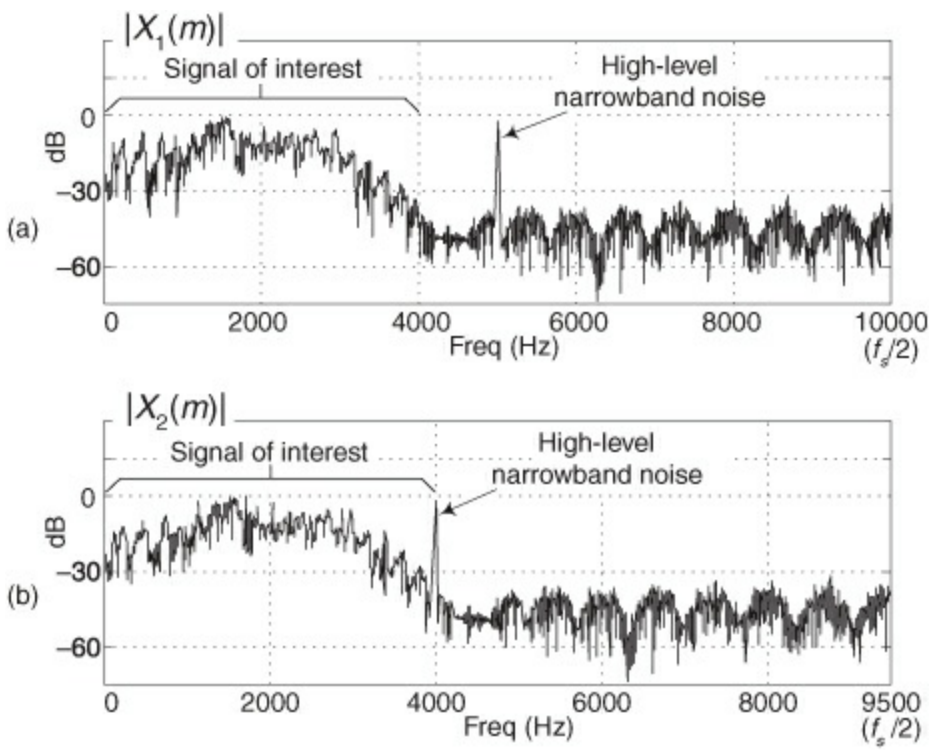
Figure P4-6



At the completion of the first FFT, the analyzer waits until Memory Bank 2 is filled with 1024 samples and then begins performing an FFT on the data in that second memory. During the second FFT computation still newer $x(n)$ time samples are loaded into Memory Bank 1. Thus the analyzer can compute 1024 FFT results as often as once every 50 microseconds, and that is the meaning of the phrase “real-time spectrum analyzer.” Here’s your problem: In a *lowpass sampling* scenario what is the maximum one-sided bandwidth B_{\max} of the analog $x(t)$ input signal for which the analyzer can perform real-time FFTs without discarding (ignoring) any discrete $x(n)$ samples? (The definition of bandwidth B_{\max} is shown in [Figure P4-6\(b\)](#).)

4.7 Here’s an interesting problem. Assume we performed lowpass sampling of an analog $x(t)$ signal, at a sample rate of $f_s = 20$ kHz, obtaining a discrete sequence $x_1(n)$. Next we perform an FFT on $x_1(n)$ to obtain the $|X_1(m)|$ FFT magnitude results presented in [Figure P4-7\(a\)](#). There we see our signal of interest in the range of 0 to 4 kHz, but we detect a high-magnitude narrowband spectral noise signal centered at 5 kHz.

Figure P4-7



Experimenting, as every good engineer should, we change the sampling rate to $f'_s = 19$ kHz, obtaining a new discrete sequence $x_2(n)$. Performing an FFT on $x_2(n)$, we obtain the $|X_2(m)|$ FFT

magnitude results presented in [Figure P4-7\(b\)](#). In our new spectral results we see our signal of interest remains in the frequency range of 0 to 4 kHz, but the narrowband spectral noise signal is now centered near 4 kHz! (If this ever happens to you in practice, to quote Veronica in the 1986 movie *The Fly*, “Be afraid. Be very afraid.”) Describe the characteristic of the analog $x(t)$ that would account for the unexpected shift in center frequency of the narrowband noise in the $|X_2(m)|$ FFT results.

4.8 In the text’s derivation of the radix-2 FFT, to simplify the algebraic notation we represented unity-magnitude complex numbers (what we called “twiddle factors”) in the following form:

$$\alpha = W_N^k.$$

If $k = 3$ and $N = 16$:

(a) Express α as a complex number in polar (complex exponential) form.

(b) Express α as a complex number in rectangular form.

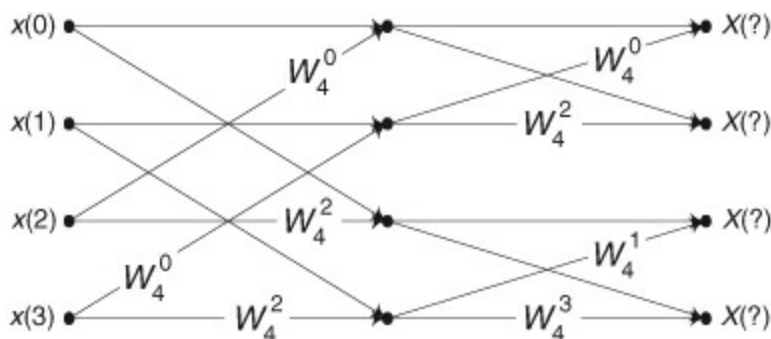
4.9 Reviewing the 8-point FFT signal-flow diagram in the text’s [Figure 4-5](#):

(a) Which $x(n)$ input samples affect the value of the FFT’s $X(2)$ output sample?

(b) Which $x(n)$ input samples affect the value of the FFT’s $X(5)$ output sample?

4.10 [Figure P4-10](#) shows a 4-point FFT using standard decimation-in-time butterflies. Redraw that FFT using *optimized* decimation-in-time butterflies as shown in the text’s [Figure 4-14\(c\)](#). In your drawing provide the correct indices for the $X(m)$ output samples.

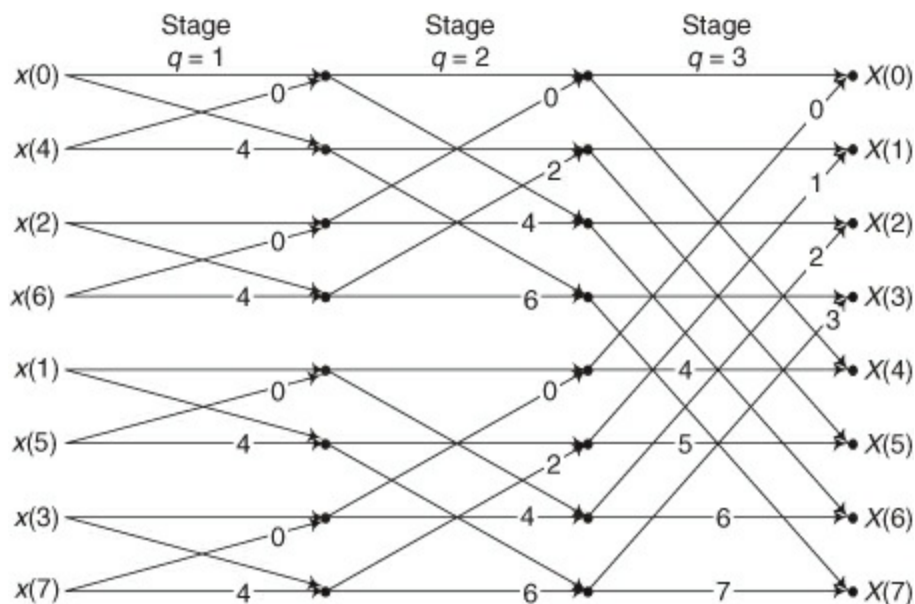
Figure P4-10



4.11 Being able to compute individual twiddle factors within an FFT can be important when implementing specialized FFTs, such as *pruned* FFTs. (Pruned FFTs are FFTs where we need not compute all N FFT output samples[Pruned FFT–1-Pruned FFT 4]). [Figure P4-11](#) shows the signal-flow diagram of a standard 8-point *decimation-in-time* (DIT) FFT with bit-reversed inputs. As in the text’s [Figure 4-8](#), the number on an arrow is the integer k of a butterfly’s

$$W_8^k = e^{-j2\pi k/8}$$

Figure P4-11



twiddle factor. Notice that the number of unique twiddle factors is different in each of the three stages. The values of the R unique twiddle factors in the q th stage of a general N -point DIT FFT are given by

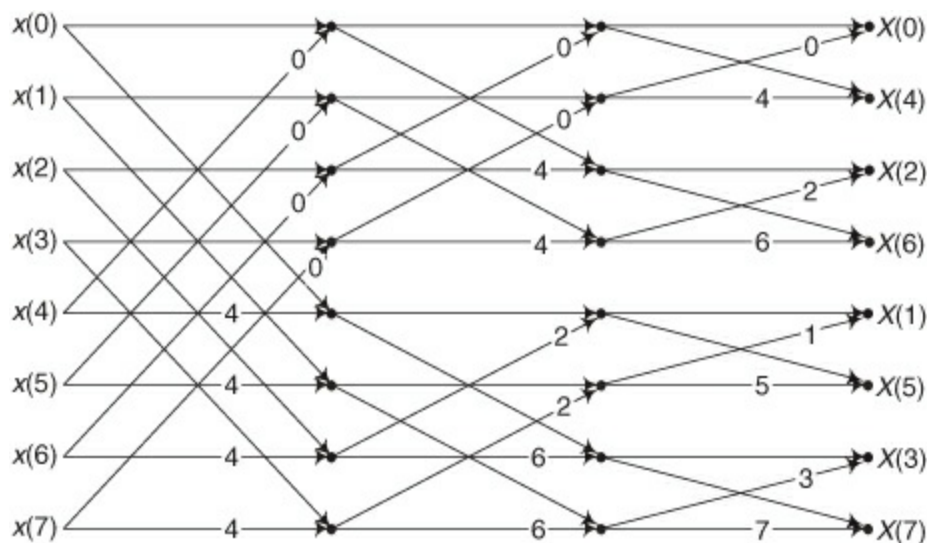
$$k\text{th twiddle factor of } q\text{th stage} = W_N^{kN/P}, \text{ for } k = 0, 1, 2, \dots, R - 1.$$

What are the expressions for the above R and P factors in terms of the FFT's q stage number?

Hint: Use the 8-point FFT in [Figure P4-11](#) as a guide to find R and P .

4.12 Let's become more familiar with the interesting internal computations of a radix-2 FFT. [Figure P4-12](#) shows the signal-flow diagram of a standard 8-point *decimation-in-time* FFT with bit-reversed outputs. In that figure, as in the text's [Figure 4-9](#), the number on an arrow is the integer k of a butterfly's $e^{-j2\pi k/8}$ twiddle factor.

Figure P4-12



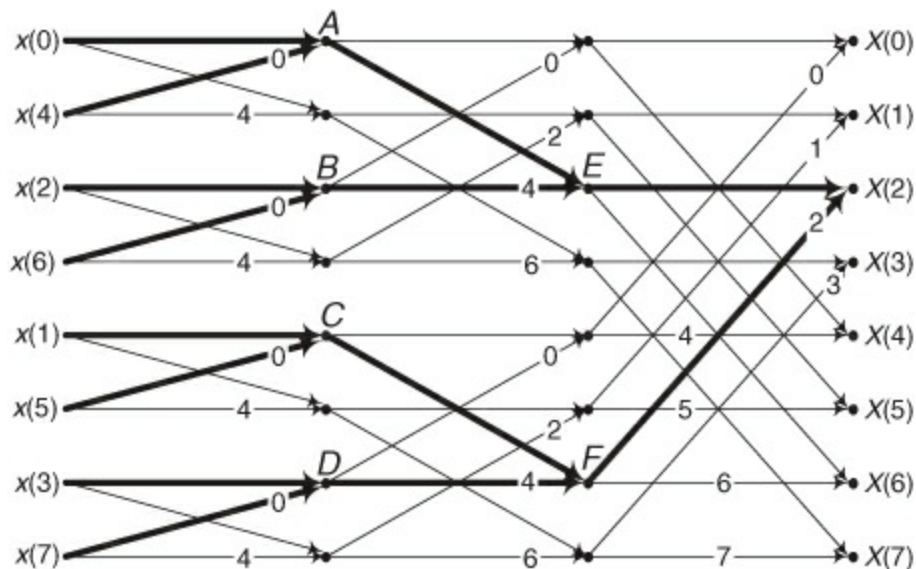
- Redraw [Figure P4-12](#), replacing the k factors with the butterflies' full complex twiddle factors in rectangular notation.
- Regarding your solution to the above Part (a), comment on any interesting properties of the twiddle factors in the FFT's first and second stages.

4.13 To reiterate the meaning and correctness of the FFT butterfly structures in the text, we examine

the 8-point *decimation-in-time* FFT with bit-reversed inputs. That FFT, the text's [Figure 4-8](#) repeated here as [Figure P4-13](#), uses our notation where a number on an arrow is the integer k of a butterfly's $e^{-j2\pi k/8}$ twiddle factor. Compute the values at sample nodes A through F , in terms of the $x(n)$ input samples, and show that the FFT's $X(2)$ output is equal to a DFT's output for $m = 2$ in

$$X(m)|_{m=2} = \sum_{n=0}^{N-1} x(n)e^{-j2\pi mn/N}|_{m=2} = \sum_{n=0}^7 x(n)e^{-j2\pi 2n/8}.$$

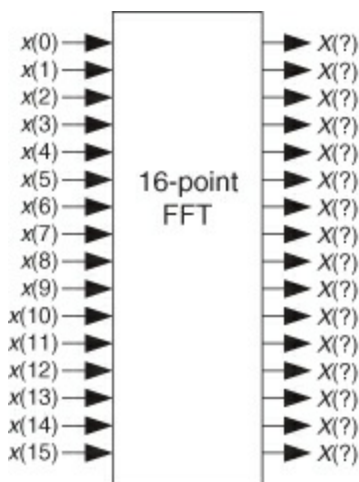
Figure P4-13



Hint: To keep the notation simple, use the term W^q to represent $e^{-j2\pi q/8}$.

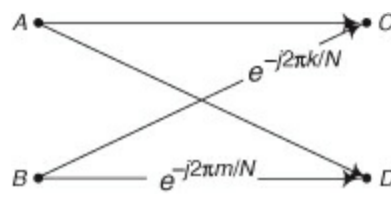
4.14 Consider the 16-point decimation-in-time FFT in [Figure P4-14](#) that is implemented in a similar manner to that shown in the text's [Figure 4-9](#). This FFT has *in-order* input data indexing. That is, the $x(n)$ input indexing is in normal numerical order from $x(0)$ to $x(15)$. What will be the order of the frequency-domain indexing of the $X(m)$ output samples for this 16-point radix-2 FFT?

Figure P4-14



4.15 Is it possible to examine the signal-flow diagram of a single standard butterfly, such as that in [Figure P4-15](#), and determine if it is a decimation-in-time (DIT) butterfly or a decimation-in-frequency (DIF) butterfly? Justify your answer.

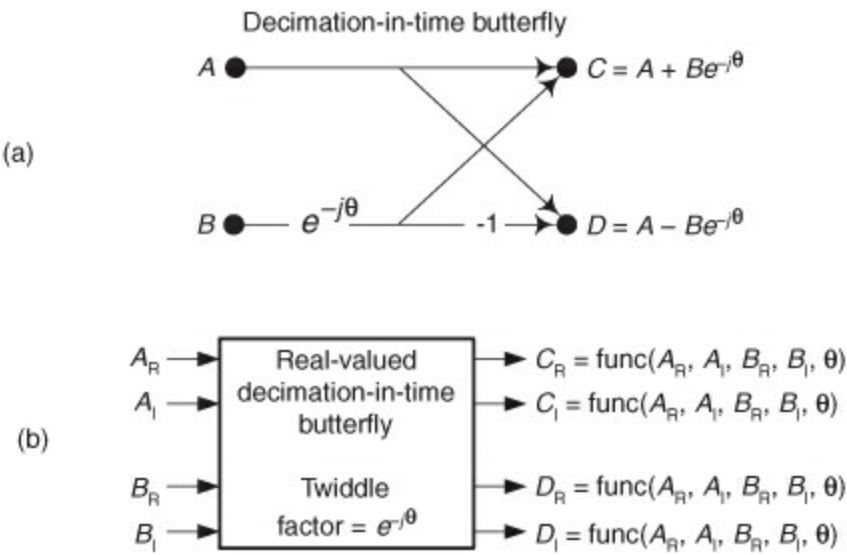
Figure P4-15



4.16 Let's explore the internal structure of a single radix-2 FFT butterfly. [Figure P4-16\(a\)](#) shows our standard notation for a decimation-in-time butterfly where the input and output samples (A , B , C , and D) are complex-valued. [Figure P4-16\(b\)](#) shows the same decimation-in-time butterfly where the input and output values are represented by real-valued samples. We use the notation that

$$A = A_R + jA_I$$

Figure P4-16



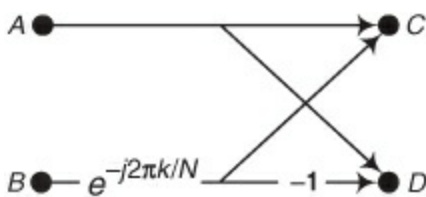
where A_R and A_I are real-valued. Draw the real-valued block diagram of what arithmetic is performed inside the rectangle in [Figure P4-16\(b\)](#). Be sure to include in your diagram the expressions (the equations) for the real-valued C_R , C_I , D_R , and D_I output samples in terms of the real-valued A_R , A_I , B_R , and B_I input samples and the twiddle factor angle θ . The solution to this problem illustrates the computational complexity of performing a single FFT butterfly.

4.17 Here's a problem that has much practical importance. It concerns the data word growth that can occur inside an FFT.

For this problem, our assumptions are:

- We are implementing an FFT using the optimized decimation-in-time FFT butterfly structure, shown in [Figure P4-17](#), to compute intermediate results.

Figure P4-17



- The complex data samples A and B are contained in 8-bit storage locations using the *sign-magnitude* number format system. (In that number format the most positive and most negative decimal numbers we can store, as binary words in an 8-bit-wide memory location, are +127 and

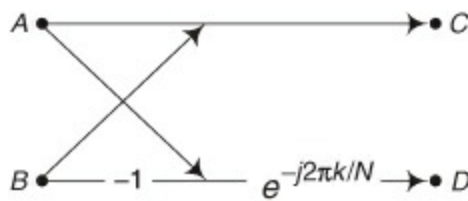
−127 respectively.)

It's difficult at first to imagine that multiplying complex samples A and B by sines and cosines (the real and imaginary parts of $e^{-j2\pi k/N}$) can lead to excessive data word growth—particularly because sines and cosines are never greater than unity. However, significant data word growth can happen within an FFT butterfly.

- (a) In our 8-bit number format scenario, what is the maximum possible decimal value of the real part of the complex output sample C ?
- (b) How many binary bits are needed for a storage register (memory location) to hold that maximum real part of the complex output sample C ?

4.18 In 2006 the scientists at the Max Planck Institute for Radio Astronomy, in Bonn, Germany, built a hardware spectrum analyzer that performs 16384-point FFTs. This massively parallel analyzer performs 1.744×10^5 such FFTs per second. Assuming that the FFTs use the optimized decimation-in-frequency FFT butterfly structure, shown in [Figure P4-18](#), and that the A and B samples are complex-valued, how many real-valued multiplies per second are being performed by the spectrum analyzer? Show your work.

Figure P4-18



References

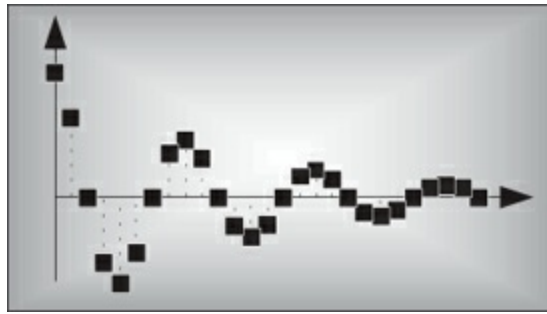
[Pruned FFT 1] Nagai, K. “Pruning the Decimation-in-Time FFT Algorithm with Frequency Shift,” *IEEE Trans. on ASSP*, Vol. ASSP-34, August 1986, pp. 1008–1010.

[Pruned FFT 2] Skinner, D. “Pruning the Decimation-in-Time FFT Algorithm,” *IEEE Trans. on ASSP*, Vol. ASSP-24, April 1976, pp. 193–194.

[Pruned FFT 3] Markel, J. D. “FFT Pruning,” *IEEE Trans. on Audio Electroacoustics*, Vol. AU-19, December 1971, pp. 305–311.

[Pruned FFT 4] Sreenivas, T., and Rao, P. “FFT Algorithm for Both Input and Output Pruning,” *IEEE Trans. on ASSP*, Vol. ASSP-27, June 1979, pp. 291–292.

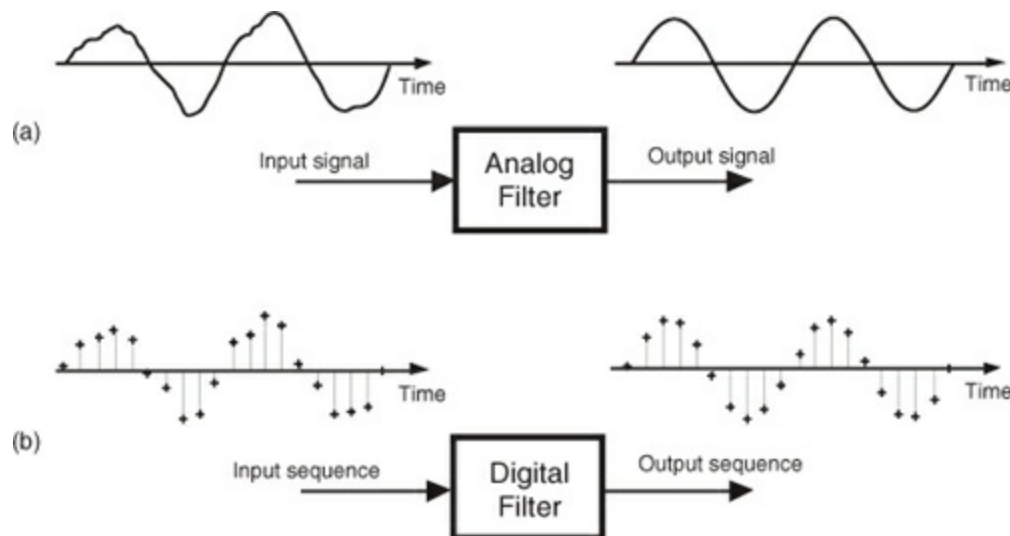
Chapter Five. Finite Impulse Response Filters



The filtering of digitized data, if not the most fundamental, is certainly the oldest discipline in the field of digital signal processing. Digital filtering's origins go back 50 years. The growing availability of digital computers in the early 1950s led to efforts in the smoothing of discrete sampled data and the analysis of discrete data control systems. However, it wasn't until the early to mid-1960s, around the time the Beatles came to America, that the analysis and development of digital equivalents of analog filters began in earnest. That's when digital signal processing experts realized that computers could go beyond the mere analysis of digitized signals into the domain of actually changing signal characteristics through filtering. Today, digital filtering is so widespread that the quantity of literature pertaining to it exceeds that of any other topic in digital signal processing. In this chapter, we introduce the fundamental attributes of digital filters, learn how to quantify their performance, and review the principles associated with the design of finite impulse response digital filters.

So let's get started by illustrating the concept of filtering a time-domain signal as shown in [Figure 5-1](#).

Figure 5-1 Filters: (a) an analog filter with a noisy tone input and a reduced-noise tone output; (b) the digital equivalent of the analog filter.



In general, filtering is the processing of a time-domain signal resulting in some change in that signal's original spectral content. The change is usually the reduction, or filtering out, of some unwanted input spectral components; that is, filters allow certain frequencies to pass while attenuating other frequencies. [Figure 5-1](#) shows both analog and digital versions of a filtering process. Where an analog filter operates on a continuous signal, a digital filter processes a sequence of discrete sample values. The digital filter in [Figure 5-1\(b\)](#), of course, can be a software program in a computer, a

programmable hardware processor, or a dedicated integrated circuit. Traditional linear digital filters typically come in two flavors: *finite impulse response* (FIR) filters and *infinite impulse response* (IIR) filters. Because FIR filters are the simplest type of digital filter to analyze, we'll examine them in this chapter and cover IIR filters in [Chapter 6](#).

5.1 An Introduction to Finite Impulse Response (FIR) Filters

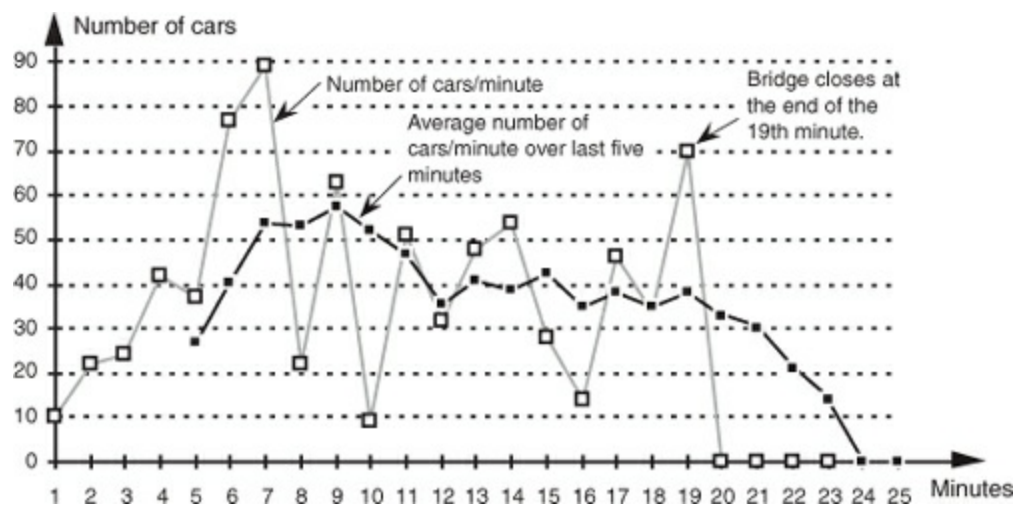
Given a finite duration of nonzero input values, an FIR filter will always have a finite duration of nonzero output values, and that's how FIR filters got their name. So, if the FIR filter's input suddenly becomes a sequence of all zeros, the filter's output will eventually be all zeros. While not sounding all that unusual, this characteristic is, however, very important, and we'll soon find out why, as we learn more about digital filters.

FIR filters use addition to calculate their outputs in a manner much the same as the process of averaging uses addition. In fact, averaging is a kind of FIR filter that we can illustrate with an example. Let's say we're counting the number of cars that pass over a bridge every minute, and we need to know the average number of cars per minute over five-minute intervals; that is, every minute we'll calculate the average number of cars/minute over the last five minutes. If the results of our car counting for the first ten minutes are those values shown in the center column of [Table 5-1](#), then the average number of cars/minute over the previous five one-minute intervals is listed in the right column of the table. We've added the number of cars for the first five one-minute intervals and divided by 5 to get our first five-minute average output value, $(10+22+24+42+37)/5 = 27$. Next we've averaged the number of cars/minute for the second to the sixth one-minute intervals to get our second five-minute average output of 40.4. Continuing, we average the number of cars/minute for the third to the seventh one-minute intervals to get our third average output of 53.8, and so on. With the number of cars/minute for the one-minute intervals represented by the dashed line in [Figure 5-2](#), we show our five-minute average output as the solid line. ([Figure 5-2](#) shows cars/minute input values beyond the first ten minutes listed in [Table 5-1](#) to illustrate a couple of important ideas to be discussed shortly.)

Table 5-1 Values for the Averaging Example

Minute index	Number of cars/minute over the last minute	Number of cars/minute averaged over the last five minutes
1	10	—
2	22	—
3	24	—
4	42	—
5	37	27
6	77	40.4
7	89	53.8
8	22	53.4
9	63	57.6
10	9	52

Figure 5-2 Averaging the number of cars/minute. The dashed line shows the individual cars/minute, and the solid line is the number of cars/minute averaged over the last five minutes.



There's much to learn from this simple averaging example. In [Figure 5-2](#), notice that the sudden changes in our input sequence of cars/minute are flattened out by our averager. The averager output sequence is considerably smoother than the input sequence. Knowing that sudden transitions in a time sequence represent high-frequency components, we can say that our averager is behaving like a lowpass filter and smoothing sudden changes in the input. Is our averager an FIR filter? It sure is—no previous averager output value is used to determine a current output value; only input values are used to calculate output values. In addition, we see that, if the bridge were suddenly closed at the end of the 19th minute, the dashed line immediately goes to zero cars/minute at the end of the 20th minute, and the averager's output in [Figure 5-2](#) approaches and settles to a value of zero by the end of the 24th minute.

[Figure 5-2](#) shows the first averager output sample occurring at the end of the 5th minute because that's when we first have five input samples to calculate a valid average. The 5th output of our averager can be denoted as $y_{ave}(5)$ where

(5-1)

$$y_{ave}(5) = \frac{1}{5} [x(1) + x(2) + x(3) + x(4) + x(5)].$$

In the general case, if the k th input sample is $x(k)$, then the n th output is

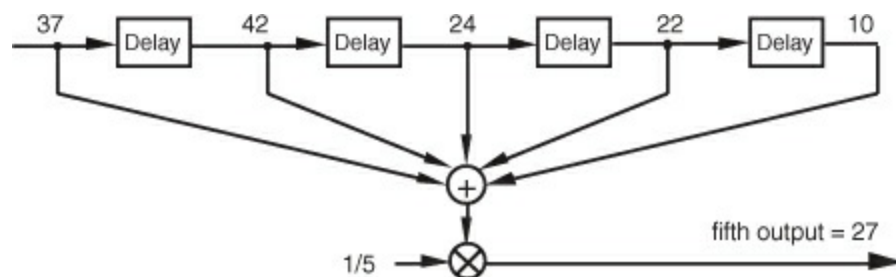
(5-2)

$$y_{ave}(n) = \frac{1}{5} [x(n-4) + x(n-3) + x(n-2) + x(n-1) + x(n)] = \frac{1}{5} \sum_{k=n-4}^n x(k).$$

Look at [Eq. \(5-2\)](#) carefully now. It states that the n th output is the average of the n th input sample and the four previous input samples.

We can formalize the digital filter nature of our averager by creating the block diagram in [Figure 5-3](#) showing how the averager calculates its output samples.

Figure 5-3 Averaging filter block diagram when the fifth input sample value, 37, is applied.



This block diagram, referred to as the filter [structure](#), is a physical depiction of how we might calculate our averaging filter outputs with the input sequence of values shifted, in order, from left to right along the top of the filter as new output calculations are performed. This structure, implementing [Eqs. \(5-1\)](#) and [\(5-2\)](#), shows those values used when the first five input sample values are available. The delay elements in [Figure 5-3](#), called *unit delays*, merely indicate a shift register arrangement where input sample values are temporarily stored during an output calculation.

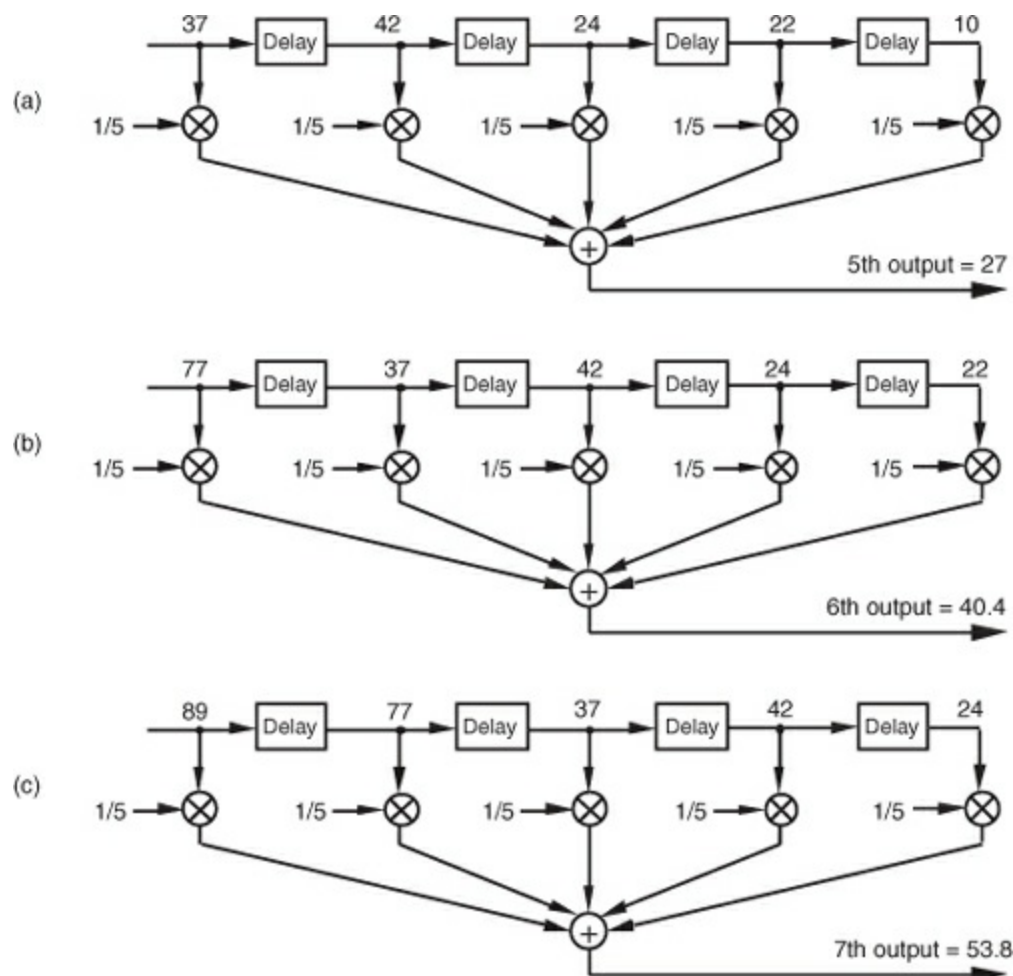
In averaging, we add five numbers and divide the sum by 5 to get our answer. In a conventional FIR filter implementation, we can just as well multiply each of the five input samples by the coefficient $1/5$ and then perform the summation as shown in [Figure 5-4\(a\)](#). Of course, the two methods in [Figures 5-3](#) and [5-4\(a\)](#) are equivalent because [Eq. \(5-2\)](#) describing the structure shown in [Figure 5-3](#) is equivalent to

$$(5-3)$$

$$y_{\text{ave}}(n) = \frac{1}{5}x(n-4) + \frac{1}{5}x(n-3) + \frac{1}{5}x(n-2) + \frac{1}{5}x(n-1) + \frac{1}{5}x(n)$$

$$= \sum_{k=n-4}^n \frac{1}{5}x(k),$$

Figure 5-4 Alternate averaging filter structure: (a) input values used for the fifth output value; (b) input values used for the sixth output value; (c) input values used for the seventh output value.



which describes the structure in [Figure 5-4\(a\)](#).[†]

[†] We've used the venerable distributive law for multiplication and addition of scalars, $a(b+c+d) = ab+ac+ad$, in moving [Eq. \(5-2\)](#)'s factor of $1/5$ inside the summation in [Eq. \(5-3\)](#).

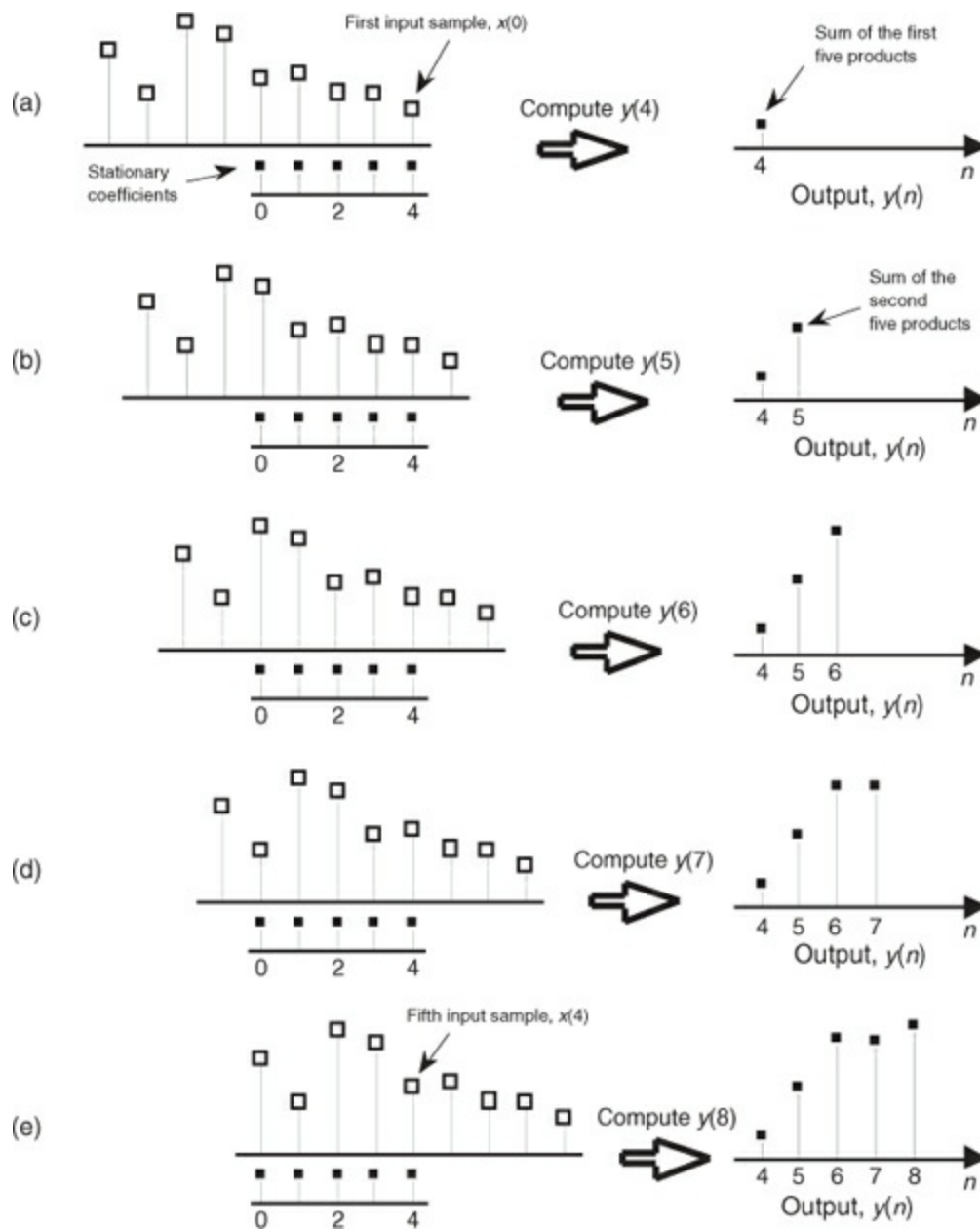
Let's make sure we understand what's happening in [Figure 5-4\(a\)](#). Each of the first five input values is multiplied by $1/5$, and the five products are summed to give the fifth filter output value. The left-to-right sample shifting is illustrated in [Figures 5-4\(b\)](#) and [5-4\(c\)](#). To calculate the filter's sixth output value, the input sequence is right-shifted, discarding the first input value of 10, and the sixth input value, 77, is accepted on the left. Likewise, to calculate the filter's seventh output value, the input sequence is right-shifted, discarding the second value of 22, and the seventh input value, 89, arrives on the left. So, when a new input sample value is applied, the filter discards the oldest sample value, multiplies the samples by the coefficients of $1/5$, and sums the products to get a single new output value. The filter's structure using this bucket brigade shifting process is often called a [*transversal filter*](#) due to the cross-directional flow of the input samples. Because we *tap off* five separate input sample values to calculate an output value, the structure in [Figure 5-4](#) is called a 5-tap tapped-delay line FIR filter, in digital filter vernacular.

One important and, perhaps, most interesting aspect of understanding FIR filters is learning how to predict their behavior when sinusoidal samples of various frequencies are applied to the input, i.e., how to estimate their frequency-domain response. Two factors affect an FIR filter's frequency response: the number of taps and the specific values used for the multiplication coefficients. We'll explore these two factors using our averaging example and, then, see how we can use them to design FIR filters. This brings us to the point where we have to introduce the C word: *convolution*. (Actually, we already slipped a convolution equation in on the reader without saying so. It was [Eq. \(5-3\)](#), and we'll examine it in more detail later.)

5.2 Convolution in FIR Filters

OK, here's where we get serious about understanding the mathematics behind FIR filters. We can graphically depict [Eq. \(5-3\)](#)'s and [Figure 5-4](#)'s calculations as shown in [Figure 5-5](#). Also, let's be formal and use the standard notation of digital filters for indexing the input samples and the filter coefficients by starting with an initial index value of zero; that is, we'll call the initial input value the 0th sample $x(0)$. The next input sample is represented by the term $x(1)$, the following input sample is called $x(2)$, and so on. Likewise, our five coefficient values will be indexed from zero to four, $h(0)$ through $h(4)$. (This indexing scheme makes the equations describing our example consistent with conventional filter notation found in the literature.)

Figure 5-5 Averaging filter convolution: (a) first five input samples aligned with the stationary filter coefficients, index $n = 4$; (b) input samples shift to the right and index $n = 5$; (c) index $n = 6$; (d) index $n = 7$; (e) index $n = 8$.



In [Eq. \(5-3\)](#) we used the factor of $1/5$ as the filter coefficients multiplied by our averaging filter's input samples. The left side of [Figure 5-5](#) shows the alignment of those coefficients, black squares, with the filter input sample values represented by the white squares. Notice in [Figures 5-5\(a\)](#) through [5-5\(e\)](#) that we're marching the input samples to the right, and, at each step, we calculate the filter output sample value using [Eq. \(5-3\)](#). The output samples on the right side of [Figure 5-5](#) match the first five values represented by the black squares in [Figure 5-2](#). The input samples in [Figure 5-5](#) are those values represented by the white squares in [Figure 5-2](#). Notice that the time order of the inputs in [Figure 5-5](#) has been reversed from the input sequence order in [Figure 5-2](#)! That is, the input sequence has been flipped in the time domain in [Figure 5-5](#). This time order reversal is what happens to the input data using the filter structure in [Figure 5-4](#).

Repeating the first part of [Eq. \(5-3\)](#) and omitting the subscript on the output term, our original FIR filter's $y(n)$ th output is given by

$$(5-4)$$

$$y(n) = \frac{1}{5}x(n-4) + \frac{1}{5}x(n-3) + \frac{1}{5}x(n-2) + \frac{1}{5}x(n-1) + \frac{1}{5}x(n).$$

Because we'll explore filters whose coefficients are not all the same value, we need to represent the individual filter coefficients by a variable, such as the term $h(k)$, for example. Thus we can rewrite

the averaging filter's output from [Eq. \(5-4\)](#) in a more general way as

(5-5)

$$y(n) = h(4)x(n-4) + h(3)x(n-3) + h(2)x(n-2) + h(1)x(n-1) + h(0)x(n)$$

$$= \sum_{k=0}^4 h(k)x(n-k),$$

where $h(0)$ through $h(4)$ all equal $1/5$. [Equation \(5-5\)](#) is a concise way of describing the filter structure in [Figure 5-4](#) and the process illustrated in [Figure 5-5](#).

Let's take [Eq. \(5-5\)](#) one step further and say, for a general M -tap FIR filter, the n th output is

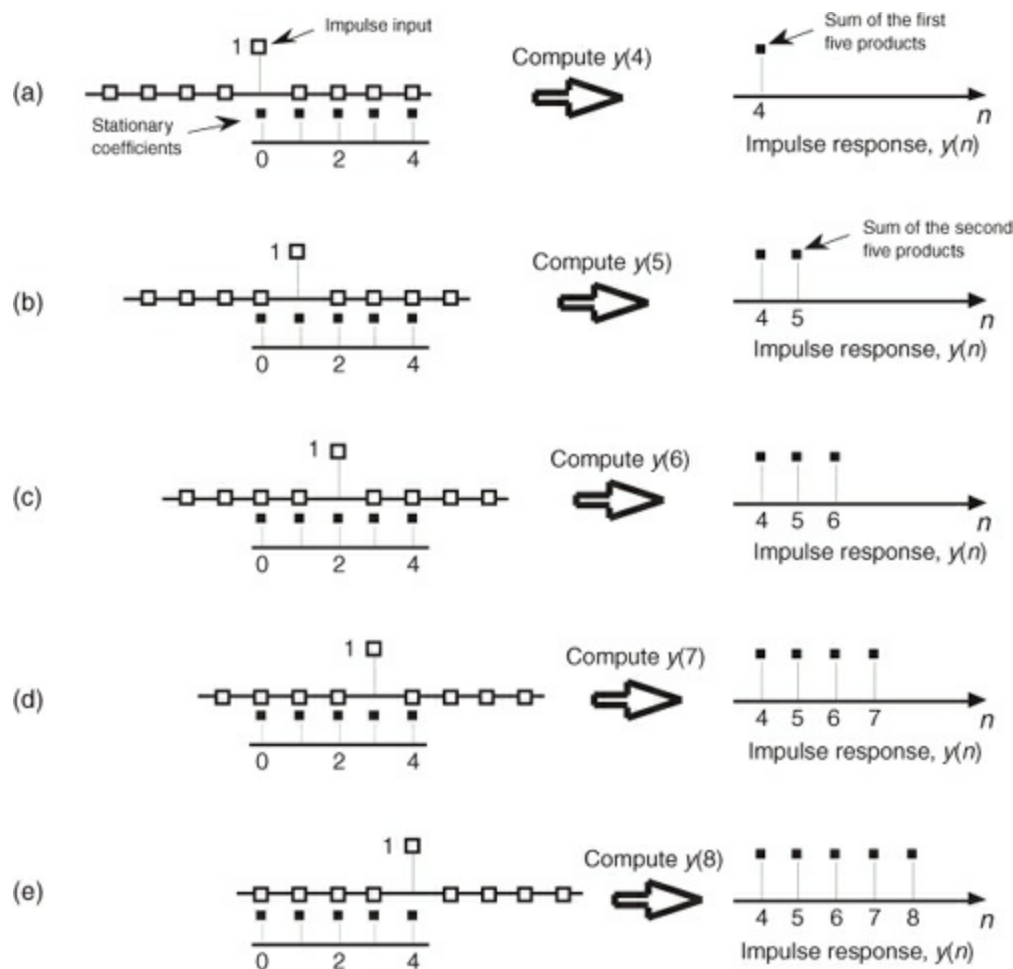
(5-6)

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k).$$

Well, there it is. [Eq. \(5-6\)](#) is the infamous convolution equation as it applies to digital FIR filters. Beginners in the field of digital signal processing often have trouble understanding the concept of convolution. It need not be that way. [Eq. \(5-6\)](#) is merely a series of multiplications followed by the addition of the products. The process is actually rather simple. We just flip the time order of an input sample sequence and start stepping the flipped sequence across the filter's coefficients as shown in [Figure 5-5](#). For each new filter input sample, we sum a series of products to compute a single filter output value.

Let's pause for a moment and introduce a new term that's important to keep in mind, the [*impulse response*](#). The impulse response of a filter is exactly what its name implies—it's the filter's output time-domain sequence when the input is a single unity-valued sample (impulse) preceded and followed by zero-valued samples. [Figure 5-6](#) illustrates this idea in the same way we determined the filter's output sequence in [Figure 5-5](#). The left side of [Figure 5-6](#) shows the alignment of the filter coefficients, black squares, with the filter input impulse sample values represented by the white squares. Again, in [Figures 5-6\(a\)](#) through [5-6\(e\)](#) we're shifting the input samples to the right, and, at each step, we calculate the filter output sample value using [Eq. \(5-4\)](#). The output samples on the right side of [Figure 5-6](#) are the filter's impulse response. Notice the key point here: the FIR filter's impulse response is identical to the five filter coefficient values. For this reason, the terms *FIR filter coefficients* and [*impulse response*](#) are synonymous. Thus, when someone refers to the impulse response of an FIR filter, they're also talking about the coefficients. Because there are a finite number of coefficients, the impulse response will be finite in time duration (finite impulse response, FIR).

Figure 5-6 Convolution of filter coefficients and an input impulse to obtain the filter's output impulse response: (a) impulse sample aligned with the first filter coefficient, index $n = 4$; (b) impulse sample shifts to the right and index $n = 5$; (c) index $n = 6$; (d) index $n = 7$; (e) index $n = 8$.



Returning to our averaging filter, recall that coefficients (or impulse response) $h(0)$ through $h(4)$ were all equal to $1/5$. As it turns out, our filter's performance can be improved by using coefficients whose values are not all the same. By "performance" we mean how well the filter passes desired signals and attenuates unwanted signals. We judge that performance by determining the shape of the filter's frequency-domain response that we obtain by the convolution property of linear systems. To describe this concept, let's repeat [Eq. \(5-6\)](#) using the abbreviated notation of

$$(5-7)$$

$$y(n) = h(k) * x(n)$$

where the $*$ symbol means convolution. ([Equation 5-7](#) is read as "y of n equals the convolution of h of k and x of n.") The process of convolution, as it applies to FIR filters, is as follows: the discrete Fourier transform (DFT) of the convolution of a filter's impulse response (coefficients) and an input sequence is equal to the product of the spectrum of the input sequence and the DFT of the impulse response. The idea we're trying to convey here is that if two time-domain sequences $h(k)$ and $x(n)$ have DFTs of $H(m)$ and $X(m)$, respectively, then the DFT of $y(n) = h(k) * x(n)$ is $H(m) \cdot X(m)$. Making this point in a more compact way, we state this relationship with the expression

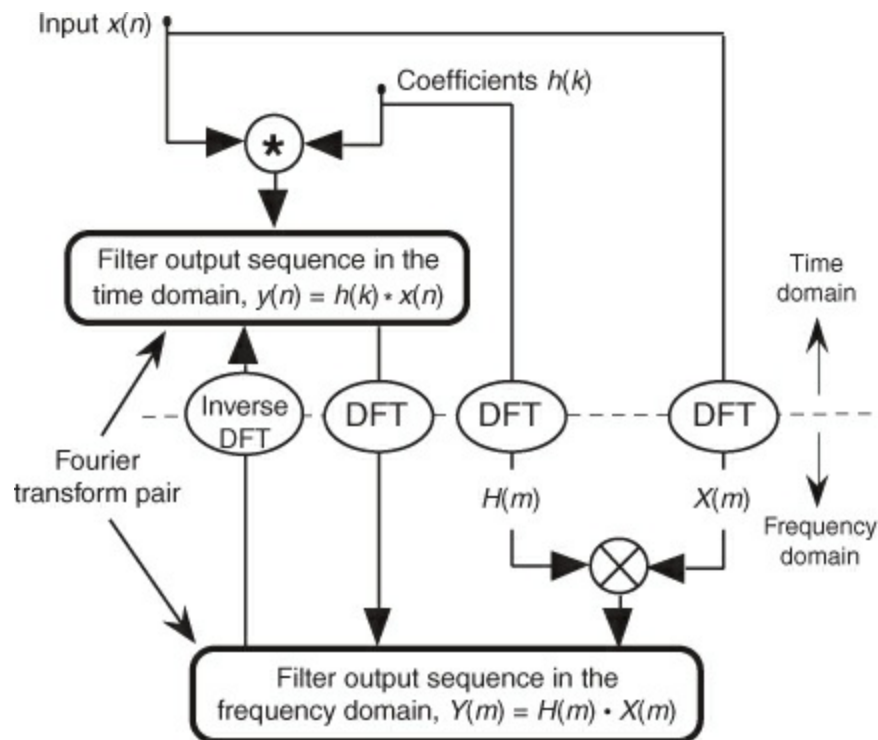
$$(5-8)$$

$$y(n) = h(k) * x(n) \xrightleftharpoons[\text{IDFT}]{\text{DFT}} H(m) \cdot X(m).$$

With IDFT indicating the inverse DFT, [Eq. \(5-8\)](#) indicates that two sequences resulting from $h(k)*x(n)$ and $H(m)\cdot X(m)$ are Fourier transform pairs. So taking the DFT of $h(k)*x(n)$ gives us the product $H(m)\cdot X(m)$ that is the spectrum of our filter output $Y(m)$. Likewise, we can determine $h(k)*x(n)$ by taking the inverse DFT of $H(m)\cdot X(m)$. The very important conclusion to learn from [Eq.](#)

(5-8) is that convolution in the time domain is equivalent to multiplication in the frequency domain. To help us appreciate this principle, [Figure 5-7](#) sketches the relationship between convolution in the time domain and multiplication in the frequency domain. The process of convolution with regard to linear systems is discussed in more detail in [Section 5.9](#). The beginner is encouraged to review that material to get a general idea of why and when the convolution process can be used to analyze digital filters.

Figure 5-7 Relationships of convolution as applied to FIR digital filters.

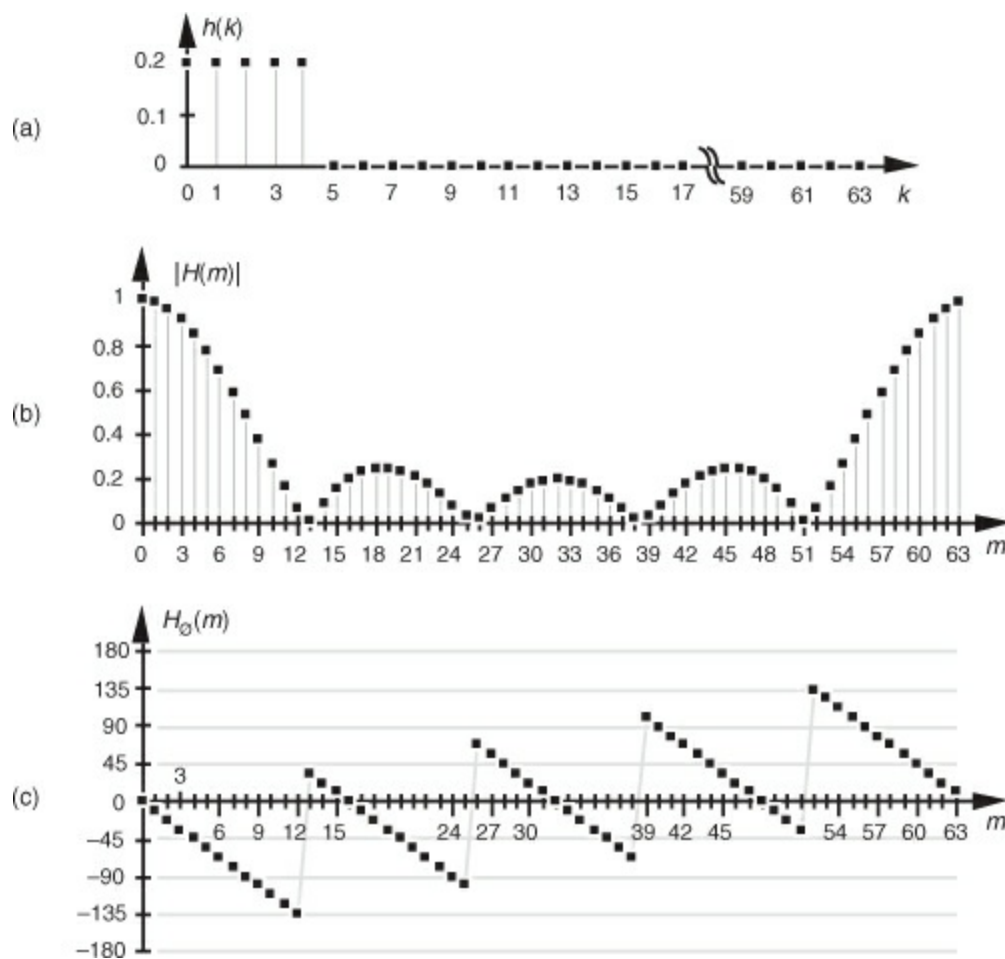


[Equation \(5-8\)](#) and the relationships in [Figure 5-7](#) tell us what we need to do to determine the frequency response of an FIR filter. The product $X(m) \cdot H(m)$ is the DFT of the filter output. Because $X(m)$ is the DFT of the filter's input sequence, the frequency response of the filter is then defined as $H(m)$, the DFT of the filter's impulse response $h(k)$.[†] Getting back to our original problem, we can determine our averaging filter's frequency-domain response by taking the DFT of the individual filter coefficients (impulse response) in [Eq. \(5-4\)](#). If we take the five $h(k)$ coefficient values of $1/5$ and append 59 zeros, we have the sequence depicted in [Figure 5-8\(a\)](#). Performing a 64-point DFT on that sequence, and normalizing the DFT magnitudes, gives us the filter's frequency magnitude response $|H(m)|$ in [Figure 5-8\(b\)](#) and phase response shown in [Figure 5-8\(c\)](#).^{††} $H(m)$ is our old friend, the $\sin(x)/x$ function from [Section 3.13](#).

[†] We use the term *impulse response* here, instead of *coefficients*, because this concept also applies to IIR filters. IIR filter frequency responses are also equal to the DFT of their impulse responses.

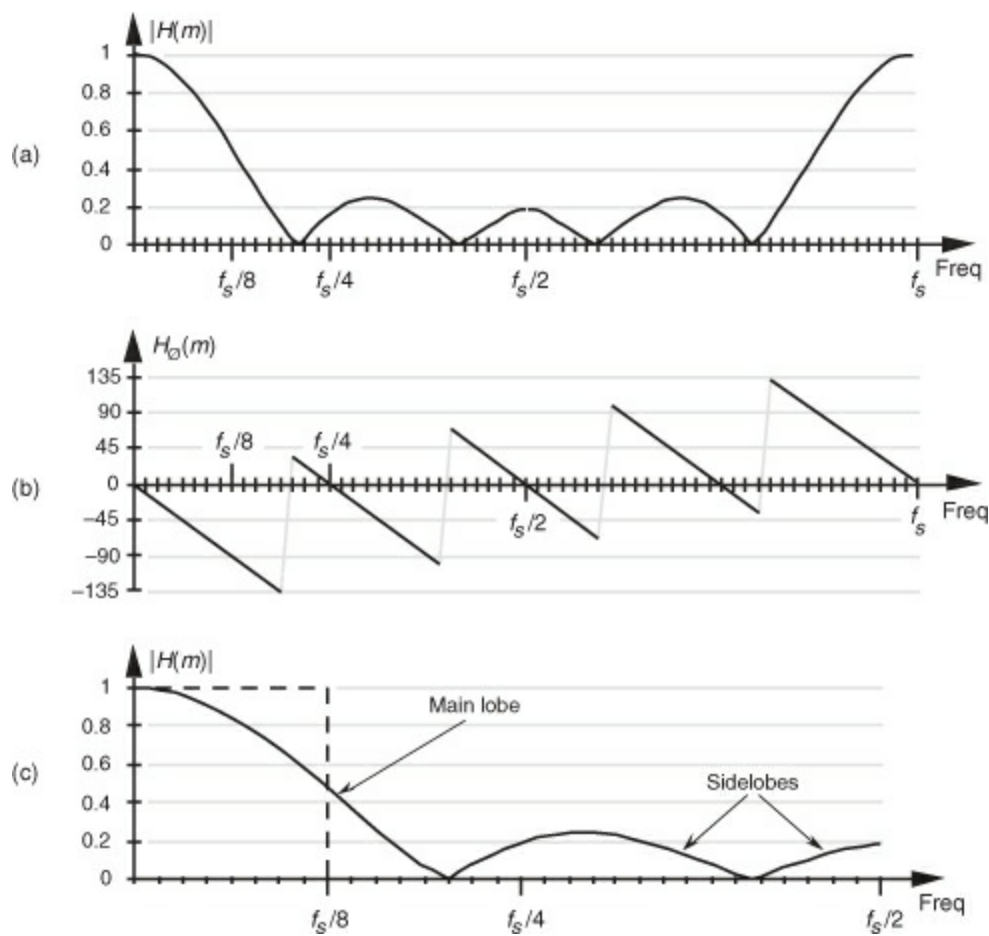
^{††} There's nothing sacred about using a 64-point DFT here. We could just as well have appended only enough zeros to take a 16- or 32-point FFT. We chose 64 points to get a frequency resolution that would make the shape of the response in [Figure 5-8\(b\)](#) reasonably smooth. Remember, the more points in the FFT, the finer the frequency granularity—right?

Figure 5-8 Averaging FIR filter: (a) filter coefficient sequence $h(k)$ with appended zeros; (b) normalized discrete frequency magnitude response $|H(m)|$ of the $h(k)$ filter coefficients; (c) phase-angle response of $H(m)$ in degrees.



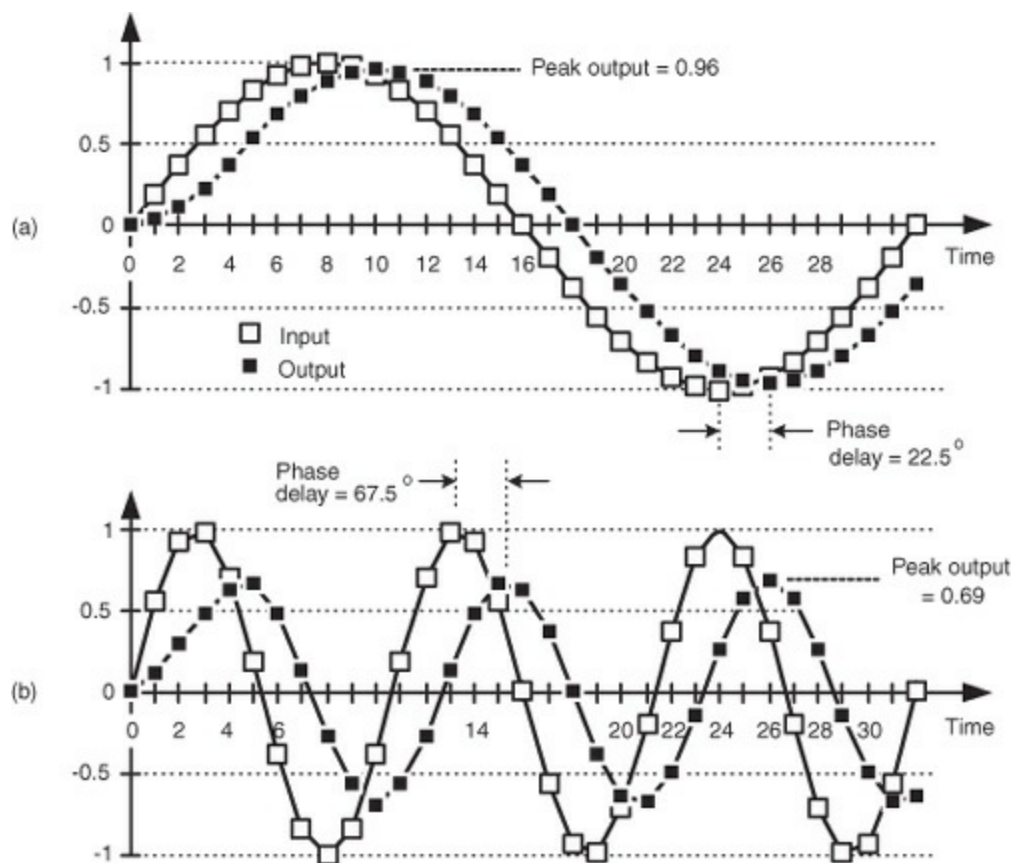
Let's relate the discrete frequency response samples in [Figures 5-8\(b\)](#) and [5-8\(c\)](#) to the physical dimension of the sample frequency f_s . We know, from [Section 3.5](#) and our experience with the DFT, that the $m = N/2$ discrete frequency sample, $m = 32$ in this case, is equal to the folding frequency, or half the sample rate, $f_s/2$. Keeping this in mind, we can convert the discrete frequency axis in [Figure 5-8](#) to that shown in [Figure 5-9](#). In [Figure 5-9\(a\)](#), notice that the filter's magnitude response is, of course, periodic in the frequency domain with a period of the equivalent sample rate f_s . Because we're primarily interested in the filter's response between 0 and half the sample rate, [Figure 5-9\(c\)](#) shows that frequency band in greater detail, affirming the notion that averaging behaves like a lowpass filter. It's a relatively poor lowpass filter compared to an arbitrary, *ideal* lowpass filter indicated by the dashed lines in [Figure 5-9\(c\)](#), but our averaging filter will attenuate higher-frequency inputs relative to its response to low-frequency input signals.

Figure 5-9 Averaging FIR filter frequency response shown as continuous curves: (a) normalized frequency magnitude response, $|H(m)|$; (b) phase-angle response of $H(m)$ in degrees; (c) the filter's magnitude response between zero Hz and half the sample rate, $f_s/2$ Hz.



We can demonstrate this by way of example. Suppose we applied a low-frequency sinewave to a 5-point averaging FIR filter as shown by the white squares in [Figure 5-10\(a\)](#). The input sinewave's frequency is $f_s/32$ Hz and its peak amplitude is unity. The filter's output sequence is shown by the black squares.

Figure 5-10 Averaging FIR filter input and output responses: (a) with an input sinewave of frequency $f_s/32$; (b) with an input sinewave of frequency $3f_s/32$.



[Figure 5-10\(a\)](#) is rich in information! First, the filter's output is a sinewave of the same frequency as the input. This is a characteristic of a linear system. We apply a single sinewave input, and the output will be a single sinewave (shifted in phase and perhaps reduced in amplitude) of the same frequency as the input. Second, notice that the initial four output samples are not exactly sinusoidal. Those output samples are the *transient response* of the filter. With tapped-delay line FIR filters, the sample length of that transient response is equal to the number of filter unit-delay elements D , after which the filter's output begins its steady-state time response.

The above transient response property is important. It means that tapped-delay line FIR filter outputs are not valid until $D+1$ input samples have been applied to the filter. That is, the output samples are not valid until the filter's delay line is filled with input data. So, for an FIR filter having $D = 70$ unit-delay elements the first 70 output samples are not valid and would be ignored in practice. WARNING: There are tapped-delay line FIR filters, used in practice, that have more unit-delay elements than nonzero-valued tap coefficients. The transient response length for those filters, measured in samples, is equal to the number of unit-delay elements, D (and is unrelated to the number of nonzero-valued tap coefficients).

The filter's output sinewave peak amplitude is reduced to a value of 0.96 and the output sinewave is delayed from the input by a phase angle of 22.5 degrees. Notice that the time delay between the input and output sinewaves, in [Figure 5-10\(a\)](#), is two samples in duration. (Although we discuss this time delay topic in more detail later, for now we'll just say that, because the filter's coefficients are symmetrical, the input/output delay measured in samples is equal to half the number of unit-delay elements in the filter's tapped-delay line.)

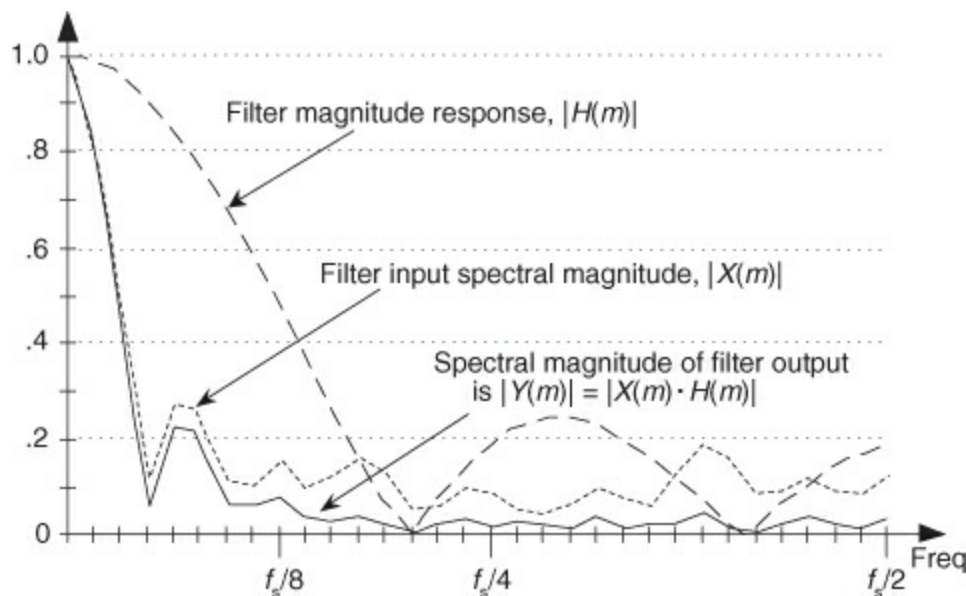
Next, if we applied a higher-frequency sinewave of $3f_s/32$ Hz to our 5-tap FIR filter as shown in [Figure 5-10\(b\)](#), the filter output is a sinewave of frequency $3f_s/32$ Hz and its peak amplitude is even further reduced to a value of 0.69. That's the nature of lowpass filters—they attenuate higher-frequency inputs more than they attenuate low-frequency inputs. As in [Figure 5-10\(a\)](#), the time delay

between the input and output sinewaves, in [Figure 5-10\(b\)](#), is two samples in duration (corresponding to a phase-angle delay of 67.5 degrees). That property, where the input/output delay does not depend on frequency, is a very beneficial property of FIR filters having symmetrical coefficients. We'll discuss this important issue again later in this chapter. In [Figure 5-10\(b\)](#) we see that the nonsinusoidal filter output transient response is even more obvious than it was in [Figure 5-10\(a\)](#).

Although the output amplitudes and phase delays in [Figure 5-10](#) were measured values from actually performing a 5-tap FIR filter process on the input sinewaves' samples, we could have obtained those amplitude and phase delay values directly from [Figures 5-8\(b\)](#) and [5-8\(c\)](#). The point is, we don't have to implement an FIR filter and apply various sinewave inputs to discover what its frequency response will be. We need merely take the DFT of the FIR filter's coefficients (impulse response) to determine the filter's frequency response as we did for [Figure 5-8](#).

[Figure 5-11](#) is another depiction of how well our 5-tap averaging FIR filter performs, where the dashed line is the filter's magnitude response $|H(m)|$, and the shaded line is the $|X(m)|$ magnitude spectrum of the filter's input values (the white squares in [Figure 5-2](#)). The solid line is the magnitude spectrum of the filter's output sequence, which is shown by the black squares in [Figure 5-2](#). So in [Figure 5-11](#), the solid output spectrum is the product of the dashed filter response curve and the shaded input spectrum, or $|X(m) \cdot H(m)|$. Again, we see that our averager does indeed attenuate the higher-frequency portion of the input spectrum.

Figure 5-11 Averaging FIR filter input magnitude spectrum, frequency magnitude response, and output magnitude spectrum.



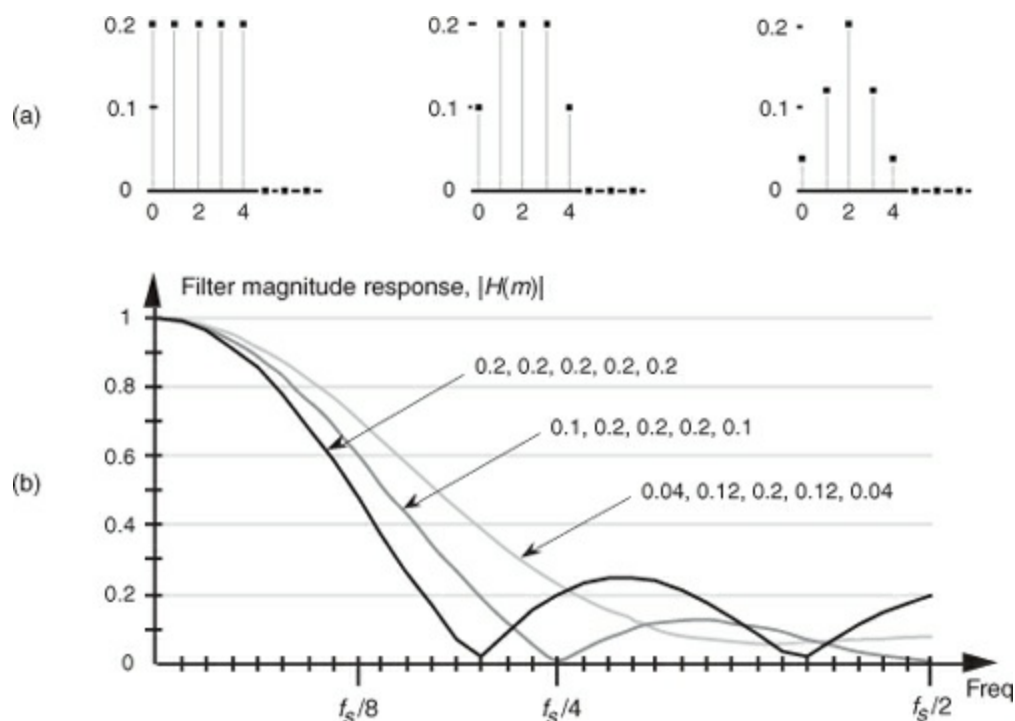
Let's pause for a moment to let all of this soak in a little. So far we've gone through the averaging filter example to establish that

- FIR filters perform time-domain convolution by summing the products of the shifted input samples and a sequence of filter coefficients,
- an FIR filter's output sequence is equal to the convolution of the input sequence and a filter's impulse response (coefficients),
- an FIR filter's frequency response is the DFT of the filter's impulse response,
- an FIR filter's output spectrum is the product of the input spectrum and the filter's frequency response, and

- convolution in the time domain and multiplication in the frequency domain are Fourier transform pairs.

OK, here's where FIR filters start to get really interesting. Let's change the values of the five filter coefficients to modify the frequency response of our 5-tap lowpass filter. In fact, [Figure 5-12\(a\)](#) shows our original five filter coefficients and two other arbitrary sets of 5-tap coefficients. [Figure 5-12\(b\)](#) compares the frequency magnitude responses of those three sets of coefficients. Again, the frequency responses are obtained by taking the DFT of the three individual sets of coefficients and plotting the magnitude of the transforms, as we did for [Figure 5-9\(c\)](#). So we see three important characteristics in [Figure 5-12](#). First, as we expected, different sets of coefficients give us different frequency magnitude responses. Second, a sudden change in the values of the coefficient sequence, such as the 0.2 to 0 transition in the first coefficient set, causes ripples, or sidelobes, in the frequency response. Third, if we minimize the suddenness of the changes in the coefficient values, such as the third set of coefficients in [Figure 5-12\(a\)](#), we reduce the sidelobe ripples in the frequency response. However, reducing the sidelobes results in increasing the main lobe width of our lowpass filter. (As we'll see, this is exactly the same effect encountered in the discussion of window functions used with the DFT in [Section 3.9](#).)

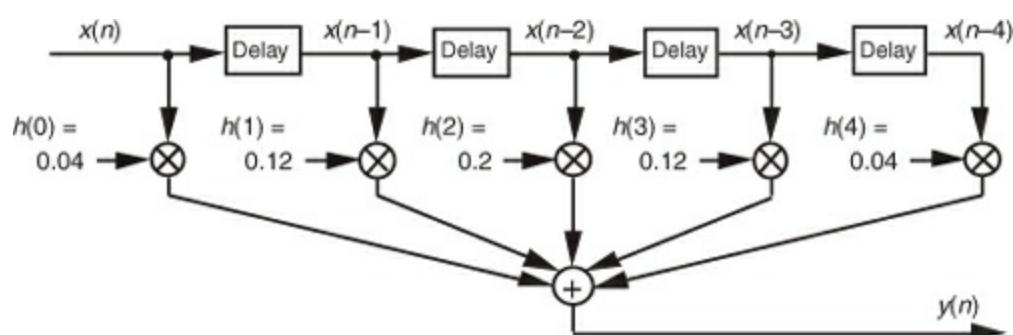
Figure 5-12 Three sets of 5-tap lowpass filter coefficients: (a) sets of coefficients: 0.2, 0.2, 0.2, 0.2, 0.2; 0.1, 0.2, 0.2, 0.2, 0.1; and 0.04, 0.12, 0.2, 0.12, 0.04; (b) frequency magnitude response of three lowpass FIR filters using those sets of coefficients.



To reiterate the function of the filter coefficients, [Figure 5-13](#) shows the 5-tap FIR filter structure using the third set of coefficients from [Figure 5-12](#). The implementation of constant-coefficient transversal FIR filters does not get any more complicated than that shown in [Figure 5-13](#). It's that simple. We can have a filter with more than 5 taps, but the input signal sample shifting, the multiplications by the constant coefficients, and the summation are all there is to it. (By constant coefficients, we don't mean coefficients whose values are all the same; we mean coefficients whose values remain unchanged, or time invariant. There is a class of digital filters, called adaptive filters, whose coefficient values are periodically changed to adapt to changing input signal parameters. While we won't discuss these adaptive filters in this introductory text, their descriptions are

available in the literature[1–5].)

Figure 5-13 Five-tap lowpass FIR filter implementation using the coefficients 0.04, 0.12, 0.2, 0.12, and 0.04.



So far, our description of an FIR filter implementation has been presented from a hardware perspective. In [Figure 5-13](#), to calculate a single filter output sample, five multiplications and five additions must take place before the arrival of the next input sample value. In a software implementation of a 5-tap FIR filter, however, all of the input data samples would be previously stored in memory. The software filter routine’s job, then, is to access different five-sample segments of the $x(n)$ input data space, perform the calculations shown in [Figure 5-13](#), and store the resulting filter $y(n)$ output sequence in an array of memory locations.[‡]

[‡] In reviewing the literature of FIR filters, the reader will often find the term z^{-1} replacing the delay function in [Figure 5-13](#). This equivalence is explained in the next chapter when we study IIR filters.

Now that we have a basic understanding of what a digital FIR filter is, let’s see what effect is had by using more than 5 filter taps by learning to design FIR filters.

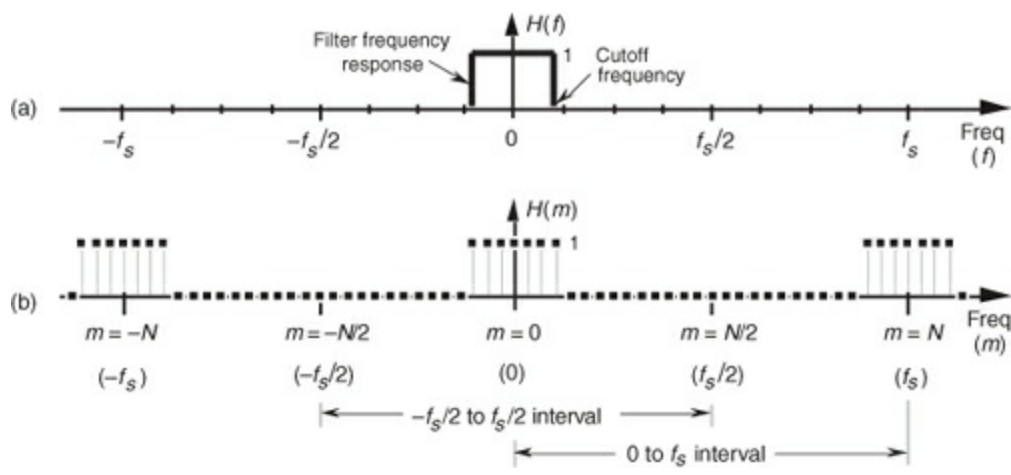
5.3 Lowpass FIR Filter Design

OK, instead of just accepting a given set of FIR filter coefficients and analyzing their frequency response, let’s reverse the process and design our own lowpass FIR filter. The design procedure starts with the determination of a *desired* frequency response followed by calculating the filter coefficients that will give us that response. There are two predominant techniques used to design FIR filters: the window method and the so-called optimum method. Let’s discuss them in that order.

5.3.1 Window Design Method

The window method of FIR filter design (also called the Fourier series method) begins with our deciding what frequency response we want for our lowpass filter. We can start by considering a continuous lowpass filter, and simulating that filter with a digital filter. We’ll define the continuous frequency response $H(f)$ to be ideal, i.e., a lowpass filter with unity gain at low frequencies and zero gain (infinite attenuation) beyond some [cutoff frequency](#), as shown in [Figure 5-14\(a\)](#). Representing this $H(f)$ response by a discrete frequency response is straightforward enough because the idea of a discrete frequency response is essentially the same as a continuous frequency response—with one important difference. As described in [Sections 2.2](#) and [3.13](#), discrete frequency-domain representations are always periodic with the period being the sample rate f_s . The discrete representation of our ideal, continuous lowpass filter $H(f)$ is the periodic response $H(m)$ depicted by the frequency-domain samples in [Figure 5-14\(b\)](#).

Figure 5-14 Lowpass filter frequency responses: (a) continuous frequency response $H(f)$; (b) periodic, discrete frequency response $H(m)$.



We have two ways to determine our lowpass filter's time-domain coefficients. The first way is algebraic:

1. Develop an expression for the discrete frequency response $H(m)$.
2. Apply that expression to the inverse DFT equation to get the time domain $h(k)$.
3. Evaluate that $h(k)$ expression as a function of time index k .

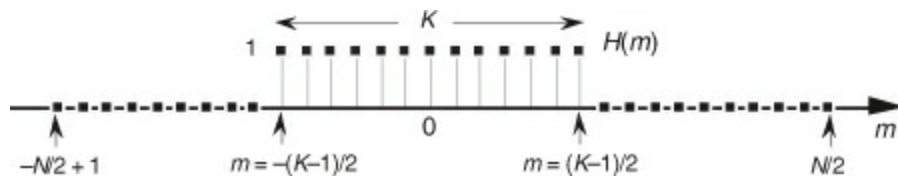
The second method is to define the individual frequency-domain samples representing $H(m)$ and then have a software routine perform the inverse DFT of those samples, giving us the FIR filter coefficients. In either method, we need only define the periodic $H(m)$ over a single period of f_s Hz. As it turns out, defining $H(m)$ in [Figure 5-14\(b\)](#) over the frequency span $-f_s/2$ to $f_s/2$ is the easiest form to analyze algebraically, and defining $H(m)$ over the frequency span 0 to f_s is the best representation if we use the inverse DFT to obtain our filter's coefficients. Let's try both methods to determine the filter's time-domain coefficients.

In the algebraic method, we can define an arbitrary discrete frequency response $H(m)$ using N samples to cover the $-f_s/2$ to $f_s/2$ frequency range and establish K unity-valued samples for the passband of our lowpass filter as shown in [Figure 5-15](#). To determine $h(k)$ algebraically we need to take the inverse DFT of $H(m)$ in the form of

(5-9)

$$h(k) = \frac{1}{N} \sum_{m=-(N/2)+1}^{N/2} H(m) e^{j2\pi mk/N},$$

Figure 5-15 Arbitrary, discrete lowpass FIR filter frequency response defined over N frequency-domain samples covering the frequency range of f_s Hz.



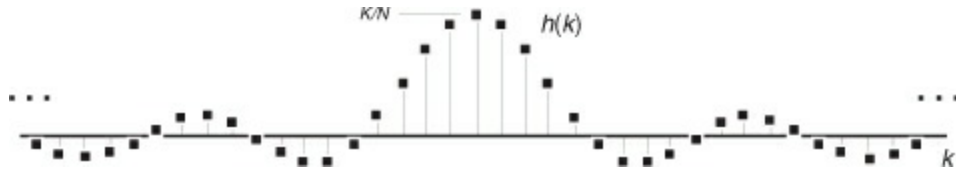
where our time-domain index is k . The solution to [Eq. \(5-9\)](#), derived in [Section 3.13](#) as Eq. (3-59), is repeated here as

(5-10)

$$h(k) = \frac{1}{N} \cdot \frac{\sin(\pi k K / N)}{\sin(\pi k / N)}.$$

If we evaluate [Eq. \(5-10\)](#) as a function of k , we get the sequence shown in [Figure 5-16](#), taking the form of the classic $\sin(x)/x$ function. By reviewing the material in [Section 3.13](#), it's easy to see the great deal of algebraic manipulation required to arrive at [Eq. \(5-10\)](#) from [Eq. \(5-9\)](#). So much algebra, in fact, with its many opportunities for making errors, that digital filter designers like to avoid evaluating [Eq. \(5-9\)](#) algebraically. They prefer to use software routines to perform inverse DFTs (in the form of an inverse FFT) to determine $h(k)$, and so will we.

Figure 5-16 Time-domain $h(k)$ coefficients obtained by evaluating [Eq. \(5-10\)](#).



We can demonstrate the software inverse DFT method of FIR filter design with an example. Let's say we need to design a lowpass FIR filter simulating the continuous frequency response shown in [Figure 5-17\(a\)](#). The discrete representation of the filter's frequency response $H(m)$ is shown in [Figure 5-17\(b\)](#), where we've used $N = 32$ points to represent the frequency-domain variable $H(f)$. Because it's equivalent to [Figure 5-17\(b\)](#) but avoids the negative values of the frequency index m , we represent the discrete frequency samples over the range 0 to f_s in [Figure 5-17\(c\)](#), as opposed to the $-f_s/2$ to $+f_s/2$ range in [Figure 5-17\(b\)](#). OK, we're almost there. Using a 32-point inverse FFT to implement a 32-point inverse DFT of the $H(m)$ sequence in [Figure 5-17\(c\)](#), we get the 32 $h(k)$ values depicted by the dots from $k = -15$ to $k = 16$ in [Figure 5-18\(a\)](#).[‡] We have one more step to perform. Because we want our final 31-tap $h(k)$ filter coefficients to be symmetrical with their peak value in the center of the coefficient sample set, we drop the $k = 16$ sample and shift the k index to the left from [Figure 5-18\(a\)](#), giving us the desired $\sin(x)/x$ form of $h(k)$ as shown in [Figure 5-18\(b\)](#). This shift of the index k will not change the frequency magnitude response of our FIR filter. (Remember from our discussion of the DFT shifting theorem in [Section 3.6](#) that a shift in the time domain manifests itself only as a linear phase shift in the frequency domain with no change in the frequency-domain magnitude.) The sequence in [Figure 5-18\(b\)](#), then, is now the coefficients we use in the convolution process of [Figure 5-5](#) to implement a lowpass FIR filter.

[‡] If you want to use this FIR design method but only have a forward FFT software routine available, [Section 13.6](#) shows a slick way to perform an inverse FFT with the forward FFT algorithm.

Figure 5-17 An ideal lowpass filter: (a) continuous frequency response $H(f)$; (b) discrete response $H(m)$ over the range $-f_s/2$ to $f_s/2$ Hz; (c) discrete response $H(m)$ over the range 0 to f_s Hz.

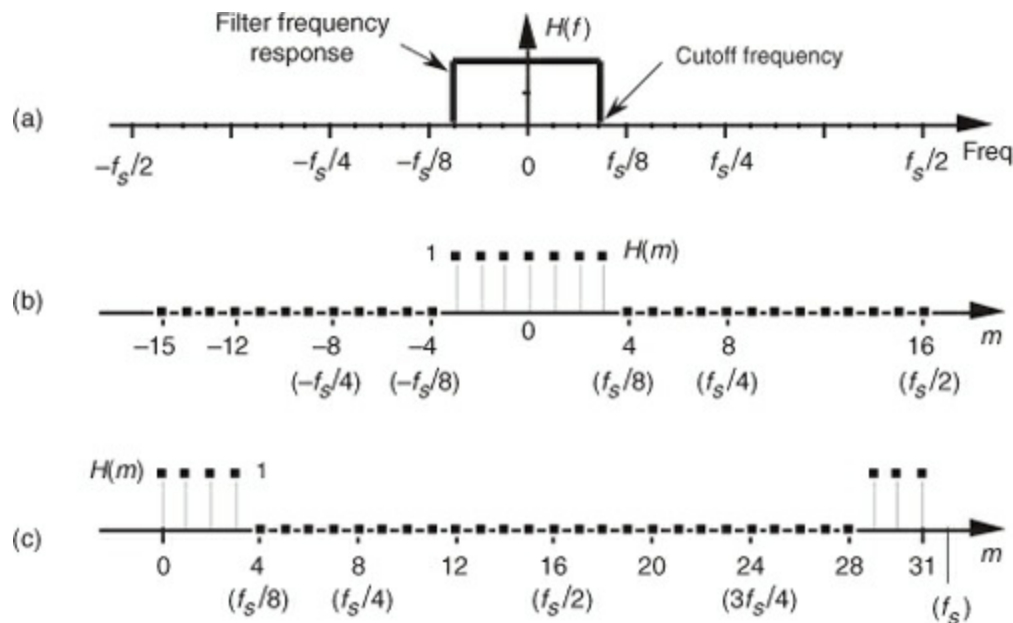
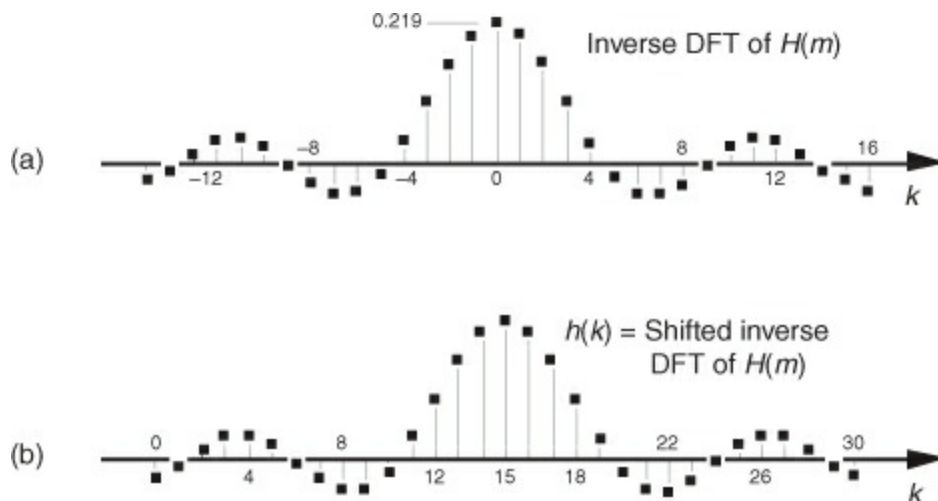
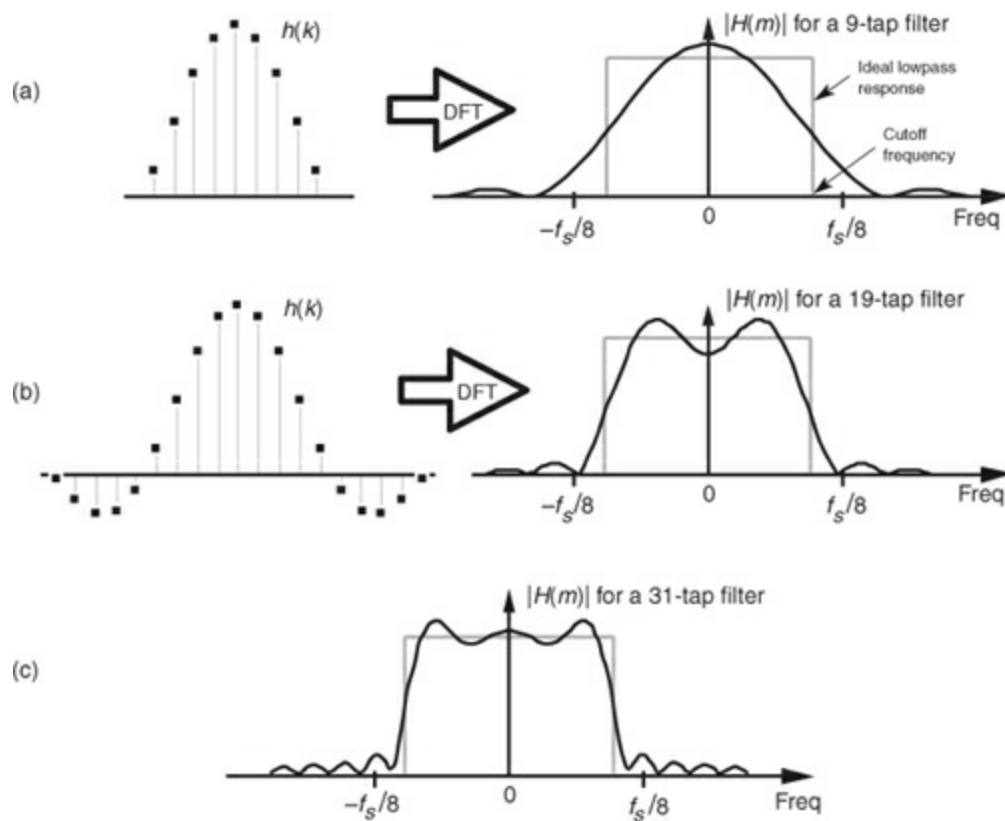


Figure 5-18 Inverse DFT of the discrete response in [Figure 5-17\(c\)](#): (a) normal inverse DFT indexing for k ; (b) symmetrical coefficients used for a 31-tap lowpass FIR filter.



It's important to demonstrate that the more $h(k)$ terms we use as filter coefficients, the closer we'll approximate our ideal lowpass filter response. Let's be conservative, just use the center nine $h(k)$ coefficients, and see what our filter response looks like. Again, our filter's magnitude response in this case will be the DFT of those nine coefficients as shown on the right side of [Figure 5-19\(a\)](#). The ideal filter's frequency response is also shown for reference as the dashed curve. (To show the details of its shape, we've used a continuous curve for $|H(m)|$ in [Figure 5-19\(a\)](#), but we have to remember that $|H(m)|$ is really a sequence of discrete values.) Notice that using nine coefficients gives us a lowpass filter, but it's certainly far from ideal. Using more coefficients to improve our situation, [Figure 5-19\(b\)](#) shows 19 coefficients and their corresponding frequency magnitude response that is beginning to look more like our desired rectangular response. Notice that magnitude fluctuations, or ripples, are evident in the passband of our $H(m)$ filter response. Continuing, using all 31 of the $h(k)$ values for our filter coefficients results in the frequency response in [Figure 5-19\(c\)](#). Our filter's response is getting better (approaching the ideal), but those conspicuous passband magnitude ripples are still present.

Figure 5-19 Coefficients and frequency responses of three lowpass filters: (a) 9-tap FIR filter; (b) 19-tap FIR filter; (c) frequency response of the full 31-tap FIR filter.



It's important that we understand why those passband ripples are in the lowpass FIR filter response in [Figure 5-19](#). Recall the above discussion of convolving the 5-tap averaging filter coefficients, or impulse response, with an input data sequence to obtain the averager's output. We established that convolution in the time domain is equivalent to multiplication in the frequency domain, which we symbolized with [Eq. \(5-8\)](#) and repeat here as

(5-11)

$$h(k) * x(n) \xrightleftharpoons[\text{IDFT}]{\text{DFT}} H(m) \cdot X(m).$$

This association between convolution in the time domain and multiplication in the frequency domain, sketched in [Figure 5-7](#), indicates that if two time-domain sequences $h(k)$ and $x(n)$ have DFTs of $H(m)$ and $X(m)$, respectively, then the DFT of $h(k) * x(n)$ is $H(m) \cdot X(m)$. No restrictions whatsoever need be placed on what the time-domain sequences $h(k)$ and $x(n)$ in [Eq. \(5-11\)](#) actually represent. As detailed later in [Section 5.9](#), convolution in one domain is equivalent to multiplication in the other domain, allowing us to state that multiplication in the time domain is equivalent to convolution in the frequency domain, or

(5-12)

$$h(k) \cdot x(n) \xrightleftharpoons[\text{IDFT}]{\text{DFT}} H(m) * X(m).$$

Now we're ready to understand why the magnitude ripples are present in [Figure 5-19](#).

Rewriting [Eq. \(5-12\)](#) and replacing the $h(k)$ and $x(n)$ expressions with $h^\infty(k)$ and $w(k)$, respectively,

(5-13)

$$h^\infty(k) \cdot w(k) \xrightleftharpoons[\text{IDFT}]{\text{DFT}} H^\infty(m) * W(m).$$

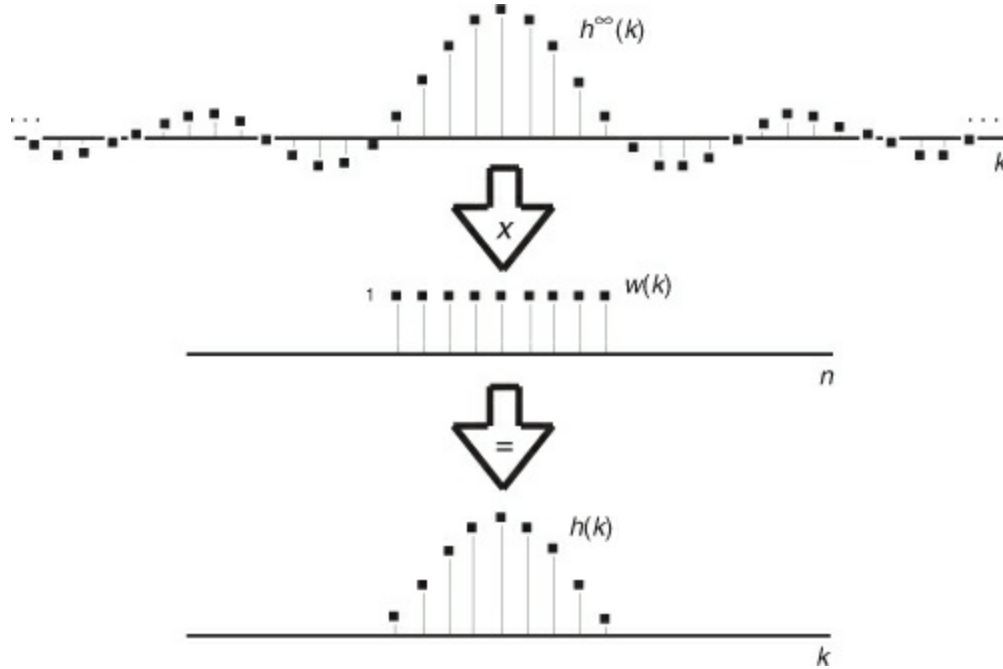
Let's say that $h^\infty(k)$ represents an infinitely long $\sin(x)/x$ sequence of ideal lowpass FIR filter coefficients and that $w(k)$ represents a window sequence that we use to truncate the $\sin(x)/x$ terms as

shown in [Figure 5-20](#). Thus, the $w(k)$ sequence is a finite-length set of unity values and its DFT is $W(m)$. The length of $w(k)$ is merely the number of coefficients, or taps, we intend to use to implement our lowpass FIR filter. With $h^\infty(k)$ defined as such, the product $h^\infty(k) \cdot w(k)$ represents the truncated set of filter coefficients $h(k)$ in [Figures 5-19\(a\)](#) and [5-19\(b\)](#). So, from [Eq. \(5-13\)](#), the FIR filter's true frequency response $H(m)$ is the convolution

$$(5-14)$$

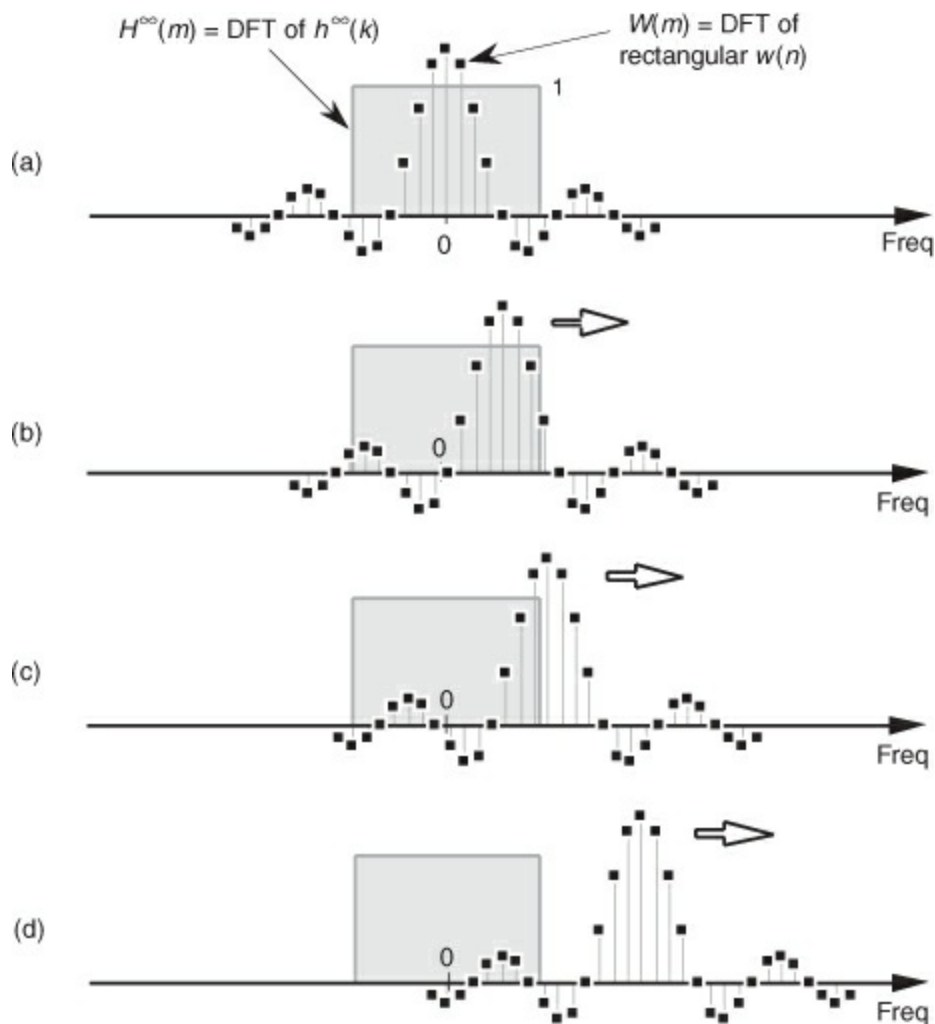
$$H(m) = H^\infty(m) * W(m).$$

Figure 5-20 Infinite $h^\infty(k)$ sequence windowed by $w(k)$ to define the final filter coefficients $h(k)$.



We depict this convolution in [Figure 5-21](#) where, to keep the figure from being so busy, we show $H^\infty(m)$ (the DFT of the $h^\infty(k)$ coefficients) as the gray rectangle. Keep in mind that it's really a sequence of constant-amplitude sample values.

Figure 5-21 Convolution $W(m) * H^\infty(m)$: (a) unshifted $W(m)$ and $H^\infty(m)$; (b) shift of $W(m)$ leading to ripples within $H(m)$'s positive-frequency passband; (c) shift of $W(m)$ causing response roll-off near $H(m)$'s positive cutoff frequency; (d) shift of $W(m)$ causing ripples beyond $H(m)$'s positive cutoff frequency.



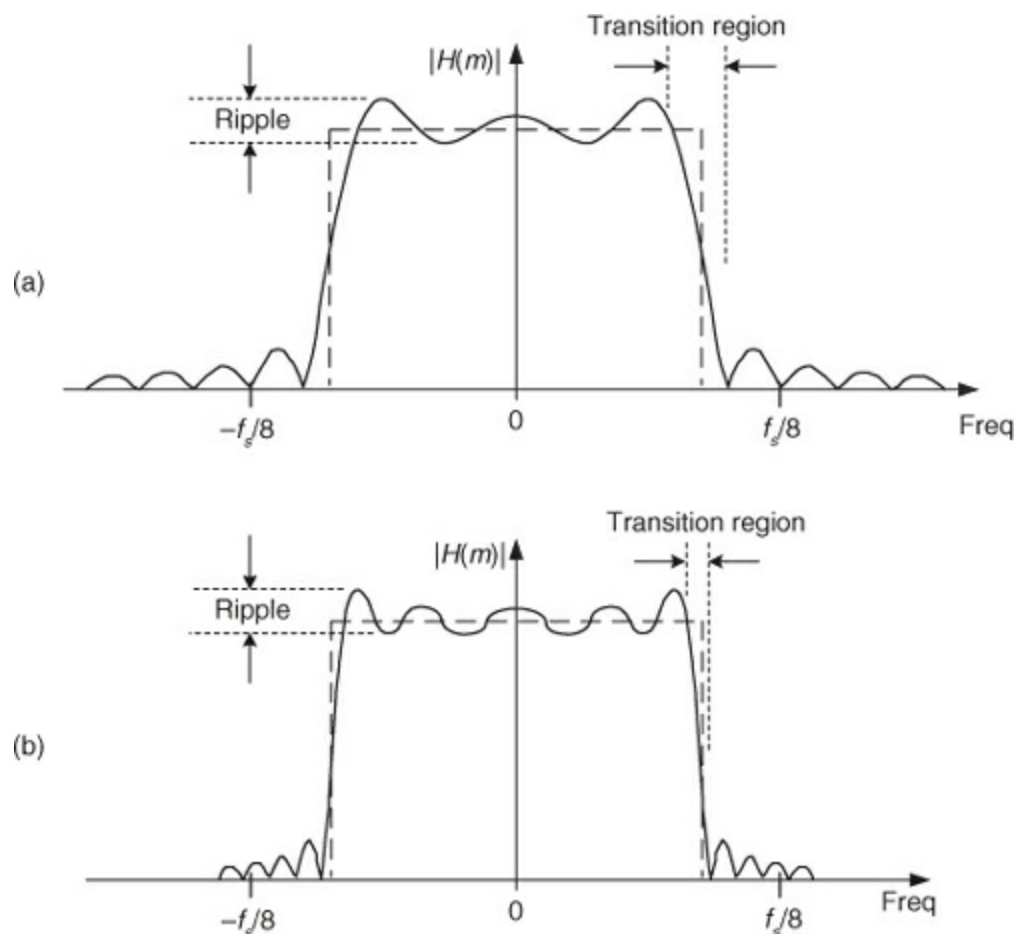
Let's look at [Figure 5-21\(a\)](#) very carefully to see why all three $|H(m)|$ s exhibit passband ripple in [Figure 5-19](#). We can view a particular sample value of the $H(m) = H^\infty(m) * W(m)$ convolution as being the sum of the products of $H^\infty(m)$ and $W(m)$ for a particular frequency shift of $W(m)$. $H^\infty(m)$ and the unshifted $W(m)$ are shown in [Figure 5-21\(a\)](#). With an assumed value of unity for all of $H^\infty(m)$, a particular $H(m)$ value is now merely the sum of the $W(m)$ samples that overlap the $H^\infty(m)$ rectangle. So, with a $W(m)$ frequency shift of 0 Hz, the sum of the $W(m)$ samples that overlap the $H^\infty(m)$ rectangle in [Figure 5-21\(a\)](#) is the value of $H(m)$ at 0 Hz. As $W(m)$ is shifted to the right to give us additional positive-frequency $H(m)$ values, we can see that the sum of the positive and negative values of $W(m)$ under the rectangle oscillates during the shifting of $W(m)$. As the convolution shift proceeds, [Figure 5-21\(b\)](#) shows why there are ripples in the passband of $H(m)$ —again, the sum of the positive and negative $W(m)$ samples under the $H^\infty(m)$ rectangle continues to vary as the $W(m)$ function is shifted. The $W(m)$ frequency shift, indicated in [Figure 5-21\(c\)](#), where the peak of $W(m)$'s main lobe is now outside the $H^\infty(m)$ rectangle, corresponds to the frequency where $H(m)$'s passband begins to roll off. [Figure 5-21\(d\)](#) shows that, as the $W(m)$ shift continues, there will be ripples in $H(m)$ beyond the positive cutoff frequency.[‡] The point of all of this is that the ripples in $H(m)$ are caused by the sidelobes of $W(m)$.

[‡] In [Figure 5-21\(b\)](#), had we started to shift $W(m)$ to the left in order to determine the negative-frequency portion of $H(m)$, we would have obtained the mirror image of the positive-frequency portion of $H(m)$.

[Figure 5-22](#) helps us answer the question “How many $\sin(x)/x$ coefficients do we have to use (or how wide must $w(k)$ be) to get nice sharp falling edges and no ripples in our $H(m)$ passband?” The answer is that we can't get there from here. It doesn't matter how many $\sin(x)/x$ coefficients (filter taps) we

use; there will always be filter passband ripple. As long as $w(k)$ is a finite number of unity values (i.e., a rectangular window of finite width), there will be sidelobe ripples in $W(m)$, and this will induce passband ripples in the final $H(m)$ frequency response. To illustrate that increasing the number of $\sin(x)/x$ coefficients doesn't reduce passband ripple, we repeat the 31-tap lowpass filter response in [Figure 5-22\(a\)](#). The frequency response, using 63 coefficients, is shown in [Figure 5-22\(b\)](#), and the passband ripple remains. We can make the filter's transition region narrower using additional $h(k)$ filter coefficients, but we cannot eliminate the passband ripple. That ripple, known as Gibbs's phenomenon, manifests itself anytime a function ($w(k)$ in this case) with an instantaneous discontinuity is represented by a Fourier series[6–8]. No finite set of sinusoids will be able to change fast enough to be exactly equal to an instantaneous discontinuity. Another way to state this Gibbs's dilemma is that, no matter how wide our $w(k)$ window is, $W(m)$ will always have sidelobe ripples. As shown in [Figure 5-22\(b\)](#), we can use more coefficients by extending the width of the rectangular $w(k)$ to narrow the filter transition region, but a wider $w(k)$ does not eliminate the filter passband ripple, nor does it even reduce their peak-to-peak ripple magnitudes, as long as $w(k)$ has sudden discontinuities.

Figure 5-22 Passband ripple and transition regions: (a) for a 31-tap lowpass filter; (b) for a 63-tap lowpass filter.



5.3.2 Windows Used in FIR Filter Design

OK. The good news is that we can minimize FIR passband ripple with window functions the same way we minimized DFT leakage in [Section 3.9](#). Here's how. Looking back at [Figure 5-20](#), by truncating the infinitely long $h^\infty(k)$ sequence through multiplication by the rectangular $w(k)$, our final $h(k)$ exhibited ripples in the frequency-domain passband. [Figure 5-21](#) shows us that the passband ripples were caused by $W(m)$'s sidelobes that, in turn, were caused by the sudden discontinuities from zero to one and one to zero in $w(k)$. If we think of $w(k)$ in [Figure 5-20](#) as a rectangular window,

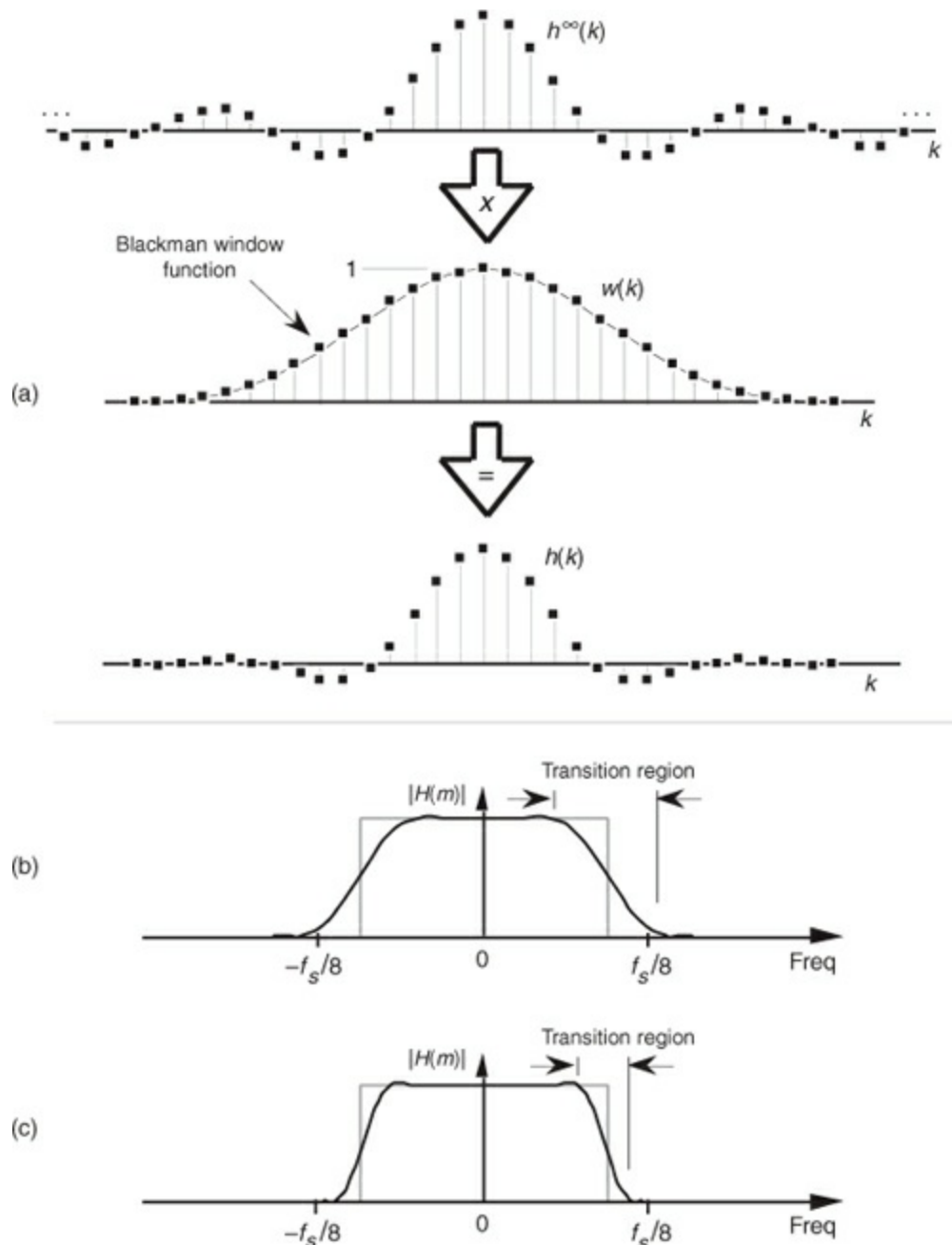
then it is $w(k)$'s abrupt amplitude changes that are the source of our filter passband ripple. The window FIR design method is the technique of reducing $w(k)$'s discontinuities by using window functions other than the rectangular window.

Consider [Figure 5-23](#) to see how a nonrectangular window function can be used to design low-ripple FIR digital filters. Imagine if we replaced [Figure 5-20](#)'s rectangular $w(k)$ with the Blackman window function whose discrete values are defined as

$$(5-15)$$

$$w(k) = 0.42 - 0.5 \cos\left(\frac{2\pi k}{N-1}\right) + 0.08 \cos\left(\frac{4\pi k}{N-1}\right), \text{ for } k = 0, 1, 2, \dots, N-1.$$

Figure 5-23 Coefficients and frequency response of a 31-tap Blackman-windowed FIR filter: (a) defining the windowed filter coefficients $h(k)$; (b) low-ripple 31-tap frequency response; (c) low-ripple 63-tap frequency response.



This situation is depicted for $N = 31$ in [Figure 5-23\(a\)](#), where [Eq. \(5-15\)](#)'s $w(k)$ looks very much like the Hanning window function in [Figure 3-17\(a\)](#). This Blackman window function results in the 31 smoothly tapered $h(k)$ coefficients at the bottom of [Figure 5-23\(a\)](#). Notice two things about the

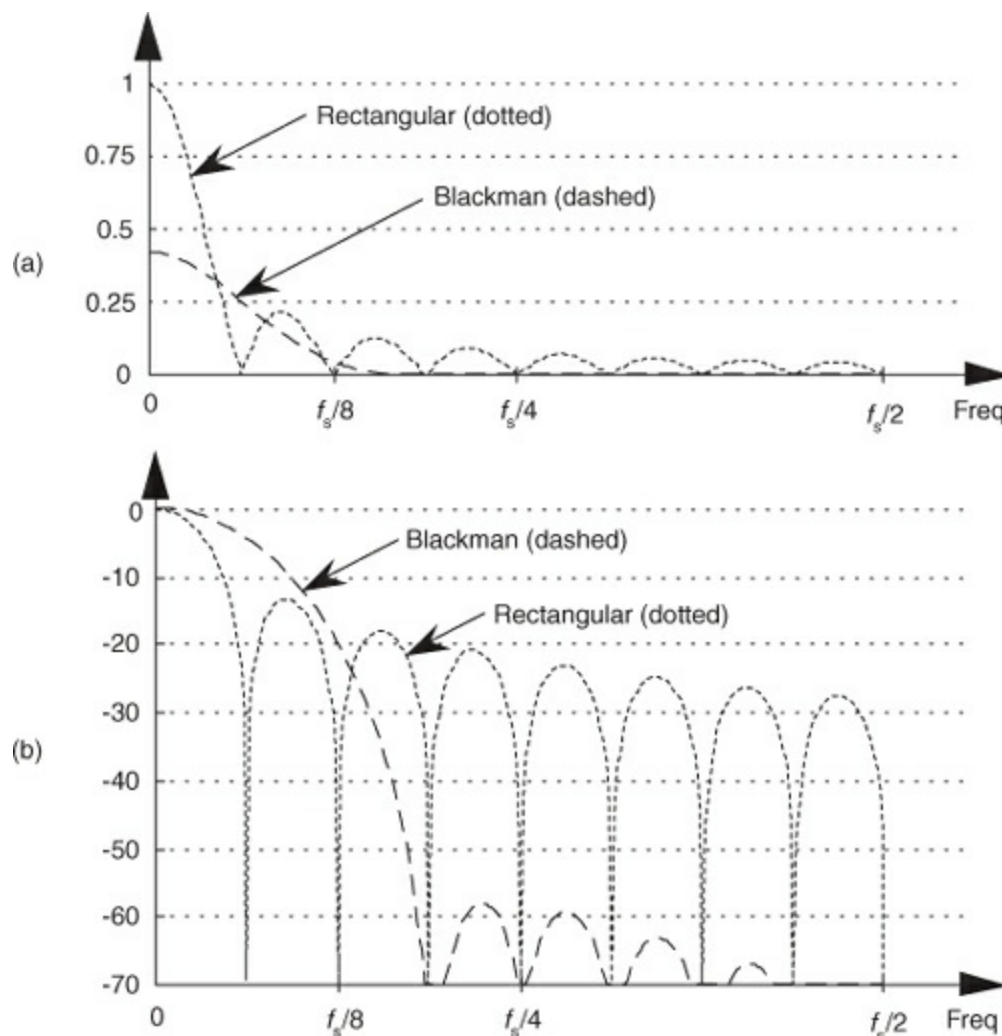
resulting $H(m)$ in [Figure 5-23\(b\)](#). First, the good news. The passband ripples are greatly reduced from those evident in [Figure 5-22\(a\)](#)—so our Blackman window function did its job. Second, the price we paid for reduced passband ripple is a wider $H(m)$ transition region. We can get a steeper filter response roll-off by increasing the number of taps in our FIR filter. [Figure 5-23\(c\)](#) shows the improved frequency response had we used a 63-coefficient Blackman window function for a 63-tap FIR filter. So using a nonrectangular window function reduces passband ripple at the expense of slower passband to stopband roll-off.

A graphical comparison of the frequency responses for the rectangular and Blackman windows is provided in [Figure 5-24](#). (The curves in [Figure 5-24](#) were obtained for the window functions defined by 16 discrete samples, to which 496 zeros were appended, applied to a 512-point DFT.) The sidelobe magnitudes of the Blackman window's $|W(m)|$ are too small to see on a linear scale. We can see those sidelobe details by plotting the two windows' frequency responses on a logarithmic scale and normalizing each plot so that their main lobe peak values are both zero dB. For a given window function, we can get the log magnitude response of $W_{dB}(m)$ by using the expression

(5-16)

$$W_{dB}(m) = 20 \cdot \log_{10} \left(\frac{|W(m)|}{|W(0)|} \right).$$

Figure 5-24 Rectangular versus Blackman window frequency magnitude responses: (a) $|W(m)|$ on a linear scale; (b) normalized logarithmic scale of $W_{dB}(m)$.



(The $|W(0)|$ term in [Eq. \(5-16\)](#) is the magnitude of $W(m)$ at the peak of the main lobe when $m = 0$.) [Figure 5-24\(b\)](#) shows us the greatly reduced sidelobe levels of the Blackman window and how that

window's main lobe is almost three times as wide as the rectangular window's main lobe.

Of course, we could have used any of the other window functions, discussed in [Section 3.9](#), for our lowpass FIR filter. That's why this FIR filter design technique is called the window design method. We pick a window function and multiply it by the $\sin(x)/x$ values from $H^o(m)$ in [Figure 5-23\(a\)](#) to get our final $h(k)$ filter coefficients. It's that simple. Before we leave the window method of FIR filter design, let's introduce two other interesting window functions.

Although the Blackman window and those windows discussed in [Section 3.9](#) are useful in FIR filter design, we have little control over their frequency responses; that is, our only option is to select some window function and accept its corresponding frequency response. Wouldn't it be nice to have more flexibility in trading off, or striking a compromise between, a window's main lobe width and sidelobe levels? Fortunately, there are two popular window functions that give us this opportunity. Called the Chebyshev (or Dolph-Chebyshev) and the Kaiser window functions, they're defined by the following formidable expressions:

(5-17)

$$w(k) = \frac{\cos \left[N \cdot \cos^{-1} \left[\alpha \cdot \cos \left(\pi \frac{m}{N} \right) \right] \right]}{\cosh [N \cdot \cosh^{-1} (\alpha)]},$$

Chebyshev window:→
(also called the Dolph-Chebyshev and the Tchebyschev window)

where $\alpha = \cosh \left(\frac{1}{N} \cosh^{-1} (10^{\gamma/20}) \right)$ and $m = 0, 1, 2, \dots, N$

(5-18)

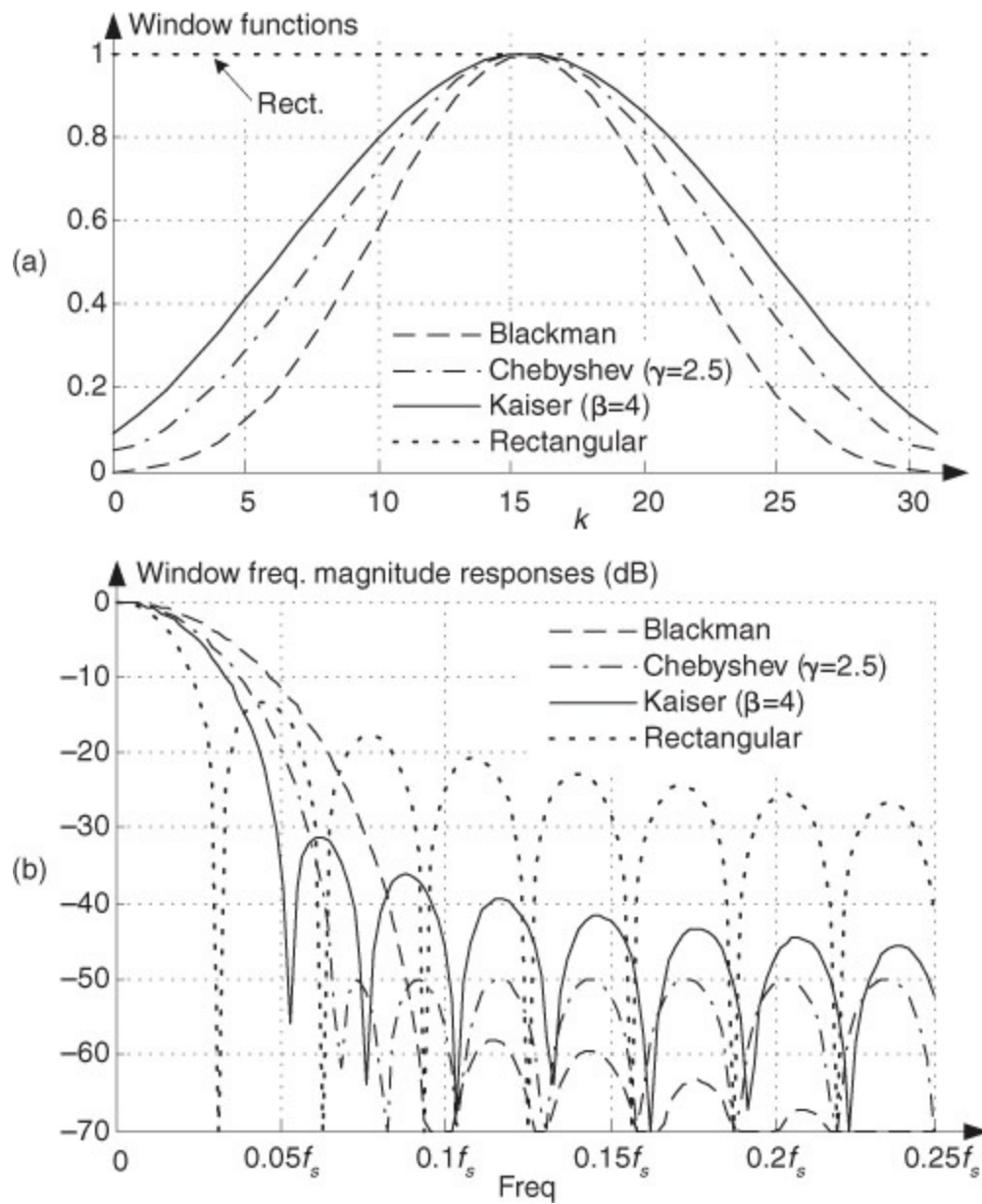
$$w(k) = \frac{I_0 \left[\beta \sqrt{1 - \left(\frac{k-p}{p} \right)^2} \right]}{I_0(\beta)},$$

Kaiser window:→
(also called the Kaiser-Bessel window)

for $k = 0, 1, 2, \dots, N-1$, and $p = (N-1)/2$.

Two typical Chebyshev and Kaiser window functions and their frequency magnitude responses are shown in [Figure 5-25](#). For comparison, the rectangular and Blackman window functions are also shown in that figure. (Again, the curves in [Figure 5-25\(b\)](#) were obtained for window functions defined by 32 discrete time samples, with 480 zeros appended, applied to a 512-point DFT.)

Figure 5-25 Typical window functions used with digital filters: (a) window coefficients in the time domain; (b) frequency-domain magnitude responses in dB.



[Equation \(5-17\)](#) was originally based on the analysis of antenna arrays using the mathematics of Chebyshev polynomials[9–11]. [Equation \(5-18\)](#) evolved from Kaiser’s approximation of prolate spheroid functions using zeroth-order Bessel functions[12–13]. For each sample of the N -length sequence inside the brackets of the numerator of [Eq. \(5-18\)](#), as well as for the β term in the denominator, the $I_0(x)$ zeroth-order Bessel function values can be approximated using

(5-18')

$$I_0(x) = \sum_{q=0}^{24} \frac{x^{2q}}{4^q \cdot (q!)^2}.$$

In theory the upper limit of the summation in [Eq. \(5-18'\)](#) should be infinity but, fortunately, 25 summations give us sufficient accuracy when evaluating $I_0(x)$.

Don’t be intimidated by the complexity of [Eqs. \(5-17\)](#) and [\(5-18\)](#)—at this point, we need not be concerned with the mathematical details of their development. We just need to realize that the γ and β control parameters give us control over the Chebyshev and Kaiser windows’ main lobe widths and the sidelobe levels.

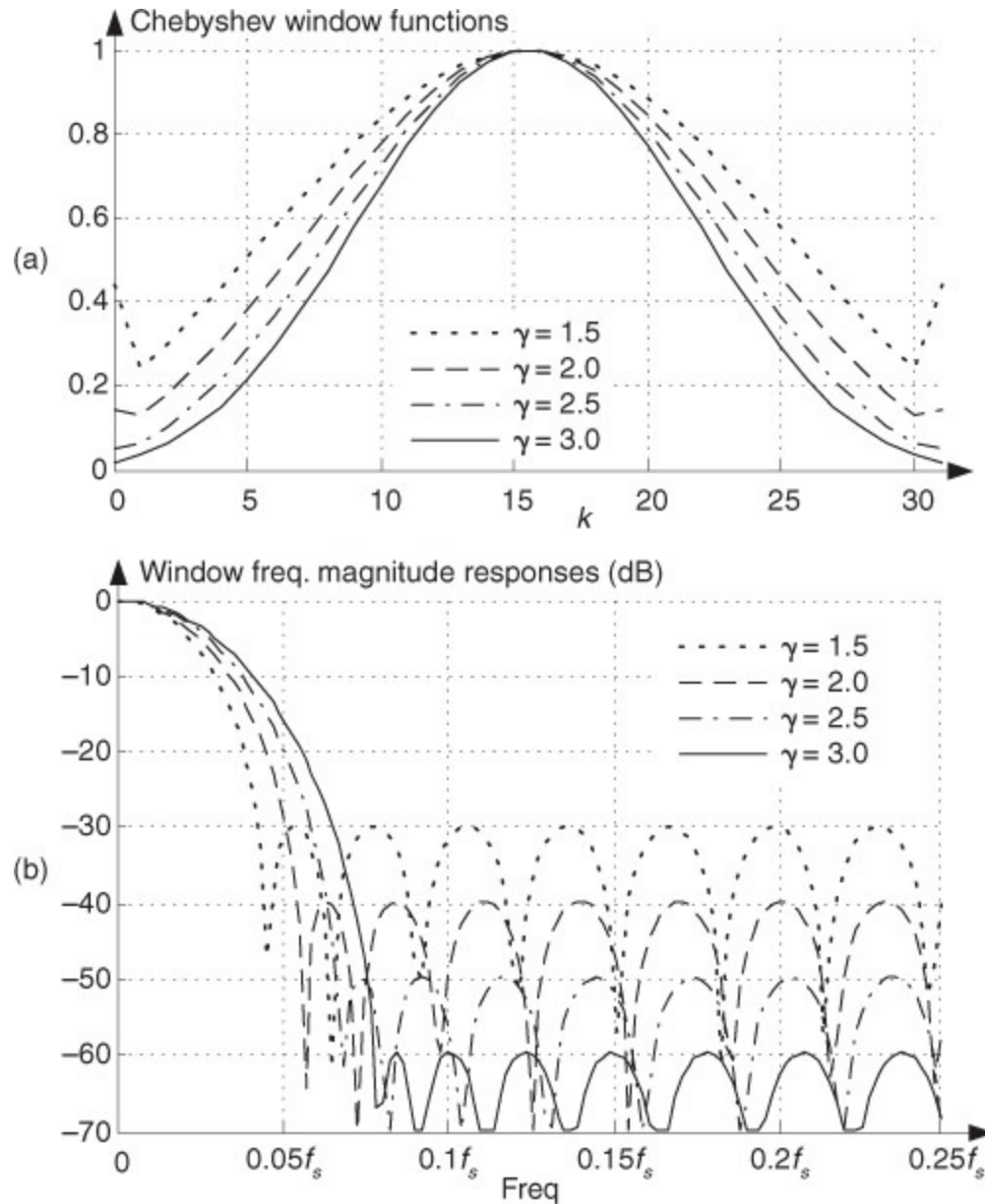
Let’s see how this works for Chebyshev window functions, having four separate values of γ , and their frequency responses shown in [Figure 5-26](#). FIR filter designers applying the window method

typically use predefined software routines to obtain their Chebyshev window coefficients. Commercial digital signal processing software packages allow the user to specify three things: the window function (Chebyshev in this case), the desired number of coefficients (the number of taps in the FIR filter), and the value of γ . Selecting different values for γ enables us to adjust the sidelobe levels and see what effect those values have on main lobe width, a capability that we didn't have with the Blackman window or the window functions discussed in [Section 3.9](#). The Chebyshev window function's stopband attenuation, in dB, is equal to

$$(5-19)$$

$$Atten_{\text{Cheb}} = -20\gamma.$$

Figure 5-26 Chebyshev window functions for various γ values: (a) window coefficients in the time domain; (b) frequency-domain magnitude responses in dB.

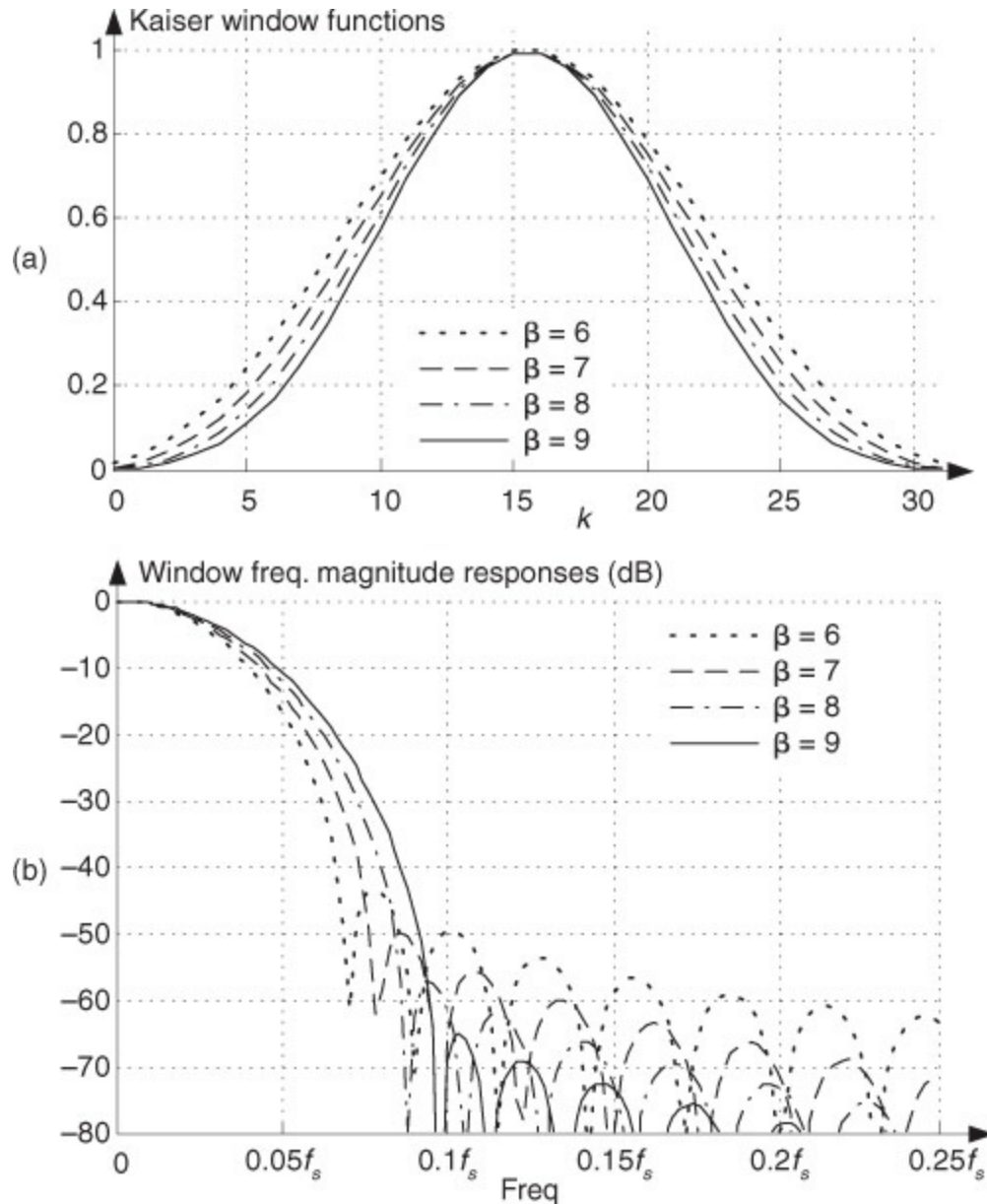


So, for example, if we needed our sidelobe levels to be no greater than -60 dB below the main lobe, we use [Eq. \(5-19\)](#) to establish a γ value of 3.0 and let the software generate the Chebyshev window coefficients.[‡]

[‡] By the way, some digital signal processing software packages require that we specify $Atten_{\text{Cheb}}$ in decibels instead of γ . That way, we don't have to bother using [Eq. \(5-19\)](#) at all.

The same process applies to the Kaiser window, as shown in [Figure 5-27](#). Commercial software packages allow us to specify β in [Eq. \(5-18\)](#) and provide us with the associated window coefficients. The curves in [Figure 5-27\(b\)](#), obtained for Kaiser window functions defined by 32 discrete samples, show that we can select the desired sidelobe levels and see what effect this has on the main lobe width.

Figure 5-27 Kaiser window functions for various β values: (a) window coefficients in the time domain; (b) frequency-domain magnitude responses in dB.



Chebyshev or Kaiser, which is the best window to use? It depends on the application. Returning to [Figure 5-25\(b\)](#), notice that, unlike the constant sidelobe peak levels of the Chebyshev window, the Kaiser window's sidelobes decrease with increased frequency. However, the Kaiser sidelobes are higher than the Chebyshev window's sidelobes near the main lobe. Our primary trade-off here is trying to reduce the sidelobe levels without broadening the main lobe too much. Digital filter designers typically experiment with various values of γ and β for the Chebyshev and Kaiser windows to get the optimum $W_{dB}(m)$ for a particular application. (For that matter, the Blackman window's very low sidelobe levels outweigh its wide main lobe in many applications.) For some reason, algorithms for computing Chebyshev window functions are not readily available in the literature of DSP. To remedy that situation, [Appendix I](#) presents a straightforward procedure for computing N -sample

Chebyshev window sequences.

To conclude this section, remember that different window functions have their own individual advantages and disadvantages for FIR filter design. Regardless of the non-rectangular window function used, they always decrease an FIR filter's passband ripple over that of the rectangular window. For the enthusiastic reader, a thorough discussion of many window functions can be found in reference [\[14\]](#).

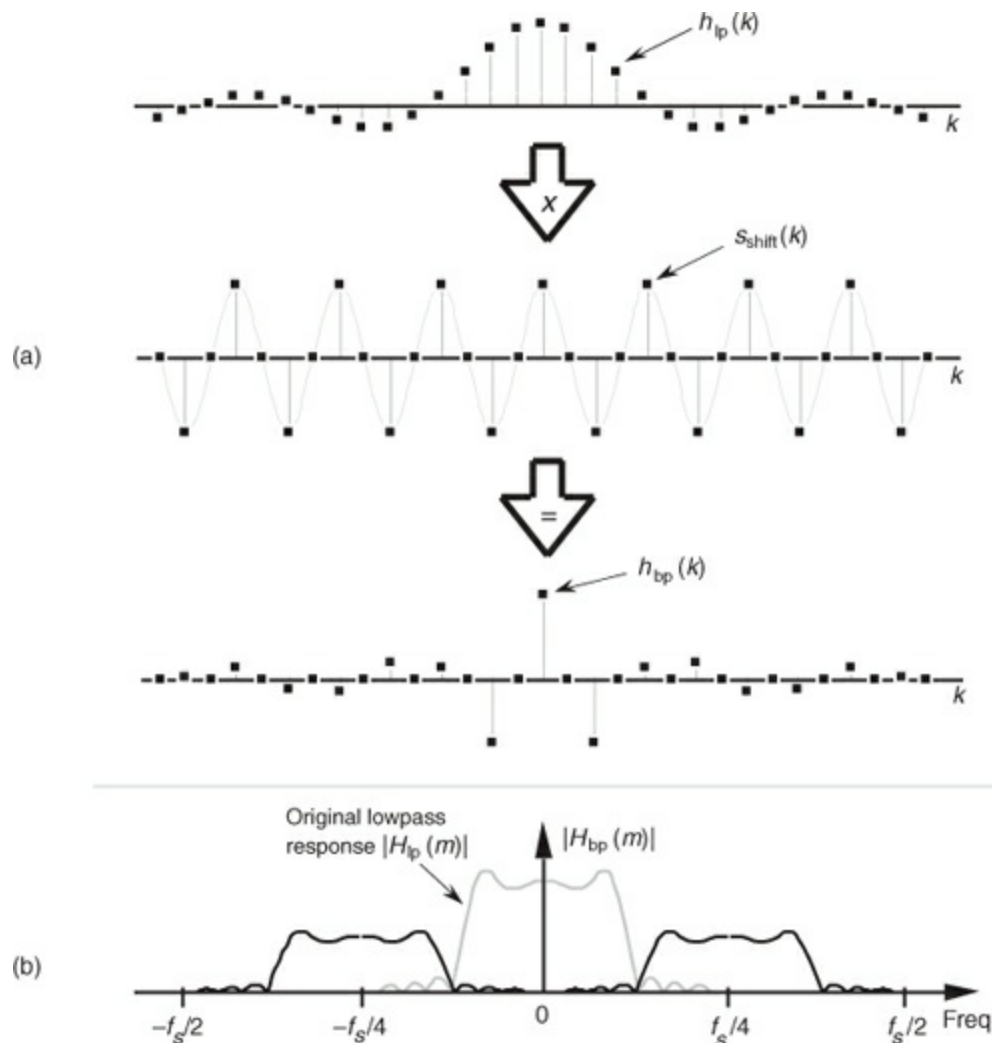
5.4 Bandpass FIR Filter Design

The window method of lowpass FIR filter design can be used as the first step in designing a bandpass FIR filter. Let's say we want a 31-tap FIR filter with the frequency response shown in [Figure 5-22\(a\)](#), but instead of being centered about zero Hz, we want the filter's passband to be centered about $f_s/4$ Hz. If we define a lowpass FIR filter's coefficients as $h_{lp}(k)$, our problem is to find the $h_{bp}(k)$ coefficients of a bandpass FIR filter. As shown in [Figure 5-28](#), we can shift $H_{lp}(m)$'s frequency response by multiplying the filter's $h_{lp}(k)$ lowpass coefficients by a sinusoid of $f_s/4$ Hz. That sinusoid is represented by the $s_{\text{shift}}(k)$ sequence in [Figure 5-28\(a\)](#), whose values are a sinewave sampled at a rate of four samples per cycle. Our final 31-tap $h_{bp}(k)$ FIR bandpass filter coefficients are

(5-20)

$$h_{bp}(k) = h_{lp}(k) \cdot s_{\text{shift}}(k) ,$$

Figure 5-28 Bandpass filter with frequency response centered at $f_s/4$: (a) generating 31-tap filter coefficients $h_{bp}(k)$; (b) frequency magnitude response $|H_{bp}(m)|$.



whose frequency magnitude response $|H_{bp}(m)|$ is shown as the solid curves in [Figure 5-28\(b\)](#). The actual magnitude of $|H_{bp}(m)|$ is half that of the original $|H_{lp}(m)|$ because half the values in $h_{bp}(k)$ are zero when $s_{shift}(k)$ corresponds exactly to $f_s/4$. This effect has an important practical implication. It means that, when we design an N -tap bandpass FIR filter centered at a frequency of $f_s/4$ Hz, we only need to perform approximately $N/2$ multiplications for each filter output sample. (There's no reason to multiply an input sample value, $x(n-k)$, by zero before we sum all the products from [Eq. \(5-6\)](#) and [Figure 5-13](#), right? We just don't bother to perform the unnecessary multiplications at all.) Of course, when the bandpass FIR filter's center frequency is other than $f_s/4$, we're forced to perform the full number of N multiplications for each FIR filter output sample.

Notice, here, that the $h_{lp}(k)$ lowpass coefficients in [Figure 5-28\(a\)](#) have not been multiplied by any window function. In practice, we'd use an $h_{lp}(k)$ that has been windowed prior to implementing [Eq. \(5-20\)](#) to reduce the passband ripple. If we wanted to center the bandpass filter's response at some frequency other than $f_s/4$, we merely need to modify $s_{shift}(k)$ to represent sampled values of a sinusoid whose frequency is equal to the desired bandpass center frequency. That new $s_{shift}(k)$ sequence would then be used in [Eq. \(5-20\)](#) to get the new $h_{bp}(k)$.

5.5 Highpass FIR Filter Design

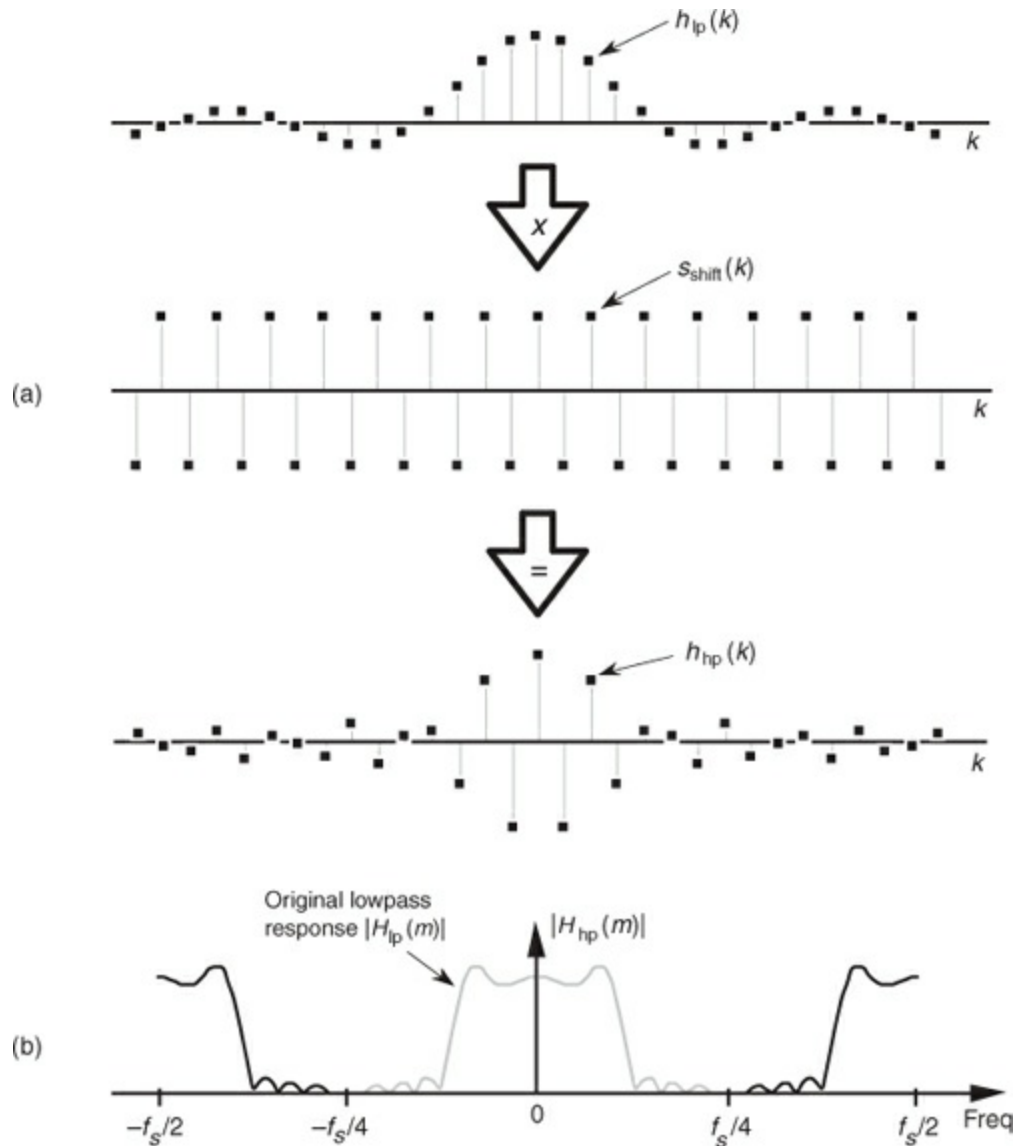
Going one step further, we can use the bandpass FIR filter design technique to design a highpass FIR filter. To obtain the coefficients for a highpass filter, we need only modify the shifting sequence $s_{shift}(k)$ to make it represent a sampled sinusoid whose frequency is $f_s/2$. This process is shown in

[Figure 5-29](#). Our final 31-tap highpass filter's $h_{\text{hp}}(k)$ coefficients are

$$(5-21)$$

$$\begin{aligned} h_{\text{hp}}(k) &= h_{\text{lp}}(k) \cdot s_{\text{shift}}(k) \\ &= h_{\text{lp}}(k) \cdot (1, -1, 1, -1, 1, -1, \text{etc.}), \end{aligned}$$

Figure 5-29 Highpass filter with frequency response centered at $f_s/2$: (a) generating 31-tap filter coefficients $h_{\text{hp}}(k)$; (b) frequency magnitude response $|H_{\text{hp}}(m)|$.



whose $|H_{\text{hp}}(m)|$ frequency response is the solid curve in [Figure 5-29\(b\)](#). Because $s_{\text{shift}}(k)$ in [Figure 5-29\(a\)](#) has alternating plus and minus ones, we can see that $h_{\text{hp}}(k)$ is merely $h_{\text{lp}}(k)$ with the sign changed for every other coefficient. Unlike $|H_{\text{bp}}(m)|$ in [Figure 5-28\(b\)](#), the $|H_{\text{hp}}(m)|$ response in [Figure 5-29\(b\)](#) has the same amplitude as the original $|H_{\text{lp}}(m)|$.

Again, notice that the $h_{\text{lp}}(k)$ lowpass coefficients in [Figure 5-29\(a\)](#) have not been modified by any window function. In practice, we'd use a windowed $h_{\text{lp}}(k)$ to reduce the passband ripple before implementing [Eq. \(5-21\)](#).

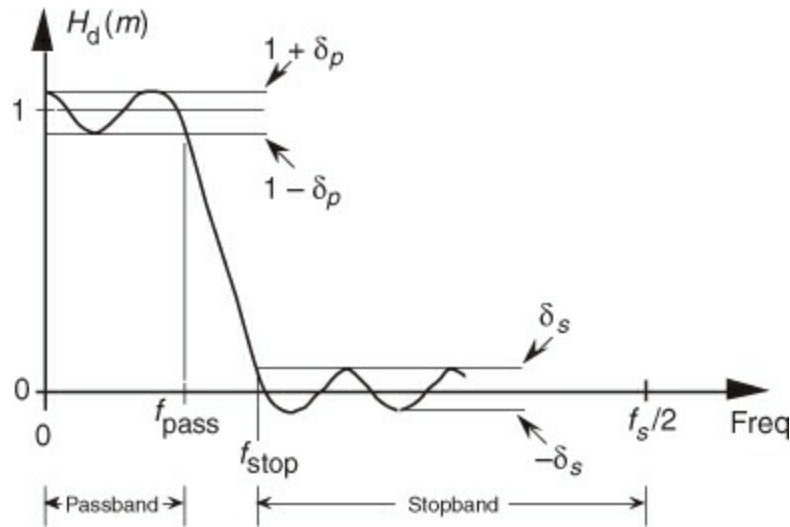
5.6 Parks-McClellan Exchange FIR Filter Design Method

Let's introduce one last FIR filter design technique that has found wide acceptance in practice. The

Parks-McClellan FIR filter design method (also called the Remez Exchange, or Optimal method[‡]) is a popular technique used to design high-performance FIR filters. To use this design method, we have to visualize a desired frequency response $H_d(m)$ like that shown in [Figure 5-30](#).

[‡] Remez is pronounced re-'mā.

Figure 5-30 Desired frequency response definition of a lowpass FIR filter using the Parks-McClellan Exchange design method.



We have to establish a desired passband cutoff frequency f_{pass} and the frequency where the attenuated stopband begins, f_{stop} . In addition, we must establish the variables δ_p and δ_s that define our desired passband and stopband ripple. Passband and stopband ripples, in decibels, are related to δ_p and δ_s by[\[15\]](#)

(5-22)

$$\text{Passband ripple} = 20 \cdot \log_{10}(1 + \delta_p)$$

and

(5-22')

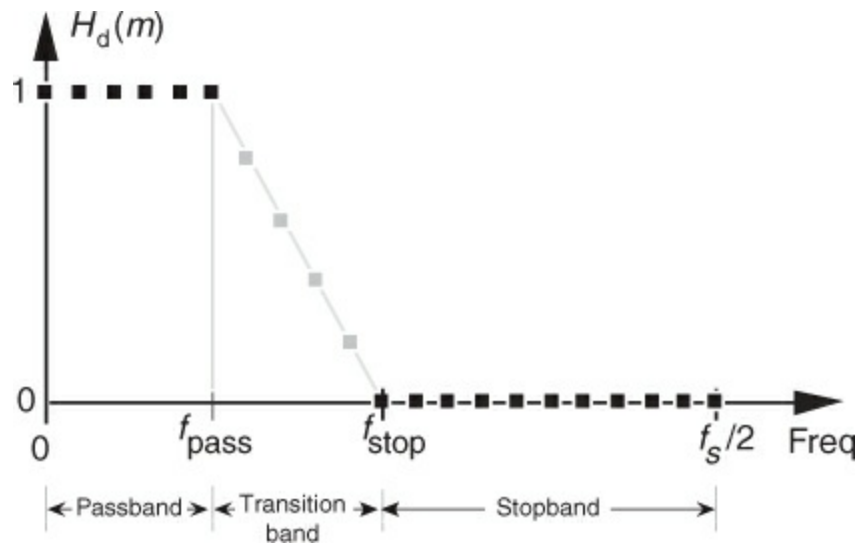
$$\text{Stopband ripple} = -20 \cdot \log_{10}(\delta_s).$$

(Some of the early journal papers describing the Parks-McClellan design method used the equally valid expression $-20 \cdot \log_{10}(\delta_p)$ to define the passband ripple in dB. However, [Eq. \(5-22\)](#) is the most common form used today.) Next, we apply these parameters to a computer software routine that generates the filter's N time-domain $h(k)$ coefficients where N is the minimum number of filter taps to achieve the desired filter response.

On the other hand, some software Parks-McClellan routines assume that we want δ_p and δ_s to be as small as possible and require us only to define the desired values of the $H_d(m)$ response as shown by the solid black dots in [Figure 5-31](#). The software then adjusts the values of the undefined (shaded dots) values of $H_d(m)$ to minimize the error between our desired and actual frequency response while minimizing δ_p and δ_s . The filter designer has the option to define some of the $H_d(m)$ values in the transition band, and the software calculates the remaining undefined $H_d(m)$ transition band values. With this version of the Parks-McClellan algorithm, the issue of most importance becomes how we define the transition region. We want to minimize its width while, at the same time, minimizing

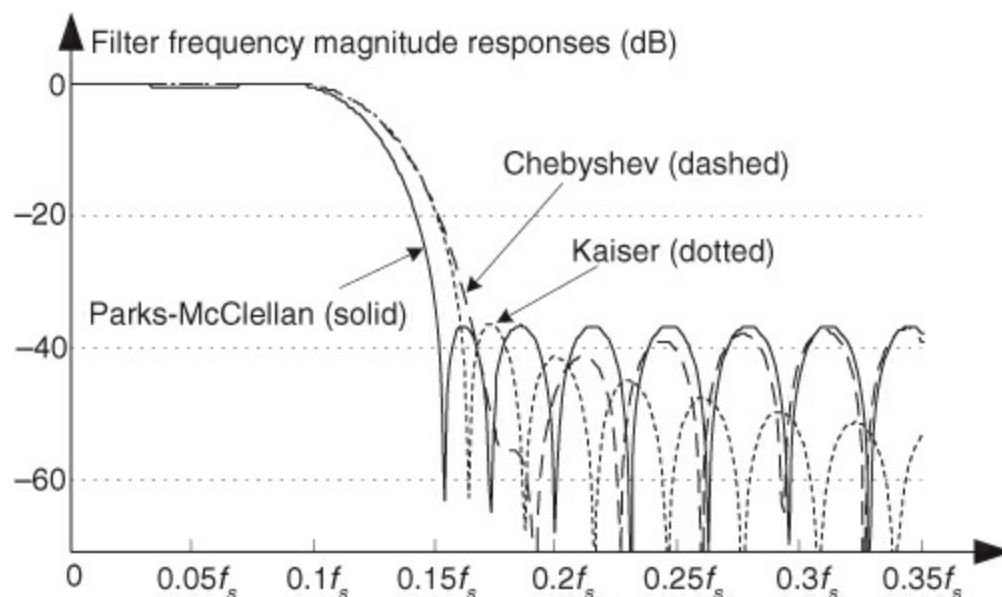
passband and stopband ripple. So exactly how we design an FIR filter using the Parks-McClellan Exchange technique is specific to the available filter design software. Although the mathematics involved in the development of the Parks-McClellan Exchange method is rather complicated, we don't have to worry about that here[16–20]. Just remember that the Parks-McClellan Exchange design method gives us a Chebyshev-type filter whose actual frequency response is as close as possible to the desired $H_d(m)$ response for a given number of filter taps.

Figure 5-31 Alternate method for defining the desired frequency response of a lowpass FIR filter using the Parks-McClellan Exchange technique.



To illustrate the advantage of the Parks-McClellan method, the solid curve in [Figure 5-32](#) shows the frequency response of a 31-tap FIR designed using this technique. For comparison, [Figure 5-32](#) also shows the frequency responses of two 31-tap FIR filters for the same passband width using the Chebyshev and Kaiser windowing techniques. Notice how the three filters have roughly the same stopband sidelobe levels, near the main lobe, but that the Parks-McClellan filter has the more desirable (steeper) transition band roll-off.

Figure 5-32 Frequency response comparison of three 31-tap FIR filters: Parks-McClellan, Chebyshev windowed, and Kaiser windowed.



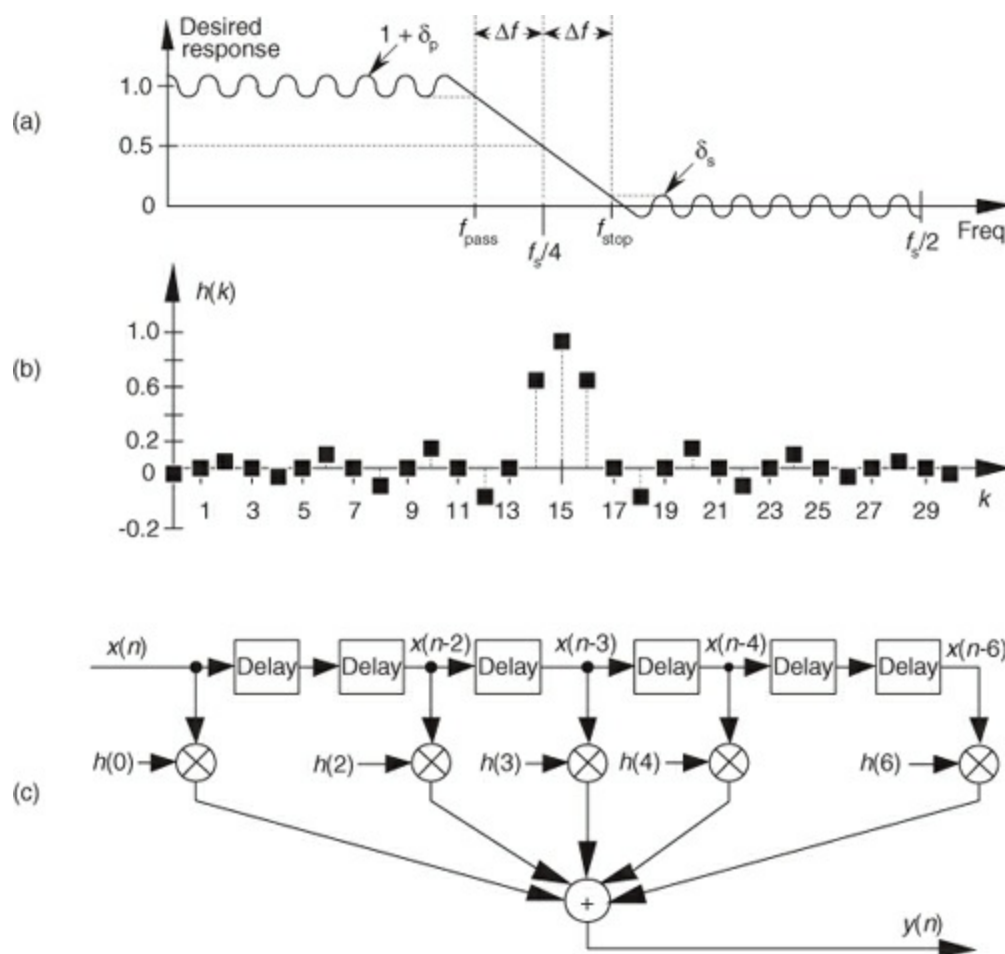
The Parks-McClellan Exchange filter design method revolutionized the art of, and has become the predominant technique for, designing linear-phase FIR filters. As a historical note, when Profs. Parks

and McClellan (James McClellan was a graduate student at the time) developed their triumphant filter design method in 1971, they submitted a paper to *Electronics Letters* to publicize their achievement. Surprisingly, the editors of *Electronics Letters* rejected the paper because the reviewers didn't believe that such a flexible, and optimized, FIR design procedure was possible. A description of Parks and McClellan's revolutionary design method was eventually published in reference [17]. That story is reminiscent of when Decca Records auditioned a group of four young musicians in 1961. Decca executives decided *not* to sign the group to a contract. You may have heard of that musical group—they were called the Beatles.

5.7 Half-band FIR Filters

There's a specialized FIR filter that's proved very useful in signal decimation and interpolation applications [21–25]. Called a *half-band* FIR filter, its frequency magnitude response is symmetrical about the $f_s/4$ point as shown in Figure 5-33(a). As such, the sum of f_{pass} and f_{stop} is $f_s/2$. When the filter has an odd number of taps, this symmetry has the beautiful property that the filter's time-domain impulse response has every other filter coefficient being zero, except the center coefficient. This enables us to avoid approximately half the number of multiplications when implementing this kind of filter. By way of example, Figure 5-33(b) shows the coefficients for a 31-tap half-band filter where Δf was defined to be approximately $f_s/32$ using the Parks-McClellan FIR filter design method.

Figure 5-33 Half-band FIR filter: (a) frequency magnitude response [transition region centered at $f_s/4$]; (b) 31-tap filter coefficients; (c) 7-tap half-band filter structure.



Notice how the alternating $h(k)$ coefficients are zero, so we perform 17 multiplications per output sample instead of the expected 31 multiplications. Stated in different words, we achieve the performance of a 31-tap filter at the computational expense of only 17 multiplies per output sample.

In the general case, for an N -tap half-band FIR filter, we'll only need to perform $(N + 1)/2 + 1$ multiplications per output sample. ([Section 13.7](#) shows a technique to further reduce the number of necessary multiplies for linear-phase tapped-delay line FIR filters, including half-band filters.) The structure of a simple seven-coefficient half-band filter is shown in [Figure 5-33\(c\)](#), with the $h(1)$ and $h(5)$ multipliers absent.

Be aware, there's a restriction on the number of half-band filter coefficients. To build linear-phase N -tap half-band FIR filters, having alternating zero-valued coefficients, $N + 1$ *must* be an integer multiple of four. If this restriction is not met, for example when $N = 9$, the first and last coefficients of the filter will both be equal to zero and can be discarded, yielding a 7-tap half-band filter.

On a practical note, there are two issues to keep in mind when we use an FIR filter design software package to design a half-band filter. First, assuming that the modeled filter has a passband gain of unity, ensure that your filter has a gain of 0.5 (–6 dB) at a frequency of $f_s/4$. Second, unavoidable numerical computation errors will yield alternate filter coefficients that are indeed very small but not exactly zero-valued as we desire. So in our filter modeling efforts, we must force those very small coefficient values to zero before we proceed to analyze half-band filter frequency responses.

You might sit back and think, “OK, these half-band filters are mildly interesting, but they're certainly not worth writing home about.” As it turns out, half-band filters are very important because they're widely used in applications with which you're familiar—like pagers, cell phones, digital receivers/televisions, CD/DVD players, etc. We'll learn more about half-band filter applications in [Chapter 10](#).

5.8 Phase Response of FIR Filters

Although we illustrated a couple of output phase shift examples for our original averaging FIR filter in [Figure 5-10](#), the subject of FIR phase response deserves additional attention. One of the dominant features of FIR filters is their linear phase response which we can demonstrate by way of example. Given the 25 $h(k)$ FIR filter coefficients in [Figure 5-34\(a\)](#), we can perform a DFT to determine the filter's $H(m)$ frequency response. The normalized real part, imaginary part, and magnitude of $H(m)$ are shown in [Figures 5-34\(b\)](#) and [5-34\(c\)](#), respectively.[‡] Being complex values, each $H(m)$ sample value can be described by its real and imaginary parts, or equivalently, by its magnitude $|H(m)|$ and its phase $H_\phi(m)$ shown in [Figure 5-35\(a\)](#).

[‡] Any DFT size greater than the $h(k)$ width of 25 is sufficient to obtain $H(m)$. The $h(k)$ sequence was padded with 103 zeros to take a 128-point DFT, resulting in the $H(m)$ sample values in [Figure 5-34](#).

Figure 5-34 FIR filter frequency response $H(m)$: (a) $h(k)$ filter coefficients; (b) real and imaginary parts of $H(m)$; (c) magnitude of $H(m)$.

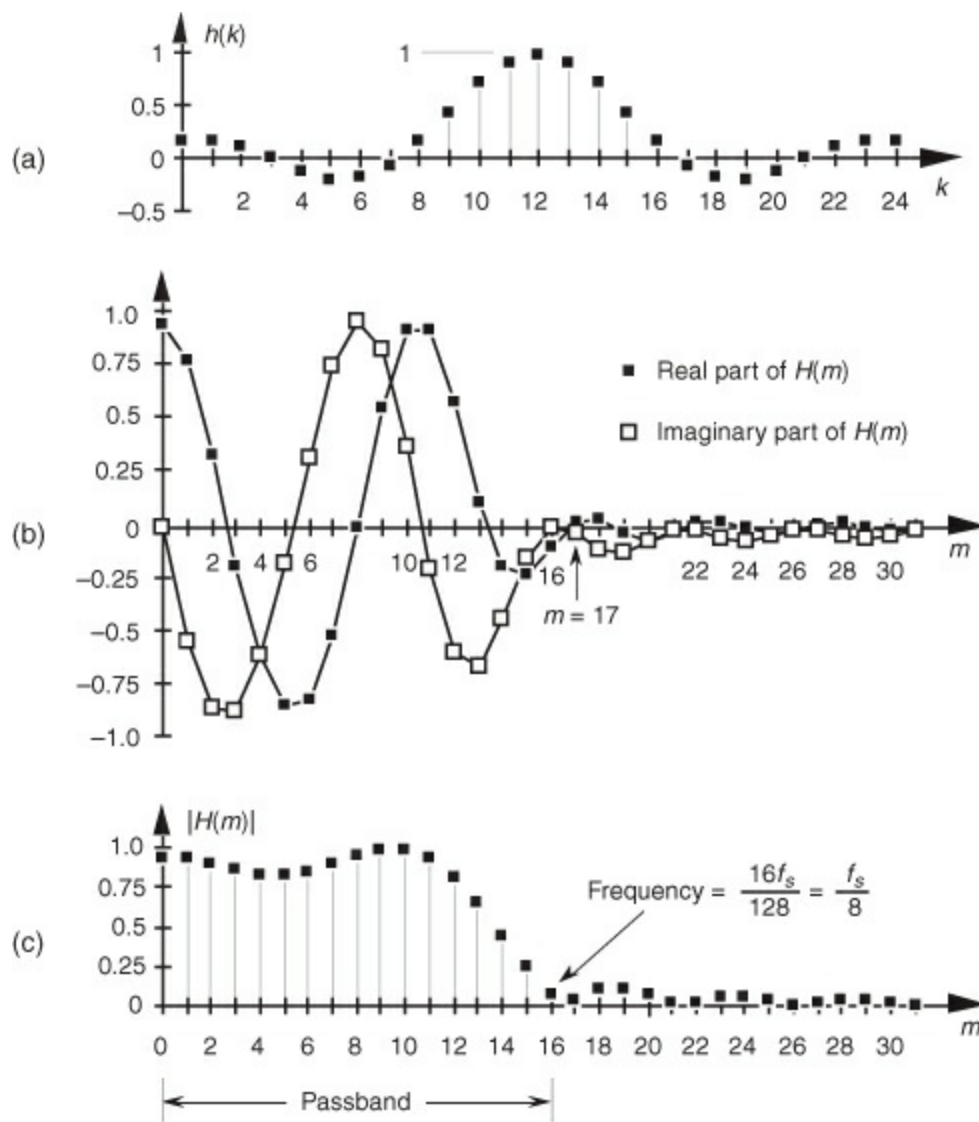
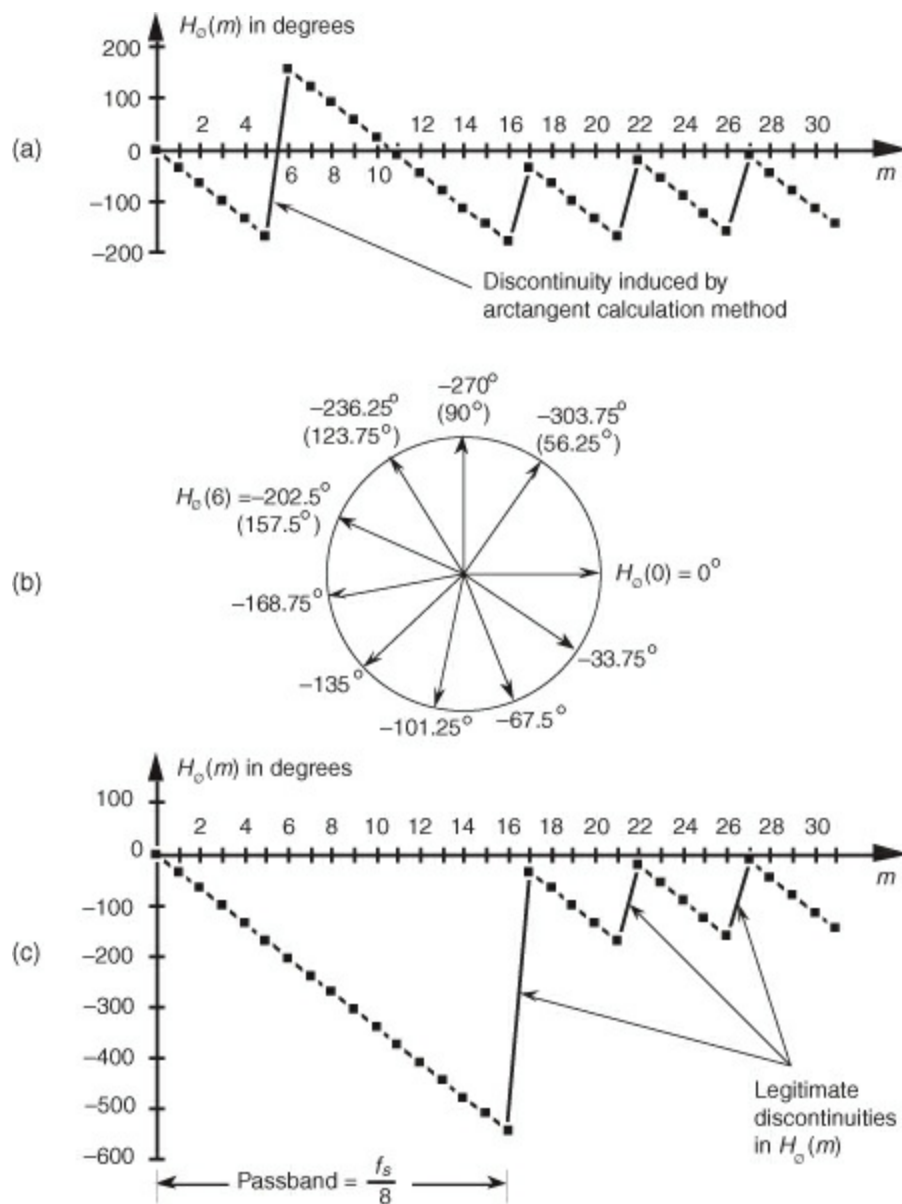


Figure 5-35 FIR filter phase response $H_0(m)$ in degrees: (a) calculated $H_0(m)$; (b) polar plot of $H_0(m)$'s first ten phase angles in degrees; (c) actual $H_0(m)$.



The phase of a complex quantity is, of course, the arctangent of the imaginary part divided by the real part, or $\phi = \tan^{-1}(\text{imag}/\text{real})$. Thus the phase of $H_0(m)$ is determined from the samples in [Figure 5-34\(b\)](#).

The phase response in [Figure 5-35\(a\)](#) certainly looks linear over selected frequency ranges, but what do we make of those sudden jumps, or discontinuities, in this phase response? If we were to plot the angles of $H_0(m)$ starting with the $m = 0$ sample on a polar graph, using the nonzero real part of $H(0)$, and the zero-valued imaginary part of $H(0)$, we'd get the zero-angled $H_0(0)$ phasor shown on the right side of [Figure 5-35\(b\)](#). Continuing to use the real and imaginary parts of $H(m)$ to plot additional phase angles results in the phasors going clockwise around the circle in increments of -33.75° . It's at the $H_0(6)$ that we discover the cause of the first discontinuity in [Figure 5-35\(a\)](#). Taking the real and imaginary parts of $H(6)$, we'd plot our phasor oriented at an angle of -202.5° . But [Figure 5-35\(a\)](#) shows that $H_0(6)$ is equal to 157.5° . The problem lies in the software routine used to generate the arctangent values plotted in [Figure 5-35\(a\)](#). The software adds 360° to any negative angles in the range of $-180^\circ > \phi \geq -360^\circ$, i.e., angles in the upper half of the circle. This makes ϕ a positive angle in the range of $0^\circ < \phi \leq 180^\circ$ and that's what gets plotted. (This apparent discontinuity between $H_0(5)$ and $H_0(6)$ is called *phase wrapping*.) So the true $H_0(6)$ of -202.5° is converted to a $+157.5^\circ$ as shown in parentheses in [Figure 5-35\(b\)](#). If we continue our polar plot for additional $H_0(m)$ values,

we'll see that their phase angles continue to decrease with an angle increment of -33.75° . If we compensate for the software's behavior and plot phase angles more negative than -180° , by *unwrapping* the phase, we get the true $H_o(m)$ shown in [Figure 5-35\(c\)](#).

Notice that $H_o(m)$ is, indeed, linear over the passband of $H(m)$. It's at $H_o(17)$ that our particular $H(m)$ experiences a polarity change of its real part while its imaginary part remains negative—this induces a true phase-angle discontinuity that really is a constituent of $H(m)$ at $m = 17$. (Additional phase discontinuities occur each time the real part of $H(m)$ reverses polarity, as shown in [Figure 5-35\(c\)](#).) The reader may wonder why we care about the linear phase response of $H(m)$. The answer, an important one, requires us to introduce the notion of group delay.

[*Group delay*](#) is defined as the negative of the derivative of the phase with respect to frequency, or $G = -d\phi/df$. For FIR filters, then, group delay is the slope of the $H_o(m)$ response curve. When the group delay is constant, as it is over the passband of all FIR filters having symmetrical coefficients, all frequency components of the filter input signal are delayed by an equal amount of time G before they reach the filter's output. This means that no phase distortion is induced in the filter's desired output signal, and this is crucial in communications signals. For amplitude modulation (AM) signals, constant group delay preserves the time waveform shape of the signal's modulation envelope. That's important because the modulation portion of an AM signal contains the signal's information. Conversely, a nonlinear phase will distort the audio of AM broadcast signals, blur the edges of television video images, blunt the sharp edges of received radar pulses, and increase data errors in digital communications signals. (Group delay is sometimes called [*envelope delay*](#) because group delay was originally the subject of analysis due to its effect on the envelope, or modulation signal, of amplitude modulation AM systems.) Of course we're not really concerned with the group delay outside the passband because signal energy outside the passband is what we're trying to eliminate through filtering.

Over the passband frequency range for a linear-phase, S -tap FIR filter, group delay has been shown to be given by

$$(5-23)$$

$$G = \frac{D \cdot t_s}{2} \text{ seconds,}$$

where $D = S-1$ is the number of unit-delay elements in the filter's delay line, and t_s is the sample period ($1/f_s$).[‡] This group delay is measured in seconds. Eliminating the t_s factor in [Eq. \(5-23\)](#) would change its dimensions to samples. The value G , measured in samples, is always an integer for odd-tap FIR filters and a noninteger for even-tap filters.

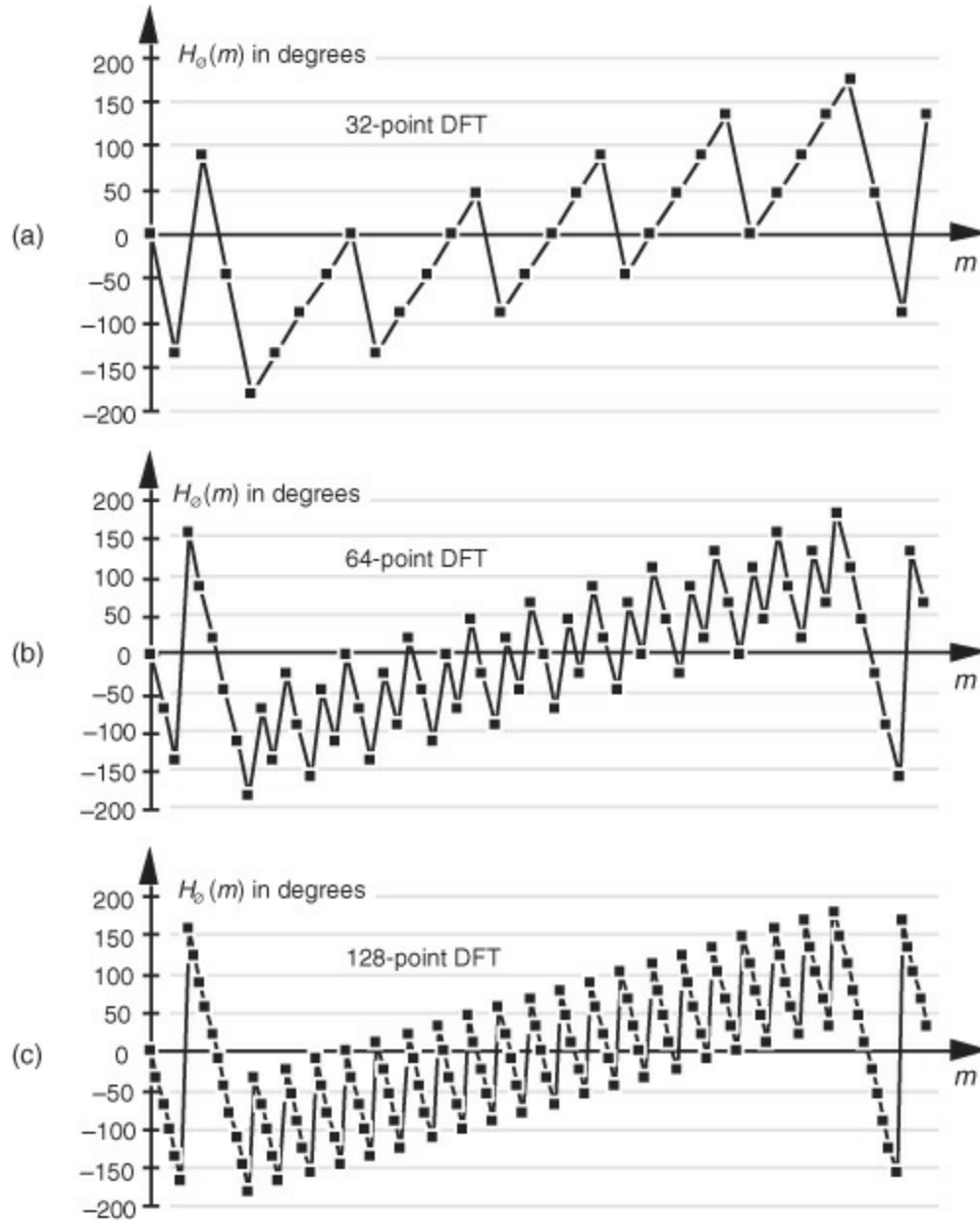
[‡] As derived in [Section 3.4](#) of reference [\[16\]](#), and page 597 of reference [\[19\]](#).

Although we used a 128-point DFT to obtain the frequency responses in [Figures 5-34](#) and [5-35](#), we could just as well have used $N = 32$ -point or $N = 64$ -point DFTs. These smaller DFTs give us the phase response curves shown in [Figures 5-36\(a\)](#) and [5-36\(b\)](#). Notice how different the phase response curves are when $N = 32$ in [Figure 5-36\(a\)](#) compared to when $N = 128$ in [Figure 5-36\(c\)](#). The phase-angle resolution is much finer in [Figure 5-36\(c\)](#). The passband phase-angle resolution, or increment $\Delta\phi$, is given by

$$(5-24)$$

$$\Delta\theta = \frac{-G \cdot 360^\circ}{N},$$

Figure 5-36 FIR filter phase response $H_\theta(m)$ in degrees: (a) calculated using a 32-point DFT; (b) using a 64-point DFT; (c) using a 128-point DFT.



where N is the number of points in the DFT. So, for our $S = 25$ -tap filter in [Figure 5-34\(a\)](#), $G = 12$, and $\Delta\theta$ is equal to $-12 \cdot 360^\circ/32 = -135^\circ$ in [Figure 5-36\(a\)](#), and $\Delta\theta$ is -33.75° in [Figure 5-36\(c\)](#). If we look carefully at the sample values in [Figure 5-36\(a\)](#), we'll see that they're all included within the samples in [Figures 5-36\(b\)](#) and [5-36\(c\)](#).

Let's conclude this FIR phase discussion by reiterating the meaning of phase response. The phase, or phase delay, at the output of an FIR filter is the phase of the first output sample relative to the phase of the filter's first input sample. Over the passband, that phase shift, of course, is a linear function of frequency. This will be true only as long as the filter has symmetrical coefficients. [Figure 5-10](#) is a good illustration of an FIR filter's output phase delay.

For FIR filters, the output phase shift measured in degrees, for the passband frequency $f = mf_s/N$, is expressed as

$$(5-25)$$

$$\text{phase delay} = H_{\phi}(mf_s / N) = m \cdot \Delta\phi = \frac{-m \cdot G \cdot 360^\circ}{N}.$$

We can illustrate [Eq. \(5-25\)](#) and show the relationship between the phase responses in [Figure 5-36](#) by considering the phase delay associated with the frequency of $f_s/32$ in [Table 5-2](#). The subject of group delay is described further in [Appendix F](#), where an example of envelope delay distortion, due to a filter’s nonlinear phase, is illustrated.

Table 5-2 Values Used in [Eq. \(5-25\)](#) for the Frequency $f_s/32$

DFT size, N	Index m	$H_{\phi}(mf_s/N)$
32	1	-135°
64	2	-135°
128	4	-135°

5.9 A Generic Description of Discrete Convolution

Although convolution was originally an analysis tool used to prove continuous signal processing theorems, we now know that convolution affects every aspect of digital signal processing. Convolution influences our results whenever we analyze or filter any finite set of data samples from a linear time-invariant system. Convolution not only constrains DFTs to be just approximations of the continuous Fourier transform; it is the reason that discrete spectra are periodic in the frequency domain. It’s interesting to note that, although we use the process of convolution to implement FIR digital filters, convolution effects induce frequency response ripple, preventing us from ever building a perfect digital filter. Its influence is so pervasive that to repeal the law of convolution, quoting a phrase from Dr. Who, would “unravel the entire causal nexus” of digital signal processing.

Convolution has always been a somewhat difficult concept for the beginner to grasp. That’s not too surprising for several reasons. Convolution’s effect on discrete signal processing is not intuitively obvious for those without experience working with discrete signals, and the mathematics of convolution does seem a little puzzling at first. Moreover, in their sometimes justified haste, many authors present the convolution equation and abruptly start using it as an analysis tool without explaining its origin and meaning. For example, this author once encountered what was called a *tutorial* article on the FFT in a professional journal that proceeded to define *convolution* merely by presenting something like that shown in [Figure 5-37](#) with no further explanation!

Figure 5-37 One very efficient, but perplexing, way of defining convolution.

$$Y_j = \sum_{k=0}^{N-1} P_k \cdot Q_{j-k}, \text{ or}$$

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ \vdots \\ Y_{N-1} \end{bmatrix} = \begin{bmatrix} Q_0 & Q_{N-1} & Q_{N-2} & \dots & Q_1 \\ Q_1 & Q_0 & Q_{N-1} & \dots & Q_2 \\ Q_2 & Q_1 & Q_0 & \dots & Q_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ Q_{N-1} & Q_{N-2} & Q_{N-3} & \dots & Q_0 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ \vdots \\ P_{N-1} \end{bmatrix}$$

Theorem: if

$$P_j \leftarrow \text{DFT} \rightarrow A_n,$$

$$Q_j \leftarrow \text{DFT} \rightarrow B_n, \text{ and}$$

$$Y_j \leftarrow \text{DFT} \rightarrow C_n,$$

then

$$C_n = N \cdot A_n \cdot B_n.$$

Unfortunately, few beginners can gain an understanding of the convolution process from [Figure 5-37](#) alone. Here, we avoid this dilemma by defining the process of convolution and gently proceed through a couple of simple convolution examples. We conclude this chapter with a discussion of the powerful convolution theorem and show why it's so useful as a qualitative tool in discrete system analysis.

5.9.1 Discrete Convolution in the Time Domain

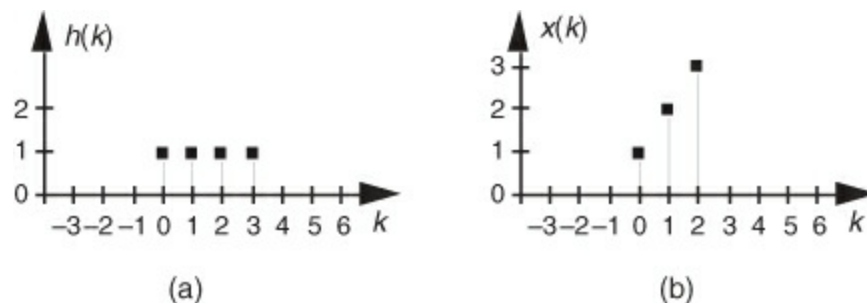
Discrete convolution is a process whose input is two sequences and that provides a single output sequence. Convolution inputs can be two time-domain sequences giving a time-domain output, or two frequency-domain input sequences providing a frequency-domain result. (Although the two input sequences must both be in the same domain for the process of convolution to have any practical meaning, their sequence lengths need not be the same.) Let's say we have two input sequences $h(k)$ of length P and $x(k)$ of length Q in the time domain. The output sequence $y(n)$ of the convolution of the two inputs is defined mathematically as

(5-26)

$$y(n) = \sum_{k=0}^{P+Q-2} h(k)x(n-k).$$

Let's examine [Eq. \(5-26\)](#) by way of example, using the $h(k)$ and $x(k)$ sequences shown in [Figure 5-38](#). In this example, we can write the terms for each $y(n)$ in [Eq. \(5-26\)](#) as

Figure 5-38 Convolution example input sequences: (a) first sequence $h(k)$ of length $P = 4$; (b) second sequence $x(k)$ of length $Q = 3$.



(5-27)

$$\begin{aligned}
y(0) &= h(0)x(0-0) + h(1)x(0-1) + h(2)x(0-2) + h(3)x(0-3) + h(4)x(0-4) + h(5)x(0-5), \\
y(1) &= h(0)x(1-0) + h(1)x(1-1) + h(2)x(1-2) + h(3)x(1-3) + h(4)x(1-4) + h(5)x(1-5), \\
y(2) &= h(0)x(2-0) + h(1)x(2-1) + h(2)x(2-2) + h(3)x(2-3) + h(4)x(2-4) + h(5)x(2-5), \\
y(3) &= h(0)x(3-0) + h(1)x(3-1) + h(2)x(3-2) + h(3)x(3-3) + h(4)x(3-4) + h(5)x(3-5), \\
y(4) &= h(0)x(4-0) + h(1)x(4-1) + h(2)x(4-2) + h(3)x(4-3) + h(4)x(4-4) + h(5)x(4-5),
\end{aligned}$$

and

$$y(5) = h(0)x(5-0) + h(1)x(5-1) + h(2)x(5-2) + h(3)x(5-3) + h(4)x(5-4) + h(5)x(5-5).$$

With $P = 4$ and $Q = 3$, we need evaluate only $4 + 3 - 1 = 6$ individual $y(n)$ terms. Because $h(4)$ and $h(5)$ are zero, we can eliminate some of the terms in [Eq. \(5-27\)](#) and evaluate the remaining $x(n-k)$ indices, giving the following expressions for $y(n)$ as

(5-28)

$$\begin{aligned}
y(0) &= h(0)x(0) + h(1)x(-1) + h(2)x(-2) + h(3)x(-3), \\
y(1) &= h(0)x(1) + h(1)x(0) + h(2)x(-1) + h(3)x(-2), \\
y(2) &= h(0)x(2) + h(1)x(1) + h(2)x(0) + h(3)x(-1), \\
y(3) &= h(0)x(3) + h(1)x(2) + h(2)x(1) + h(3)x(0), \\
y(4) &= h(0)x(4) + h(1)x(3) + h(2)x(2) + h(3)x(1),
\end{aligned}$$

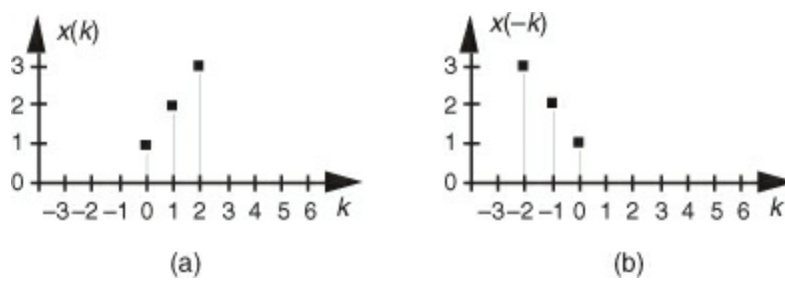
and

$$y(5) = h(0)x(5) + h(1)x(4) + h(2)x(3) + h(3)x(2).$$

Looking at the indices of the $h(k)$ and $x(k)$ terms in [Eq. \(5-28\)](#), we see two very important things occurring. First, convolution is merely the summation of a series of products—so the process itself is not very complicated. Second, notice that, for a given $y(n)$, $h(k)$'s index is increasing as $x(k)$'s index is decreasing. This fact has led many authors to introduce a new sequence $x(-k)$ and use that new sequence to graphically illustrate the convolution process. The $x(-k)$ sequence is simply our original $x(k)$ reflected about the 0 index of the k axis as shown in [Figure 5-39](#). Defining $x(-k)$ as such enables us to depict the products and summations of [Eq. \(5-28\)](#)'s convolution as in [Figure 5-40](#); that is, we can now align the $x(-k)$ samples with the samples of $h(k)$ for a given n index to calculate $y(n)$. As shown in [Figure 5-40\(a\)](#), the alignment of $h(k)$ and $x(n-k)$, for $n = 0$, yields $y(0) = 1$. This is the result of the first line in [Eq. \(5-28\)](#) repeated on the right side of [Figure 5-40\(a\)](#). The calculation of $y(1)$, for $n = 1$, is depicted in [Figure 5-40\(b\)](#), where $x(n-k)$ is shifted one element to the right, resulting in $y(1) = 3$. We continue this $x(n-k)$ shifting and incrementing n until we arrive at the last nonzero convolution result of $y(5)$ shown in [Figure 5-40\(f\)](#). So, performing the convolution of $h(k)$ and $x(k)$ comprises

1. plotting both the $h(k)$ and $x(k)$ sequences,
2. flipping the $x(k)$ sequence around the $k = 0$ sample to obtain $x(-k)$,
3. summing the products of $h(k)$ and $x(0-k)$ for all k to yield $y(0)$,
4. shifting the $x(-k)$ sequence one sample to the right,
5. summing the products of $h(k)$ and $x(1-k)$ for all k to obtain $y(1)$, and
6. continuing to shift $x(-k)$ and sum products until there's no overlap of $h(k)$ and the shifted $x(n-k)$, in which case all further $y(n)$ output samples are zero and we're done.

Figure 5-39 Convolution example input sequence: (a) second sequence $x(k)$ of length 3; (b) reflection of the second sequence about the $k = 0$ index.

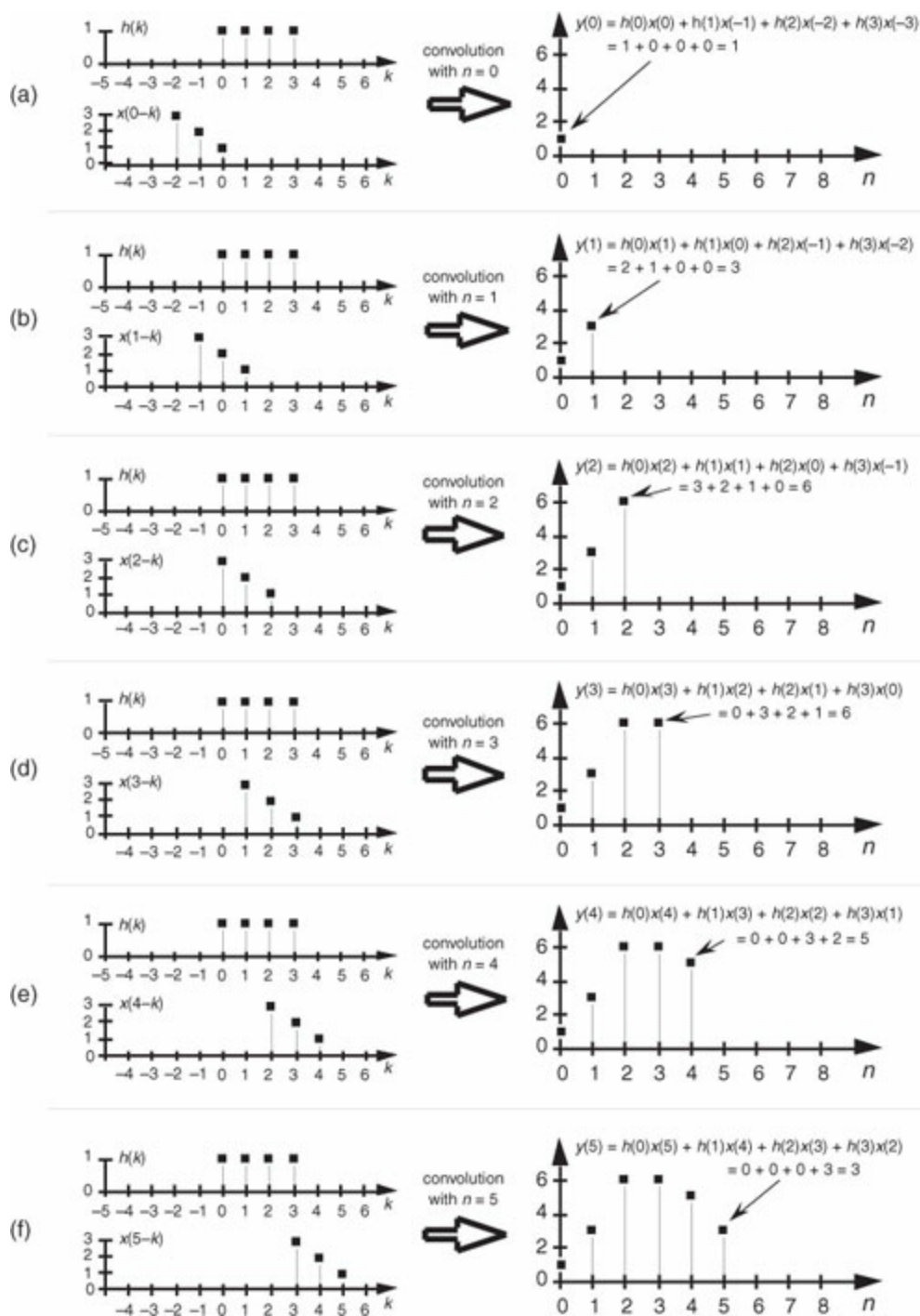


The full convolution of our $h(k)$ and $x(k)$ is the $y(n)$ sequence on the right side of [Figure 5-40\(f\)](#). We've scanned the $x(-k)$ sequence across the $h(k)$ sequence and summed the products where the sequences overlap. By the way, notice that the $y(n)$ sequence in [Figure 5-40\(f\)](#) has six elements where $h(k)$ had a length of four and $x(k)$ was of length three. In the general case, if $h(k)$ is of length P and $x(k)$ is of length Q , the length of $y(n)$ will have a sequence length of L , where

$$(5-29)$$

$$L = P + Q - 1.$$

Figure 5-40 Graphical depiction of the convolution of $h(k)$ and $x(k)$ in [Figure 5-38](#).



At this point, it's fair for the beginner to ask, "OK, so what? What does this *strange* convolution process have to do with digital signal processing?" The answer to that question lies in understanding the effects of the convolution theorem.

5.9.2 The Convolution Theorem

The convolution theorem is a fundamental constituent of digital signal processing. It impacts our results anytime we filter or Fourier transform discrete data. To see why this is true, let's simplify the notation of [Eq. \(5-26\)](#) and use the abbreviated form

$$(5-30)$$

$$y(n) = h(k) * x(k),$$

where, again, the "*" symbol means convolution. The convolution theorem may be stated as follows: If two time-domain sequences $h(k)$ and $x(k)$ have DFTs of $H(m)$ and $X(m)$, respectively, then the DFT of $h(k) * x(k)$ is the product $H(m) \cdot X(m)$. Likewise, the inverse DFT of $H(m) \cdot X(m)$ is $h(k) * x(k)$.

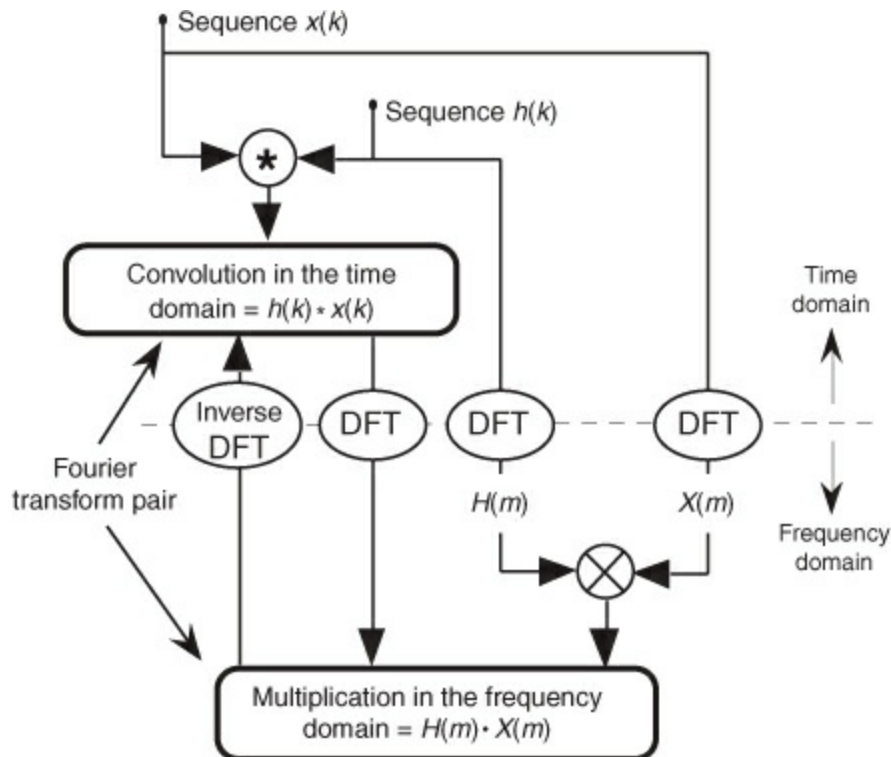
We can represent this relationship with the expression

(5-31)

$$h(k) * x(k) \xrightleftharpoons[\text{IDFT}]{\text{DFT}} H(m) \cdot X(m).$$

[Equation \(5-31\)](#) tells us that two sequences resulting from $h(k) * x(k)$ and $H(m) \cdot X(m)$ are Fourier transform pairs. So, taking the DFT of $h(k) * x(k)$ always gives us $H(m) \cdot X(m)$. Likewise, we can determine $h(k) * x(k)$ by taking the inverse DFT of $H(m) \cdot X(m)$. The important point to learn from [Eq. \(5-31\)](#) is that convolution in the time domain is equivalent to multiplication in the frequency domain. (We won't derive the convolution theorem here because its derivation is readily available to the interested reader[[26–29](#)].) To help us appreciate this principle, [Figure 5-41](#) sketches the relationship between convolution in the time domain and multiplication in the frequency domain.

Figure 5-41 Relationships of the convolution theorem.



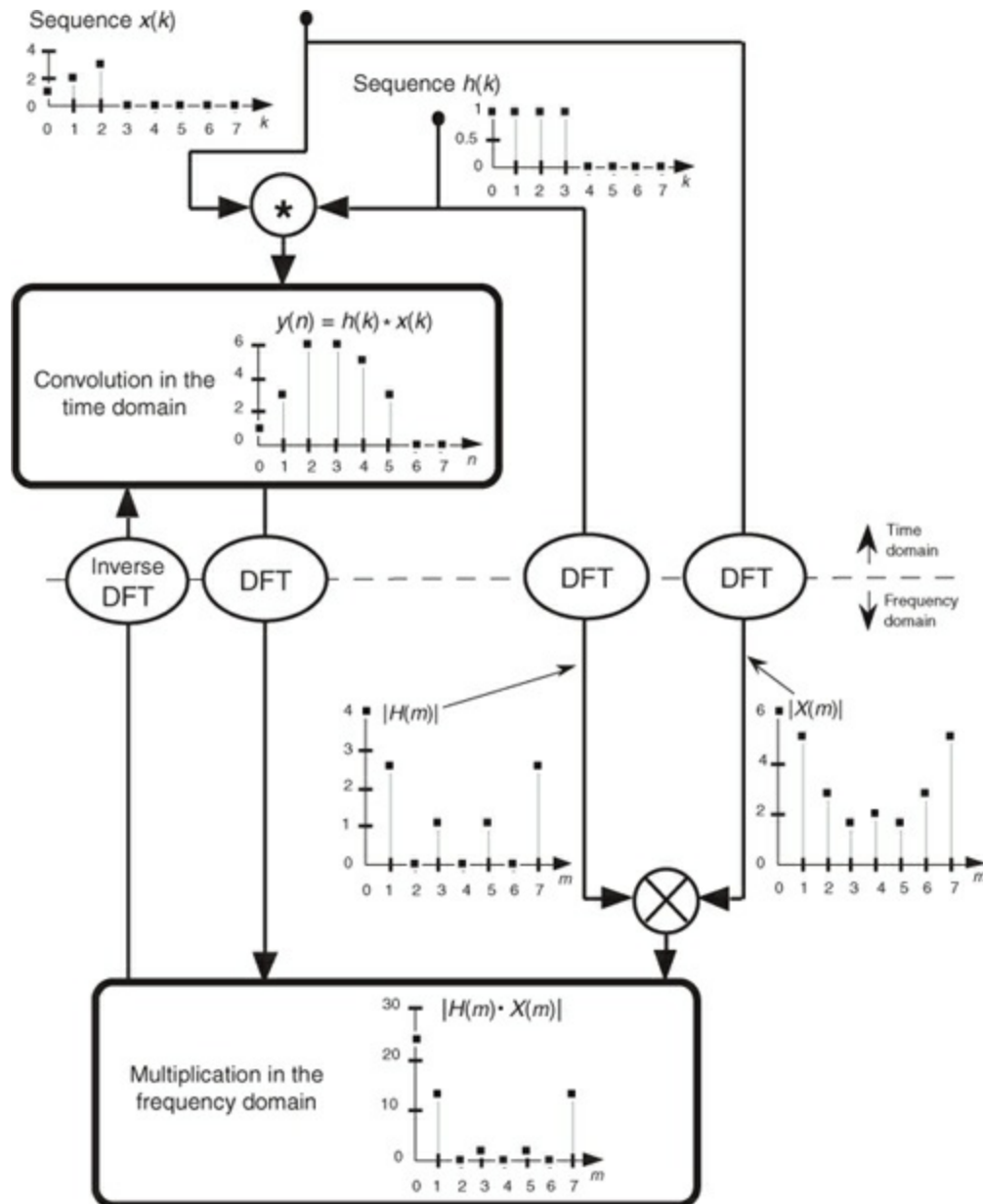
We can easily illustrate the convolution theorem by taking 8-point DFTs of $h(k)$ and $x(k)$ to get $H(m)$ and $X(m)$, respectively, and listing these values as in [Table 5-3](#). (Of course, we have to pad $h(k)$ and $x(k)$ with zeros, so they both have lengths of 8 to take 8-point DFTs.) Tabulating the inverse DFT of the product $H(m) \cdot X(m)$ allows us to verify [Eq. \(5-31\)](#), as listed in the last two columns of [Table 5-3](#), where the acronym IDFT again means inverse DFT. The values from [Table 5-3](#) are shown in [Figure 5-42](#). (For simplicity, only the magnitudes of $H(m)$, $X(m)$, and $H(m) \cdot X(m)$ are shown in the figure.) We need to become comfortable with convolution in the time domain because, as we've learned, it's the process used in FIR filters. As detailed in [Section 5.2](#), we perform discrete time-domain FIR filtering by convolving an input sequence, $x(n)$ say, with the impulse response $h(k)$ of a filter, and for FIR filters that impulse response happens to also be the filter's coefficients.[‡] The result of that convolution is a filtered time-domain sequence whose spectrum is modified (multiplied) by the filter's frequency response $X(m)$. [Section 13.10](#) describes a clever scheme to perform FIR filtering efficiently using the FFT algorithm to implement convolution.

[‡] As we'll see in [Chapter 6](#), the coefficients used for an infinite impulse response (IIR) filter are not equal to that filter's impulse response.

Table 5-3 Convolution Values of $h(k)$ and $x(k)$ from [Figure 5-38](#)

Index k or m	$h(k)$	$x(k)$	DFT of $h(k) = H(m)$	DFT of $x(k) = X(m)$	$H(m) \cdot X(m)$	IDFT of $H(m) \cdot X(m)$	$h(k) * x(k)$
0	1	1	$4.0 + j0.0$	$6.0 + j0.0$	$24.0 + j0.0$	$1.0 + j0.0$	1
1	1	2	$1.00 - j2.41$	$2.41 - j4.41$	$-8.24 - j10.24$	$3.0 + j0.0$	3
2	1	3	0	$-2.0 - j2.0$	0	$6.0 + j0.0$	6
3	1	0	$1.00 - j0.41$	$-0.41 + j1.58$	$0.24 + j1.75$	$6.0 + j0.0$	6
4	0	0	0	$2.0 + j0.0$	0	$5.0 + j0.0$	5
5	0	0	$1.00 + j0.41$	$-0.41 - j1.58$	$0.24 - j1.75$	$3.0 + j0.0$	3
6	0	0	0	$-2.00 + j2.00$	0	$0.0 + j0.0$	0
7	0	0	$1.00 + j2.41$	$2.41 + j4.41$	$-8.24 + j10.24$	$0.0 + j0.0$	0

Figure 5-42 Convolution relationships of $h(k)$, $x(k)$, $H(m)$, and $X(m)$ from [Figure 5-38](#).



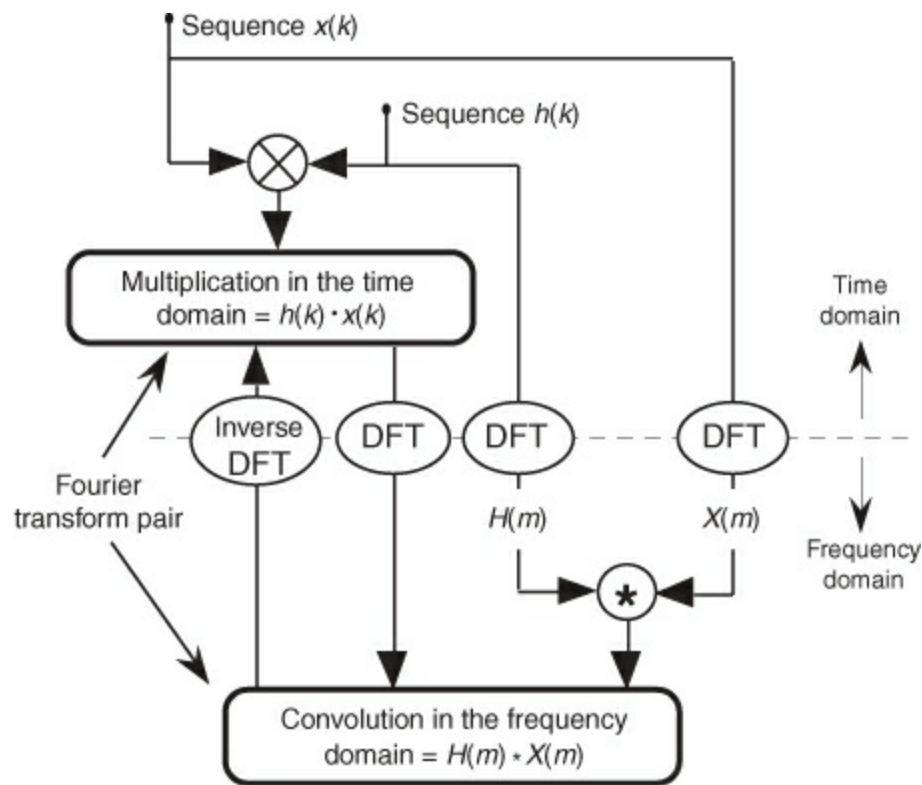
Because of the duality of the convolution theorem, we could have swapped the time and frequency domains in our discussion of convolution and multiplication being a Fourier transform pair. This means that, similar to [Eq. \(5-31\)](#), we can also write

$$(5-32)$$

$$h(k) \cdot x(k) \xrightleftharpoons[\text{IDFT}]{\text{DFT}} H(m) * X(m).$$

So the convolution theorem can be stated more generally as *Convolution in one domain is equivalent to multiplication in the other domain*. [Figure 5-43](#) shows the relationship between multiplication in the time domain and convolution in the frequency domain. [Equation \(5-32\)](#) is the fundamental relationship used in the process of windowing time-domain data to reduce DFT leakage, as discussed in [Section 3.9](#).

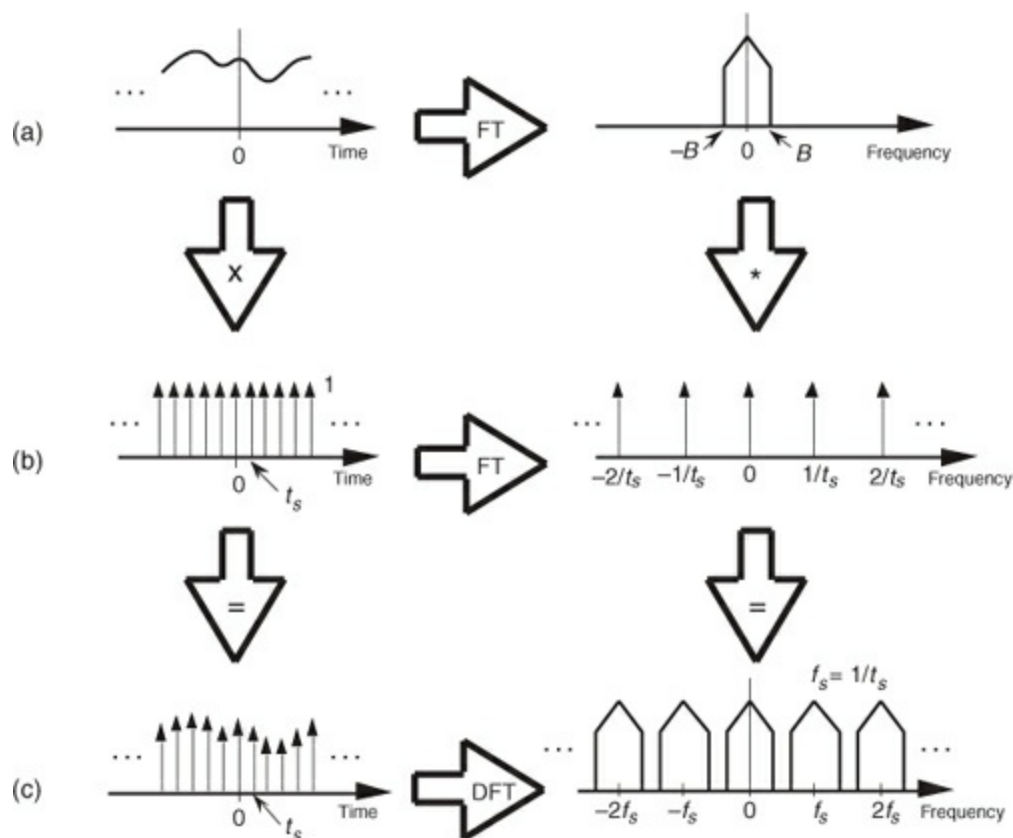
Figure 5-43 Relationships of the convolution theorem related to multiplication in the time domain.



5.9.3 Applying the Convolution Theorem

The convolution theorem is useful as a qualitative tool in predicting the effects of different operations in discrete linear time-invariant systems. For example, many authors use the convolution theorem to show why periodic sampling of continuous signals results in discrete samples whose spectra are periodic in the frequency domain. Consider the real continuous time-domain waveform in [Figure 5-44\(a\)](#), with the one-sided spectrum of bandwidth B . Being a real signal, of course, its spectrum is symmetrical about 0 Hz. (In [Figure 5-44](#), the large right-pointing arrows represent Fourier transform operations.) Sampling this waveform is equivalent to multiplying it by a sequence of periodically spaced impulses, [Figure 5-44\(b\)](#), whose values are unity. If we say that the sampling rate is f_s samples/second, then the sample period $t_s = 1/f_s$ seconds. The result of this multiplication is the sequence of discrete time-domain impulses shown in [Figure 5-44\(c\)](#). We can use the convolution theorem to help us predict what the frequency-domain effect is of this multiplication in the time domain. From our theorem, we now realize that the spectrum of the time-domain product must be the convolution of the original spectra. Well, we know what the spectrum of the original continuous waveform is. What about the spectrum of the time-domain impulses? It has been shown that the spectrum of periodic impulses, whose period is t_s seconds, is also periodic impulses in the frequency domain with a spacing of f_s Hz as shown in [Figure 5-44\(b\)\[30\]](#).

Figure 5-44 Using convolution to predict the spectral replication effects of periodic sampling.



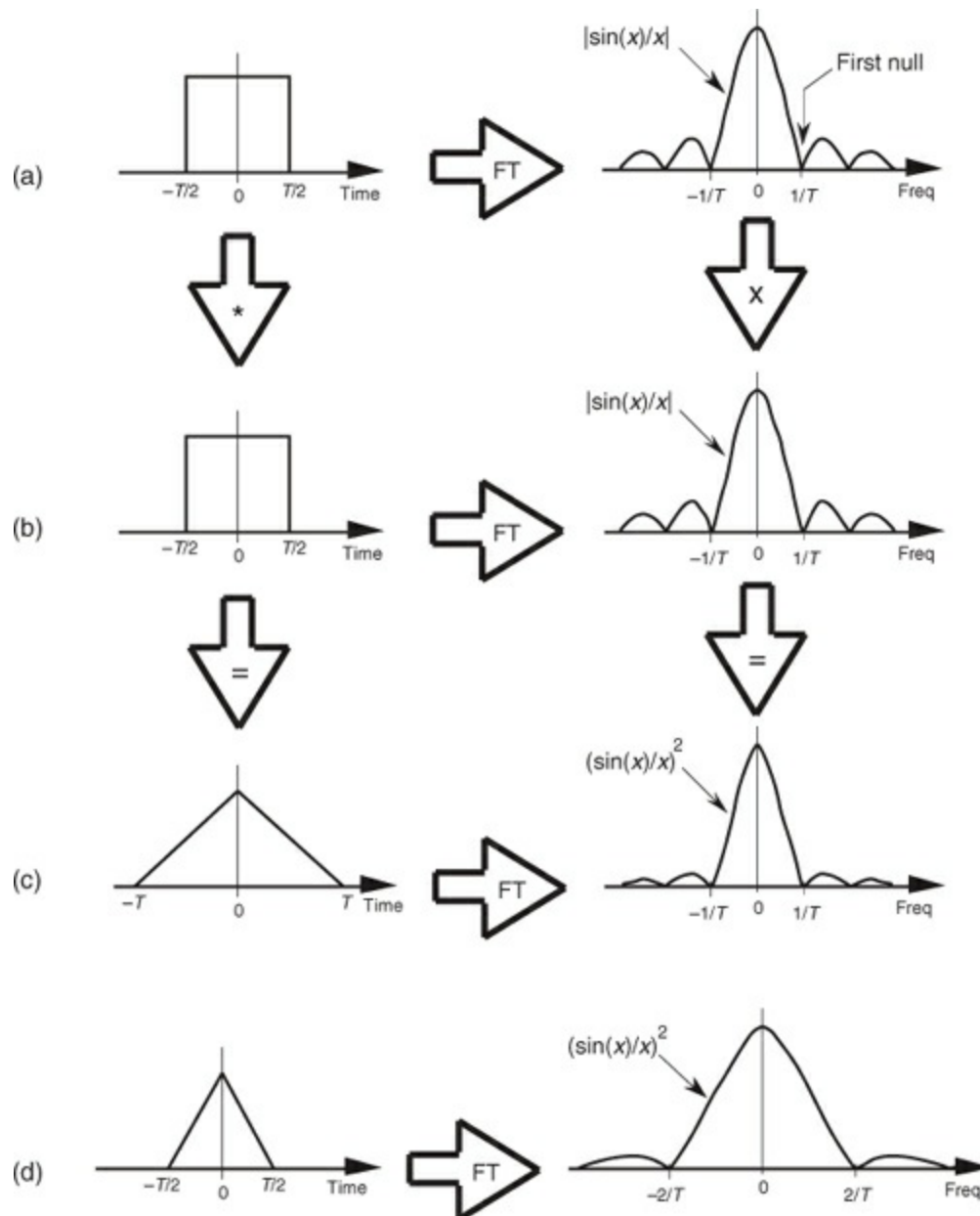
Now, all we have to do is convolve the two spectra. In this case, convolution is straightforward because both of the frequency-domain functions are symmetrical about the zero-Hz point, and flipping one of them about zero Hz is superfluous. So we merely slide one of the functions across the other and plot the product of the two. The convolution of the original waveform spectrum and the spectral impulses results in replications of the waveform spectrum every f_s Hz, as shown in [Figure 5-44\(c\)](#). This discussion reiterates the fact that the DFT is always periodic with a period of f_s Hz.

Here's another example of how the convolution theorem can come in handy when we try to understand digital signal processing operations. This author once used the theorem to resolve the puzzling result, at the time, of a triangular window function having its first frequency response null at twice the frequency of the first null of a rectangular window function. The question was "If a rectangular time-domain function of width T has its first spectral null at $1/T$ Hz, why does a triangular time-domain function of width T have its first spectral null at $2/T$ Hz?" We can answer this question by considering convolution in the time domain.

Look at the two rectangular time-domain functions shown in [Figures 5-45\(a\)](#) and [5-45\(b\)](#). If their widths are each T seconds, their spectra are shown to have nulls at $1/T$ Hz as depicted in the frequency-domain functions in [Figures 5-45\(a\)](#) and [5-45\(b\)](#). We know that the frequency magnitude responses will be the absolute value of the classic $\sin(x)/x$ function.[‡] If we convolve those two rectangular time-domain functions of width T , we'll get the triangular function shown in [Figure 5-45\(c\)](#). Again, in this case, flipping one rectangular function about the zero time axis is unnecessary. To convolve them, we need only scan one function across the other and determine the area of their overlap. The time shift where they overlap the most happens to be a zero time shift. Thus, our resultant convolution has a peak at a time shift of zero seconds because there's 100 percent overlap. If we slide one of the rectangular functions in either direction, the convolution decreases linearly toward zero. When the time shift is $T/2$ seconds, the rectangular functions have a 50 percent overlap. The convolution is zero when the time shift is T seconds—that's when the two rectangular functions cease to overlap.

† The $\sin(x)/x$ function was introduced in our discussion of window functions in [Section 3.9](#) and is covered in greater detail in [Section 3.13](#).

Figure 5-45 Using convolution to show that the Fourier transform of a triangular function has its first null at twice the frequency of the Fourier transform of a rectangular function.



Notice that the triangular convolution result has a width of $2T$, and that's really the key to answering our question. Because convolution in the time domain is equivalent to multiplication in the frequency domain, the Fourier transform magnitude of our $2T$ -width triangular function is the $|\sin(x)/x|$ in [Figure 5-45\(a\)](#) times the $|\sin(x)/x|$ in [Figure 5-45\(b\)](#), or the $(\sin(x)/x)^2$ function in [Figure 5-45\(c\)](#). If a triangular function of width $2T$ has its first frequency-domain null at $1/T$ Hz, then the same function of width T must have its first frequency null at $2/T$ Hz as shown in [Figure 5-45\(d\)](#), and that's what we set out to show. Comparison of [Figures 5-45\(c\)](#) and [5-45\(d\)](#) illustrates a fundamental Fourier transform property that compressing a function in the time domain results in an expansion of its corresponding frequency-domain representation.

We cannot overemphasize the importance of the convolution theorem as an analysis tool. As an aside, for years I thought convolution was a process developed in the second half of the twentieth century to help us analyze discrete-time signal processing systems. Later I learned that statisticians had been using convolution since the late 1800s. In statistics the probability density function (PDF) of the sum

of two random variables is the convolution of their individual PDFs.

5.10 Analyzing FIR Filters

There are two popular ways to analyze tapped-delay line, nonrecursive FIR filters. The first way uses continuous-time Fourier algebra, and the second way uses the discrete Fourier transform. (By “analyze an FIR filter” we mean determining the FIR filter’s frequency response based on known filter coefficients.) Let’s quickly review the two FIR filter analysis methods.

5.10.1 Algebraic Analysis of FIR Filters

The algebraic method used to analyze nonrecursive FIR filters uses the discrete-time Fourier transform (DTFT) equation. Linear system theory tells us that the frequency response of a linear system (our filter) is the Fourier transform of that system’s impulse response. Because a tapped-delay line FIR filter’s impulse response is equal to its coefficient values, we proceed by expressing the Fourier transform of the filter’s coefficients. In [Section 3.14](#) we learned that we can describe the continuous Fourier transform of a discrete sequence using the DTFT expressed as

(5-33)

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-jn\omega}.$$

Modifying [Eq. \(5-33\)](#)’s notation to correspond to the DTFT of an FIR filter having N coefficients (impulse response) represented by $h(k)$, where index $k = 0, 1, 2, \dots, N-1$, we can express the filter’s complex frequency response as

(5-34)

$$\begin{aligned} H(\omega) &= \sum_{k=0}^{N-1} h(k)e^{-jk\omega} \\ &= h(0)e^{-j0\omega} + h(1)e^{-j\omega} + h(2)e^{-j2\omega} + \dots + h(N-1)e^{-j(N-1)\omega}. \end{aligned}$$

$H(\omega)$ is an $(N-1)$ th-order polynomial, and this is why, for example, a 6-tap FIR filter is often called a *5th-order FIR filter*. In [Eq. \(5-34\)](#) the digital frequency variable ω is continuous and ranges from 0 to 2π radians/sample, corresponding to a continuous-time frequency range of 0 to f_s Hz.

Let’s see how [Eq. \(5-34\)](#) is used to determine the frequency response of an FIR filter. Assume we have a 4-tap FIR filter whose coefficients are $h(k) = [0.2, 0.4, 0.4, 0.2]$. In this case our continuous $H(\omega)$ equation becomes

(5-35)

$$\begin{aligned} H(\omega) &= \sum_{k=0}^3 h(k)e^{-jk\omega} = h(0)e^{-j0\omega} + h(1)e^{-j\omega} + h(2)e^{-j2\omega} + h(3)e^{-j3\omega} \\ &= 0.2 + 0.4e^{-j\omega} + 0.4e^{-j2\omega} + 0.2e^{-j3\omega}. \end{aligned}$$

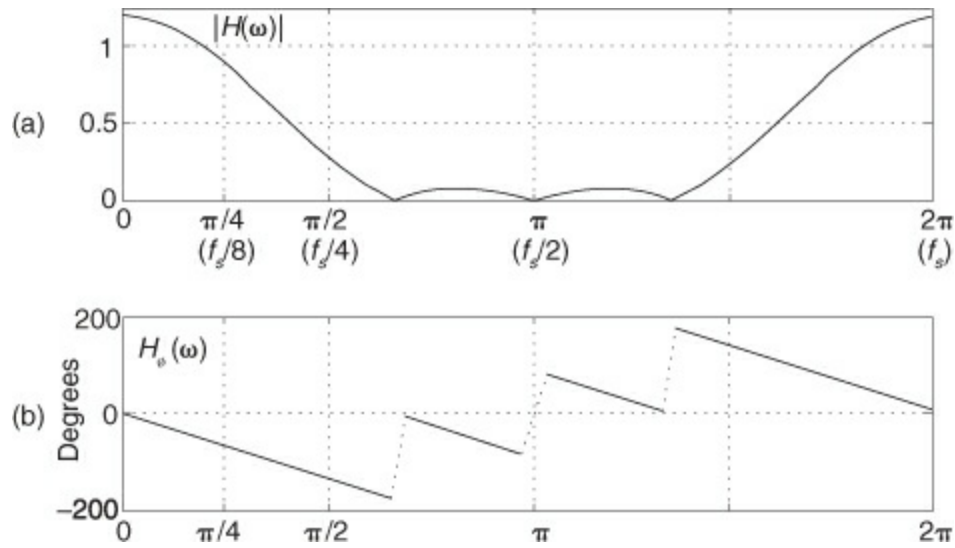
We can, if we wish, write the complex 3rd-order [Eq. \(5-35\)](#) in rectangular form as

(5-36)

$$\begin{aligned} H(\omega) &= 0.2 + 0.4\cos(\omega) + 0.4\cos(2\omega) + 0.2\cos(3\omega) \\ &\quad -j[0.4\sin(\omega) + 0.4\sin(2\omega) + 0.2\sin(3\omega)]. \end{aligned}$$

Evaluating [Eq. \(5-35\)](#), or [Eq. \(5-36\)](#), and plotting the magnitude of the continuous complex $H(\omega)$ function results in the curve in [Figure 5-46\(a\)](#). To compute the continuous $H_\theta(\omega)$ phase function, we merely take the arctangent of the ratio of the imaginary part over the real part of $H(\omega)$, yielding the $H_\theta(\omega)$ phase response in [Figure 5-46\(b\)](#).

Figure 5-46 FIR filter frequency response: (a) magnitude; (b) phase.



In practice, evaluating [Eq. \(5-34\)](#) would be performed using some sort of commercial math software, where code must be written to compute a sampled version of the continuous $H(\omega)$. Rather than writing the code to implement [Eq. \(5-34\)](#), fortunately we can conveniently compute an FIR filter's $H(\omega)$ frequency response using software that performs the discrete Fourier transform. That's the subject we discuss next.

5.10.2 DFT Analysis of FIR Filters

The most convenient way to determine an FIR filter's frequency response is to perform the discrete Fourier transform (DFT) of the filter's coefficients. This analysis method is popular because the DFT is built into most commercial signal processing software packages such as MathCAD, LabView, MATLAB, etc. (In fact, in a pinch, we can even compute DFTs with Microsoft Excel.) The DFT of an FIR filter's coefficients is computed using

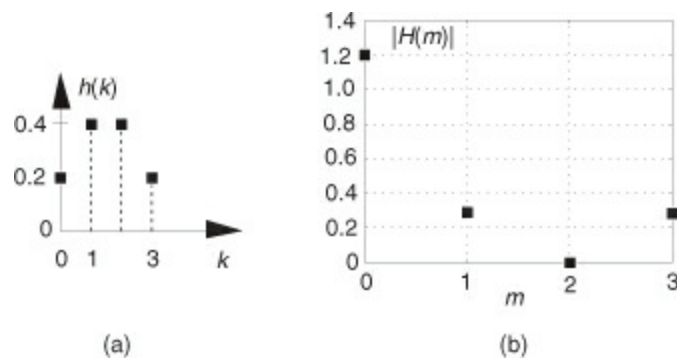
(5-37)

$$H(m) = \sum_{k=0}^{N-1} h(k) e^{-j2\pi mk/N},$$

which we normally implement with the high-speed fast Fourier transform (FFT) algorithm. Variables m and n both range from 0 to $N-1$.

Ah, but there's trouble in paradise because [Eq. \(5-37\)](#) poses a problem. If we perform a 4-point DFT of the above 4-tap FIR filter coefficients, $h(k) = [0.2, 0.4, 0.4, 0.2]$ as shown in [Figure 5-47\(a\)](#), we obtain the $|H(m)|$ samples in [Figure 5-47\(b\)](#). That $|H(m)|$ sequence reveals very little about the frequency response of the 4-tap FIR filter. We need more $|H(m)|$ frequency-domain information. That is, we need improved frequency *resolution*.

Figure 5-47 Four-tap FIR filter: (a) impulse response; (b) 4-point DFT frequency magnitude response.

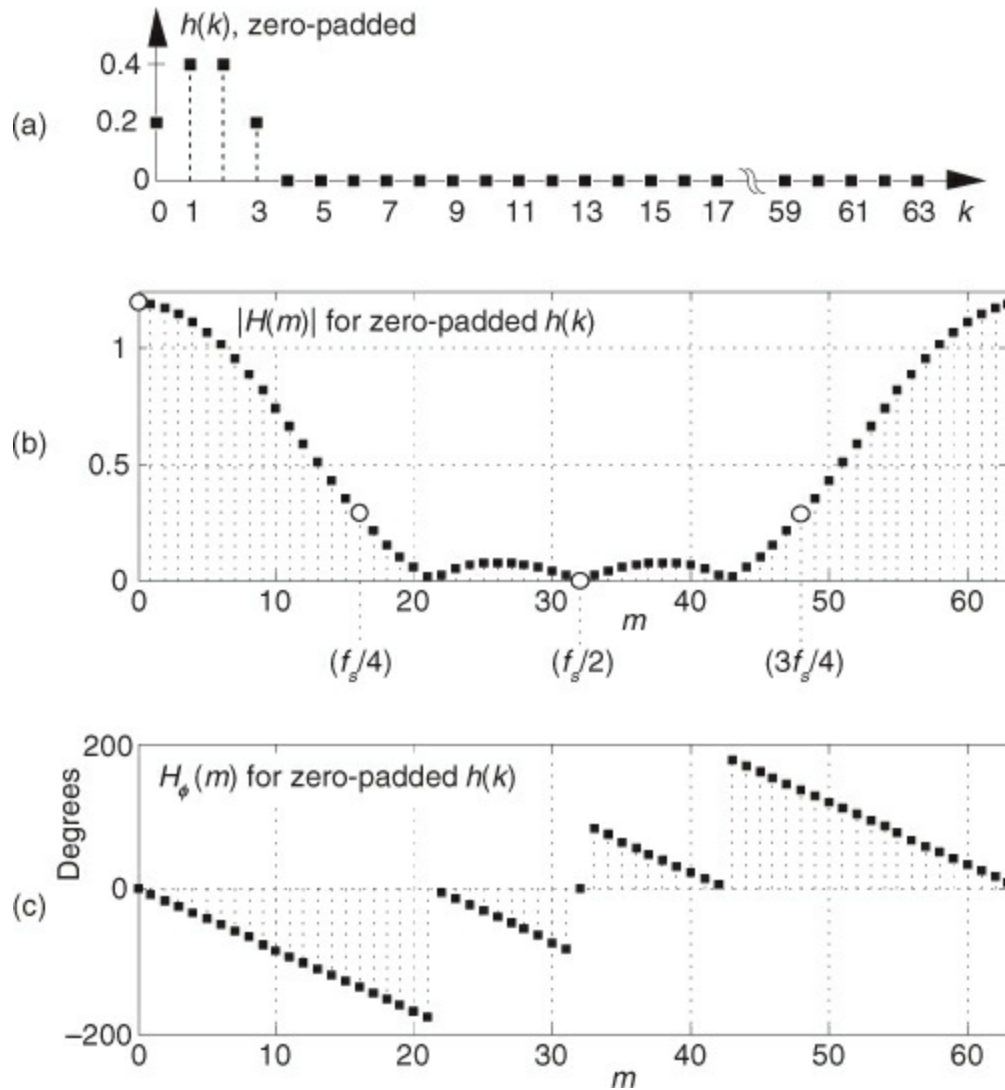


Fortunately we can obtain a finer-granularity version of $H(m)$ by *zero padding* the $h(k)$ coefficients with zero-valued samples and performing a larger-sized DFT. [Figure 5-48\(a\)](#) shows the 4-tap FIR filter's coefficients padded with 60 zero-valued samples. Performing a 64-point DFT on that padded $h(k)$ sequence yields the higher-resolution discrete $|H(m)|$ magnitude response sequence shown in [Figure 5-48\(b\)](#). Sequence $|H(m)|$ is, of course, computed using

$$(5-38)$$

$$|H(m)| = \sqrt{H_{\text{real}}(m)^2 + H_{\text{imag}}(m)^2}$$

Figure 5-48 High-resolution FIR filter frequency response: (a) zero-padded $h(k)$; (b) discrete magnitude response; (c) phase response.



where $H_{\text{real}}(m)$ and $H_{\text{imag}}(m)$ are the real and imaginary parts computed using [Eq. \(5-37\)](#). The circular white dots in [Figure 5-48\(b\)](#) correspond to the square dots in [Figure 5-47\(b\)](#).

Remember, now, a filter's complex $H(m)$ frequency response sequence is

$$(5-38')$$

$$H(m) = |H(m)| e^{jH_\phi(m)},$$

comprising a real-valued $|H(m)|$ magnitude response times a complex $e^{jH_\phi(m)}$ phase response. The real-valued phase-angle samples, shown in [Figure 5-48\(c\)](#), are computed using

$$(5-39)$$

$$H_\phi(m) = \tan^{-1} \left(\frac{H_{\text{imag}}(m)}{H_{\text{real}}(m)} \right).$$

So, our FIR filter analysis *rule of thumb* is to append a sequence of zero-valued samples (whose length is, say, $10N$) to an N -tap filter's $h(k)$ impulse response. Appending those zero-valued samples is called *zero padding* the $h(k)$ sequence. Next we compute the DFT of that padded sequence. Of course the final zero-padded sequence should have a length that is an integer power of two so that we can use the FFT to compute the high-resolution $H(m)$.

By the way, it doesn't matter if the zero-valued samples are placed before or after the original $h(k)$ coefficients prior to performing the DFT. The computed high-resolution $|H(m)|$ magnitude sequence will be the same in either case, and the resulting $H_\phi(m)$ phase samples in the two cases will differ only by a constant phase angle. (The DFT shifting theorem discussed in [Section 3.6](#) explains why this is true.)

5.10.3 FIR Filter Group Delay Revisited

We mentioned in [Section 5.8](#) how a constant time delay, what we formally refer to as *group delay*, through a filter was crucial in many applications. A constant group delay means a filter has a linear phase response over its passband and will induce no phase distortion in its output signals. Here we explore the concept of group delay a bit further.

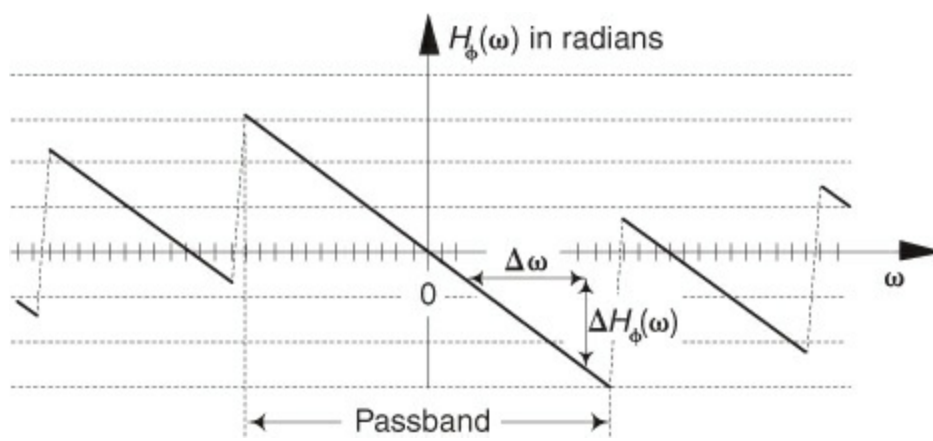
The group delay, as a function of frequency, of a filter having a frequency response of $H(\omega) = |H(\omega)|e^{jH_\phi(\omega)}$ is the negative of the derivative of the filter's $H_\phi(\omega)$ phase response with respect to frequency ω and is expressed as

$$(5-40)$$

$$G(\omega) = \frac{-dH_\phi(\omega)}{d(\omega)} \text{ samples}$$

where digital frequency ω is continuous and ranges from $-\pi$ to π radians/sample, corresponding to a continuous-time frequency range of $-f_s/2$ to $f_s/2$ Hz. Because the dimensions of $H_\phi(\omega)$ are radians, and the dimensions of ω are radians/sample, the dimensions of group delay $G(\omega)$ are time measured in samples. We graphically depict the notion of the group delay, for a lowpass filter, in [Figure 5-49](#).

Figure 5-49 FIR filter group delay derived from a filter's phase response.



For example, the complex-valued frequency response of a K -tap moving average filter is

(5-41)

$$H_{\text{ma}}(\omega) = e^{-j\omega(K-1)/2} \frac{\sin(\omega K / 2)}{\sin(\omega / 2)},$$

where the subscript “ma” means moving average. As such, from [Eq. \(5-41\)](#) the phase response of a $K = 5$ -tap moving average filter is

(5-42)

$$H_{\phi, K=5}(\omega) = -\omega(5-1)/2 = -2\omega.$$

Using [Eq. \(5-40\)](#), the group delay of a $K = 5$ -tap moving average filter is

(5-43)

$$G_{\text{ma}, K=5}(\omega) = \frac{-dH_{\phi, K=5}(\omega)}{d(\omega)} = \frac{-d(-2\omega)}{d(\omega)} = 2 \text{ samples.}$$

Luckily for us, [Eq. \(5-40\)](#) becomes very simple to evaluate if an N -tap FIR filter’s $h(k)$ coefficients (impulse response samples) are symmetrical. By “symmetrical” we mean $h(k)$ coefficients that abide by

(5-44)

$$h(k) = h(N-k-1)$$

where $0 \leq k \leq (N-1)/2$ when N is odd, and $0 \leq k \leq (N/2)-1$ when N is even. [Equation \(5-44\)](#) merely means that the first coefficient equals the last coefficient, the second coefficient equals the next to the last coefficient, and so on. All of the FIR filters we’ve discussed, so far, fall into this category.

OK, here’s the point we’re making. For symmetrical-coefficient FIR filters that comply with [Eq. \(5-44\)](#), their group delay is simple to compute. The group delay of such filters, measured in samples, is a constant equal to half the number of delay elements in the filter’s tapped-delay line structure. That is,

(5-45)

$$G_{\text{samples}} = \frac{D}{2} \text{ samples}$$

where D is the number of unit-delay elements in the filter’s delay line. Measured in seconds, a symmetrical FIR filter’s group delay is

(5-46)

$$G_{\text{seconds}} = \frac{Dt_s}{2} = \frac{D}{2f_s} \text{ seconds}$$

where t_s is the reciprocal of the filter's f_s input signal sample rate.

We can now make the following all-encompassing statement: The group delay of a tapped-delay line FIR digital filter, whose impulse response is symmetric, is equal to

(5-47)

$$G_{\text{samples}} = \frac{\text{impulse response length} - 1}{2} \text{ samples.}$$

For our purposes, we view a linear-phase FIR filter's group delay as simply the time delay through the filter. That is, if the group delay of a filter is G samples, then the filter's output sequence is delayed by G samples relative the filter's input sequence.

It's worth mentioning at this point that although we have not yet discussed such filter networks, if a tapped-delay line (FIR) network has an antisymmetrical impulse response defined by

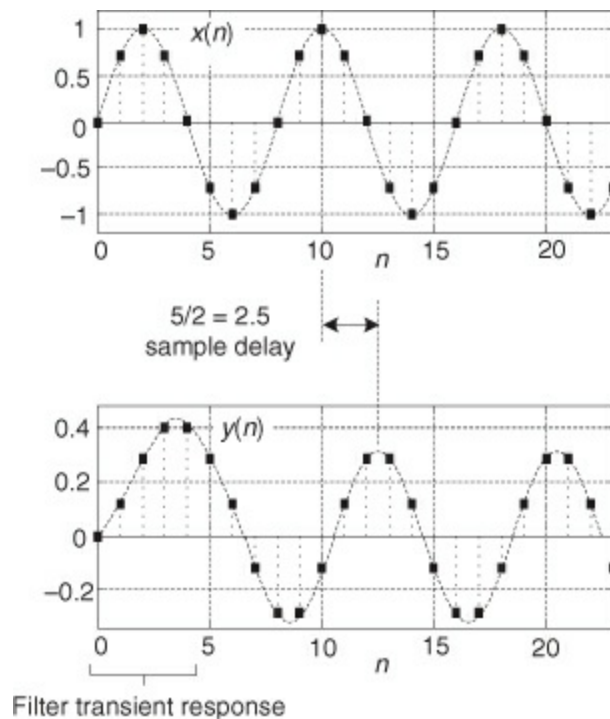
(5-48)

$$h(k) = -h(N-k-1)$$

where $0 \leq k \leq (N-1)/2$ when N is odd and $0 \leq k \leq (N/2)-1$ when N is even, such a network also has a linear phase response and its group delay is also described by [Eq. \(5-47\)](#). Digital differentiators and Hilbert transformers, discussed in later chapters, fall into this category.

At this point, looking at [Eq. \(5-45\)](#), the DSP novice may wonder, "If D is an odd number, how is it possible to have a discrete signal sequence delayed by a noninteger number of samples?" The answer to this sensible question is illustrated in [Figure 5-50](#), where $x(n)$ is a sinusoidal sequence applied to a symmetrical FIR filter having 6 taps ($D = 5$ delay elements in the tapped-delay line). There we see that the sinusoidal sequence is preserved at the filter's $y(n)$ output and delayed relative to input $x(n)$ by a group delay value of exactly $5/2$ samples. In the lingo of digital filters, the behavior in [Figure 5-50](#) is called *fractional delay*.

Figure 5-50 Group delay of a 6-tap (5 delay elements) FIR filter.



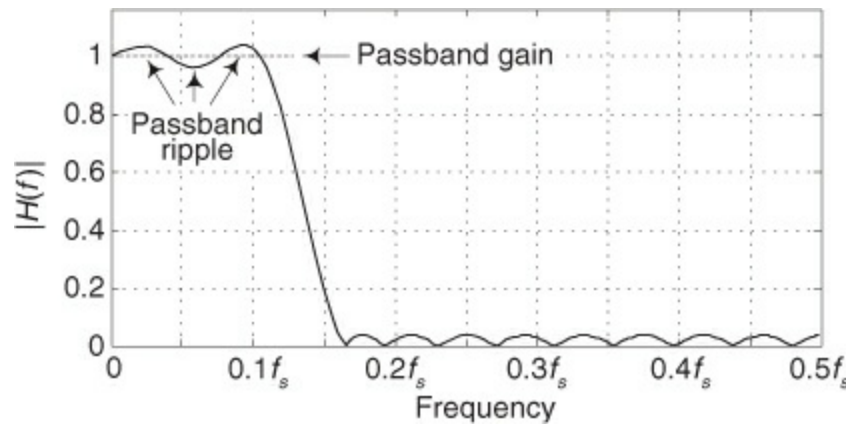
Again, constant group delay—linear phase—is a desirable filter property because the spectral components in the filter's output signal will suffer no phase distortion. Stated in different words: all

spectral components within a linear-phase filter's passband will be delayed by the same amount of time as they pass through the filter. If a linear-phase filter's input is a complicated digital communications signal, rich in spectral-phase complexity representing digital data, the spectral-phase relationships and the digital data are preserved undistorted at the filter's output. Their linear-phase property is the reason we use FIR filters!

5.10.4 FIR Filter Passband Gain

One FIR filter property that is of interest is the filter's *passband gain*. The standard definition of passband gain is that it is the filter's passband magnitude response level around which the passband ripple fluctuates, as shown by the lowpass filter in [Figure 5-51](#) where the passband gain equals unity. In practice we design filters to have very small passband ripple, so a lowpass filter's passband gain is roughly equal to its DC gain (gain at zero Hz), which is the sum of the filter's impulse response sequence, i.e., the sum of the FIR filter's coefficients. (We leave the proof of this as a homework problem.) Most commercial FIR filter design software packages compute filter coefficients such that their passband gain is unity.

Figure 5-51 FIR filter passband gain definition.



5.10.5 Estimating the Number of FIR Filter Taps

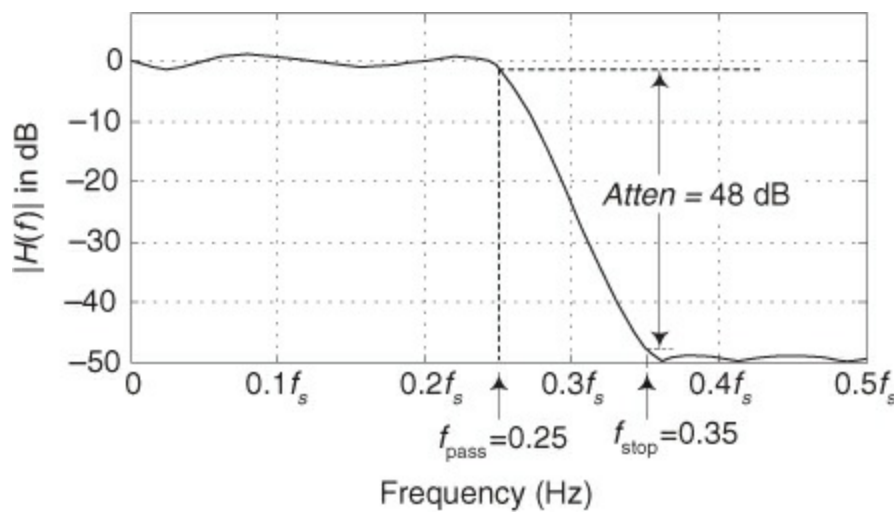
Our final topic regarding the analysis of FIR filters is: How do we estimate the number of filter taps (coefficients), N , that can satisfy a given frequency magnitude response of an FIR filter? Several authors have proposed empirical relationships for estimating N for traditional tapped-delay line lowpass FIR filters based on the desired passband ripple, stopband attenuation, and transition region width[[24,31–33](#)]. A particularly simple expression proposed by Prof. Fred Harris for N , giving results consistent with other estimates for passband ripple values near 0.1 dB, is

(5-49)

$$N_{\text{FIR}} \approx \frac{\text{Atten}}{22(f_{\text{stop}} - f_{\text{pass}})}$$

where *Atten* is the filter's desired stopband attenuation measured in dB, and f_{pass} and f_{stop} are frequencies normalized to the f_s sample rate in Hz as illustrated in [Figure 5-52](#). For example, $f_{\text{pass}} = 0.2$ means that the continuous-time frequency of f_{pass} is $0.2f_s$ Hz.

Figure 5-52 Example FIR filter frequency definitions.



As an example, let's obtain a rough estimate of the number of lowpass FIR filter taps (coefficients) needed to achieve the magnitude response shown in [Figure 5-52](#). Assuming $f_s = 1000$ Hz, we want the end of a lowpass filter's passband to be at 250 Hz, the beginning of the stopband is 350 Hz, and we need a stopband attenuation of 48 dB. Applying those values to [Eq. \(5-49\)](#), we have

(5-50)

$$N_{\text{FIR}} \approx \frac{48}{22(350 / 1000 - 250 / 1000)} = \frac{48}{22(0.35 - 0.25)} = 21.8.$$

Taking the integer closest to 21.8, i.e., 22, we then state that the lowpass filter in [Figure 5-52](#) can be built using a 22-tap FIR filter. We'll use [Eq. \(5-49\)](#) many times in later chapters of this book.

References

- [1] Shynk, J. J. "Adaptive IIR Filtering," *IEEE ASSP Magazine*, April 1989.
- [2] Laundrie, A. "Adaptive Filters Enable Systems to Track Variations," *Microwaves & RF*, September 1989.
- [3] Bullock, S. R. "High Frequency Adaptive Filter," *Microwave Journal*, September 1990.
- [4] Haykin, S. S. *Adaptive Filter Theory*, Prentice Hall, Englewood Cliffs, New Jersey, 1986.
- [5] Goodwin, G. C., and Sin, K. S. *Adaptive Filtering Prediction and Control*, Prentice Hall, Englewood Cliffs, New Jersey, 1984.
- [6] Gibbs, J. W. *Nature*, Vol. 59, 1899, p. 606.
- [7] Stockham, T. G. "High-Speed Convolution and Correlation with Applications to Digital Filtering," [Chapter 7](#) in *Digital Processing of Signals*, ed. by B. Gold et al., McGraw-Hill, New York, 1969, pp. 203–232.
- [8] Wait, J. V. "Digital Filters," in *Active Filters: Lumped, Distributed, Integrated, Digital, and Parametric*, ed. by L. P. Huelsman, McGraw-Hill, New York, 1970, pp. 200–277.
- [9] Dolph, C. L. "A Current Distribution for Broadside Arrays Which Optimizes the Relationship Between Beam Width and Side-Lobe Level," *Proceedings of the IRE*, Vol. 35, June 1946.
- [10] Barbieri, D. "A Method for Calculating the Current Distribution of Chebyshev Arrays," *Proceedings of the IRE*, Vol. 40, January 1952.
- [11] Cook, C. E., and Bernfeld, M. *Radar Signals*, Academic Press, New York, 1967, pp. 178–180.
- [12] Kaiser, J. F. "Digital Filters," in *System Analysis by Digital Computer*, ed. by F. F. Kuo and J.

F. Kaiser, John Wiley and Sons, New York, 1966, pp. 218–277.

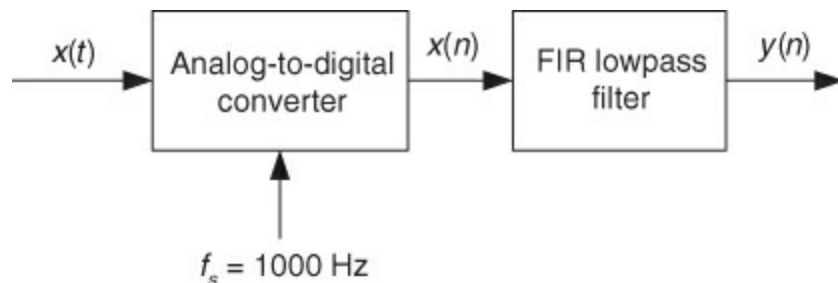
- [13] Williams, C. S. *Designing Digital Filters*, Prentice Hall, Englewood Cliffs, New Jersey, 1986, p. 117.
- [14] Harris, F. J. “On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform,” *Proceedings of the IEEE*, Vol. 66, No. 1, January 1978.
- [15] McClellan, J. H., Parks, T. W., and Rabiner, L. R. “A Computer Program for Designing Optimum FIR Linear Phase Digital Filters,” *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-21, No. 6, December 1973, p. 515.
- [16] Rabiner, L. R., and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1975, p. 136.
- [17] Parks, T. W., and McClellan, J. H. “Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase,” *IEEE Trans. on Circuit Theory*, Vol. CT-19, March 1972.
- [18] McClellan, J. H., and Parks, T. W. “A Unified Approach to the Design of Optimum FIR Linear Phase Digital Filters,” *IEEE Trans. on Circuit Theory*, Vol. CT-20, November 1973.
- [19] Rabiner, L. R., McClellan, J. H., and Parks, T. W. “FIR Digital Filter Design Techniques Using Weighted Chebyshev Approximation,” *Proceedings of the IEEE*, Vol. 63, No. 4, April 1975.
- [20] Oppenheim, A. V., and Schaffer, R. W. *Discrete Time Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1989, p. 478.
- [21] Funderburk, D. M., and Park, S. “Implementation of a C-QUAM AM-Stereo Receiver Using a General Purpose DSP Device,” *RF Design*, June 1993.
- [22] Harris Semiconductor Inc. “A Digital, 16-Bit, 52 Msps Halfband Filter,” *Microwave Journal*, September 1993.
- [23] Ballanger, M. G. “Computation Rate and Storage Estimation in Multirate Digital Filtering with Half-Band Filters,” *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-25, No. 4, August 1977.
- [24] Crochiere, R. E., and Rabiner, L. R. “Decimation and Interpolation of Digital Signals—A Tutorial Review,” *Proceedings of the IEEE*, Vol. 69, No. 3, March 1981, p. 318.
- [25] Ballanger, M. G., Daguet, J. L., and Lepagnol, G. P. “Interpolation, Extrapolation, and Reduction of Computational Speed in Digital Filters,” *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-22, No. 4, August 1974.
- [26] Oppenheim, A. V., Willsky, A. S., and Young, I. T. *Signals and Systems*, Prentice Hall, Englewood Cliffs, New Jersey, 1983, p. 212.
- [27] Stearns, S. *Digital Signal Analysis*, Hayden Book Co., Rochelle Park, New Jersey, 1975, p. 93.
- [28] Oppenheim, A. V., and Schaffer, R. W. *Discrete Time Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1989, p. 58.
- [29] Rabiner, L. R., and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1975, p. 59.
- [30] Oppenheim, A. V., Willsky, A. S., and Young, I. T. *Signals and Systems*, Prentice Hall, Englewood Cliffs, New Jersey, 1983, p. 201.

- [31] Rorabaugh, C. *DSP Primer*, McGraw-Hill, New York, 1999, pp. 278–279.
- [32] Kaiser, J. “Nonrecursive Digital Filter Design Using Io-Sinh Window Function,” *Proc. 1974 IEEE Int. Symp. Circuits Systems*, April 1974, pp. 20–23.
- [33] Harris, F. *Multirate Signal Processing for Communication Systems*, Prentice Hall, Upper Saddle River, New Jersey, 2004, pp. 56–57.

Chapter 5 Problems

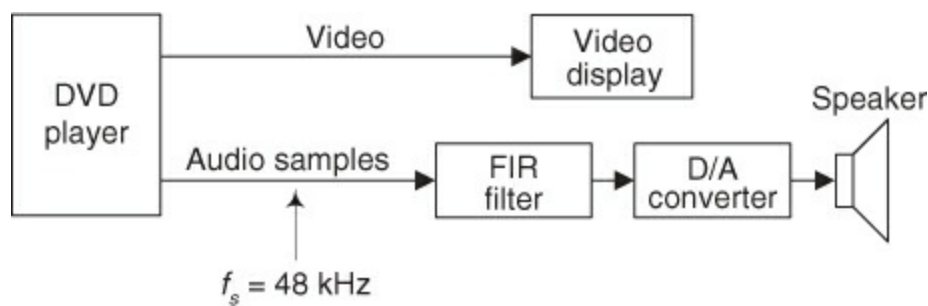
- 5.1** We first introduced the notion of [impulse response](#) in [Chapter 1](#), and here in [Chapter 5](#) we discussed the importance of knowing the impulse response of FIR filter networks. With that said, if the $y(n)$ output of a discrete system is equal to the system’s $x(n)$ input sequence:
- (a) Draw the unit impulse response of such a system.
 - (b) Draw the block diagram (structure) of that system.
 - (c) What is the frequency magnitude response of such a system? Prove your answer.
- 5.2** Consider a simple analog signal defined by $x(t) = \cos(2\pi 800t)$ shown in [Figure P5-2](#). The FIR lowpass filter has a passband extending from -400 Hz to $+400$ Hz, a passband gain of unity, a transition region width of 20 Hz, and a stopband attenuation of 60 dB.
- (a) Draw the spectral magnitude of $x(n)$ showing all spectral components in the range of $-2f_s$ to $+2f_s$.
 - (b) Draw the spectral magnitude of $y(n)$ showing all spectral components in the range of $-2f_s$ to $+2f_s$.
 - (c) What is the time-domain peak amplitude of the sinusoidal $y(n)$ output?

Figure P5-2



- 5.3** Assume we want to filter the audio signal from a digital video disc (DVD) player as shown in [Figure P5-3](#). The filtered audio signal drives, by way of a digital-to-analog (D/A) converter, a speaker. For the audio signal to have acceptable time synchronization with the video signal, video engineers have determined that the time delay of the filter must be no greater than 6×10^{-3} seconds. If the f_s sample rate of the audio is 48 kHz, what is the maximum number of taps in the FIR filter that will satisfy the time delay restriction? (Assume a linear-phase FIR filter, and zero time delay through the D/A converter.)

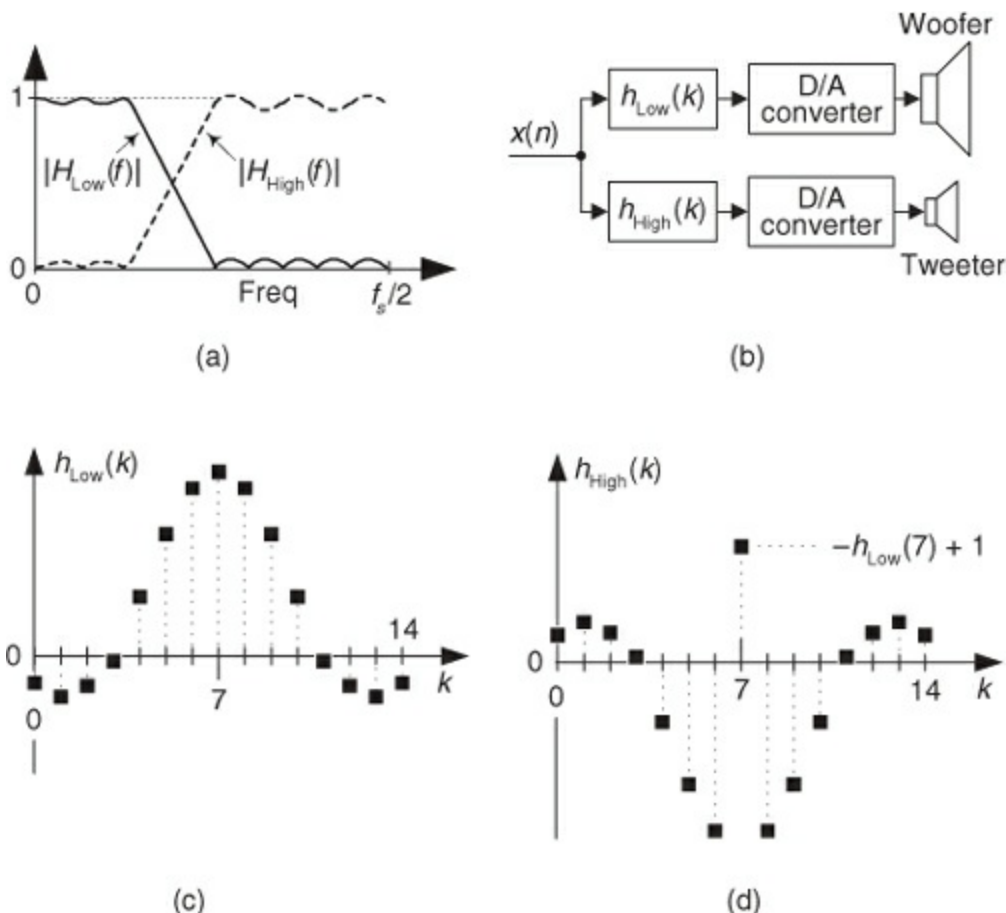
Figure P5-3



5.4 There are times when we want to build a lowpass filter and a highpass filter that are *complementary*. By “complementary” we mean that a highpass filter’s passband covers the frequency range defined by a lowpass filter’s stopband range. This idea is illustrated in [Figure P5-4\(a\)](#). An example of such filters is an audio system, shown in [Figure P5-4\(b\)](#), where the low-frequency spectral components of an $x(n)$ audio signal drive, by way of a digital-to-analog (D/A) converter, a low-frequency speaker (woofer). Likewise, the high-frequency spectral components of $x(n)$ drive a high-frequency speaker (tweeter). Audio enthusiasts call [Figure P5-4\(b\)](#) a “crossover” network. Assuming that the lowpass filter is implemented with a 15-tap FIR filter whose $h_{\text{Low}}(k)$ coefficients are those in [Figure P5-4\(c\)](#), the complementary highpass filter will have the coefficients shown in [Figure P5-4\(d\)](#). Highpass coefficients $h_{\text{High}}(k)$ are defined by

$$h_{\text{High}}(k) = \begin{cases} -h_{\text{Low}}(k), & \text{when } k \neq 7 \\ -h_{\text{Low}}(k) + 1, & \text{when } k = 7. \end{cases}$$

Figure P5-4

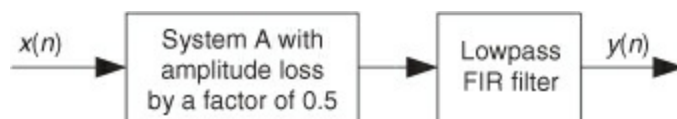


Here is the problem: Draw a block diagram of a system that performs the process in [P5-4\(b\)](#) where only the $h_{\text{Low}}(k)$ lowpass FIR filter need be implemented.

5.5 Think about a discrete System A, shown in [Figure P5-5](#), that has an undesirable amplitude (gain)

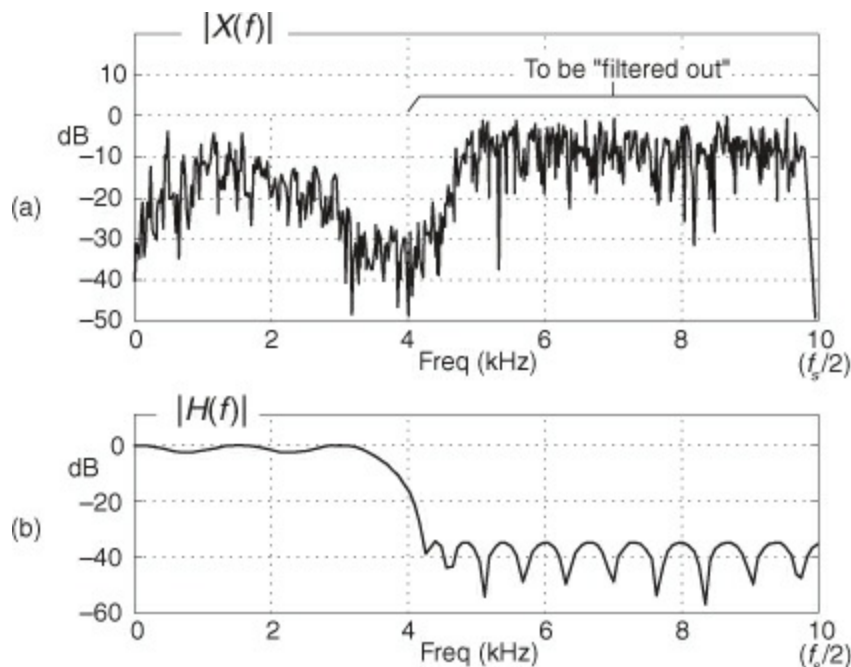
loss by a factor 0.5 (−6 dB), whose output requires lowpass linear-phase filtering. What can we do in the design of the lowpass FIR filter so the filter has an amplitude gain of 2 to compensate for System A’s amplitude loss?

Figure P5-5



5.6 Let’s assume we have an $x(n)$ time sequence, whose f_s sample rate is 20 kHz, and its $|X(f)|$ spectral magnitude is that shown in [Figure P5-6\(a\)](#). We are required to design a linear-phase lowpass FIR filter that will attenuate the undesired high-frequency noise indicated in [Figure P5-6\(a\)](#). So we design a lowpass FIR filter whose frequency magnitude response is the $|H(f)|$ shown in [Figure P5-6\(b\)](#) and assume our filter design exercise is complete. Sometime later, unfortunately, we learn that the original $x(n)$ sequence’s sample rate was *not* 20 kHz, but is in fact 40 kHz.

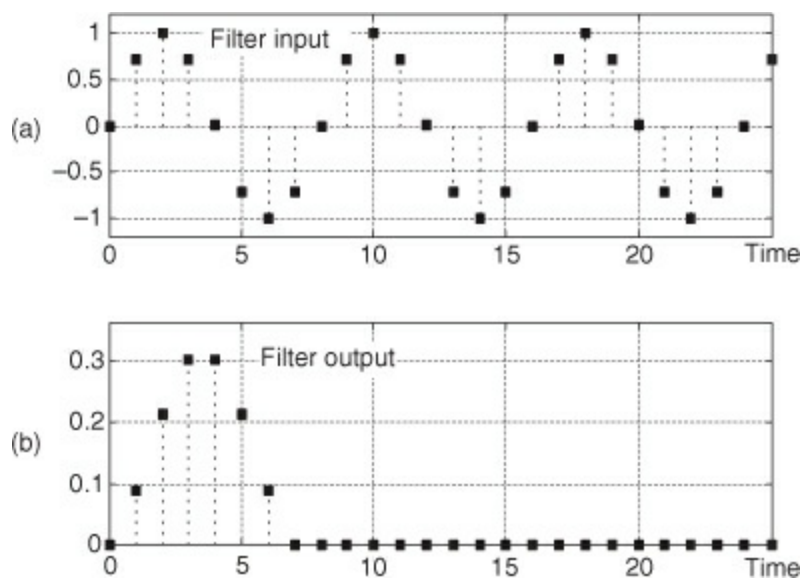
Figure P5-6



Here is the problem: What must we do to our lowpass filter’s $h(k)$ coefficients, originally designed based on a 20 kHz sample rate, so that they will still attenuate $x(n)$ ’s undesired high-frequency noise when the f_s sample rate is actually 40 kHz?

5.7 Here is an interesting little problem. Think about applying the sinusoidal input sequence shown in [Figure P5-7\(a\)](#) to an 8-point moving average FIR filter. The filter’s output sequence is that depicted in [Figure P5-7\(b\)](#).

Figure P5-7



(a) What characteristic of the filter's frequency response causes the filter's output sequence to go to all zeros as shown in [Figure P5-7\(b\)](#)?

(b) In [Figure P5-7\(b\)](#), what do we call those initial nonzero-valued filter output samples?

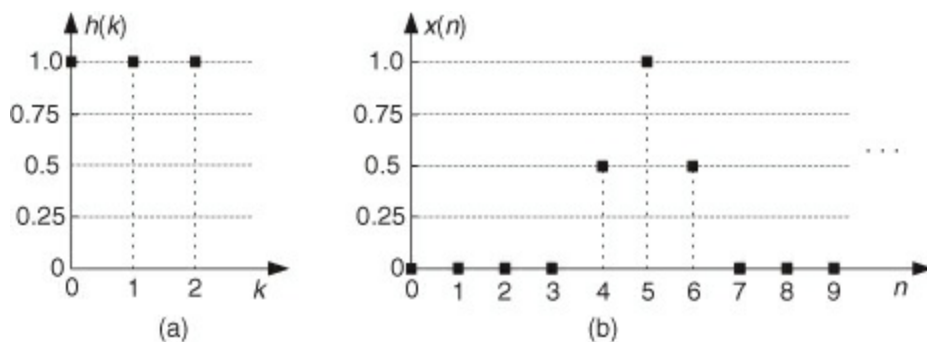
5.8 Are abrupt (sudden) changes in the amplitude of a continuous, or discrete, signal associated with low or high frequencies?

5.9 Consider an FIR filter whose impulse response is shown in [Figure P5-9\(a\)](#). Given the $x(n)$ filter input sequence shown in [Figure P5-9\(b\)](#):

(a) What is the length, measured in samples, of the nonzero-valued samples of the filter's output sequence?

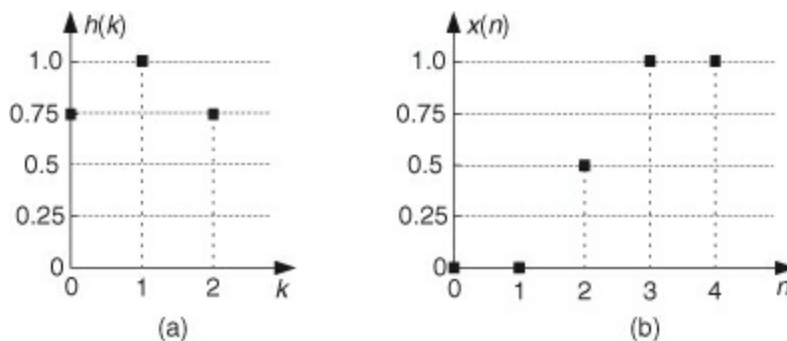
(b) What is the maximum sample value of the filter's output sequence?

Figure P5-9



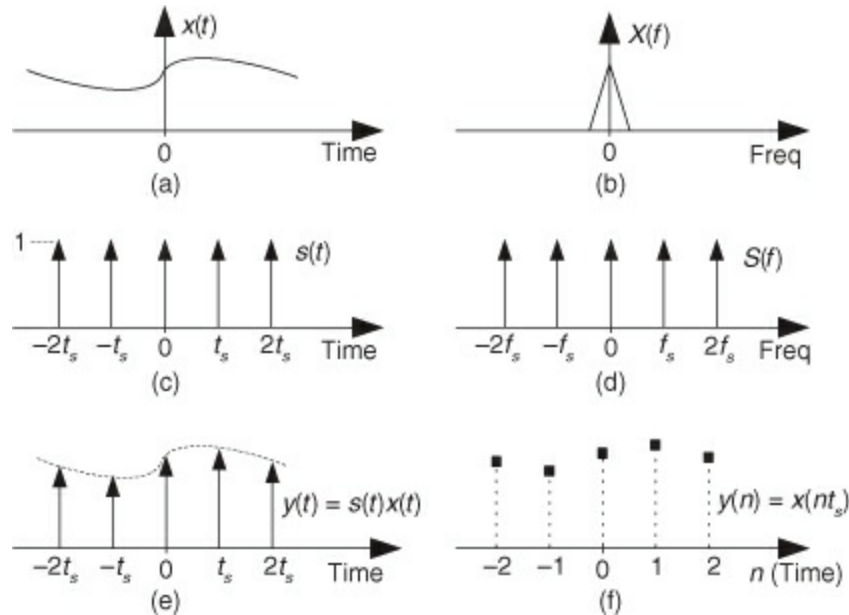
5.10 Consider an FIR filter whose impulse response is that shown in [Figure P5-10\(a\)](#). Given the $x(n)$ filter input sequence shown in [Figure P5-10\(b\)](#), draw the filter's output sequence.

Figure P5-10



5.11 Regarding the material in this chapter, it's educational to revisit the idea of periodic sampling that was presented in [Chapter 2](#). Think about a continuous $x(t)$ signal in [Figure P5-11\(a\)](#) whose spectrum is depicted in [Figure P5-11\(b\)](#). Also, consider the continuous periodic infinitely narrow impulses, $s(t)$, shown in [Figure P5-11\(c\)](#). Reference [28] provides the algebraic acrobatics to show that the spectrum of $s(t)$ is the continuous infinitely narrow impulses, $S(f)$, shown in [Figure P5-11\(d\)](#). If we multiply the $x(t)$ signal by the $s(t)$ impulses, we obtain the continuous $y(t) = s(t)x(t)$ impulse signal shown by the arrows in [Figure P5-11\(e\)](#).

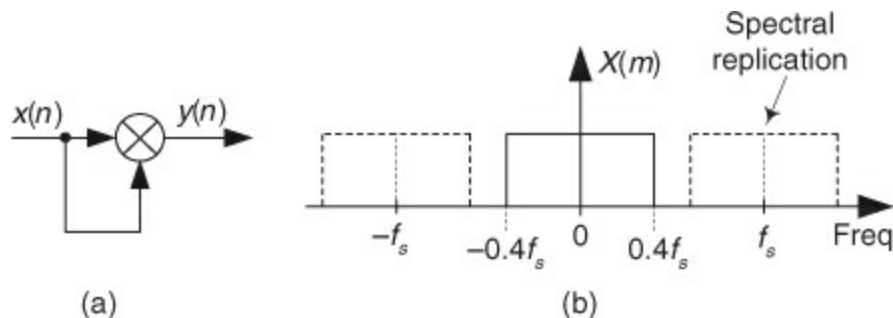
Figure P5-11



Now, if we use an analog-to-digital converter to represent those $y(t)$ impulse values as a sequence of discrete samples, we obtain the $y(n)$ sequence shown in [Figure P5-11\(f\)](#). Here is the problem: Briefly discuss what we learned in this [Chapter 5](#) that tells us the spectrum of the $y(n)$ samples comprises periodic replications of the $X(f)$ in [Figure P5-11\(b\)](#). Your brief discussion should confirm the material in [Chapter 2](#) which stated that discrete-time sequences have periodic (replicated) spectra.

5.12 Now that we're familiar with the powerful *convolution theorem*, think about the discrete system shown in [Figure P5-12\(a\)](#).

Figure P5-12



Given that $x(n)$'s spectrum is the $X(m)$ shown in [Figure P5-12\(b\)](#):

- Draw the $Y(m)$ spectrum of sequence $y(n)$. (We're not worried about the vertical axis scale here, merely the frequency axis and spectral shape of $Y(m)$.)
- Will aliasing errors occur in the $y(n) = x(n)^2$ output? (That is, will spectral replications in $Y(m)$

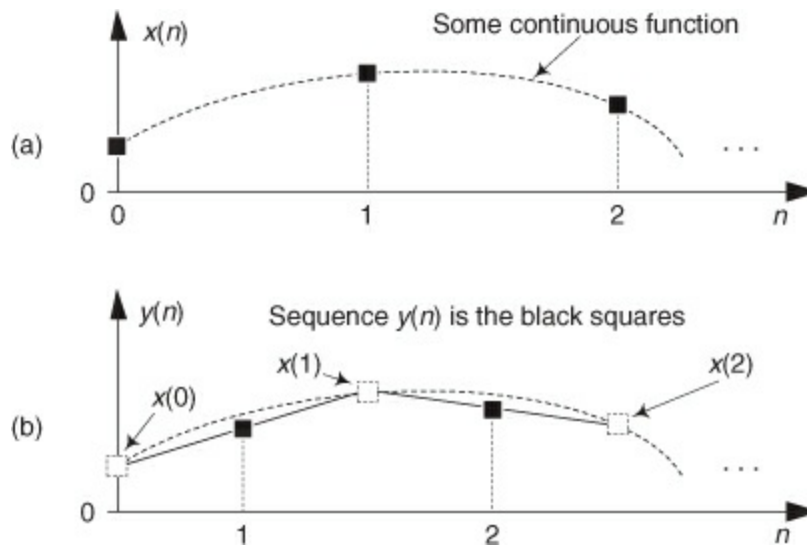
overlap each other?)

- (c) What is $x(n)$'s maximum one-sided bandwidth that will avoid aliasing errors in $y(n)$? (Stated in different words, what is the maximum one-sided bandwidth of $x(n)$ that will avoid overlapped spectral replications in $Y(m)$?)

5.13 It's likely that you have heard of the process called *linear interpolation*. It's a computationally simple (but not terribly accurate) scheme for estimating sample values of a continuous function in between some given $x(n)$ sample values of that function. For the $x(n)$ time samples in [Figure P5-13\(a\)](#), linear interpolation is the process of computing the intermediate $y(n)$ samples shown as the black squares in [Figure P5-13\(b\)](#). That is, the interpolated sample $y(1)$ is the value lying on the center of the straight line connecting $x(0)$ and $x(1)$, the interpolated sample $y(2)$ is the value lying on the center of the straight line connecting $x(1)$ and $x(2)$, and so on. Given this process of linear interpolation:

- (a) What is the equation defining $y(n)$ in terms of the $x(n)$ samples?
- (b) The implementation of linear interpolation is often called a *filter* because we build interpolators using tapped-delay line structures, just like standard FIR filter structures. Draw the block diagram of a linear interpolation *filter* that computes $y(n)$ from the input $x(n)$ sequence.

Figure P5-13



5.14 Consider a linear-phase lowpass FIR filter whose coefficients are

$$h_1(k) = [-0.8, 1.6, 25.5, 47, 25.5, 1.6, -0.8],$$

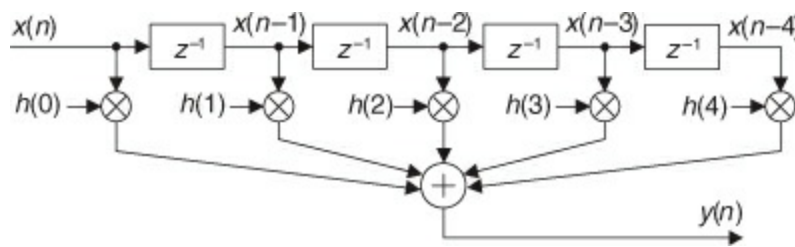
and whose DC gain, $H_1(0)$, is equal to 99.6. If we change those coefficients to

$$h_2(k) = [-0.8, 1.6, Q, 47, Q, 1.6, -0.8],$$

we obtain a new DC gain equal to 103.6. What is the value of Q ?

5.15 [Figure P5-15](#) shows a linear-phase 5-tap FIR filter.

Figure P5-15



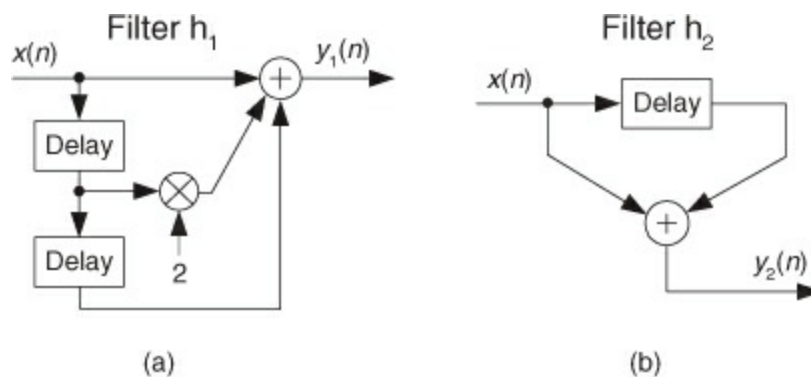
DSP engineers always seek to reduce the number of multipliers in their systems. Redesign the filter in [Figure P5-15](#) to a form that reduces the number of necessary multiplications per output sample. Draw the block diagram of your new design.

Hint: Write the difference equation for the $y(n)$ output sequence, and recall the relationships between the filter's coefficients.

- 5.16** The two linear-phase lowpass filters in [Figure P5-16](#) have very similar frequency responses, but those responses are not identical except at a single frequency. If we replaced Filter h_1 with Filter h_2 to reduce our filtering computational workload, determine the frequency, ω_0 , where the two $H_1(\omega)$ and $H_2(\omega)$ frequency responses are equal.

Hint: Begin by creating closed-form equations for $H_1(\omega)$ and $H_2(\omega)$ using the discrete-time Fourier transform (DTFT).

Figure P5-16



- 5.17** The following is a useful problem regarding the 3-tap nonrecursive FIR filter shown in [Figure P5-17\(a\)](#). The problem's solution shows us how to design computationally efficient narrowband-noise reduction filters. If $|h_1| \leq 2$, the filter will have an $|H(\omega)|$ frequency magnitude response having two nulls at $\pm\omega_n$ as shown in [Figure P5-17\(b\)](#). (Here, the frequency axis value of π radians/sample corresponds to a cyclic frequency of half the sample rate, $f_s/2$.)

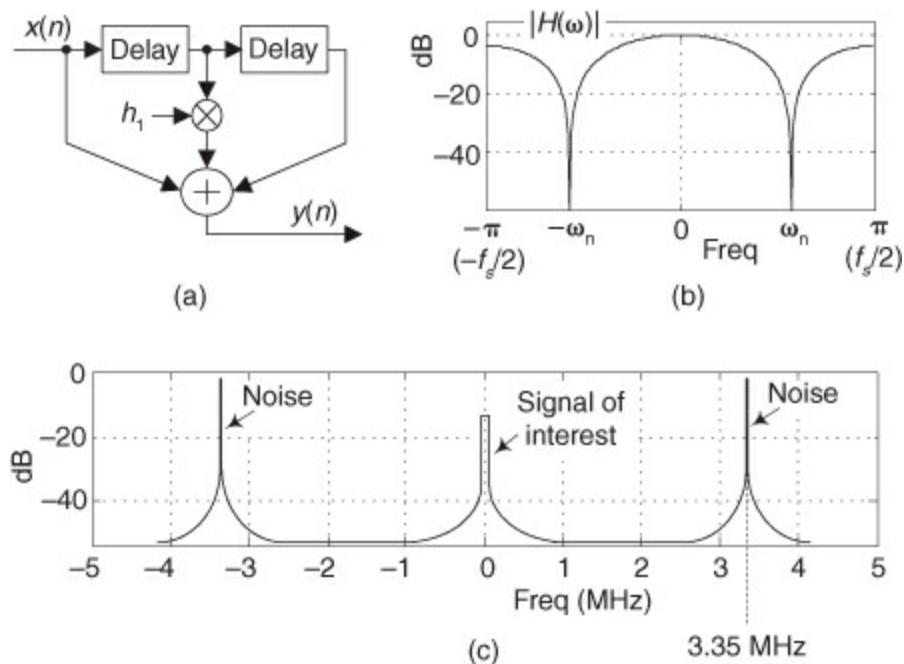
- (a)** Assume we have a low-frequency signal of interest that's contaminated with high-level narrowband noise located at ± 3.35 MHz when the sample rate is $f_s = 8.25$ MHz as shown in [Figure P5-17\(c\)](#). To attenuate that noise, for what value of h_1 will the 3-tap FIR filter's nulls be located at the noise center frequency of ± 3.35 MHz? Show your work.

Hint: Use the discrete-time Fourier transform (DTFT) of the filter's impulse response to create a closed-form equation for the filter's $H(\omega)$ frequency response in terms of the coefficient h_1 and frequency ω . Next, obtain the expression for h_1 in terms of the filter's null frequency ω_n .

- (b)** What is the DC gain (gain at zero Hz) of our 3-tap FIR filter?

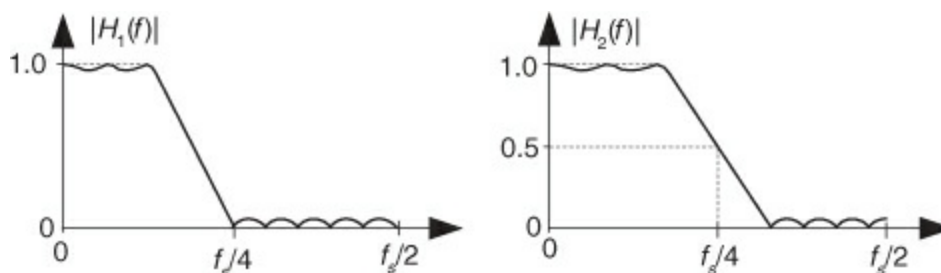
- (c)** Explain why the filter has a linear, or nonlinear, phase response.

Figure P5-17



- 5.18** What characteristic must the coefficients of an FIR filter have to ensure that its frequency-domain phase response is a linear function of frequency (i.e., linear phase)?
- 5.19** Quickfilter Technologies Inc. produces a tapped-delay line FIR filter chip (Part #QF1D512) that has an astounding $N = 512$ taps. When a new filter input sample is applied to the chip, how many addition operations must this chip perform to compute a single filter output sample?
- 5.20** Intersil Corp. produces an HSP5021 *down-converter* integrated circuit containing a symmetrical-coefficient FIR filter having 255 taps. If the down-converter chip's input signal sample rate is $f_s = 8$ MHz, what is the group delay (delay through the filter) of their 255-tap FIR filter measured in samples?
- 5.21** Assume we have digitized an analog signal at an f_s sample rate of 2×10^6 samples/second. Next we pass the samples through a 70-tap linear-phase lowpass FIR filter whose cutoff frequency (end of the passband) is 600 kHz. What would be the time delay, measured in seconds, between the lowpass filter's input and output for a sinusoidal tone whose frequency is 200 kHz?
- 5.22** Think about two linear-phase FIR filters whose frequency magnitude responses are shown in [Figure P5-22](#).

Figure P5-22

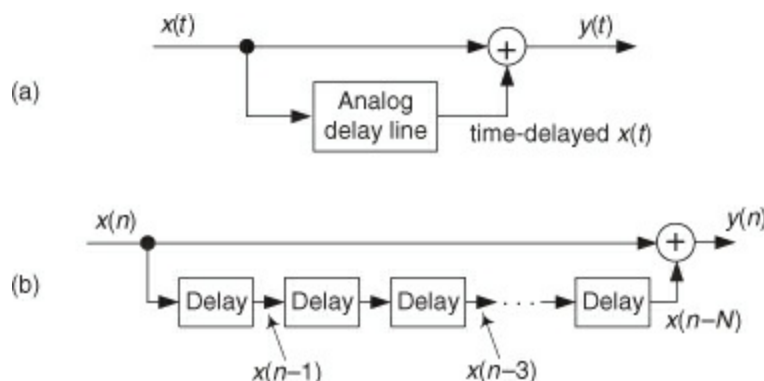


- (a)** Let's assume that filter $H_1(f)$ is a 17-tap FIR filter. What is the group delay of this linear-phase filter measured in samples?
- (b)** Next, let's assume that filter $H_2(f)$ is a 17-tap *half-band* FIR filter. $H_2(f)$, like all half-band FIR

filters, has a gain of 0.5 at the frequency $f = f_s/4$. What is the group delay of this linear-phase $H_2(f)$ filter, measured in samples?

5.23 Reverberation, a kind of echo, is a popular *audio effect* applied to guitar music. (Most commercial electric guitar amplifiers have a reverberation capability.) In the world of continuous signals reverberation is implemented with an analog delay line as shown in [Figure P5-23\(a\)](#). That analog delay line is typically a kind of *speaker* at one end of a coiled metal spring, and a kind of *microphone* at the other end of the spring. However, analog reverberation units have no convenient way to control the amount of time delay, and unfortunately their hardware is physically large.

Figure P5-23



Making use of digital signal processing on the other hand, the process of reverberation seems easy to implement using a delay line network like that shown in [Figure P5-23\(b\)](#). For the digital reverberation process to be usable, however, it must have a constant gain, where

$$\text{Gain} = \frac{y(n)}{x(n)} = \text{constant},$$

over the full operating frequency range of the system. That is, we want our reverberator to have a flat frequency magnitude response. (By “Gain” we mean the steady-state gain after the delay line is filled with input samples.)

(a) Assume we have the [Figure P5-23\(b\)](#) delay line with $N = 8$ delay elements. What is the $N = 8$ digital reverberator’s $h(n)$ time-domain impulse response?

(b) What is the equation for the digital reverberator’s $|H(\omega)|$ frequency magnitude response?

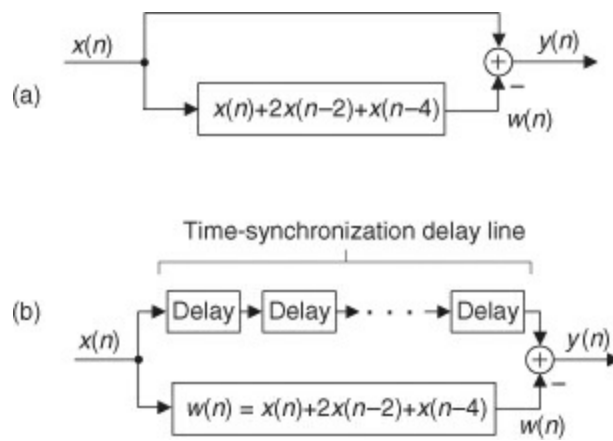
Hint: Use what you learned in [Section 3.14](#), and don’t forget your trigonometric identities.

(c) Draw a rough sketch of the $|H(\omega)|$ frequency magnitude response from Part (b). (This curve shows us how well simple digital delay-line reverberators work.)

5.24 There are digital filtering schemes that use the process conceptually shown in [Figure P5-24\(a\)](#). In that parallel-path filter the $x(n)$ input is filtered to generate sequence $w(n)$. The network’s $y(n)$ output is the $x(n)$ input sequence minus the $w(n)$ sequence. The $w(n)$ sequence is defined by

$$w(n) = x(n) + 2x(n-2) + x(n-4).$$

Figure P5-24



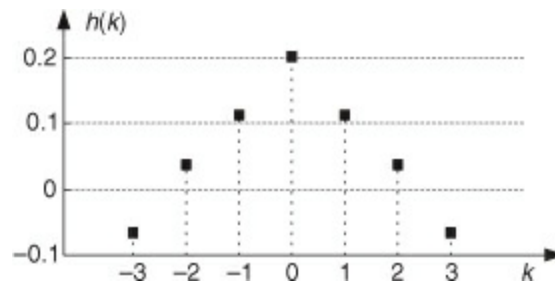
The actual implementation of such a parallel-path filter is shown in [Figure P5-24\(b\)](#) where the multi-element delay line in the upper path of [Figure P5-24\(b\)](#) is needed for time alignment to compensate for the time delay of the bottom-path FIR filter. How many unit-delay elements must be used in the upper path in [Figure P5-24\(b\)](#)?

5.25 As we stated in [Section 5.10](#), a lowpass FIR filter's frequency magnitude response at zero Hz (DC) is equal to the sum of the filter's impulse response samples (sum of the filter's coefficients). Prove this important lowpass FIR filter property.

5.26 Although we didn't state it explicitly in the text, the continuous frequency magnitude response of a symmetrical 7-tap FIR filter (for example, an FIR filter whose $h(k)$ coefficients are indexed as shown in [Figure P5-26](#)) can be computed using

$$|H(\omega)| = |h(0) + 2 \sum_{k=1}^3 h(k) \cos(\omega k)|.$$

Figure P5-26



(The normalized frequency range is $-\pi \leq \omega \leq \pi$ where ω is a continuous normalized angle with $\omega = \pi$ corresponding to a cyclic frequency of $f_s/2$ Hz.) There are two reasons we introduce the above $|H(\omega)|$ expression:

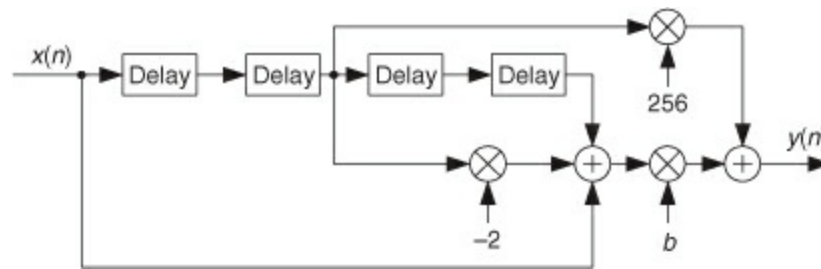
- Such $|H(\omega)|$ equations can be used to compute the magnitude responses of linear-phase FIR filters, having an odd number of taps, when no FFT software routine is available.
- You won't be surprised when you see, in the literature of DSP, FIR filter frequency magnitude response equations such as the above summation of cosine functions.

Derive the general equation for the $|H(\omega)|$ for an N -tap symmetrical FIR filter's magnitude response, when N is odd. Show your work.

Hint: Use what you learned in [Section 3.14](#), and pay careful attention to the range of the k index in [Figure P5-26](#). Notice how $h(0)$ is the center coefficient! Also, don't forget our friend Leonhard Euler.

- 5.27** Assume a commercial data acquisition device has the ability to implement a 191-tap digital FIR filter. What is the narrowest transition region width ($f_{\text{stop}} - f_{\text{pass}}$), stated in terms of f_s , we can expect to achieve for a lowpass FIR filter using this device if we desire at least 55 dB of stopband attenuation?
- 5.28** Texas Instruments Inc. produces a video processing chip (Part #TMS320DM646x) containing the FIR filter shown in [Figure P5-28](#). Coefficient b , defined by the user, controls the frequency magnitude response of the filter.
- (a) What is the time-domain difference equation for the filter?
- (b) Does the filter have a linear-phase frequency response? Justify your answer.
- (c) What is the group delay of the filter measured in samples?

Figure P5-28



- 5.29** Here is a fun problem proving that you have actually been performing convolutions since you were a child. Show how the multiplication (computing the product) of the two numbers 24 and 13 can be performed by convolving their digits.

Chapter Six. Infinite Impulse Response Filters

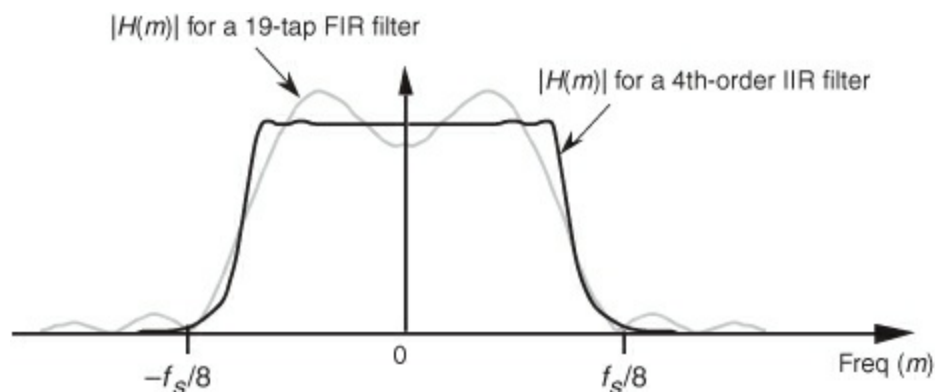


Infinite impulse response (IIR) digital filters are fundamentally different from FIR filters because practical IIR filters always require feedback. Where FIR filter output samples depend only on past input samples, each IIR filter output sample depends on previous input samples *and* previous filter output samples. IIR filters' *memory* of past outputs is both a blessing and a curse. As in all feedback systems, perturbations at the IIR filter input could, depending on the design, cause the filter output to become unstable and oscillate indefinitely. This characteristic of possibly having an infinite duration of nonzero output samples, even if the input becomes all zeros, is the origin of the phrase *infinite impulse response*. It's interesting at this point to know that, relative to FIR filters, IIR filters have more complicated structures (block diagrams), are harder to design and analyze, and do not have linear phase responses. Why in the world, then, would anyone use an IIR filter? Because they are very efficient. IIR filters require far fewer multiplications per filter output sample to achieve a given frequency magnitude response. From a hardware standpoint, this means that IIR filters can be very fast, allowing us to build real-time IIR filters that operate over much higher sample rates than FIR filters.[‡]

[‡] At the end of this chapter, we briefly compare the advantages and disadvantages of IIR filters relative to FIR filters.

To illustrate the utility of IIR filters, [Figure 6-1](#) contrasts the frequency magnitude responses of what's called a 4th-order lowpass IIR filter and the 19-tap FIR filter of [Figure 5-19\(b\)](#) from [Chapter 5](#). Where the 19-tap FIR filter in [Figure 6-1](#) requires 19 multiplications per filter output sample, the 4th-order IIR filter requires only 9 multiplications for each filter output sample. Not only does the IIR filter give us reduced passband ripple and a sharper filter roll-off, it does so with less than half the multiplication workload of the FIR filter.

Figure 6-1 Comparison of the frequency magnitude responses of a 19-tap lowpass FIR filter and a 4th-order lowpass IIR filter.



Recall from [Section 5.3](#) that to force an FIR filter's frequency response to have very steep transition regions, we had to design an FIR filter with a very long impulse response. The longer the impulse response, the more ideal our filter frequency response will become. From a hardware standpoint, the

maximum number of FIR filter taps we can have (the length of the impulse response) depends on how fast our hardware can perform the required number of multiplications and additions to get a filter output value before the next filter input sample arrives. IIR filters, however, can be designed to have impulse responses that are longer than their number of taps! Thus, IIR filters can give us much better filtering for a given number of multiplications per output sample than FIR filters. With this in mind, let's take a deep breath, flex our mathematical muscles, and learn about IIR filters.

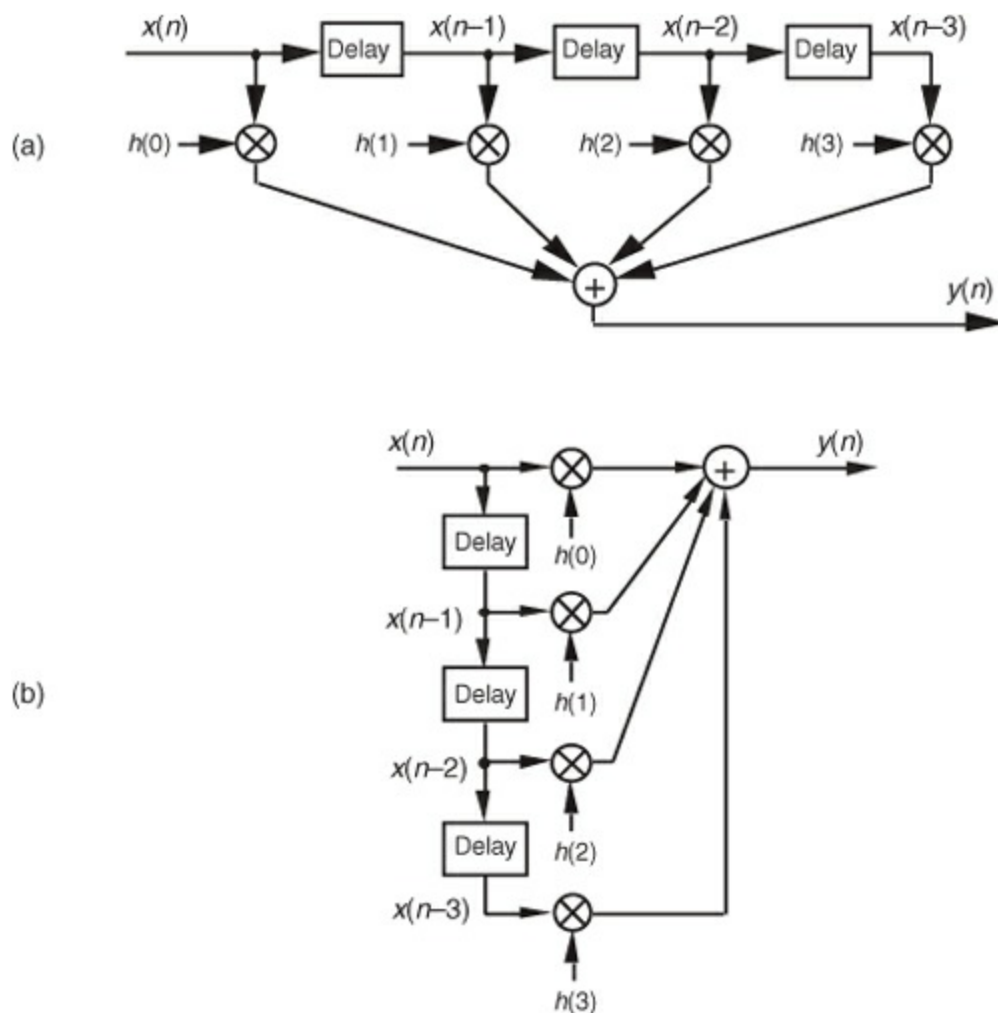
6.1 An Introduction to Infinite Impulse Response Filters

Given a finite duration of nonzero input values, an IIR filter will have an infinite duration of nonzero output samples. So, if the IIR filter's input suddenly becomes a sequence of all zeros, the filter's output could conceivably remain nonzero forever. This peculiar attribute of IIR filters comes about because of the way they're realized, i.e., the feedback structure of their delay units, multipliers, and adders. Understanding IIR filter structures is straightforward if we start by recalling the building blocks of an FIR filter. [Figure 6-2\(a\)](#) shows the now familiar structure of a 4-tap FIR digital filter that implements the time-domain FIR equation

$$(6-1)$$

$$y(n) = h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + h(3)x(n-3).$$

Figure 6-2 FIR digital filter structures: (a) traditional FIR filter structure; (b) rearranged, but equivalent, FIR filter structure.

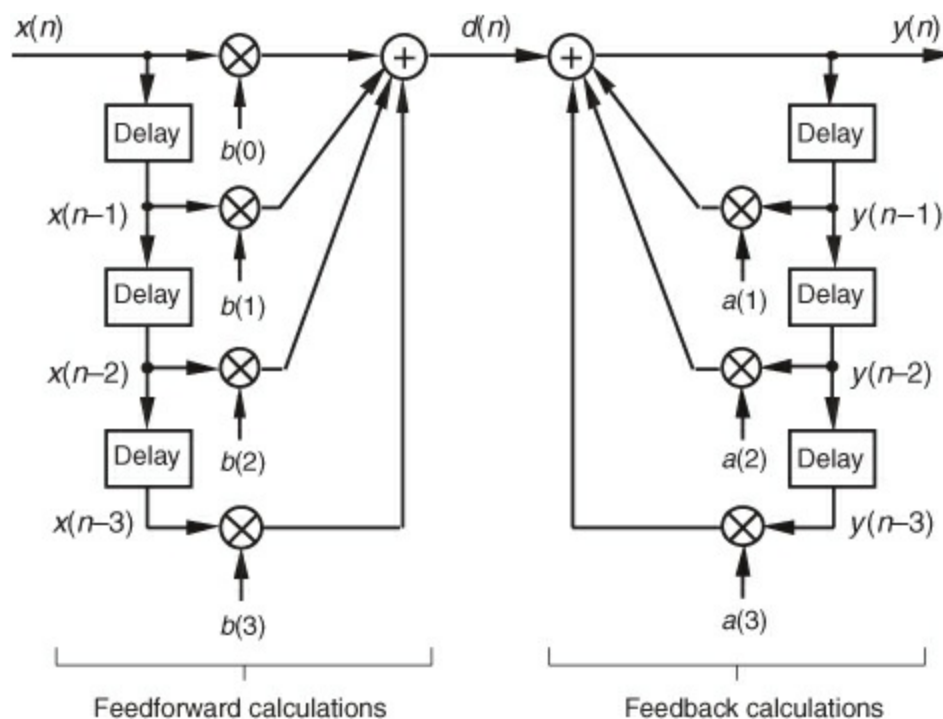


Although not specifically called out as such in [Chapter 5](#), [Eq. \(6-1\)](#) is known as a *difference equation*. To appreciate how past filter output samples are used in the structure of IIR filters, let's

begin by reorienting our FIR structure in [Figure 6-2\(a\)](#) to that of [Figure 6-2\(b\)](#). Notice how the structures in [Figure 6-2](#) are computationally identical, and both are implementations, or realizations, of [Eq. \(6-1\)](#).

We can now show how past filter output samples are combined with past input samples by using the IIR filter structure in [Figure 6-3](#). Because IIR filters have two sets of coefficients, we'll use the standard notation of the variables $b(k)$ to denote the feedforward coefficients and the variables $a(k)$ to indicate the feedback coefficients in [Figure 6-3](#). OK, the difference equation describing the IIR filter in [Figure 6-3](#) is

Figure 6-3 IIR digital filter structure showing feedforward and feedback calculations.



(6-2)

$$y(n) = b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + b(3)x(n-3)$$

$$+ a(1)y(n-1) + a(2)y(n-2) + a(3)y(n-3).$$

Look at [Figure 6-3](#) and [Eq. \(6-2\)](#) carefully. It's important to convince ourselves that [Figure 6-3](#) really is a valid implementation of [Eq. \(6-2\)](#) and that, conversely, difference equation [Eq. \(6-2\)](#) fully describes the IIR filter structure in [Figure 6-3](#). Keep in mind, now, that the sequence $y(n)$ in [Figure 6-3](#) is not the same $y(n)$ sequence that's shown in [Figure 6-2](#). The $d(n)$ sequence in [Figure 6-3](#) is equal to the $y(n)$ sequence in [Figure 6-2](#).

By now you're probably wondering, "Just how do we determine those $a(k)$ and $b(k)$ IIR filter coefficients if we actually want to design an IIR filter?" Well, fasten your seat belt because this is where we get serious about understanding IIR filters. Recall from the last chapter concerning the window method of lowpass FIR filter design that we defined the frequency response of our desired FIR filter, took the inverse Fourier transform of that frequency response, and then shifted that transform result to get the filter's time-domain impulse response. Happily, due to the nature of transversal FIR filters, the desired $h(k)$ filter coefficients turned out to be exactly equal to the impulse response sequence. Following that same procedure with IIR filters, we could define the desired frequency response of our IIR filter and then take the inverse Fourier transform of that response to

yield the filter's time-domain impulse response. The bad news is that there's no direct method for computing the IIR filter's $a(k)$ and $b(k)$ coefficients from the impulse response! Unfortunately, the FIR filter design techniques that we've learned so far simply cannot be used to design IIR filters. Fortunately for us, this wrinkle can be ironed out by using one of several available methods of designing IIR filters.

Standard IIR filter design techniques fall into three basic classes: the impulse invariance, bilinear transform, and optimization methods. These design methods use a discrete sequence, mathematical transformation process known as the z -transform whose origin is the Laplace transform traditionally used in the analyzing of continuous systems. With that in mind, let's start this IIR filter analysis and design discussion by briefly reacquainting ourselves with the fundamentals of the Laplace transform.

6.2 The Laplace Transform

The Laplace transform is a mathematical method of solving linear differential equations that has proved very useful in the fields of engineering and physics. This transform technique, as it's used today, originated from the work of the brilliant English physicist Oliver Heaviside.[‡] The fundamental process of using the Laplace transform goes something like the following:

[‡] Heaviside (1850–1925), who was interested in electrical phenomena, developed an efficient algebraic process of solving differential equations. He initially took a lot of heat from his contemporaries because they thought his work was not sufficiently justified from a mathematical standpoint. However, the discovered correlation of Heaviside's methods with the rigorous mathematical treatment of the French mathematician Marquis Pierre Simon de Laplace's (1749–1827) operational calculus verified the validity of Heaviside's techniques.

Step 1: A time-domain differential equation is written that describes the input/output relationship of a physical system (and we want to find the output function that satisfies that equation with a given input).

Step 2: The differential equation is Laplace transformed, converting it to an algebraic equation.

Step 3: Standard algebraic techniques are used to determine the desired output function's equation in the Laplace domain.

Step 4: The desired Laplace output equation is, then, inverse Laplace transformed to yield the desired time-domain output function's equation.

This procedure, at first, seems cumbersome because it forces us to *go the long way around*, instead of just solving a differential equation directly. The justification for using the Laplace transform is that although solving differential equations by classical methods is a very powerful analysis technique for all but the most simple systems, it can be tedious and (for some of us) error prone. The reduced complexity of using algebra outweighs the extra effort needed to perform the required forward and inverse Laplace transformations. This is especially true now that tables of forward and inverse Laplace transforms exist for most of the commonly encountered time functions. Well-known properties of the Laplace transform also allow practitioners to decompose complicated time functions into combinations of simpler functions and, then, use the tables. (Tables of Laplace transforms allow us to translate quickly back and forth between a time function and its Laplace transform—analogue to, say, a German-English dictionary if we were studying the German language.[‡]) Let's briefly look at a few of the more important characteristics of the Laplace transform that will prove useful as we make our way toward the discrete z -transform used to design and analyze IIR digital filters.

[‡] Although tables of commonly encountered Laplace transforms are included in almost every system analysis textbook, very

comprehensive tables are also available[1–3].

The Laplace transform of a continuous time-domain function $f(t)$, where $f(t)$ is defined only for positive time ($t > 0$), is expressed mathematically as

(6-3)

$$F(s) = \int_0^{\infty} f(t)e^{-st} dt.$$

$F(s)$ is called “the Laplace transform of $f(t)$,” and the variable s is the complex number

(6-4)

$$s = \sigma + j\omega.$$

A more general expression for the Laplace transform, called the *bilateral* or *two-sided transform*, uses negative infinity ($-\infty$) as the lower limit of integration. However, for the systems that we’ll be interested in, where system conditions for negative time ($t < 0$) are not needed in our analysis, the *one-sided* [Eq. \(6-3\)](#) applies. Those systems, often referred to as *causal systems*, may have initial conditions at $t = 0$ that must be taken into account (velocity of a mass, charge on a capacitor, temperature of a body, etc.), but we don’t need to know what the system was doing prior to $t = 0$.

In [Eq. \(6-4\)](#), σ is a real number and ω is frequency in radians/second. Because e^{-st} is dimensionless, the exponent term s must have the dimension of 1/time, or frequency. That’s why the Laplace variable s is often called a complex frequency.

To put [Eq. \(6-3\)](#) into words, we can say that it requires us to multiply, point for point, the function $f(t)$ by the complex function e^{-st} for a given value of s . (We’ll soon see that using the function e^{-st} here is not accidental; e^{-st} is used because it’s the general form for the solution of linear differential equations.) After the point-for-point multiplications, we find the area under the curve of the function $f(t)e^{-st}$ by summing all the products. That area, a complex number, represents the value of the Laplace transform for the particular value of $s = \sigma + j\omega$ chosen for the original multiplications. If we were to go through this process for all values of s , we’d have a full description of $F(s)$ for every value of s .

I like to think of the Laplace transform as a continuous function, where the complex value of that function for a particular value of s is a correlation of $f(t)$ and a damped complex e^{-st} sinusoid whose frequency is ω and whose damping factor is σ . What do these complex sinusoids look like? Well, they are rotating phasors described by

(6-5)

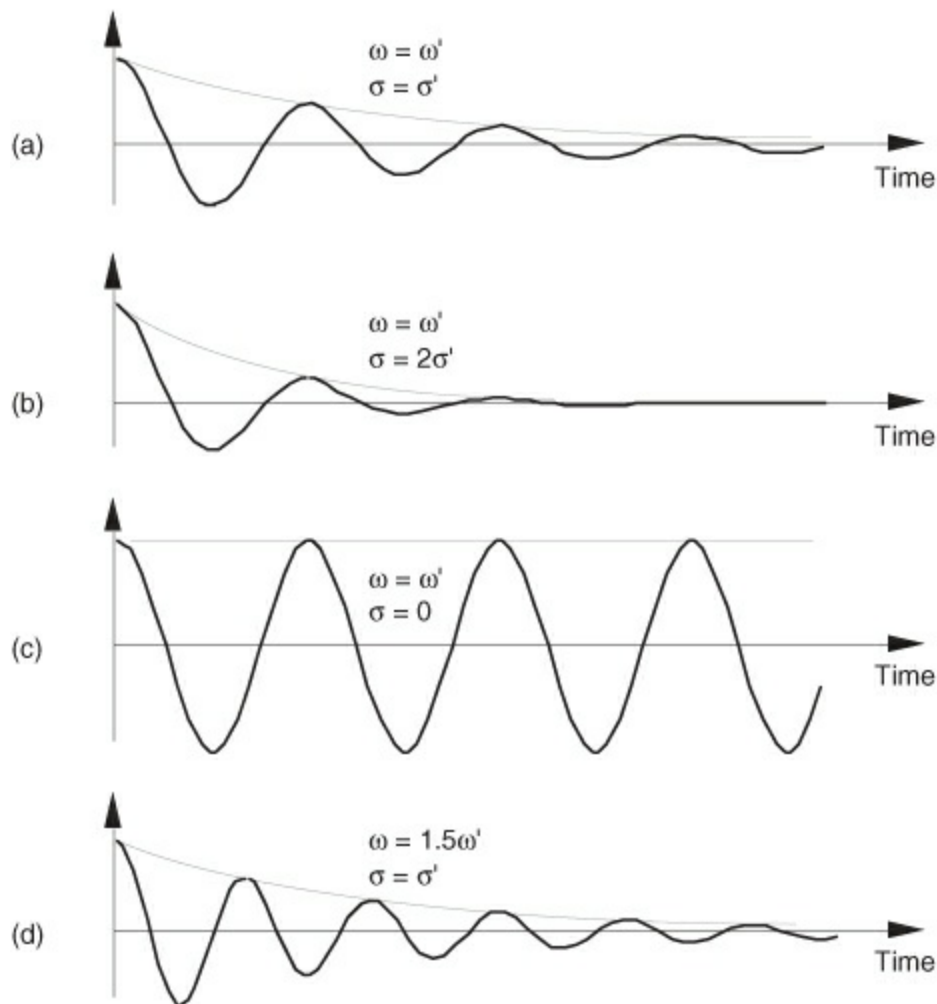
$$e^{-st} = e^{-(\sigma + j\omega)t} = e^{-\sigma t} e^{-j\omega t} = \frac{e^{-j\omega t}}{e^{\sigma t}}.$$

From our knowledge of complex numbers, we know that $e^{-j\omega t}$ is a unity-magnitude phasor rotating clockwise around the origin of a complex plane at a frequency of ω radians/second. The denominator of [Eq. \(6-5\)](#) is a real number whose value is one at time $t = 0$. As t increases, the denominator $e^{\sigma t}$ gets larger (when σ is positive), and the complex e^{-st} phasor’s magnitude gets smaller as the phasor rotates on the complex plane. The tip of that phasor traces out a curve spiraling in toward the origin of the complex plane. One way to visualize a complex sinusoid is to consider its real and imaginary parts individually. We do this by expressing the complex e^{-st} sinusoid from [Eq. \(6-5\)](#) in rectangular form as

$$e^{-st} = \frac{e^{-j\omega t}}{e^{\sigma t}} = \frac{\cos(\omega t)}{e^{\sigma t}} - j \frac{\sin(\omega t)}{e^{\sigma t}}.$$

[Figure 6-4](#) shows the real parts (cosine) of several complex sinusoids with different frequencies and different damping factors. In [Figure 6-4\(a\)](#), the complex sinusoid's frequency is the arbitrary ω' , and the damping factor is the arbitrary σ' . So the real part of $F(s)$, at $s = \sigma' + j\omega'$, is equal to the correlation of $f(t)$ and the wave in [Figure 6-4\(a\)](#). For different values of s , we'll correlate $f(t)$ with different complex sinusoids as shown in [Figure 6-4](#). (As we'll see, this correlation is very much like the correlation of $f(t)$ with various sine and cosine waves when we were calculating the discrete Fourier transform.) Again, the real part of $F(s)$, for a particular value of s , is the correlation of $f(t)$ with a cosine wave of frequency ω and a damping factor of σ , and the imaginary part of $F(s)$ is the correlation of $f(t)$ with a sinewave of frequency ω and a damping factor of σ .

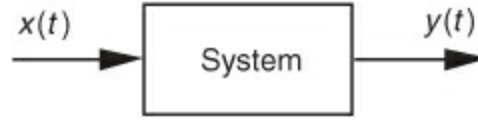
Figure 6-4 Real part (cosine) of various e^{-st} functions, where $s = \sigma + j\omega$, to be correlated with $f(t)$.



Now, if we associate each of the different values of the complex s variable with a point on a complex plane, rightfully called the s -plane, we could plot the real part of the $F(s)$ correlation as a surface above (or below) that s -plane and generate a second plot of the imaginary part of the $F(s)$ correlation as a surface above (or below) the s -plane. We can't plot the full complex $F(s)$ surface on paper because that would require four dimensions. That's because s is complex, requiring two dimensions, and $F(s)$ is itself complex and also requires two dimensions. What we can do, however, is graph the magnitude $|F(s)|$ as a function of s because this graph requires only three dimensions. Let's do that as we demonstrate this notion of an $|F(s)|$ surface by illustrating the Laplace transform in a tangible way.

Say, for example, that we have the linear system shown in [Figure 6-5](#). Also, let's assume that we can relate the $x(t)$ input and the $y(t)$ output of the linear time-invariant physical system in [Figure 6-5](#) with the following messy homogeneous constant-coefficient differential equation:

Figure 6-5 System described by [Eq. \(6-6\)](#). The system's input and output are the continuous-time functions $x(t)$ and $y(t)$ respectively.



(6-6)

$$a_2 \frac{d^2 y(t)}{dt^2} + a_1 \frac{dy(t)}{dt} + a_0 y(t) = b_1 \frac{dx(t)}{dt} + b_0 x(t).$$

We'll use the Laplace transform toward our goal of figuring out how the system will behave when various types of input functions are applied, i.e., what the $y(t)$ output will be for any given $x(t)$ input.

Let's slow down here and see exactly what [Figure 6-5](#) and [Eq. \(6-6\)](#) are telling us. First, if the system is time invariant, then the a_n and b_n coefficients in [Eq. \(6-6\)](#) are constant. They may be positive or negative, zero, real or complex, but they do not change with time. If the system is electrical, the coefficients might be related to capacitance, inductance, and resistance. If the system is mechanical with masses and springs, the coefficients could be related to mass, coefficient of damping, and coefficient of resilience. Then, again, if the system is thermal with masses and insulators, the coefficients would be related to thermal capacity and thermal conductance. To keep this discussion general, though, we don't really care what the coefficients actually represent.

OK, [Eq. \(6-6\)](#) also indicates that, ignoring the coefficients for the moment, the sum of the $y(t)$ output plus derivatives of that output is equal to the sum of the $x(t)$ input plus the derivative of that input. Our problem is to determine exactly what input and output functions satisfy the elaborate relationship in [Eq. \(6-6\)](#). (For the stout-hearted, classical methods of solving differential equations could be used here, but the Laplace transform makes the problem much simpler for our purposes.) Thanks to Laplace, the complex exponential time function of e^{st} is the one we'll use. It has the beautiful property that it can be differentiated any number of times without destroying its original form. That is,

(6-7)

$$\frac{d(e^{st})}{dt} = se^{st}, \quad \frac{d^2(e^{st})}{dt^2} = s^2 e^{st}, \quad \frac{d^3(e^{st})}{dt^3} = s^3 e^{st}, \dots, \quad \frac{d^n(e^{st})}{dt^n} = s^n e^{st}.$$

If we let $x(t)$ and $y(t)$ be functions of e^{st} , $x(e^{st})$ and $y(e^{st})$, and use the properties shown in [Eq. \(6-7\)](#), [Eq. \(6-6\)](#) becomes

(6-8)

$$a_2 s^2 y(e^{st}) + a_1 s y(e^{st}) + a_0 y(e^{st}) = b_1 s x(e^{st}) + b_0 x(e^{st}),$$

or

$$(a_2 s^2 + a_1 s + a_0) y(e^{st}) = (b_1 s + b_0) x(e^{st}).$$

Although it's simpler than [Eq. \(6-6\)](#), we can further simplify the relationship in the last line in [Eq. \(6-8\)](#) by considering the ratio of $y(e^{st})$ over $x(e^{st})$ as the Laplace transfer function of our system in [Figure 6-5](#). If we call that ratio of polynomials the transfer function $H(s)$,

(6-9)

$$H(s) = \frac{y(e^{st})}{x(e^{st})} = \frac{b_1s + b_0}{a_2s^2 + a_1s + a_0}.$$

To indicate that the original $x(t)$ and $y(t)$ have the identical functional form of e^{st} , we can follow the standard Laplace notation of capital letters and show the transfer function as

(6-10)

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_1s + b_0}{a_2s^2 + a_1s + a_0},$$

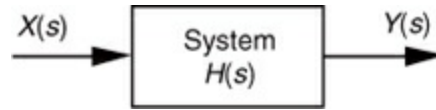
where the output $Y(s)$ is given by

(6-11)

$$Y(s) = X(s) \frac{b_1s + b_0}{a_2s^2 + a_1s + a_0} = X(s)H(s).$$

[Equation \(6-11\)](#) leads us to redraw the original system diagram in a form that highlights the definition of the transfer function $H(s)$ as shown in [Figure 6-6](#).

Figure 6-6 Linear system described by [Eqs. \(6-10\)](#) and [\(6-11\)](#). The system's input is the Laplace function $X(s)$, its output is the Laplace function $Y(s)$, and the system transfer function is $H(s)$.



The cautious reader may be wondering, “Is it really valid to use this Laplace analysis technique when it’s strictly based on the system’s $x(t)$ input being some function of e^{st} , or $x(e^{st})$?” The answer is that the Laplace analysis technique, based on the complex exponential $x(e^{st})$, is valid because all practical $x(t)$ input functions can be represented with complex exponentials, for example,

- a constant, $c = ce^{0t}$,
- sinusoids, $\sin(\omega t) = (e^{j\omega t} - e^{-j\omega t})/2j$ or $\cos(\omega t) = (e^{j\omega t} + e^{-j\omega t})/2$,
- a monotonic exponential, e^{at} , and
- an exponentially varying sinusoid, $e^{-at} \cos(\omega t)$.

With that said, if we know a system’s transfer function $H(s)$, we can take the Laplace transform of any $x(t)$ input to determine $X(s)$, multiply that $X(s)$ by $H(s)$ to get $Y(s)$, and then inverse Laplace transform $Y(s)$ to yield the time-domain expression for the output $y(t)$. In practical situations, however, we usually don’t go through all those analytical steps because it’s the system’s transfer function $H(s)$ in which we’re most interested. Being able to express $H(s)$ mathematically or graph the surface $|H(s)|$ as a function of s will tell us the two most important properties we need to know about the system under analysis: is the system stable, and if so, what is its frequency response?

“But wait a minute,” you say. “[Equations \(6-10\)](#) and [\(6-11\)](#) indicate that we have to know the $Y(s)$ output before we can determine $H(s)$!” Not really. All we really need to know is the time-domain differential equation like that in [Eq. \(6-6\)](#). Next we take the Laplace transform of that differential equation and rearrange the terms to get the $H(s)$ ratio in the form of [Eq. \(6-10\)](#). With practice, systems designers can look at a diagram (block, circuit, mechanical, whatever) of their system and promptly

write the Laplace expression for $H(s)$. Let's use the concept of the Laplace transfer function $H(s)$ to determine the stability and frequency response of simple continuous systems.

6.2.1 Poles and Zeros on the s -Plane and Stability

One of the most important characteristics of any system involves the concept of stability. We can think of a system as stable if, given any bounded input, the output will always be bounded. This sounds like an easy condition to achieve because most systems we encounter in our daily lives are indeed stable. Nevertheless, we have all experienced instability in a system containing feedback. Recall the annoying *howl* when a public address system's microphone is placed too close to the loudspeaker. A sensational example of an unstable system occurred in western Washington when the first Tacoma Narrows Bridge began oscillating on the afternoon of November 7, 1940. Those oscillations, caused by 42 mph winds, grew in amplitude until the bridge destroyed itself. For IIR digital filters with their built-in feedback, instability would result in a filter output that's not at all representative of the filter input; that is, our filter output samples would not be a filtered version of the input; they'd be some strange oscillating or pseudo-random values—a situation we'd like to avoid if we can, right? Let's see how.

We can determine a continuous system's stability by examining several different examples of $H(s)$ transfer functions associated with linear time-invariant systems. Assume that we have a system whose Laplace transfer function is of the form of [Eq. \(6-10\)](#), the coefficients are all real, and the coefficients b_1 and a_2 are equal to zero. We'll call that Laplace transfer function $H_1(s)$, where

(6-12)

$$H_1(s) = \frac{b_0}{a_1 s + a_0} = \frac{b_0 / a_1}{s + a_0 / a_1}.$$

Notice that if $s = -a_0/a_1$, the denominator in [Eq. \(6-12\)](#) equals zero and $H_1(s)$ would have an infinite magnitude. This $s = -a_0/a_1$ point on the s -plane is called a *pole*, and that pole's location is shown by the "x" in [Figure 6-7\(a\)](#). Notice that the pole is located exactly on the negative portion of the real σ axis. If the system described by H_1 were at rest and we disturbed it with an impulse like $x(t)$ input at time $t = 0$, its continuous time-domain $y(t)$ output would be the damped exponential curve shown in [Figure 6-7\(b\)](#). We can see that $H_1(s)$ is stable because its $y(t)$ output approaches zero as time passes. By the way, the distance of the pole from the $\sigma = 0$ axis, a_0/a_1 for our $H_1(s)$, gives the decay rate of the $y(t)$ impulse response. To illustrate why the term *pole* is appropriate, [Figure 6-8\(b\)](#) depicts the three-dimensional surface of $|H_1(s)|$ above the s -plane. Look at [Figure 6-8\(b\)](#) carefully and see how we've reoriented the s -plane axis. This new axis orientation allows us to see how the $H_1(s)$ system's frequency magnitude response can be determined from its three-dimensional s -plane surface. If we examine the $|H_1(s)|$ surface at $\sigma = 0$, we get the bold curve in [Figure 6-8\(b\)](#). That bold curve, the intersection of the vertical $\sigma = 0$ plane (the $j\omega$ axis plane) and the $|H_1(s)|$ surface, gives us the frequency magnitude response $|H_1(\omega)|$ of the system—and that's one of the things we're after here. The bold $|H_1(\omega)|$ curve in [Figure 6-8\(b\)](#) is shown in a more conventional way in [Figure 6-8\(c\)](#). [Figures 6-8\(b\)](#) and [6-8\(c\)](#) highlight the very important property that the Laplace transform is a more general case of the Fourier transform because if $\sigma = 0$, then $s = j\omega$. In this case, the $|H_1(s)|$ curve for $\sigma = 0$ above the s -plane becomes the $|H_1(\omega)|$ curve above the $j\omega$ axis in [Figure 6-8\(c\)](#).

Figure 6-7 Descriptions of $H_1(s)$: (a) pole located at $s = \sigma + j\omega = -a_0/a_1 + j0$ on the s -plane; (b) time-domain $y(t)$ impulse response of the system.

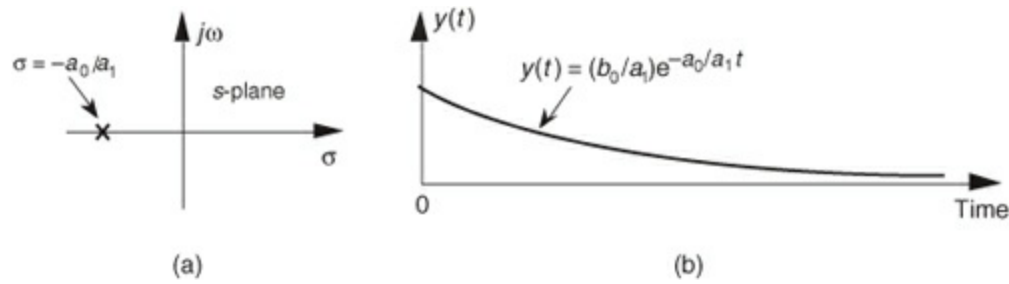
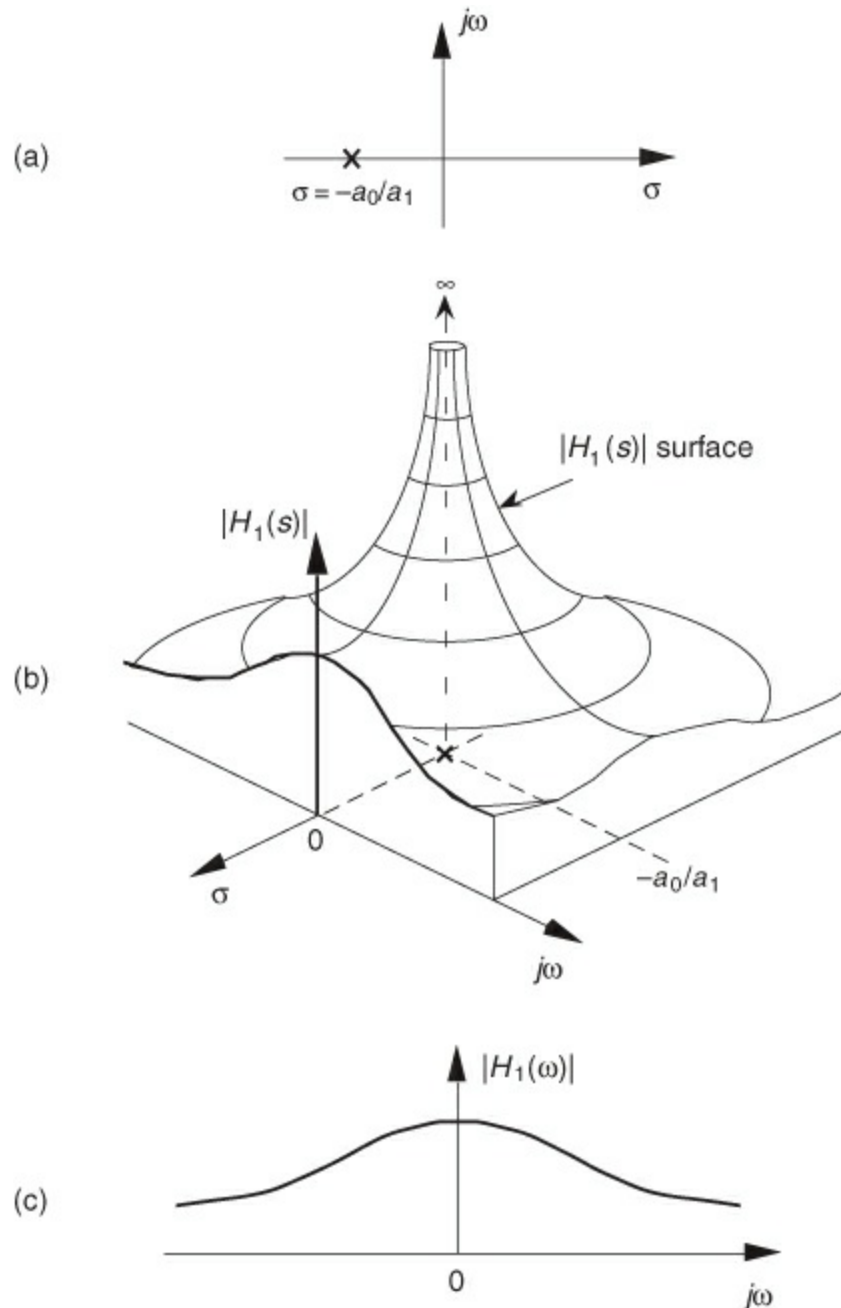


Figure 6-8 Further depictions of $H_1(s)$: (a) pole located at $\sigma = -a_0/a_1$ on the s -plane; (b) $|H_1(s)|$ surface; (c) curve showing the intersection of the $|H_1(s)|$ surface and the vertical $\sigma = 0$ plane. This is the conventional depiction of the $|H_1(\omega)|$ frequency magnitude response.



Another common system transfer function leads to an impulse response that oscillates. Let's think about an alternate system whose Laplace transfer function is of the form of [Eq. \(6-10\)](#), the coefficient b_0 equals zero, and the coefficients lead to complex terms when the denominator polynomial is

factored. We'll call this particular 2nd-order transfer function $H_2(s)$, where

(6-13)

$$H_2(s) = \frac{b_1 s}{a_2 s^2 + a_1 s + a_0} = \frac{(b_1 / a_2) s}{s^2 + (a_1 / a_2) s + a_0 / a_2}.$$

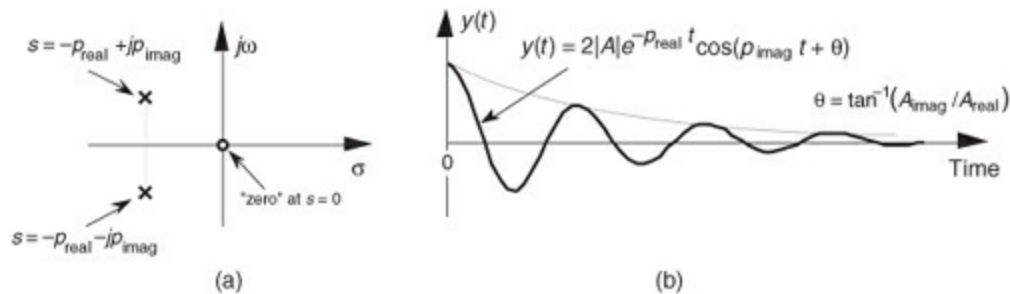
(By the way, when a transfer function has the Laplace variable s in both the numerator and denominator, the order of the overall function is defined by the largest exponential order of s in either the numerator or the denominator polynomial. So our $H_2(s)$ is a 2nd-order transfer function.) To keep the following equations from becoming too messy, let's factor its denominator and rewrite [Eq. \(6-13\)](#) as

(6-14)

$$H_2(s) = \frac{As}{(s+p)(s+p^*)}$$

where $A = b_1/a_2$, $p = p_{\text{real}} + jp_{\text{imag}}$, and $p^* = p_{\text{real}} - jp_{\text{imag}}$ (complex conjugate of p). Notice that if s is equal to $-p$ or $-p^*$, one of the polynomial roots in the denominator of [Eq. \(6-14\)](#) will equal zero, and $H_2(s)$ will have an infinite magnitude. Those two complex poles, shown in [Figure 6-9\(a\)](#), are located off the negative portion of the real σ axis. If the H_2 system were at rest and we disturbed it with an impulselike $x(t)$ input at time $t = 0$, its continuous time-domain $y(t)$ output would be the damped sinusoidal curve shown in [Figure 6-9\(b\)](#). We see that $H_2(s)$ is stable because its oscillating $y(t)$ output, like a plucked guitar string, approaches zero as time increases. Again, the distance of the poles from the $\sigma = 0$ axis ($-p_{\text{real}}$) gives the decay rate of the sinusoidal $y(t)$ impulse response. Likewise, the distance of the poles from the $j\omega = 0$ axis ($\pm p_{\text{imag}}$) gives the frequency of the sinusoidal $y(t)$ impulse response. Notice something new in [Figure 6-9\(a\)](#). When $s = 0$, the numerator of [Eq. \(6-14\)](#) is zero, making the transfer function $H_2(s)$ equal to zero. Any value of s where $H_2(s) = 0$ is sometimes of interest and is usually plotted on the s -plane as the little circle, called a *zero*, shown in [Figure 6-9\(a\)](#). At this point we're not very interested in knowing exactly what p and p^* are in terms of the coefficients in the denominator of [Eq. \(6-13\)](#). However, an energetic reader could determine the values of p and p^* in terms of a_0 , a_1 , and a_2 by using the following well-known quadratic factorization formula: Given the 2nd-order polynomial $f(s) = as^2 + bs + c$, then $f(s)$ can be factored as

Figure 6-9 Descriptions of $H_2(s)$: (a) poles located at $s = p_{\text{real}} \pm jp_{\text{imag}}$ on the s -plane; (b) time-domain $y(t)$ impulse response of the system.

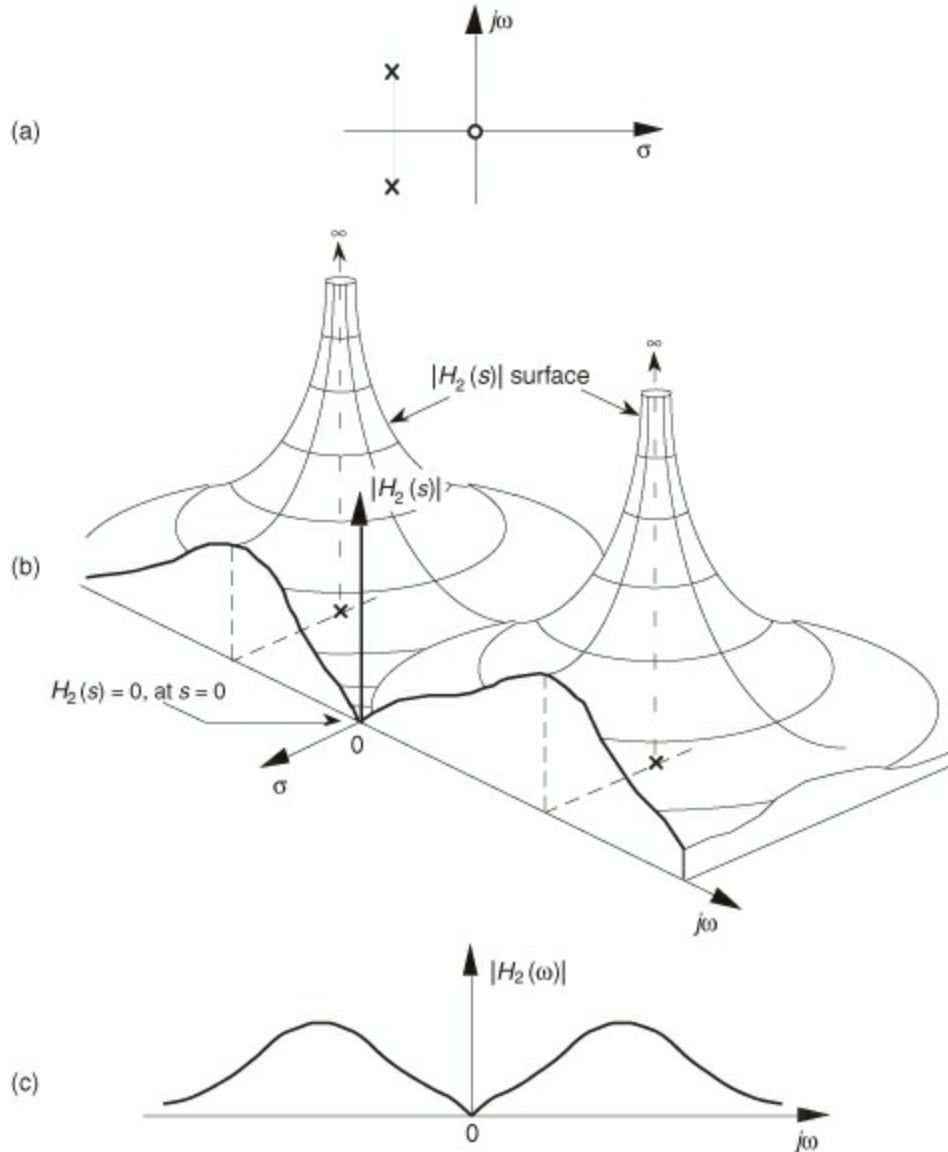


(6-15)

$$f(s) = as^2 + bs + c = \left(s + \frac{b}{2a} + \sqrt{\frac{b^2 - 4ac}{4a^2}} \right) \cdot \left(s + \frac{b}{2a} - \sqrt{\frac{b^2 - 4ac}{4a^2}} \right).$$

[Figure 6-10\(b\)](#) illustrates the $|H_2(s)|$ surface above the s -plane. Again, the bold $|H_2(\omega)|$ curve in [Figure 6-10\(b\)](#) is shown in the conventional way in [Figure 6-10\(c\)](#) to indicate the frequency magnitude response of the system described by [Eq. \(6-13\)](#). Although the three-dimensional surfaces in [Figures 6-8\(b\)](#) and [6-10\(b\)](#) are informative, they're also unwieldy and unnecessary. We can determine a system's stability merely by looking at the locations of the poles on the two-dimensional s -plane.

Figure 6-10 Further depictions of $H_2(s)$: (a) poles and zero locations on the s -plane; (b) $|H_2(s)|$ surface; (c) $|H_2(\omega)|$ frequency magnitude response curve.



To further illustrate the concept of system stability, [Figure 6-11](#) shows the s -plane pole locations of several example Laplace transfer functions and their corresponding time-domain impulse responses. We recognize [Figures 6-11\(a\)](#) and [6-11\(b\)](#), from our previous discussion, as indicative of stable systems. When disturbed from their at-rest condition, they respond and, at some later time, return to that initial condition. The single pole location at $s = 0$ in [Figure 6-11\(c\)](#) is indicative of the $1/s$ transfer function of a single element of a linear system. In an electrical system, this $1/s$ transfer function could be a capacitor that was charged with an impulse of current, and there's no discharge path in the circuit. For a mechanical system, [Figure 6-11\(c\)](#) would describe a kind of spring that's compressed with an impulse of force and, for some reason, remains under compression. Notice, in [Figure 6-11\(d\)](#), that if an $H(s)$ transfer function has conjugate poles located exactly on the $j\omega$ axis ($\sigma = 0$), the system will go into oscillation when disturbed from its initial condition. This situation, called

conditional stability, happens to describe the intentional transfer function of electronic oscillators. Instability is indicated in [Figures 6-11\(e\)](#) and [6-11\(f\)](#). Here, the poles lie to the right of the $j\omega$ axis. When disturbed from their initial at-rest condition by an impulse input, their outputs grow without bound.[‡] See how the value of σ , the real part of s , for the pole locations is the key here? When $\sigma < 0$, the system is well behaved and stable; when $\sigma = 0$, the system is conditionally stable; and when $\sigma > 0$, the system is unstable. So we can say that when σ is located on the right half of the s -plane, the system is unstable. We show this characteristic of linear continuous systems in [Figure 6-12](#). Keep in mind that real-world systems often have more than two poles, and a system is only as stable as its least stable pole. For a system to be stable, all of its transfer-function poles must lie on the left half of the s -plane.

[‡] Impulse response testing in a laboratory can be an important part of the system design process. The difficult part is generating a true impulselike input. If the system is electrical, for example, although somewhat difficult to implement, the input $x(t)$ impulse would be a very short-duration voltage or current pulse. If, however, the system were mechanical, a whack with a hammer would suffice as an $x(t)$ impulse input. For digital systems, on the other hand, an impulse input is easy to generate; it's a single unity-valued sample preceded and followed by all zero-valued samples.

Figure 6-11 Various $H(s)$ pole locations and their time-domain impulse responses: (a) single pole at $\sigma < 0$; (b) conjugate poles at $\sigma < 0$; (c) single pole located at $\sigma = 0$; (d) conjugate poles located at $\sigma = 0$; (e) single pole at $\sigma > 0$; (f) conjugate poles at $\sigma > 0$.

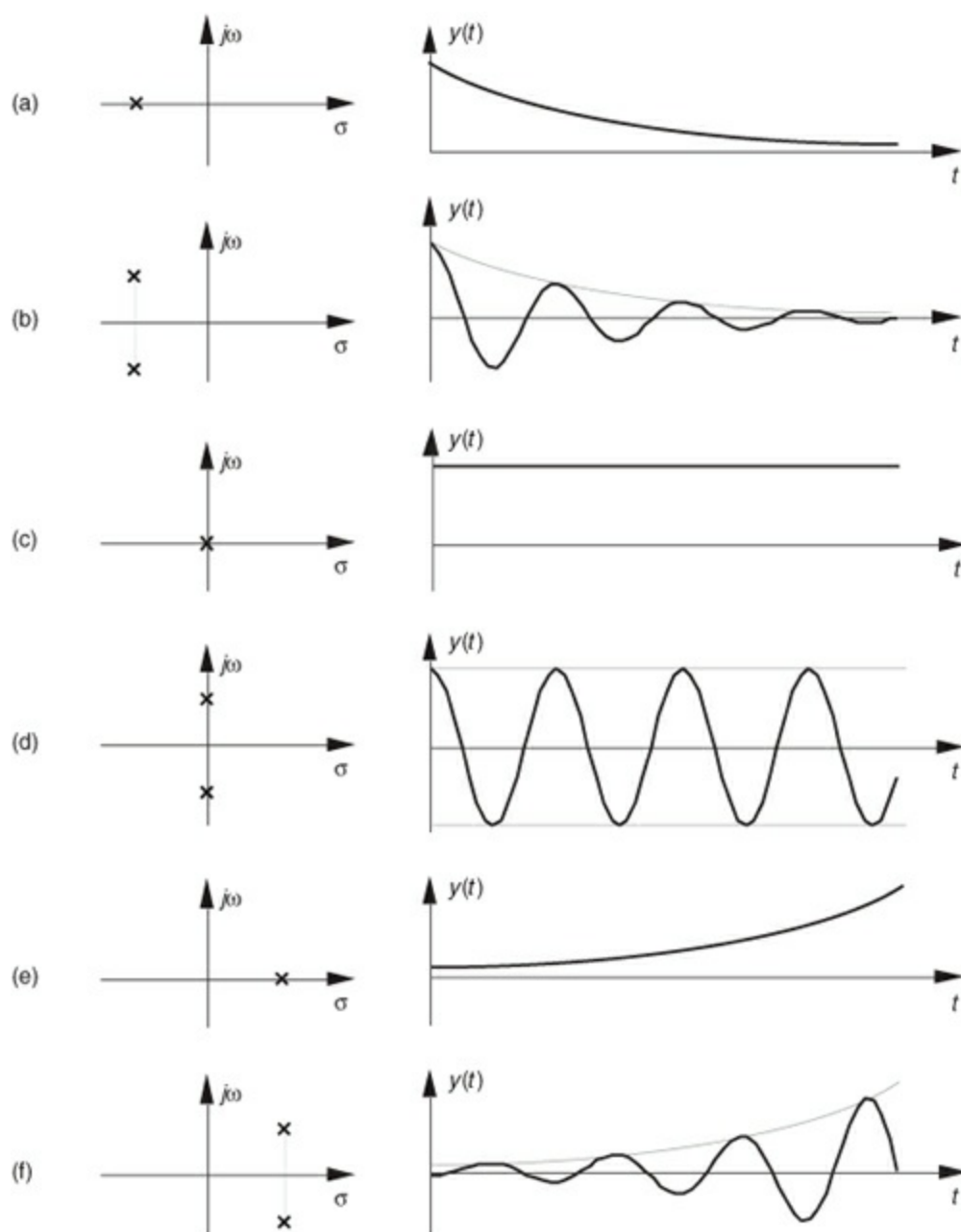
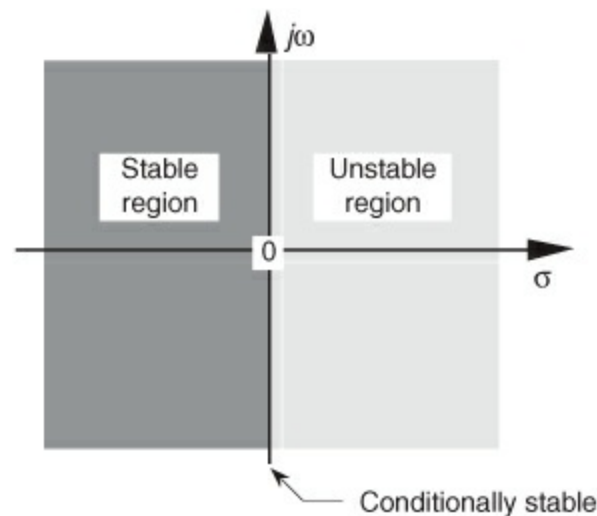


Figure 6-12 The Laplace s -plane showing the regions of stability and instability for pole locations for linear continuous systems.



To consolidate what we've learned so far: $H(s)$ is determined by writing a linear system's time-domain differential equation and taking the Laplace transform of that equation to obtain a Laplace expression in terms of $X(s)$, $Y(s)$, s , and the system's coefficients. Next we rearrange the Laplace

expression terms to get the $H(s)$ ratio in the form of [Eq. \(6-10\)](#). (The really slick part is that we do not have to know what the time-domain $x(t)$ input is to analyze a linear system!) We can get the expression for the continuous frequency response of a system just by substituting $j\omega$ for s in the $H(s)$ equation. To determine system stability, the denominator polynomial of $H(s)$ is factored to find each of its roots. Each root is set equal to zero and solved for s to find the location of the system poles on the s -plane. Any pole located to the right of the $j\omega$ axis on the s -plane will indicate an unstable system.

OK, returning to our original goal of understanding the z -transform, the process of analyzing IIR filter systems requires us to replace the Laplace transform with the z -transform and to replace the s -plane with a z -plane. Let's introduce the z -transform, determine what this new z -plane is, discuss the stability of IIR filters, and design and analyze a few simple IIR filters.

6.3 The z -Transform

The z -transform is the discrete-time cousin of the continuous Laplace transform.[‡] While the Laplace transform is used to simplify the analysis of continuous differential equations, the z -transform facilitates the analysis of discrete difference equations. Let's define the z -transform and explore its important characteristics to see how it's used in analyzing IIR digital filters.

[‡] In the early 1960s, James Kaiser, after whom the Kaiser window function is named, consolidated the theory of digital filters using a mathematical description known as the z -transform[\[4,5\]](#). Until that time, the use of the z -transform had generally been restricted to the field of discrete control systems[\[6-9\]](#).

The z -transform of a discrete sequence $h(n)$, expressed as $H(z)$, is defined as

(6-16)

$$H(z) = \sum_{n=-\infty}^{\infty} h(n)z^{-n}$$

where the variable z is complex. Where [Eq. \(6-3\)](#) allowed us to take the Laplace transform of a continuous signal, the z -transform is performed on a discrete $h(n)$ sequence, converting that sequence into a continuous function $H(z)$ of the continuous complex variable z . Similarly, as the function e^{-st} is the general form for the solution of linear differential equations, z^{-n} is the general form for the solution of linear difference equations. Moreover, as a Laplace function $F(s)$ is a continuous surface above the s -plane, the z -transform function $H(z)$ is a continuous surface above a z -plane. To whet your appetite, we'll now state that if $H(z)$ represents an IIR filter's z -domain transfer function, evaluating the $H(z)$ surface will give us the filter's frequency magnitude response, and $H(z)$'s pole and zero locations will determine the stability of the filter.

We can determine the frequency response of an IIR digital filter by expressing z in polar form as $z = re^{j\omega}$, where r is a magnitude and ω is the angle. In this form, the z -transform equation becomes

(6-16')

$$H(z) = H(re^{j\omega}) = \sum_{n=-\infty}^{\infty} h(n)(re^{j\omega})^{-n} = \sum_{n=-\infty}^{\infty} h(n)r^{-n}(e^{-j\omega n}).$$

[Equation \(6-16'\)](#) can be interpreted as the Fourier transform of the product of the original sequence $h(n)$ and the exponential sequence r^{-n} . When r equals one, [Eq. \(6-16'\)](#) simplifies to the Fourier transform. Thus on the z -plane, the contour of the $H(z)$ surface for those values where $|z| = 1$ is the

Fourier transform of $h(n)$. If $h(n)$ represents a filter impulse response sequence, evaluating the $H(z)$ transfer function for $|z| = 1$ yields the frequency response of the filter. So where on the z -plane is $|z| = 1$? It's a circle with a radius of one, centered about the $z = 0$ point. This circle, so important that it's been given the name *unit circle*, is shown in [Figure 6-13](#). Recall that the $j\omega$ frequency axis on the continuous Laplace s -plane was linear and ranged from $-\infty$ to $+\infty$ radians/second. The ω frequency axis on the complex z -plane, however, spans only the range from $-\pi$ to $+\pi$ radians. With this relationship between the $j\omega$ axis on the Laplace s -plane and the unit circle on the z -plane, we can see that the z -plane frequency axis is equivalent to coiling the s -plane's $j\omega$ axis about the unit circle on the z -plane as shown in [Figure 6-14](#).

Figure 6-13 Unit circle on the complex z -plane.

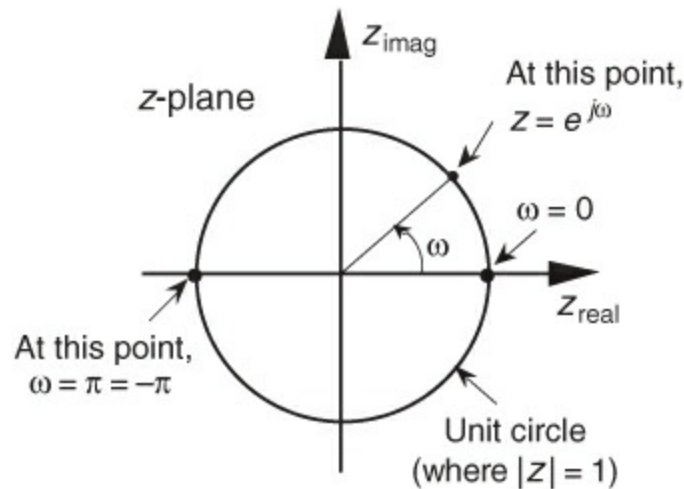
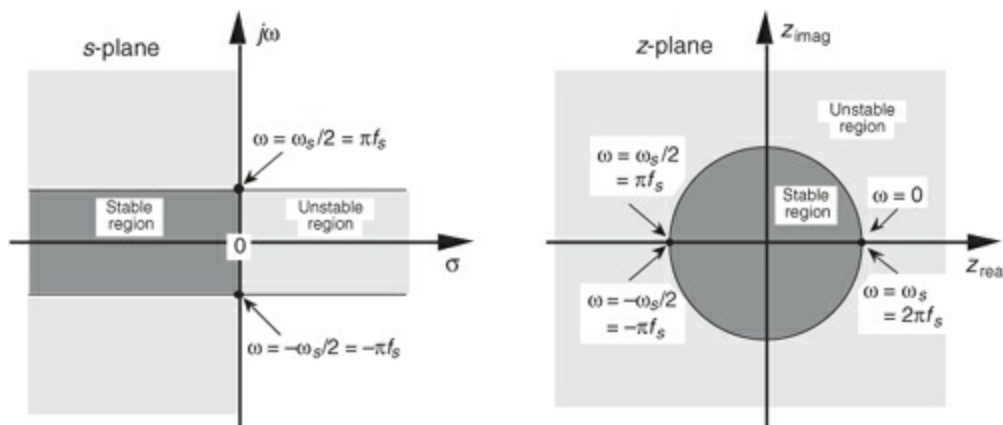


Figure 6-14 Mapping of the Laplace s -plane to the z -plane. All frequency values are in radians/second.



Then, frequency ω on the z -plane is not a distance along a straight line, but rather an angle around a circle. With ω in [Figure 6-13](#) being a general normalized angle in radians ranging from $-\pi$ to $+\pi$, we can relate ω to an equivalent f_s sampling rate by defining a new frequency variable $\omega_s = 2\pi f_s$ in radians/second. The periodicity of discrete frequency representations, with a period of $\omega_s = 2\pi f_s$ radians/second or f_s Hz, is indicated for the z -plane in [Figure 6-14](#). Where a walk along the $j\omega$ frequency axis on the s -plane could take us to infinity in either direction, a trip on the ω frequency path on the z -plane leads us in circles (on the unit circle). [Figure 6-14](#) shows us that only the $-\pi f_s$ to $+\pi f_s$ radians/second frequency range for ω can be accounted for on the z -plane, and this is another example of the universal periodicity of the discrete frequency domain. (Of course, the $-\pi f_s$ to $+\pi f_s$ radians/second range corresponds to a cyclic frequency range of $-f_s/2$ to $+f_s/2$.) With the perimeter of

the unit circle being $z = e^{j\omega}$, later, we'll show exactly how to substitute $e^{j\omega}$ for z in a filter's $H(z)$ transfer function, giving us the filter's frequency response.

6.3.1 Poles, Zeros, and Digital Filter Stability

One of the most important characteristics of the z -plane is that the region of filter stability is mapped to the inside of the unit circle on the z -plane. Given the $H(z)$ transfer function of a digital filter, we can examine that function's pole locations to determine filter stability. If all poles are located inside the unit circle, the filter will be stable. On the other hand, if any pole is located outside the unit circle, the filter will be unstable.

For example, if a causal filter's $H(z)$ transfer function has a single pole at location p on the z -plane, its transfer function can be represented by

(6-17)

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

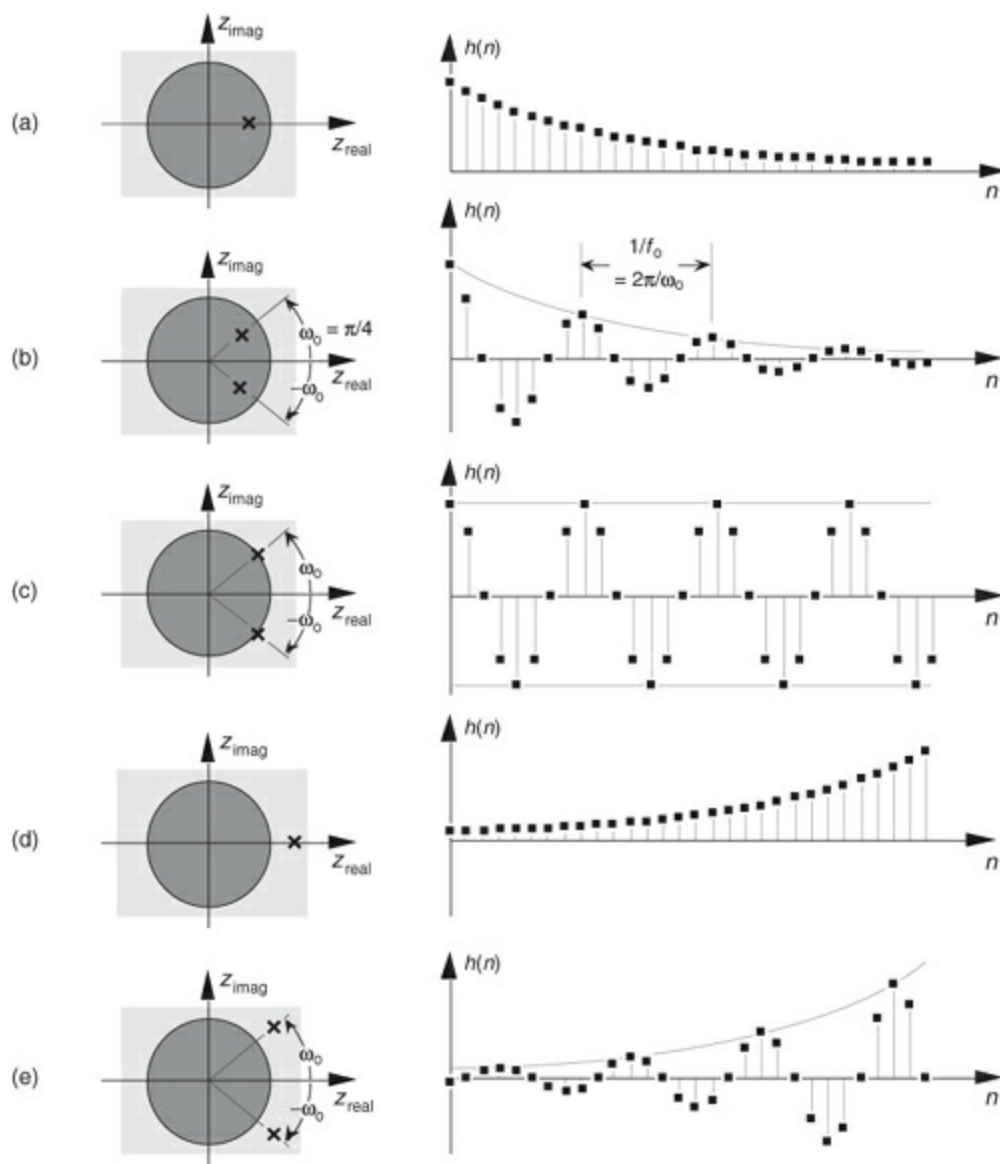
and the filter's time-domain impulse response sequence is

(6-17')

$$h(n) = p^n \cdot u(n)$$

where $u(n)$ represents a unit step (all ones) sequence beginning at time $n = 0$. [Equation \(6-17'\)](#) tells us that as time advances, the impulse response will be p raised to larger and larger powers. When the pole location p has a magnitude less than one, as shown in [Figure 6-15\(a\)](#), the $h(n)$ impulse response sequence is unconditionally bounded in amplitude. And a value of $|p| < 1$ means that the pole must lie inside the z -plane's unit circle.

Figure 6-15 Various $H(z)$ pole locations and their discrete time-domain impulse responses: (a) single pole inside the unit circle; (b) conjugate poles located inside the unit circle; (c) conjugate poles located on the unit circle; (d) single pole outside the unit circle; (e) conjugate poles located outside the unit circle.



[Figure 6-15](#) shows the z -plane pole locations of several example z -domain transfer functions and their corresponding discrete time-domain impulse responses. It's a good idea for the reader to compare the z -plane and discrete-time responses of [Figure 6-15](#) with the s -plane and the continuous-time responses of [Figure 6-11](#). The $y(n)$ outputs in [Figures 6-15\(d\)](#) and [6-15\(e\)](#) show examples of how unstable filter outputs increase in amplitude, as time passes, whenever their $x(n)$ inputs are nonzero. To avoid this situation, any IIR digital filter that we design should have an $H(z)$ transfer function with all of its individual poles inside the unit circle. Like a chain that's only as strong as its weakest link, an IIR filter is only as stable as its least stable pole.

The ω_0 oscillation frequency of the impulse responses in [Figures 6-15\(c\)](#) and [6-15\(e\)](#) is, of course, proportional to the angle of the conjugate pole pairs from the z_{real} axis, or ω_0 radians/second corresponding to $f_0 = \omega_0/2\pi$ Hz. Because the intersection of the $-z_{\text{real}}$ axis and the unit circle, point $z = -1$, corresponds to π radians (or πf_s radians/second $= f_s/2$ Hz), the ω_0 angle of $\pi/4$ in [Figure 6-15](#) means that $f_0 = f_s/8$ and our $y(n)$ will have eight samples per cycle of f_0 .

6.4 Using the z -Transform to Analyze IIR Filters

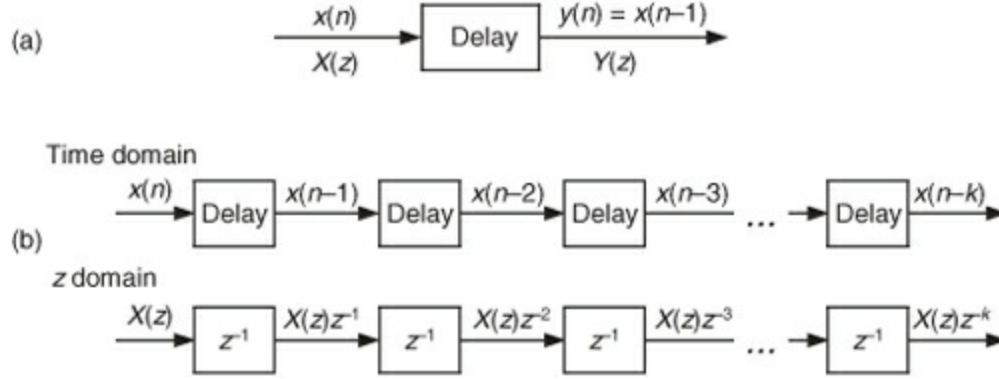
We have one last concept to consider before we can add the z -transform to our collection of digital signal processing tools. We need to determine how to represent [Figure 6-3](#)'s delay operation as part of our z -transform filter analysis equations. To do this, assume we have a sequence $x(n)$ whose z -

transform is $X(z)$ and a sequence $y(n) = x(n-1)$ whose z -transform is $Y(z)$ as shown in [Figure 6-16\(a\)](#). The z -transform of $y(n)$ is, by definition,

$$(6-18)$$

$$Y(z) = \sum_{n=-\infty}^{\infty} y(n)z^{-n} = \sum_{n=-\infty}^{\infty} x(n-1)z^{-n}.$$

Figure 6-16 Time- and z -domain delay element relationships: (a) single delay; (b) multiple delays.



Now if we let $k = n-1$, then $Y(z)$ becomes

$$(6-19)$$

$$Y(z) = \sum_{k=-\infty}^{\infty} x(k)z^{-(k+1)} = \sum_{k=-\infty}^{\infty} x(k)z^{-k}z^{-1},$$

which we can write as

$$(6-20)$$

$$Y(z) = z^{-1} \sum_{k=-\infty}^{\infty} x(k)z^{-k} = z^{-1} [X(z)].$$

Thus, the effect of a single unit of time delay is to multiply the z -transform of the undelayed sequence by z^{-1} .

6.4.1 z -Domain IIR Filter Analysis

Interpreting a unit time delay to be equivalent to the z^{-1} operator leads us to the relationship shown in [Figure 6-16\(b\)](#), where we can say $X(z)z^0 = X(z)$ is the z -transform of $x(n)$, $X(z)z^{-1}$ is the z -transform of $x(n)$ delayed by one sample, $X(z)z^{-2}$ is the z -transform of $x(n)$ delayed by two samples, and $X(z)z^{-k}$ is the z -transform of $x(n)$ delayed by k samples. So a transfer function of z^{-k} is equivalent to a delay of kt_s seconds from the instant when $t = 0$, where t_s is the period between data samples, or one over the sample rate. Specifically, $t_s = 1/f_s$. Because a delay of one sample is equivalent to the factor z^{-1} , the unit time delay symbol used in [Figures 6-2](#) and [6-3](#) is usually indicated by the z^{-1} operator as in [Figure 6-16\(b\)](#).

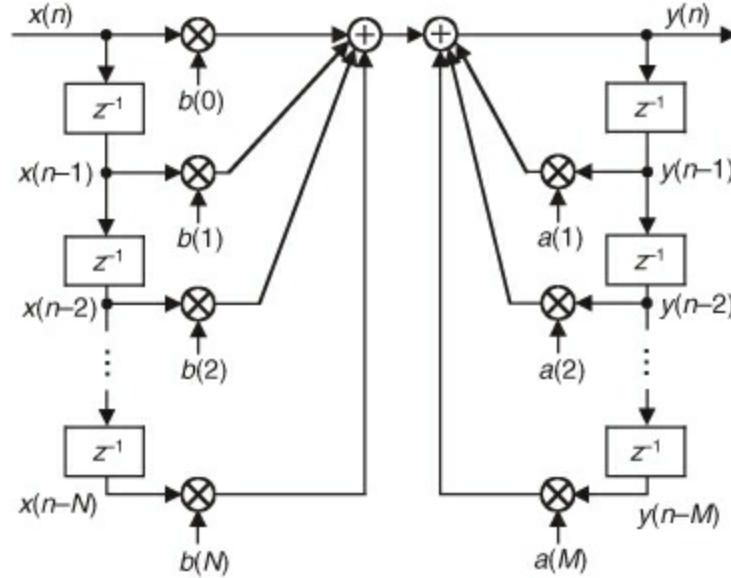
Let's pause for a moment and consider where we stand so far. Our acquaintance with the Laplace transform with its s -plane, the concept of stability based on $H(s)$ pole locations, the introduction of the z -transform with its z -plane poles, and the concept of the z^{-1} operator signifying a single unit of time delay has led us to our goal: the ability to inspect an IIR filter difference equation or filter structure (block diagram) and immediately write the filter's z -domain transfer function $H(z)$.

Accordingly, by evaluating an IIR filter's $H(z)$ transfer function appropriately, we can determine the filter's frequency response and its stability. With those ambitious thoughts in mind, let's develop the z -domain equations we need to analyze IIR filters. Using the relationships of [Figure 6-16\(b\)](#), we redraw [Figure 6-3](#) as a general M th-order IIR filter using the z^{-1} operator as shown in [Figure 6-17](#). (In hardware, those z^{-1} operations are memory locations holding successive filter input and output sample values. When implementing an IIR filter in a software routine, the z^{-1} operation merely indicates sequential memory locations where input and output sequences are stored.) The IIR filter structure in [Figure 6-17](#) is called the *Direct Form I* structure. The time-domain difference equation describing the general M th-order IIR filter, having N feedforward stages and M feedback stages, in [Figure 6-17](#) is

$$(6-21)$$

$$y(n) = b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + \dots + b(N)x(n-N) \\ + a(1)y(n-1) + a(2)y(n-2) + \dots + a(M)y(n-M).$$

Figure 6-17 General (Direct Form I) structure of an M th-order IIR filter, having N feedforward stages and M feedback stages, with the z^{-1} operator indicating a unit time delay.



In the z -domain, that IIR filter's output can be expressed by

$$(6-22)$$

$$Y(z) = b(0)X(z) + b(1)X(z)z^{-1} + b(2)X(z)z^{-2} + \dots + b(N)X(z)z^{-N} \\ + a(1)Y(z)z^{-1} + a(2)Y(z)z^{-2} + \dots + a(M)Y(z)z^{-M}$$

where $Y(z)$ and $X(z)$ represent the z -transform of $y(n)$ and $x(n)$. Look [Eqs. \(6-21\)](#) and [\(6-22\)](#) over carefully and see how the unit time delays translate to negative powers of z in the z -domain expression. A more compact notation for $Y(z)$ is

$$(6-23)$$

$$Y(z) = X(z) \sum_{k=0}^N b(k)z^{-k} + Y(z) \sum_{k=1}^M a(k)z^{-k}.$$

OK, now we've arrived at the point where we can describe the transfer function of a general IIR filter. Rearranging [Eq. \(6-23\)](#), to collect like terms, we write

$$(6-24)$$

$$Y(z) \left[1 - \sum_{k=1}^M a(k)z^{-k} \right] = X(z) \sum_{k=0}^N b(k)z^{-k}.$$

Finally, we define the filter's z -domain transfer function as $H(z) = Y(z)/X(z)$, where $H(z)$ is given by

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^N b(k)z^{-k}}{1 - \sum_{k=1}^M a(k)z^{-k}}.$$

Just as with Laplace transfer functions, the [order](#) of our z -domain transfer function and the order of our filter are defined by the largest exponential order of z in either the numerator or the denominator in [Eq. \(6-25\)](#).

There are two things we need to know about an IIR filter: its frequency response and whether or not the filter is stable. [Equation \(6-25\)](#) is the origin of that information. We can evaluate the denominator of [Eq. \(6-25\)](#) to determine the positions of the filter's poles on the z -plane indicating the filter's stability. Next, from [Eq. \(6-25\)](#) we develop an expression for the IIR filter's frequency response.

Remember, now, just as the Laplace transfer function $H(s)$ in [Eq. \(6-9\)](#) was a complex-valued surface on the s -plane, $H(z)$ is a complex-valued surface above, or below, the z -plane. The intersection of that $H(z)$ surface and the perimeter of a cylinder representing the $z = e^{j\omega}$ unit circle is the filter's complex frequency response. This means that substituting $e^{j\omega}$ for z in [Eq. \(6-25\)](#)'s transfer function gives us the expression for the filter's $H(\omega)$ frequency response as

$$H(\omega) = H(z) \Big|_{z=e^{j\omega}} = \frac{\sum_{k=0}^N b(k)e^{-jk\omega}}{1 - \sum_{k=1}^M a(k)e^{-jk\omega}}.$$

In rectangular form, using Euler's identity, $e^{-j\omega} = \cos(\omega) - j\sin(\omega)$, the filter's $H(\omega)$ frequency response is

$$H(\omega) = \frac{\sum_{k=0}^N b(k) \cdot \cos(k\omega) - j \sum_{k=0}^N b(k) \cdot \sin(k\omega)}{1 - \sum_{k=1}^M a(k) \cdot \cos(k\omega) - j \sum_{k=1}^M a(k) \cdot \sin(k\omega)}.$$

Shortly, we'll use the above expressions to analyze an actual IIR filter.

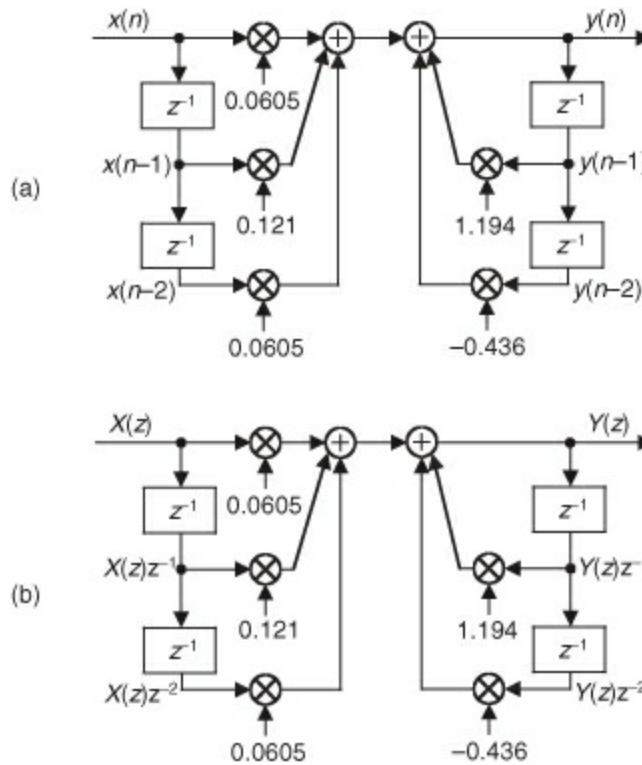
Pausing a moment to gather our thoughts, we realize that $H(\omega)$ is the ratio of complex functions and we can use [Eq. \(6-27\)](#) to compute the magnitude and phase response of IIR filters as a function of the frequency ω . And again, just what is ω ? It's the normalized frequency represented by the angle around the unit circle in [Figure 6-13](#), having a range of $-\pi \leq \omega \leq +\pi$ radians/sample. In terms of our old friend f_s Hz, [Eq. \(6-27\)](#) applies over the equivalent frequency range of $-f_s/2$ to $+f_s/2$ Hz. So, for example, if digital data is arriving at the filter's input at a rate of $f_s = 1000$ samples/second, we could use [Eq. \(6-](#)

27) to plot the filter's frequency magnitude response over the frequency range of -500 Hz to $+500$ Hz.

6.4.2 IIR Filter Analysis Example

Although [Eqs. \(6-25\)](#) and [\(6-26\)](#) look somewhat complicated at first glance, let's illustrate their simplicity and utility by analyzing the simple 2nd-order lowpass IIR filter in [Figure 6-18\(a\)](#) whose positive cutoff frequency is $\omega = \pi/5$ ($f_s/10$ Hz).

Figure 6-18 Second-order lowpass IIR filter example.



By inspection, we can write the filter's time-domain difference equation as

(6-28)

$$y(n) = 0.0605 \cdot x(n) + 0.121 \cdot x(n-1) + 0.0605 \cdot x(n-2) \\ + 1.194 \cdot y(n-1) - 0.436 \cdot y(n-2).$$

There are two ways to obtain the z-domain expression of our filter. The first way is to look at [Eq. \(6-28\)](#) and by inspection write

(6-29)

$$Y(z) = 0.0605 \cdot X(z) + 0.121 \cdot X(z)z^{-1} + 0.0605 \cdot X(z)z^{-2} \\ + 1.194 \cdot Y(z)z^{-1} - 0.436 \cdot Y(z)z^{-2}.$$

The second way to obtain the desired z-domain expression is to redraw [Figure 6-18\(a\)](#) with the z-domain notation as in [Figure 6-18\(b\)](#). Then by inspection of [Figure 6-18\(b\)](#) we could have written [Eq. \(6-29\)](#).

A piece of advice for the reader to remember: although not obvious in this IIR filter analysis example, it's often easier to determine a digital network's transfer function using the z-domain notation of [Figure 6-18\(b\)](#) rather than using the time-domain notation of [Figure 6-18\(a\)](#). (Writing the z-domain expression for a network based on the [Figure 6-18\(b\)](#) notation, rather than writing a time-domain expression based on the [Figure 6-18\(a\)](#) time notation, generally yields fewer unknown variables in our network analysis equations.) Over the years of analyzing digital networks, I regularly remind

myself, “z-domain produces less pain.” Keep this advice in mind if you attempt to solve the homework problems at the end of this chapter.

Back to our example: We can obtain the desired $H(z)$ filter transfer function by rearranging [Eq. \(6-29\)](#), or by using [Eq. \(6-25\)](#). Either method yields

(6-30)

$$H(z) = \frac{Y(z)}{X(z)} = \frac{0.0605 \cdot z^0 + 0.121 \cdot z^{-1} + 0.0605 \cdot z^{-2}}{1 - 1.194 \cdot z^{-1} + 0.436 \cdot z^{-2}}.$$

Replacing z with $e^{j\omega}$, we see that the frequency response of our example IIR filter is

(6-31)

$$H(\omega) = \frac{0.0605 \cdot e^{-j0\omega} + 0.121 \cdot e^{-j1\omega} + 0.0605 \cdot e^{-j2\omega}}{1 - 1.194 \cdot e^{-j1\omega} + 0.436 \cdot e^{-j2\omega}}.$$

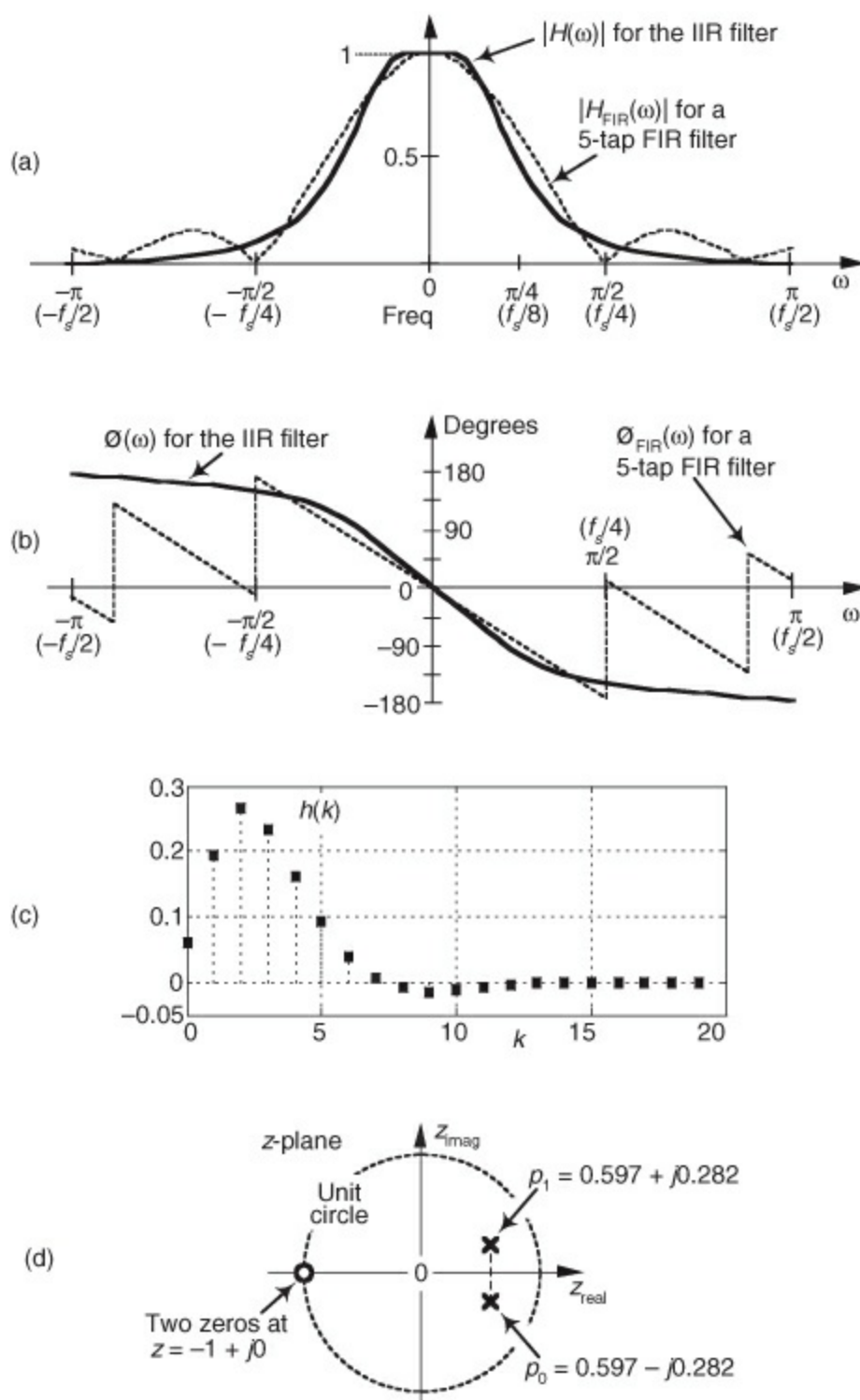
We’re almost there. Remembering Euler’s equations and that $\cos(0) = 1$ and $\sin(0) = 0$, we can write the rectangular form of $H(\omega)$ as

(6-32)

$$H(\omega) = \frac{0.0605 + 0.121 \cdot \cos(1\omega) + 0.0605 \cdot \cos(2\omega) - j[0.121 \cdot \sin(1\omega) + 0.0605 \cdot \sin(2\omega)]}{1 - 1.194 \cdot \cos(1\omega) + 0.436 \cdot \cos(2\omega) + j[1.194 \cdot \sin(1\omega) - 0.436 \cdot \sin(2\omega)]}.$$

[Equation \(6-32\)](#) is what we’re after here, and if we compute that messy expression’s magnitude over the frequency range of $-\pi \leq \omega \leq \pi$, we produce the $|H(\omega)|$ shown as the solid curve in [Figure 6-19\(a\)](#). For comparison purposes we also show a 5-tap lowpass FIR filter magnitude response in [Figure 6-19\(a\)](#). Although both filters require the same computational workload, five multiplications per filter output sample, the lowpass IIR filter has the superior frequency magnitude response. Notice the steeper magnitude response roll-off and lower sidelobes of the IIR filter relative to the FIR filter. (To make this IIR and FIR filter comparison valid, the coefficients used for both filters were chosen so that each filter would approximate the ideal lowpass frequency response shown in [Figure 5-17\(a\)](#).)

Figure 6-19 Performances of the example IIR filter (solid curves) in [Figure 6-18](#) and a 5-tap FIR filter (dashed curves): (a) magnitude responses; (b) phase responses; (c) IIR filter impulse response; (d) IIR filter poles and zeros.



A word of warning here. It's easy to inadvertently reverse some of the signs for the terms in the denominator of [Eq. \(6-32\)](#), so be careful if you attempt these calculations at home. Some authors avoid this problem by showing the $a(k)$ coefficients in [Figure 6-17](#) as negative values, so that the summation in the denominator of [Eq. \(6-25\)](#) is always positive. Moreover, some commercial software IIR design routines provide $a(k)$ coefficients whose signs must be reversed before they can be applied to the IIR structure in [Figure 6-17](#). (If, while using software routines to design or analyze IIR filters, your results are very strange or unexpected, the first thing to do is reverse the signs of the $a(k)$ coefficients and see if that doesn't solve the problem.)

The solid curve in [Figure 6-19\(b\)](#) is our IIR filter's $\emptyset(\omega)$ phase response. Notice its nonlinearity relative to the FIR filter's phase response. (Remember, now, we're only interested in the filter phase responses over the lowpass filter's passband. So those phase discontinuities for the FIR filter are of no consequence.) Phase nonlinearity is inherent in IIR filters and, based on the ill effects of nonlinear

phase introduced in the group delay discussion of [Section 5.8](#), we must carefully consider its implications whenever we decide to use an IIR filter instead of an FIR filter in any given application. The question any filter designer must ask and answer is “How much phase distortion can I tolerate to realize the benefits of the reduced computational workload and high data rates afforded by IIR filters?”

[Figure 6-19\(c\)](#) shows our filter’s time-domain $h(k)$ impulse response. Knowing that the filter’s phase response is nonlinear, we should expect the impulse response to be asymmetrical as it indeed is. That figure also illustrates why the term *infinite impulse response* is used to describe IIR filters. If we used infinite-precision arithmetic in our filter implementation, the $h(k)$ impulse response would be infinite in duration. In practice, of course, a filter’s output samples are represented by a finite number of binary bits. This means that a stable IIR filter’s $h(k)$ samples will decrease in amplitude, as time index k increases, and eventually reach an amplitude level that’s less than the smallest representable binary value. After that, all future $h(k)$ samples will be zero-valued.

To determine our IIR filter’s stability, we must find the roots of the 2nd-order polynomial of $H(z)$ ’s denominator in [Eq. \(6-30\)](#). Those roots are the z -plane poles of $H(z)$ and if their magnitudes are less than one, the IIR filter is stable. To determine the two pole locations, p_0 and p_1 , first we multiply $H(z)$ by z^2/z^2 to obtain polynomials with positive exponents. After doing so, $H(z)$ becomes

(6-33)

$$H(z) = \frac{0.0605z^2 + 0.121z + 0.0605}{z^2 - 1.194z + 0.436}.$$

Factoring [Eq. \(6-33\)](#) using the quadratic factorization formula from [Eq. \(6-15\)](#), we obtain the ratio of factors

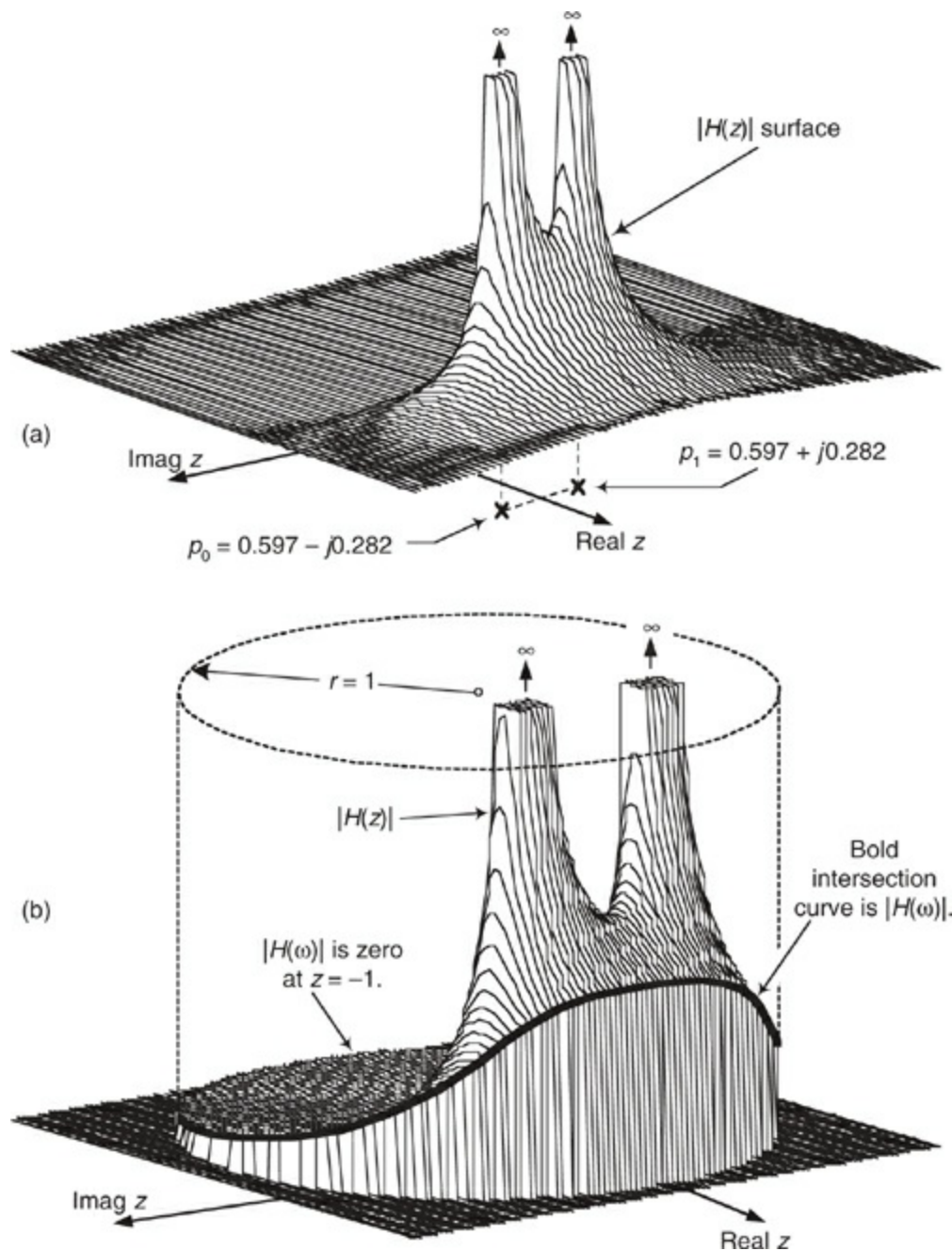
(6-34)

$$H(z) = \frac{(z - z_0)(z - z_1)}{(z - p_0)(z - p_1)} = \frac{(z + 1)(z + 1)}{(z - 0.597 + j0.282)(z - 0.597 - j0.282)}.$$

So when $z = p_0 = 0.597 - j0.282$, or when $z = p_1 = 0.597 + j0.282$, the filter’s $H(z)$ transfer function’s denominator is zero and $|H(z)|$ is infinite. We show the p_0 and p_1 pole locations in [Figure 6-19\(d\)](#). Because those pole locations are inside the unit circle (their magnitudes are less than one), our example IIR filter is unconditionally stable. The two factors in the numerator of [Eq. \(6-34\)](#) correspond to two z -plane zeros at $z = z_0 = z_1 = -1$ (at a continuous-time frequency of $\pm f_s/2$), shown in [Figure 6-19\(d\)](#).

To help us understand the relationship between the poles/zeros of $H(z)$ and the magnitude of the $H(z)$ transfer function, we show a crude depiction of the $|H(z)|$ surface as a function of z in [Figure 6-20\(a\)](#).

Figure 6-20 IIR filter’s $|H(z)|$ surface: (a) pole locations; (b) frequency magnitude response.



Continuing to review the $|H(z)|$ surface, we can show its intersection with the unit circle as the bold curve in [Figure 6-20\(b\)](#). Because $z = re^{j\omega}$, with r restricted to unity, then $z = e^{j\omega}$ and the bold curve is $|H(z)|_{|z|=1} = |H(\omega)|$, representing the lowpass filter's frequency magnitude response on the z -plane. If we were to unwrap the bold $|H(\omega)|$ curve in [Figure 6-20\(b\)](#) and lay it on a flat surface, we would have the $|H(\omega)|$ curve in [Figure 6-19\(a\)](#). Neat, huh?

6.5 Using Poles and Zeros to Analyze IIR Filters

In the last section we discussed methods for finding an IIR filter's z -domain $H(z)$ transfer function in order to determine the filter's frequency response and stability. In this section we show how to use a digital filter's pole/zero locations to analyze that filter's frequency-domain performance. To understand this process, first we must identify the two most common algebraic forms used to express a filter's z -domain transfer function.

6.5.1 IIR Filter Transfer Function Algebra

We have several ways to write the $H(z) = Y(z)/X(z)$ z -domain transfer function of an IIR filter. For example, similar to [Eq. \(6-30\)](#), we can write $H(z)$ in the form of a ratio of polynomials in negative

powers of z . For a 4th-order IIR filter such an $H(z)$ expression would be

(6-35)

$$H(z) = \frac{b(0) + b(1)z^{-1} + b(2)z^{-2} + b(3)z^{-3} + b(4)z^{-4}}{1 + a(1)z^{-1} + a(2)z^{-2} + a(3)z^{-3} + a(4)z^{-4}}.$$

Expressions like [Eq. \(6-35\)](#) are super-useful because we can replace z with $e^{j\omega}$ to obtain an expression for the frequency response of the filter. We used that substitution in the last section.

On the other hand, multiplying [Eq. \(6-35\)](#) by z^4/z^4 , we can express $H(z)$ in the *polynomial* form

(6-36)

$$H(z) = \frac{b(0)z^4 + b(1)z^3 + b(2)z^2 + b(3)z + b(4)}{z^4 + a(1)z^3 + a(2)z^2 + a(3)z + a(4)}.$$

Expressions in the form of [Eq. \(6-36\)](#) are necessary so we can factor (find the roots of) the polynomials to obtain values (locations) of the numerator zeros and denominator poles, such as in the following *factored* form:

(6-37)

$$H(z) = \frac{(z - z_0)(z - z_1)(z - z_2)(z - z_3)}{(z - p_0)(z - p_1)(z - p_2)(z - p_3)}.$$

Such an $H(z)$ transfer function has four zeros (z_0, z_1, z_2 , and z_3) and four poles (p_0, p_1, p_2 , and p_3). We're compelled to examine a filter's $H(z)$ transfer function in the factored form of [Eq. \(6-37\)](#) because the p_k pole values tell us whether or not the IIR filter is stable. If the magnitudes of all p_k poles are less than one, the filter is stable. The filter zeros, z_k , do not affect filter stability.

As an aside, while we won't encounter such filters until [Chapter 7](#) and [Chapter 10](#), it is possible to have a digital filter whose transfer function, in the factored form of [Eq. \(6-37\)](#), has common (identical) factors in its numerator and denominator. Those common factors produce a zero and a pole that lie exactly on top of each other. Like matter and anti-matter, such zero-pole combinations annihilate each other, leaving neither a zero nor a pole at that z -plane location.

Multiplying the factors in [Eq. \(6-37\)](#), a process called "expanding the transfer function" allows us to go from the factored form of [Eq. \(6-37\)](#) to the polynomial form in [Eq. \(6-36\)](#). As such, in our digital filter analysis activities we can translate back and forth between the polynomial and factored forms of $H(z)$.

Next we review the process of analyzing a digital filter given the filter's poles and zeros.

6.5.2 Using Poles/Zeros to Obtain Transfer Functions

As it turns out, we can analyze an IIR filter's frequency-domain performance based solely on the filter's poles and zeros. Given that we know the values of a filter's z_k zeros and p_k poles, we can write the factored form of the filter's transfer function as

(6-38)

$$H(z) = \frac{G_1(z-z_0)(z-z_1)(z-z_2)(z-z_3)(z-z_4)\dots}{G_2(z-p_0)(z-p_1)(z-p_2)(z-p_3)(z-p_4)\dots}$$

$$= \frac{G(z-z_0)(z-z_1)(z-z_2)(z-z_3)(z-z_4)\dots}{(z-p_0)(z-p_1)(z-p_2)(z-p_3)(z-p_4)\dots}$$

where $G = G_1/G_2$ is an arbitrary gain constant. Thus, knowing a filter's z_k zeros and p_k poles, we can determine the filter's transfer function to within a constant scale factor G .

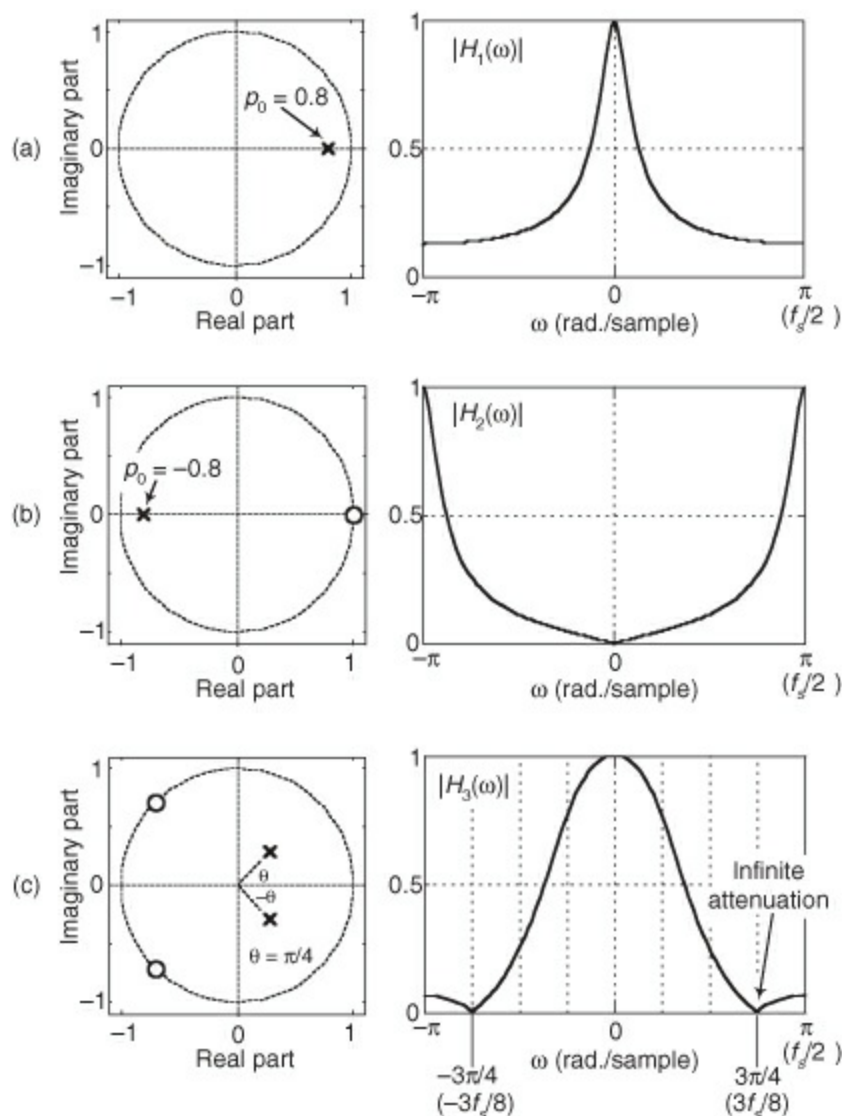
Again, filter zeros are associated with decreased frequency magnitude response, and filter poles are associated with increased frequency magnitude response. For example, if we know that a filter has no z -plane zeros, and one pole at $p_0 = 0.8$, we can write its transfer function as

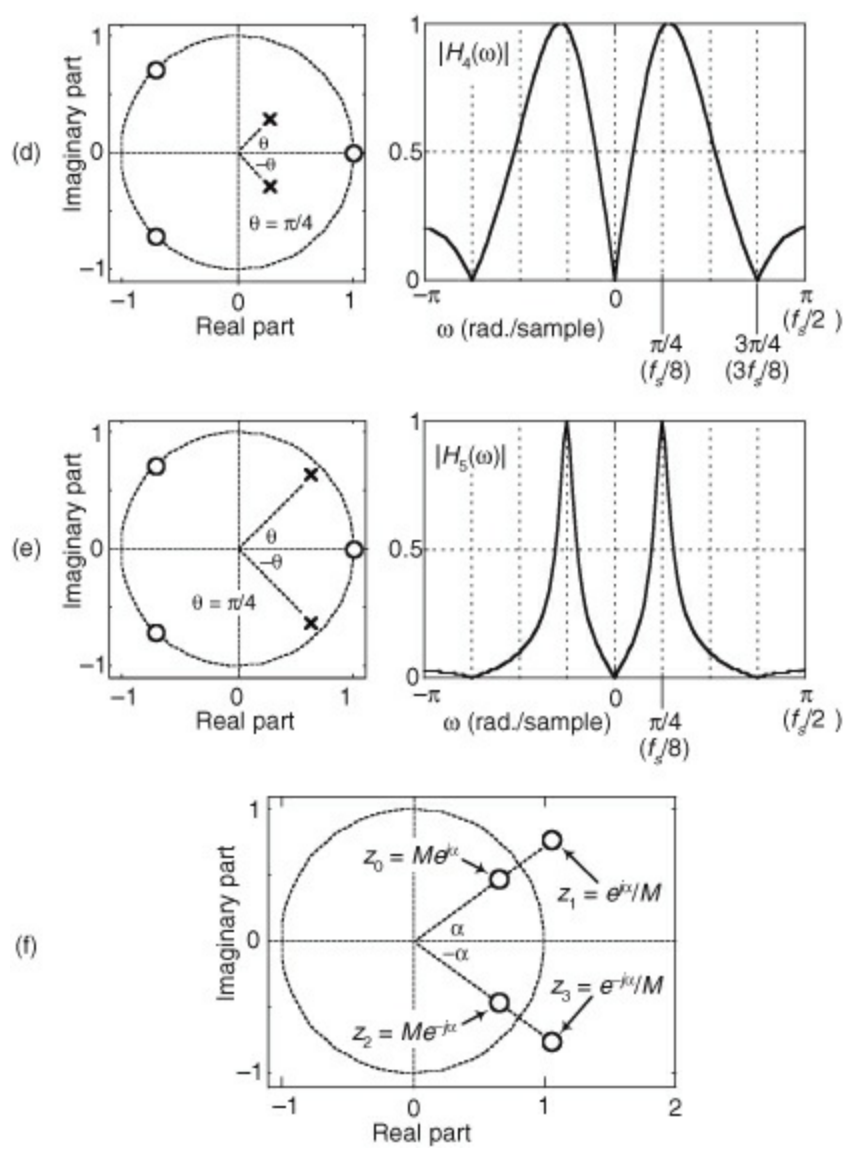
(6-39)

$$H_1(z) = \frac{G}{z-0.8}.$$

The characteristics of such a filter are depicted in [Figure 6-21\(a\)](#). The $|H_1(\omega)|$ frequency magnitude response in the figure is normalized so that the peak magnitude is unity. Because the p_0 pole is closest to the $\omega = 0$ radians/sample frequency point ($z = 1$) on the unit circle, the filter is a lowpass filter. Additionally, because $|p_0|$ is less than one, the filter is unconditionally stable.

Figure 6-21 IIR filter poles/zeros and normalized frequency magnitude responses.





If a filter has a zero at $z_0 = 1$, and a pole at $p_0 = -0.8$, we write its transfer function as

$$(6-40)$$

$$H_2(z) = \frac{G(z-1)}{z-(-0.8)} = \frac{Gz-G}{z+0.8}.$$

The characteristics of this filter are shown in [Figure 6-21\(b\)](#). Because the pole is closest to the $\omega = \pi$ radians/sample frequency point ($z = -1$) on the unit circle, the filter is a highpass filter. Notice that the zero located at $z = 1$ causes the filter to have infinite attenuation at $\omega = 0$ radians/sample (zero Hz). Because this pole/zero filter analysis topic is so important, let us look at several more pole/zero examples.

Consider a filter having two complex conjugate zeros at $-0.707 \pm j0.707$, as well as two complex conjugate poles at $0.283 \pm j0.283$. This filter's transfer function is

$$(6-41)$$

$$\begin{aligned} H_3(z) &= \frac{G[z - (-0.707 + j0.707)] \cdot [z - (-0.707 - j0.707)]}{[z - (0.283 + j0.283)] \cdot [z - (0.283 - j0.283)]} \\ &= \frac{G(z + 0.707 - j0.707) \cdot (z + 0.707 + j0.707)}{(z - 0.283 - j0.283) \cdot (z - 0.283 + j0.283)}. \end{aligned}$$

The properties of this $H_3(z)$ filter are presented in [Figure 6-21\(c\)](#). The two poles on the right side of

the z -plane make this a lowpass filter having a wider passband than the above $H_1(z)$ lowpass filter. Two zeros are on the unit circle at angles of $\omega = \pm 3\pi/4$ radians, causing the filter to have infinite attenuation at the frequencies $\omega = \pm 3\pi/4$ radians/sample ($\pm 3f_s/8$ Hz) as seen in the $|H_3(\omega)|$ magnitude response.

If we add a z -plane zero at $z = 1$ to the above $H_3(z)$, we create an $H_4(z)$ filter whose transfer function is

$$(6-42)$$

$$H_4(z) = \frac{G(z-1) \cdot (z+0.707-j0.707) \cdot (z+0.707+j0.707)}{(z-0.283-j0.283) \cdot (z-0.283+j0.283)}.$$

The characteristics of this filter are shown in [Figure 6-21\(d\)](#). The zero at $z = 1$ yields infinite attenuation at $\omega = 0$ radians/sample (zero Hz), creating a bandpass filter. Because the p_0 and p_1 poles of $H_4(z)$ are oriented at angles of $\theta = \pm\pi/4$ radians, the filter's passbands are centered in the vicinity of frequencies $\omega = \pm\pi/4$ radians/sample ($\pm f_s/8$ Hz).

Next, if we increase the magnitude of the $H_4(z)$ filter's poles, making them equal to $0.636 \pm j0.636$, we position the conjugate poles much closer to the unit circle as shown by the $H_5(z)$ characteristics in [Figure 6-21\(e\)](#). The $H_5(z)$ filter transfer function is

$$(6-43)$$

$$H_5(z) = \frac{G(z-1) \cdot (z+0.707-j0.707) \cdot (z+0.707+j0.707)}{(z-0.636-j0.636) \cdot (z-0.636+j0.636)}.$$

There are two issues to notice in this scenario. First, poles near the unit circle now have a much more profound effect on the filter's magnitude response. The poles' infinite gains cause the $H_5(z)$ passbands to be very narrow (sharp). Second, when a pole is close to the unit circle, the center frequency of its associated passband can be accurately estimated to be equal to the pole's angle. That is, [Figure 6-21\(e\)](#) shows us that with the poles' angles being $\theta = \pm\pi/4$ radians, the center frequencies of the narrow passbands are *very* nearly equal to $\omega = \pm\pi/4$ radians/sample ($\pm f_s/8$ Hz).

For completeness, one last pole/zero topic deserves mention. Consider a finite impulse response (FIR) filter—a digital filter whose $H(z)$ transfer function denominator is unity. For an FIR filter to have linear phase each z -plane zero located at $z = z_0 = Me^{j\alpha}$, where $M \neq 1$, must be accompanied by a zero having an angle of $-\alpha$ and a magnitude of $1/M$. (Proof of this restriction is available in reference [\[10\]](#).) We show this restriction in [Figure 6-21\(f\)](#) where the z_0 zero is accompanied by the z_3 zero. If the FIR filter's transfer function polynomial has real-valued b_k coefficients, then a z_0 zero not on the z -plane's real axis will be accompanied by a complex conjugate zero at $z = z_2$. Likewise, for the FIR filter to have linear phase the z_2 zero must be accompanied by the z_1 zero. Of course, the z_1 and the z_3 zeros are complex conjugates of each other.

To conclude this section, we provide the following brief list of z -plane pole/zero properties that we should keep in mind as we work with digital filters:

- Filter poles are associated with increased frequency magnitude response (gain).
- Filter zeros are associated with decreased frequency magnitude response (attenuation).

- To be unconditionally stable all filter poles must reside inside the unit circle.
- Filter zeros do not affect filter stability.
- The closer a pole (zero) is to the unit circle, the stronger will be its effect on the filter's gain (attenuation) at the frequency associated with the pole's (zero's) angle.
- A pole (zero) located on the unit circle produces infinite filter gain (attenuation).
- If a pole is at the same z -plane location as a zero, they cancel each other.
- Poles or zeros located at the origin of the z -plane do not affect the frequency response of the filter.
- Filters whose transfer function denominator (numerator) polynomial has real-valued coefficients have poles (zeros) located on the real z -plane axis, or pairs of poles (zeros) that are complex conjugates of each other.
- For an FIR filter (transfer function denominator is unity) to have linear phase, any zero on the z -plane located at $z_0 = Me^{j\alpha}$, where z_0 is not on the unit circle and α is not zero, must be accompanied by a reciprocal zero whose location is $1/z_0 = e^{-j\alpha}/M$.
- What the last two bullets mean is that if an FIR filter has real-valued coefficients, is linear phase, and has a z -plane zero not located on the real z -plane axis or on the unit circle, that z -plane zero is a member of a “gang of four” zeros. If we know the z -plane location of one of those four zeros, then we know the location of the other three zeros.

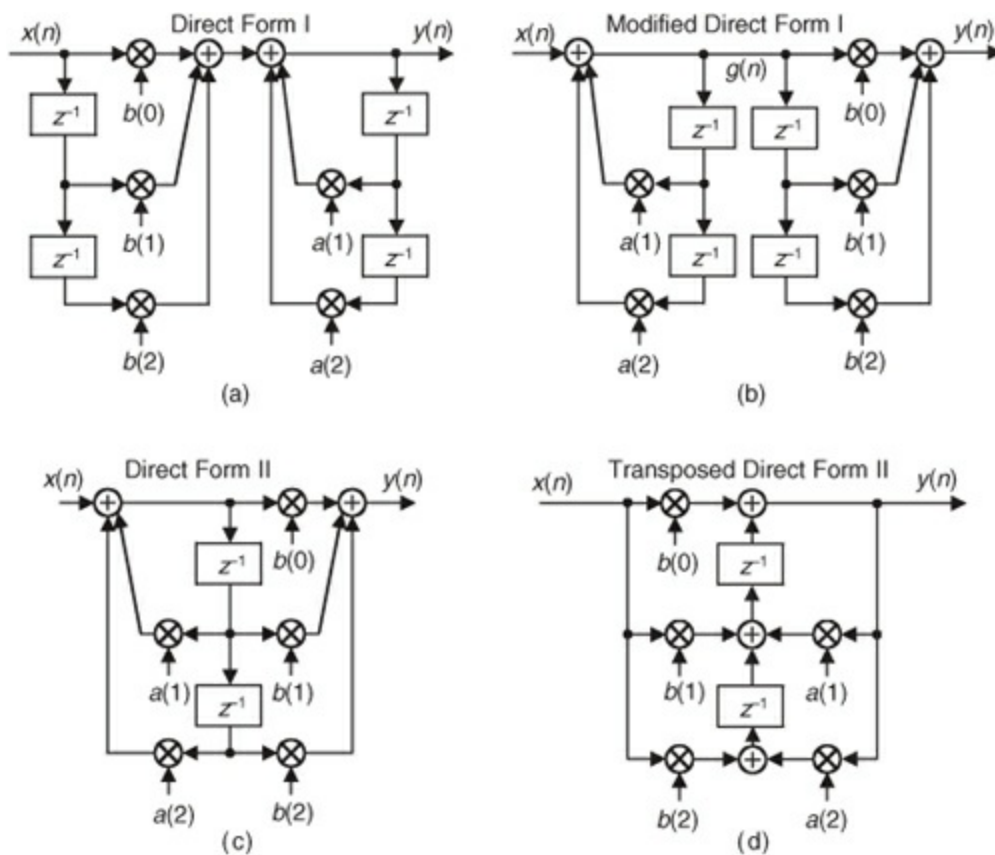
6.6 Alternate IIR Filter Structures

In the literature of DSP, it's likely that you will encounter IIR filters other than the Direct Form I structure of the IIR filter in [Figure 6-17](#). This point of our IIR filter studies is a good time to introduce those alternate IIR filter structures (block diagrams).

6.6.1 Direct Form I, Direct Form II, and Transposed Structures

The Direct Form I structure of the IIR filter in [Figure 6-17](#) can be converted to several alternate forms. It's easy to explore this idea by assuming that there are two equal-length delay lines, letting $M = N = 2$ as in [Figure 6-22\(a\)](#), and thinking of the feedforward and feedback portions as two separate filter stages. Because both stages of the filter are linear and time invariant, we can swap them, as shown in [Figure 6-22\(b\)](#), with no change in the $y(n)$ output.

Figure 6-22 Rearranged 2nd-order IIR filter structures: (a) Direct Form I; (b) modified Direct Form I; (c) Direct Form II; (d) transposed Direct Form II.



The two identical delay lines in [Figure 6-22\(b\)](#) provide the motivation for this reorientation. Because the sequence $g(n)$ is being shifted down along both delay lines in [Figure 6-22\(b\)](#), we can eliminate one of the delay paths and arrive at the simplified Direct Form II filter structure shown in [Figure 6-22\(c\)](#), where only half the delay storage registers are required compared to the Direct Form I structure.

Another popular IIR structure is the *transposed* form of the Direct Form II filter. We obtain a transposed form by starting with the Direct Form II filter, convert its signal nodes to adders, convert its adders to signal nodes, reverse the direction of its arrows, and swap $x(n)$ and $y(n)$. (The transposition steps can also be applied to FIR filters.) Following these steps yields the transposed Direct Form II structure given in [Figure 6-22\(d\)](#).

All the filters in [Figure 6-22](#) have the same performance just so long as infinite-precision arithmetic is used. However, using quantized binary arithmetic to represent our filter coefficients, and with truncation or rounding being used to combat binary overflow errors, the various filters in [Figure 6-22](#) exhibit different quantization noise and stability characteristics. In fact, the transposed Direct Form II structure was developed because it has improved behavior over the Direct Form II structure when fixed-point binary arithmetic is used. Common consensus among IIR filter designers is that the Direct Form I filter has the most resistance to coefficient quantization and stability problems. We'll revisit these finite-precision arithmetic issues in [Section 6.7](#).

By the way, because of the feedback nature of IIR filters, they're often referred to as *recursive* filters. Similarly, FIR filters are often called *nonrecursive* filters. A common misconception is that all *recursive* filters are IIR. This not true because FIR filters can be implemented with recursive structures. ([Chapters 7](#) and [10](#) discuss filters having feedback but whose impulse responses are finite in duration.) So, the terminology *recursive* and *nonrecursive* should be applied to a filter's structure, and the terms *IIR* and *FIR* should only be used to describe the time duration of the filter's impulse response[\[11,12\]](#).

6.6.2 The Transposition Theorem

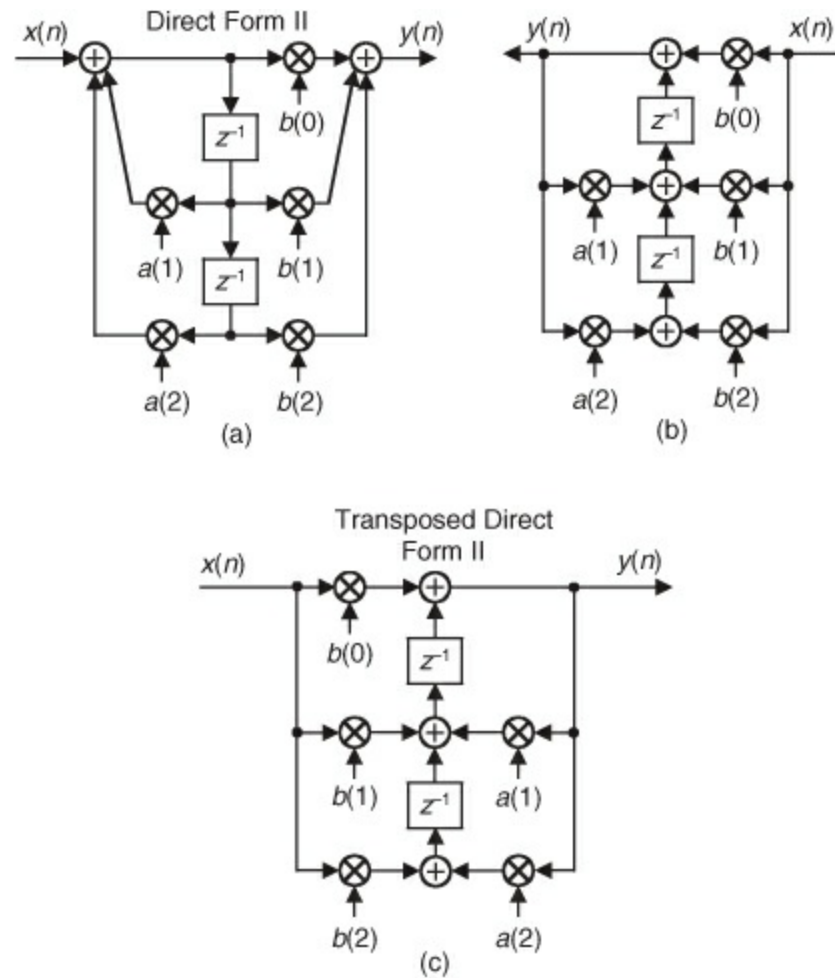
There is a process in DSP that allows us to change the structure (the block diagram implementation) of a linear time-invariant digital network without changing the network's transfer function (its frequency response). That network conversion process follows what is called the *transposition theorem*. That theorem is important because a transposed version of some digital network might be easier to implement, or may exhibit more accurate processing, than the original network.

We primarily think of the transposition theorem as it relates to digital filters, so below are the steps to transpose a digital filter (or any linear time-invariant network for that matter):

- 1. Reverse the direction of all signal-flow arrows.
- 2. Convert all adders to signal nodes.
- 3. Convert all signal nodes to adders.
- 4. Swap the $x(n)$ input and $y(n)$ output labels.

An example of this transposition process is shown in [Figure 6-23](#). The Direct Form II IIR filter in [Figure 6-23\(a\)](#) is transposed to the structure shown in [Figure 6-23\(b\)](#). By convention, we flip the network in [Figure 6-23\(b\)](#) from left to right so that the $x(n)$ input is on the left as shown in [Figure 6-23\(c\)](#).

Figure 6-23 Converting a Direct Form II filter to its transposed form.



Notice that the transposed filter contains the same number of delay elements, multipliers, and addition operations as the original filter, and both filters have the same transfer function given by

(6-44)

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b(0) + b(1)z^{-1} + b(2)z^{-2}}{1 - a(1)z^{-1} - a(2)z^{-2}}.$$

When implemented using infinite-precision arithmetic, the Direct Form II and the transposed Direct Form II filters have identical frequency response properties. As mentioned in [Section 6.6.1](#), however, the transposed Direct Form II structure is less susceptible to the errors that can occur when finite-precision binary arithmetic, for example, in a 16-bit processor, is used to represent data values and filter coefficients within a filter implementation. That property is because Direct Form II filters implement their (possibly high-gain) feedback pole computations before their feedforward zeros computations, and this can lead to problematic (large) intermediate data values which must be truncated. The transposed Direct Form II filters, on the other hand, implement their zeros computations first followed by their pole computations.

6.7 Pitfalls in Building IIR Filters

There's an old saying in engineering: "It's one thing to design a system on paper, and another thing to actually build one and make it work." (Recall the Tacoma Narrows Bridge episode!) Fabricating a working system based on theoretical designs can be difficult in practice. Let's see why this is often true for IIR digital filters.

Again, the IIR filter structures in [Figures 6-18](#) and [6-22](#) are called Direct Form implementations of an IIR filter. That's because they're all equivalent to directly implementing the general time-domain expression for an M th-order IIR filter given in [Eq. \(6-21\)](#). As it turns out, there can be stability problems and frequency response distortion errors when Direct Form implementations are used for high-order filters. Such problems arise because we're forced to represent the IIR filter coefficients and results of intermediate filter calculations with binary numbers having a finite number of bits. There are three major categories of finite-word-length errors that plague IIR filter implementations: coefficient quantization, overflow errors, and roundoff errors.

Coefficient quantization (limited-precision coefficients) will result in filter pole and zero shifting on the z -plane, and a frequency magnitude response that may not meet our requirements, and the response distortion worsens for higher-order IIR filters.

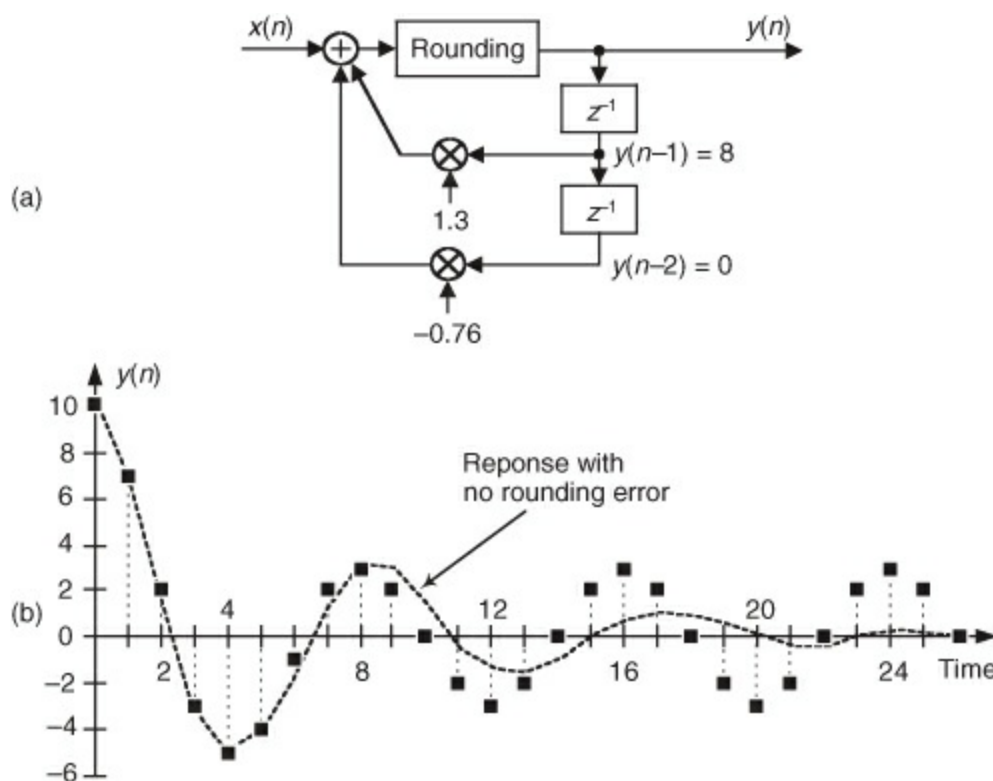
Overflow, the second finite-word-length effect that troubles IIR filters, is what happens when the result of an arithmetic operation is too large to be represented in the fixed-length hardware registers assigned to contain that result. Because we perform so many additions when we implement IIR filters, overflow is always a potential problem. With no precautions being taken to handle overflow, large nonlinearity errors can result in our filter output samples—often in the form of *overflow oscillations*.

The most common way of dealing with binary overflow errors is called *roundoff*, or rounding, where a data value is represented by, or rounded off to, the b -bit binary number that's nearest the unrounded data value. It's usually valid to treat roundoff errors as a random process, but conditions occur in IIR filters where rounding can cause the filter output to oscillate forever even when the filter input sequence is all zeros. This situation, caused by the roundoff noise being highly correlated with the signal, going by the names *limit cycles* and *deadband effects*, has been well analyzed in the literature[[13,14](#)]. We can demonstrate limit cycles by considering the 2nd-order IIR filter in [Figure 6-24\(a\)](#) whose time-domain expression is

(6-45)

$$y(n) = x(n) + 1.3y(n-1) - 0.76y(n-2).$$

Figure 6-24 Limit cycle oscillations due to rounding: (a) 2nd-order IIR filter; (b) one possible time-domain response of the IIR filter.



Let's assume this filter rounds the adder's output to the nearest integer value. If the situation ever arises where $y[-2] = 0$, $y[-1] = 8$, and $x[0]$ and all successive $x[n]$ inputs are zero, the filter output goes into endless oscillation as shown in [Figure 6-24\(b\)](#). If this filter were to be used in an audio application, when the input signal went silent the listener could end up hearing an audio tone instead of silence. The dashed line in [Figure 6-24\(b\)](#) shows the filter's stable response to this particular situation if no rounding is used. With rounding, however, this IIR filter certainly lives up to its name. (Welcome to the world of binary arithmetic!)

There are several ways to reduce the ill effects of coefficient quantization errors and limit cycles. We can increase the word widths of the hardware registers that contain the results of intermediate calculations. Because roundoff limit cycles affect the least significant bits of an arithmetic result, larger word sizes diminish the impact of limit cycles should they occur. To avoid filter input sequences of all zeros, some practitioners add a *dither sequence* to the filter's input signal sequence. A dither sequence is a sequence of low-amplitude pseudo-random numbers that interferes with an IIR filter's roundoff error generation tendency, allowing the filter output to reach zero should the input signal remain at zero. Dithering, however, decreases the effective signal-to-noise ratio of the filter output[12]. Finally, to avoid limit cycle problems, we can just use an FIR filter. Because FIR filters by definition have finite-length impulse responses, and have no feedback paths, they cannot support output oscillations of any kind.

As for overflow errors, we can eliminate them if we increase the word width of hardware registers so overflow never takes place in the IIR filter. Filter input signals can be scaled (reduced in amplitude by multiplying signals within the filter by a factor less than one) so overflow is avoided. We discuss such filter scaling in [Section 6.9](#). Overflow oscillations can be avoided by using saturation arithmetic logic where signal values aren't permitted to exceed a fixed limit when an overflow condition is detected[15,16]. It may be useful for the reader to keep in mind that when the signal data is represented in two's complement arithmetic, multiple summations resulting in

intermediate overflow errors cause no problems if we can guarantee that the final magnitude of the sum of the numbers is not too large for the final accumulator register. Of course, standard floating-point number formats can greatly reduce the errors associated with overflow oscillations and limit cycles[17]. (We discuss floating-point number formats in [Chapter 12](#).)

These quantized coefficient and overflow errors, caused by finite-width words, have different effects depending on the IIR filter structure used. Referring to [Figure 6-22](#), practice has shown the Direct Form II structure to be the most error-prone IIR filter implementation.

The most popular technique for minimizing the errors associated with finite-word-length widths is to design IIR filters comprising a cascade string, or parallel combination, of low-order filters. The next section tells us why.

6.8 Improving IIR Filters with Cascaded Structures

Filter designers minimize IIR filter stability and quantization noise problems in high-performance filters by implementing combinations of cascaded lower-performance filters. Before we consider this design idea, let's review several important issues regarding the behavior of combinations of multiple filters.

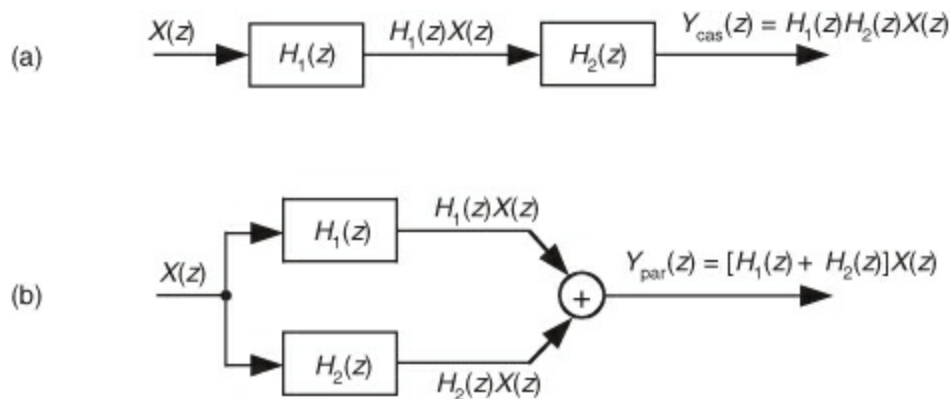
6.8.1 Cascade and Parallel Filter Properties

Here we summarize the *combined* behavior of linear time-invariant filters (be they IIR or FIR) connected in cascade and in parallel. As indicated in [Figure 6-25\(a\)](#), the resultant transfer function of two cascaded filter transfer functions is the product of those functions, or

(6-46)

$$H_{\text{cas}}(z) = H_1(z)H_2(z)$$

Figure 6-25 Combinations of two filters: (a) cascaded filters; (b) parallel filters.



with an overall frequency response of

(6-47)

$$H_{\text{cas}}(\omega) = H_1(\omega)H_2(\omega).$$

It's also important to know that the resultant impulse response of cascaded filters is

(6-48)

$$h_{\text{cas}}(k) = h_1(k)*h_2(k),$$

where “*” means convolution.

As shown in [Figure 6-25\(b\)](#), the combined transfer function of two filters connected in parallel is the

sum of their transfer functions, or

(6-49)

$$H_{\text{par}}(z) = H_1(z) + H_2(z)$$

with an overall frequency response of

(6-50)

$$H_{\text{par}}(\omega) = H_1(\omega) + H_2(\omega).$$

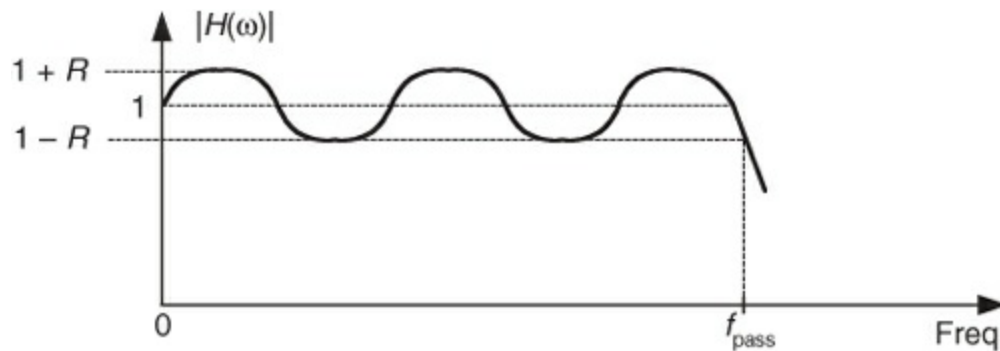
The resultant impulse response of parallel filters is the sum of their individual impulse responses, or

(6-51)

$$h_{\text{par}}(k) = h_1(k) + h_2(k).$$

While we are on the subject of cascaded filters, let's develop a rule of thumb for estimating the combined passband ripple of the two cascaded filters in [Figure 6-25\(a\)](#). The cascaded passband ripple is a function of each individual filter's passband ripple. If we represent an arbitrary filter's peak passband ripple on a linear (not dB) vertical axis as shown in [Figure 6-26](#), we can begin our cascaded ripple estimation.

Figure 6-26 Definition of filter passband ripple R .



From [Eq. \(6-47\)](#), the upper bound (the peak) of a cascaded filter's passband response, $1 + R_{\text{cas}}$, is the product of the two $H_1(\omega)$ and $H_2(\omega)$ filters' peak passband responses, or

(6-52)

$$1 + R_{\text{cas}} = (1 + R_1)(1 + R_2) = 1 + R_1 + R_2 + R_1R_2.$$

For small values of R_1 and R_2 , the R_1R_2 term becomes negligible, and we state our rule of thumb as

(6-53)

$$R_{\text{cas}} \approx R_1 + R_2.$$

Thus, in designs using two cascaded filters it's prudent to specify their individual passband ripple values to be roughly half the desired R_{cas} ripple specification for the final combined filter, or

(6-54)

$$R_1 = R_2 \approx R_{\text{cascaded}}/2.$$

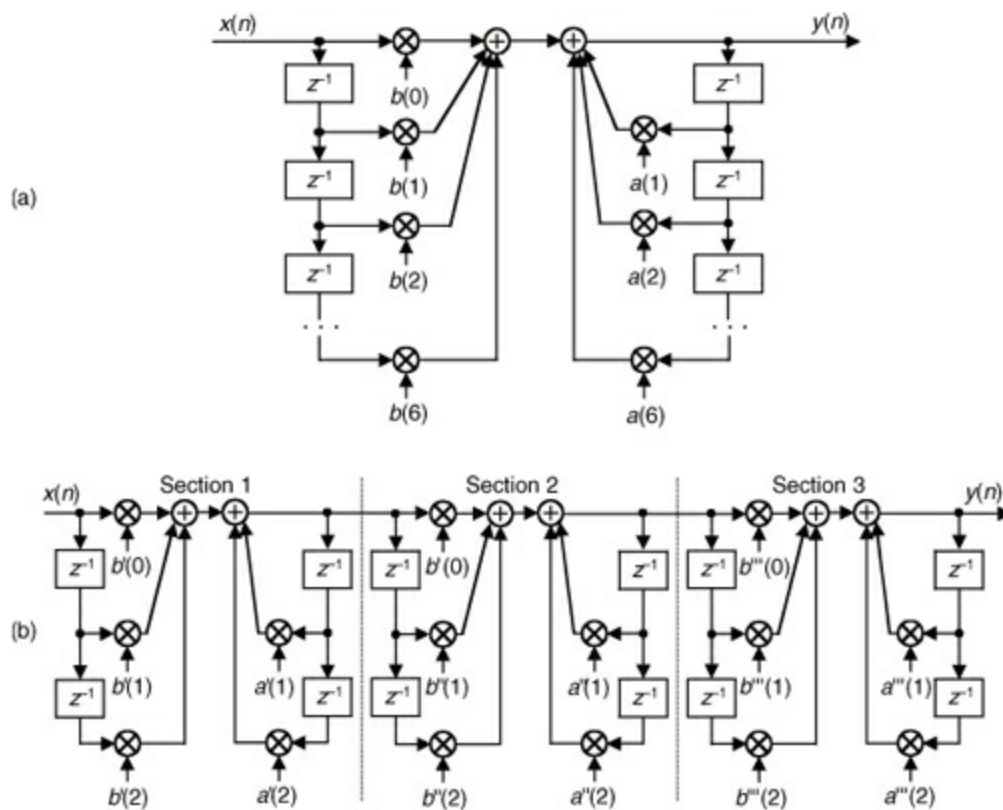
6.8.2 Cascading IIR Filters

Experienced filter designers routinely partition high-order IIR filters into a string of 2nd-order IIR filters arranged in cascade because these lower-order filters are easier to design, are less susceptible

to coefficient quantization errors and stability problems, and their implementations allow easier data word scaling to reduce the potential overflow effects of data word size growth.

Optimizing the partitioning of a high-order filter into multiple 2nd-order filter sections is a challenging task, however. For example, say we have the 6th-order Direct Form I filter in [Figure 6-27\(a\)](#) that we want to partition into three 2nd-order sections. In factoring the 6th-order filter's $H(z)$ transfer function, we could get up to three separate sets of feedforward coefficients in the factored $H(z)$ numerator: $b'(k)$, $b''(k)$, and $b'''(k)$. Likewise, we could have up to three separate sets of feedback coefficients in the factored denominator: $a'(k)$, $a''(k)$, and $a'''(k)$. Because there are three 2nd-order sections, there are three factorial, or six, ways of *pairing* the sets of coefficients. Notice in [Figure 6-27\(b\)](#) how the first section uses the $a'(k)$ and $b'(k)$ coefficients, and the second section uses the $a''(k)$ and $b''(k)$ coefficients. We could just as well have interchanged the sets of coefficients so the first 2nd-order section uses the $a'(k)$ and $b''(k)$ coefficients, and the second section uses the $a''(k)$ and $b'(k)$ coefficients. So, there are six different mathematically equivalent ways of combining the sets of coefficients. Add to this the fact that for each different combination of low-order sections there are three factorial distinct ways those three separate 2nd-order sections can be arranged in cascade.

Figure 6-27 IIR filter partitioning: (a) initial 6th-order IIR filter; (b) three 2nd-order sections.



This means if we want to partition a $2M$ -order IIR filter into M distinct 2nd-order sections, there are M factorial squared, $(M!)^2$, ways to do so. As such, there are then $(3!)^2 = 24$ different cascaded filters we could obtain when going from [Figure 6-27\(a\)](#) to [Figure 6-27\(b\)](#). To further complicate this filter partitioning problem, the errors due to coefficient quantization will, in general, be different for each possible filter combination. Although full details of this subject are outside the scope of this introductory text, ambitious readers can find further material on optimizing cascaded filter sections in references [\[14\]](#) and [\[18\]](#), and in Part 3 of reference [\[19\]](#).

One simple (although perhaps not optimum) method for arranging cascaded 2nd-order sections has been proposed [\[18\]](#). First, factor a high-order IIR filter's $H(z)$ transfer function into a ratio of the form

$$H(z) = \frac{(z - z_0)(z - z_1)(z - z_2)(z - z_3)(z - z_4)(z - z_5) \dots}{(z - p_0)(z - p_1)(z - p_2)(z - p_3)(z - p_4)(z - p_5) \dots}$$

with the z_k zeros in the numerator and p_k poles in the denominator. (Ideally you have a signal processing software package to perform the factorization.) Next, the 2nd-order section assignments go like this:

1. Find the pole, or pole pair, in $H(z)$ closest to the unit circle.
2. Find the zero, or zero pair, closest to the pole, or pole pair, found in Step 1.
3. Combine those poles and zeros into a single 2nd-order filter section. This means your first 2nd-order section may be something like

(6-56)

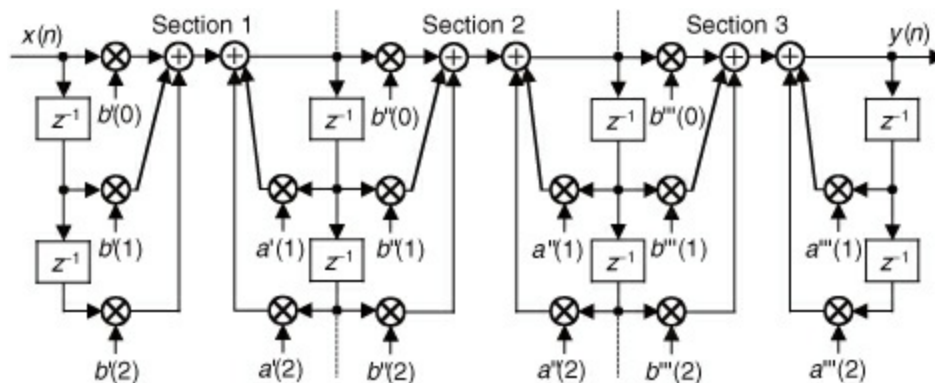
$$H_1(z) = \frac{(z - z_4)(z - z_5)}{(z - p_0)(z - p_1)}$$

4. Repeat Steps 1 to 3 until all poles and zeros have been combined into 2nd-order sections.
5. The final ordering (cascaded sequence) of the sections is based on how far the sections' poles are from the unit circle. Order the sections in either increasing or decreasing pole distances from the unit circle.
6. Implement your filter as cascaded 2nd-order sections in the order from Step 5.

In digital filter vernacular, a 2nd-order IIR filter is called a *biquad* for two reasons. First, the filter's z -domain transfer function includes two quadratic polynomials. Second, the word *biquad* sounds cool.

By the way, we started our 2nd-order sectioning discussion with a high-order Direct Form I filter in [Figure 6-27\(a\)](#). We chose that filter form because it's the structure most resistant to coefficient quantization and overflow problems. As seen in [Figure 6-27\(a\)](#), we have redundant delay elements. These can be combined, as shown in [Figure 6-28](#), to reduce our temporary storage requirements as we did with the Direct Form II structure in [Figure 6-22](#).

Figure 6-28 Cascaded Direct Form I filters with reduced temporary data storage.



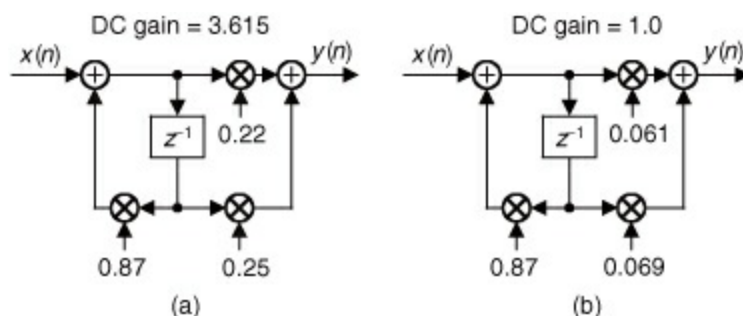
There's much material in the literature concerning finite word effects as they relate to digital IIR filters. (References [\[18\]](#), [\[20\]](#), and [\[21\]](#) discuss quantization noise effects in some detail as well as providing extensive bibliographies on the subject.)

6.9 Scaling the Gain of IIR Filters

In order to impose limits on the magnitudes of data values within an IIR filter, we may wish to change the passband gain of that filter[22,23].

For example, consider the 1st-order lowpass IIR filter in [Figure 6-29\(a\)](#) that has a DC gain (gain at zero Hz) of 3.615. (This means that, just as with FIR filters, the sum of the IIR filter's impulse response samples is equal to the DC gain of 3.615.)

Figure 6-29 Lowpass IIR filters: (a) DC gain = 3.615; (b) DC gain = 1.



The DC gain of an IIR filter is the sum of the filter's feedforward coefficients divided by 1 minus the sum of the filter's feedback coefficients. (We leave the proof of that statement as a homework problem.) That is, the DC gain of the [Figure 6-29\(a\)](#) 1st-order filter is

(6-57)

$$\text{DC gain} = \frac{b(0) + b(1)}{1 - a(1)} = \frac{0.22 + 0.25}{1 - 0.87} = 3.615.$$

Now let's say we want, for some reason, the filter's DC gain to be one (unity gain). This is easy to accomplish. We merely divide the filter's feedforward coefficients by the original DC gain as

(6-58)

$$b_{\text{new}}(0) = \frac{0.22}{3.615} = 0.061, \text{ and } b_{\text{new}}(1) = \frac{0.25}{3.615} = 0.069.$$

Doing so gives us a new filter whose feedforward coefficients are those shown in [Figure 6-29\(b\)](#). That new lowpass filter has a DC gain of one. Changing a filter's coefficients in this way is called *filter scaling*. Happily, this filter scaling does not change the shape of the original filter's frequency magnitude or phase response.

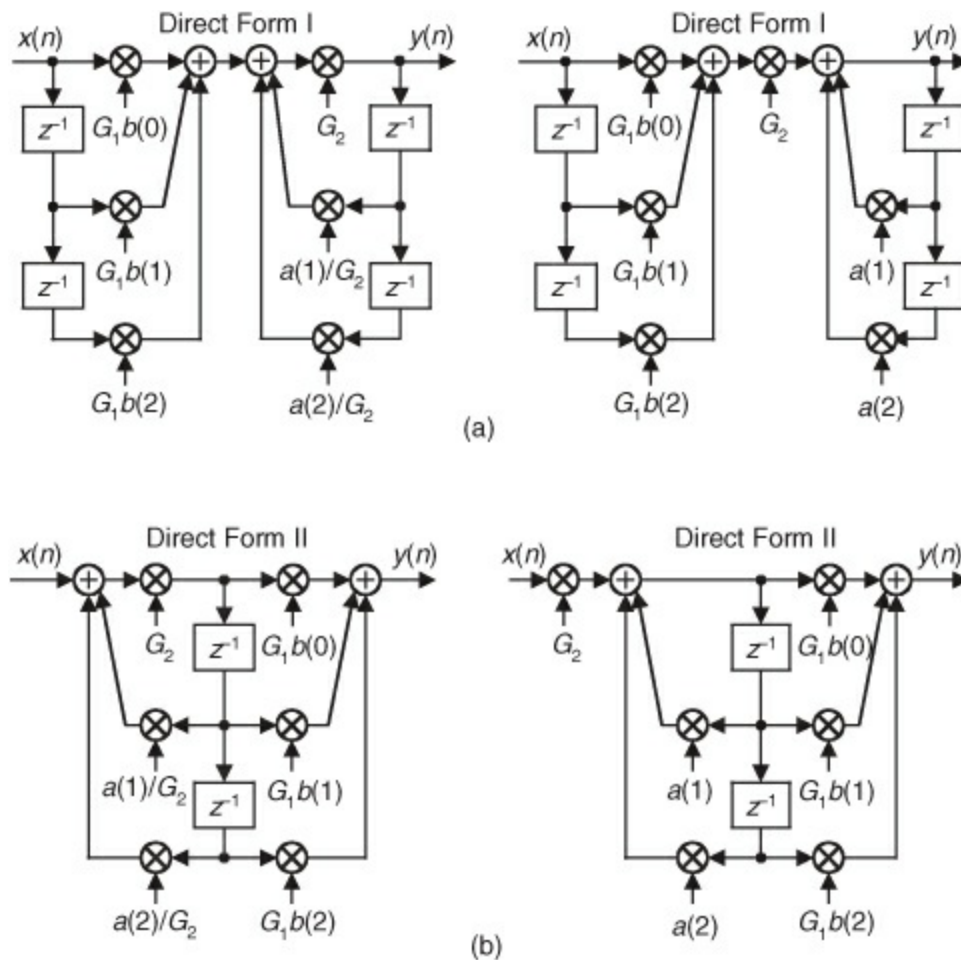
Likewise, to force the passband gain of a highpass filter to be unity, we divide the filter's feedforward coefficients by the original filter's frequency magnitude response at $f_s/2$ (half the sampling rate).

Unlike passive analog (continuous-time) filters that operate by attenuating spectral energy in their stopbands, digital IIR filters operate by amplifying spectral energy in their passbands. Because of this positive passband gain behavior, there is another type of IIR filter scaling that's used in many situations. It is possible that an IIR filter may have a passband gain so high that the filter generates internal sample values too large to be accommodated by the hardware, with its internal binary number format, used to implement the filter. Stated in different words, it's possible for a filter to generate internal data values so large that they overflow the registers in which the data is to be stored. This situation can also occur when multiple 2nd-order IIR filters are cascaded as discussed in [Section 6.8](#).

In such cases, should we wish to reduce the passband gain of an IIR filter without changing the shape

of its frequency magnitude or phase responses, we can do so by implementing one of the filters shown in [Figure 6-30](#).

Figure 6-30 Scaled IIR filter structures: (a) Direct Form I; (b) Direct Form II.



If an IIR filter has an original passband gain of G_{IIR} , we can change that passband gain by modifying the original filter's coefficients using the scalar G_1 and G_2 gain factors shown in [Figure 6-30](#). Changing a filter's coefficients in this way is also called *filter scaling*. The passband gain of a scaled filter is

$$(6-59)$$

$$G_{\text{IIR-scaled}} = G_1 G_2 G_{\text{IIR}}.$$

The general philosophy in these matters is to choose factors G_1 and G_2 so that we preserve the filter's output signal quality (called the *signal-to-noise ratio*, SNR, as discussed in [Chapter 12](#) and [Appendix D](#)) as much as possible. This means keeping all internal sample values as large as can be accommodated by the filter hardware registers. The problem is, there's no simple way to determine the values of G_1 and G_2 . The suggested procedure is to select one of the [Figure 6-30](#) implementations and apply the expected input signal to the filter. Next we experiment with different values for gain factors G_1 and G_2 from [Eq. \(6-59\)](#) until the final filter gain, $G_{\text{IIR-scaled}}$, is an acceptable value. Following that, we select an alternate [Figure 6-30](#) filter structure and experiment with different values for gains G_1 and G_2 to see if we can improve on the previous scaled-filter structure.

For computational efficiency reasons, if we're able to set G_2 to be the reciprocal of an integer power of two, then we can eliminate one of the multiplies in [Figure 6-30](#). That is, in this scenario the multiply by G_2 operation can then be implemented with binary right shifts. Then again, perhaps

factors G_1 and G_2 can be chosen so that one of the modified filter coefficients is unity in order to eliminate a multiply operation.

Now that we have some understanding of the performance and implementation structures of IIR filters, let's briefly introduce three filter design techniques. These IIR design methods go by the impressive names of *impulse invariance*, *bilinear transform*, and *optimized* methods. The first two methods use *analytical*, pencil and paper algebra, filter design techniques to approximate continuous analog filters. (By "analog filters" we mean those hardware filters made up of resistors, capacitors, and perhaps operational amplifiers.)

Because analog filter design methods are very well understood, designers can take advantage of an abundant variety of analog filter design techniques to design, say, a digital IIR Butterworth filter with its very flat passband response, or perhaps go with a Chebyshev filter with its fluctuating passband response and sharper passband-to-stopband cutoff characteristics. The *optimized methods* (by far the most popular way of designing IIR filters) comprise linear algebra algorithms available in commercial filter design software packages.

The impulse invariance, bilinear transform filter design methods are somewhat involved, so a true DSP beginner is justified in skipping those subjects upon first reading this book. However, those filter design topics may well be valuable sometime in your future as your DSP knowledge, experience, and challenges grow.

6.10 Impulse Invariance IIR Filter Design Method

The impulse invariance method of IIR filter design is based upon the notion that we can design a discrete filter whose time-domain impulse response is a sampled version of the impulse response of a continuous analog filter. If that analog filter (often called the *prototype filter*) has some desired frequency response, then our IIR filter will yield a discrete approximation of that desired response. The impulse response equivalence of this design method is depicted in [Figure 6-31](#), where we use the conventional notation of δ to represent an impulse function and $h_c(t)$ is the analog filter's impulse response. We use the subscript "c" in [Figure 6-31\(a\)](#) to emphasize the continuous nature of the analog filter. [Figure 6-31\(b\)](#) illustrates the definition of the discrete filter's impulse response: the filter's time-domain output sequence when the input is a single unity-valued sample (impulse) preceded and followed by all zero-valued samples. Our goal is to design a digital filter whose impulse response is a sampled version of the analog filter's continuous impulse response. Implied in the correspondence of the continuous and discrete impulse responses is the property that we can map each pole on the s -plane for the analog filter's $H_c(s)$ transfer function to a pole on the z -plane for the discrete IIR filter's $H(z)$ transfer function. What designers have found is that the impulse invariance method does yield useful IIR filters, as long as the sampling rate is high relative to the bandwidth of the signal to be filtered. In other words, IIR filters designed using the impulse invariance method are susceptible to aliasing problems because practical analog filters cannot be perfectly band-limited. Aliasing will occur in an IIR filter's frequency response as shown in [Figure 6-32](#).

Figure 6-31 Impulse invariance design equivalence of (a) analog filter continuous impulse response; (b) digital filter discrete impulse response.

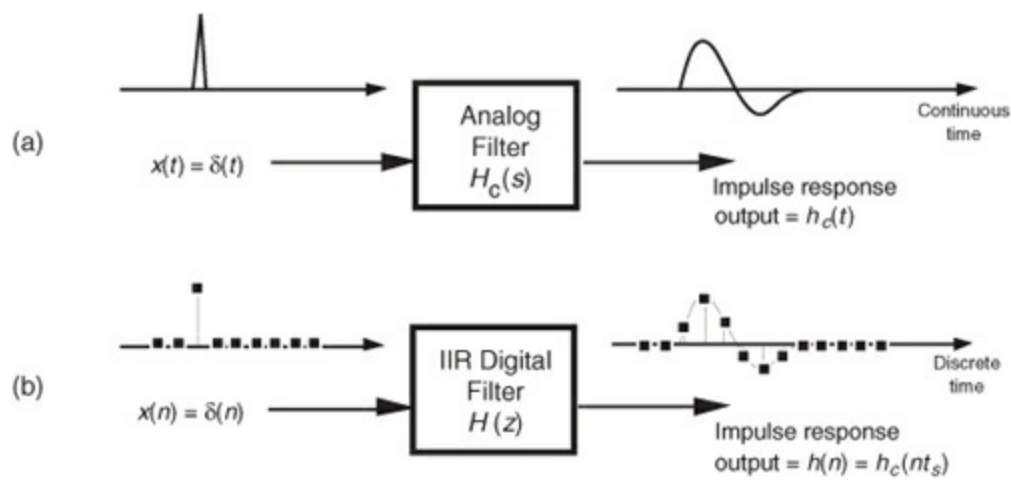
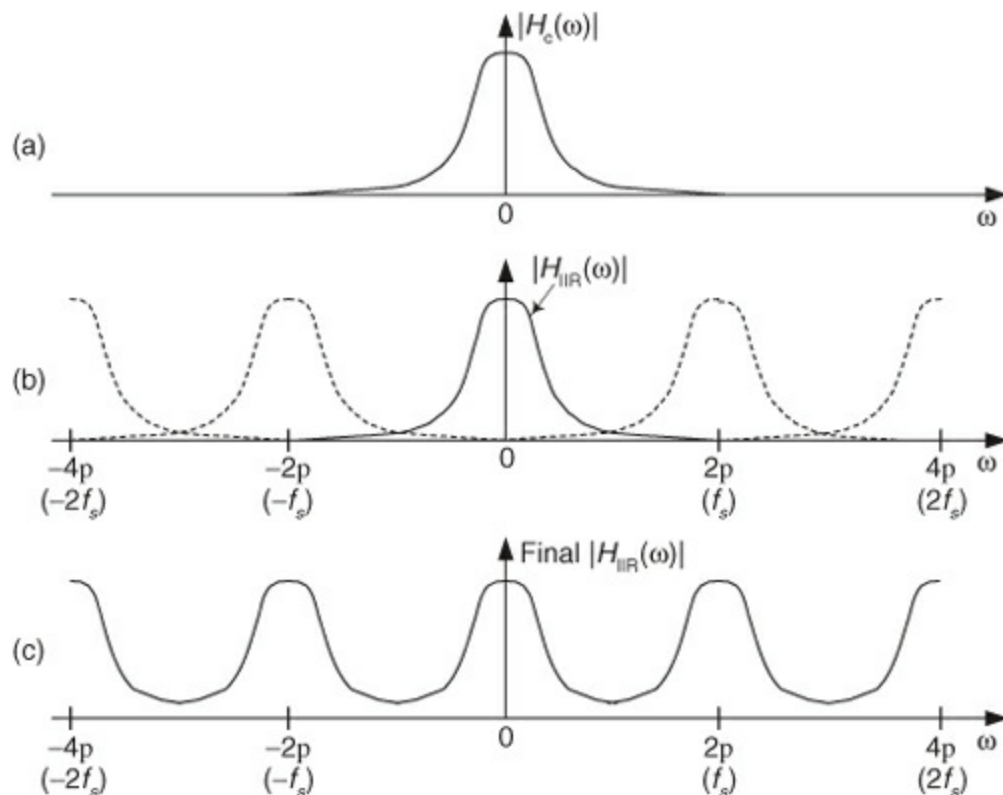


Figure 6-32 Aliasing in the impulse invariance design method: (a) prototype analog filter magnitude response; (b) replicated magnitude responses where $H_{\text{IIR}}(\omega)$ is the discrete Fourier transform of $h(n) = h_c(nt_s)$; (c) potential resultant IIR filter magnitude response with aliasing effects.



From what we've learned in [Chapter 2](#) about the spectral replicating effects of sampling, if [Figure 6-32\(a\)](#) is the spectrum of the continuous $h_c(t)$ impulse response, then the spectrum of the discrete $h_c(nt_s)$ sample sequence is the replicated spectra in [Figure 6-32\(b\)](#).

In [Figure 6-32\(c\)](#) we show the possible effect of aliasing where the dashed curve is a desired $H_{\text{IIR}}(\omega)$ frequency magnitude response. However, the actual frequency magnitude response, indicated by the solid curve, can occur when we use the impulse invariance design method. For this reason, we prefer to make the sample frequency f_s as large as possible to minimize the overlap between the primary frequency response curve and its replicated images spaced at multiples of $\pm f_s$ Hz.

Due to the aliasing behavior of the impulse invariance design method, this filter design process should never be used to design highpass digital filters. To see how aliasing can affect IIR filters designed with this method, let's list the necessary impulse invariance design steps and then go through a lowpass filter design example.

There are two different methods for designing IIR filters using impulse invariance. The first method, which we'll call Method 1, requires that an inverse Laplace transform as well as a z-transform be performed[24,25]. The second impulse invariance design technique, Method 2, uses a direct substitution process to avoid the inverse Laplace and z-transformations at the expense of needing partial fraction expansion algebra necessary to handle polynomials[20,21,26,27]. Both of these methods seem complicated when described in words, but they're really not as difficult as they sound. Let's compare the two methods by listing the steps required for each of them. The impulse invariance design Method 1 goes like this:

Method 1, Step 1: Design (or have someone design for you) a prototype analog filter with the desired frequency response.[‡] The result of this step is a continuous Laplace transfer function $H_c(s)$ expressed as the ratio of two polynomials, such as

[‡] In a lowpass filter design, for example, the filter type (Chebyshev, Butterworth, elliptic), filter order (number of poles), and the cutoff frequency are parameters to be defined in this step.

(6-60)

$$H_c(s) = \frac{b(N)s^N + b(N-1)s^{N-1} + \dots + b(1)s + b(0)}{a(M)s^M + a(M-1)s^{M-1} + \dots + a(1)s + a(0)} = \frac{\sum_{k=0}^N b(k)s^k}{\sum_{k=0}^M a(k)s^k},$$

which is the general form of [Eq. \(6-10\)](#) with $N < M$, and $a(k)$ and $b(k)$ are constants.

Method 1, Step 2: Determine the analog filter's continuous time-domain impulse response $h_c(t)$ from the $H_c(s)$ Laplace transfer function. I hope this can be done using Laplace tables as opposed to actually evaluating an inverse Laplace transform equation.

Method 1, Step 3: Determine the digital filter's sampling frequency f_s , and calculate the sample period as $t_s = 1/f_s$. The f_s sampling rate is chosen based on the absolute frequency, in Hz, of the prototype analog filter. Because of the aliasing problems associated with this impulse invariance design method, later, we'll see why f_s should be made as large as is practical.

Method 1, Step 4: Find the z-transform of the continuous $h_c(t)$ to obtain the IIR filter's z-domain transfer function $H(z)$ in the form of a ratio of polynomials in z .

Method 1, Step 5: Substitute the value (not the variable) t_s for the continuous variable t in the $H(z)$ transfer function obtained in Step 4. In performing this step, we are ensuring, as in [Figure 6-31](#), that the IIR filter's discrete $h(n)$ impulse response is a sampled version of the continuous filter's $h_c(t)$ impulse response so that $h(n) = h_c(nt_s)$, for $0 \leq n \leq \infty$.

Method 1, Step 6: Our $H(z)$ from Step 5 will now be of the general form

(6-61)

$$H(z) = \frac{b(N)z^{-N} + b(N-1)z^{-(N-1)} + \dots + b(1)z^{-1} + b(0)}{a(M)z^{-M} + a(M-1)z^{-(M-1)} + \dots + a(1)z^{-1} + a(0)} = \frac{\sum_{k=0}^N b(k)z^{-k}}{1 - \sum_{k=1}^M a(k)z^{-k}}.$$

Because the process of sampling the continuous impulse response results in a digital filter frequency response that's scaled by a factor of $1/t_s$, many filter designers find it appropriate to include the t_s factor in [Eq. \(6-61\)](#). So we can rewrite [Eq. \(6-61\)](#) as

(6-62)

$$H(z) = \frac{Y(z)}{X(z)} = \frac{t_s \sum_{k=0}^N b(k)z^{-k}}{1 - \sum_{k=1}^M a(k)z^{-k}}.$$

Incorporating the value of t_s in [Eq. \(6-62\)](#), then, makes the IIR filter time-response scaling independent of the sampling rate, and the discrete filter will have the same gain as the prototype analog filter.[‡]

[‡] Some authors have chosen to include the t_s factor in the discrete $h(n)$ impulse response in the above Step 4, that is, make $h(n) = t_s h_c(nt_s)$ [20, 28]. The final result of this, of course, is the same as that obtained by including t_s as described in Step 6.

Method 1, Step 7: Because [Eq. \(6-61\)](#) is in the form of [Eq. \(6-25\)](#), by inspection, we can express the filter's time-domain difference equation in the general form of [Eq. \(6-21\)](#) as

(6-63)

$$y(n) = b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + \dots + b(N)x(n-N) \\ + a(1)y(n-1) + a(2)y(n-2) + \dots + a(M)y(n-M).$$

Choosing to incorporate t_s , as in [Eq. \(6-62\)](#), to make the digital filter's gain equal to the prototype analog filter's gain by multiplying the $b(k)$ coefficients by the sample period t_s leads to an IIR filter time-domain expression of the form

(6-64)

$$y(n) = t_s \cdot [b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + \dots + b(N)x(n-N)] \\ + a(1)y(n-1) + a(2)y(n-2) + \dots + a(M)y(n-M).$$

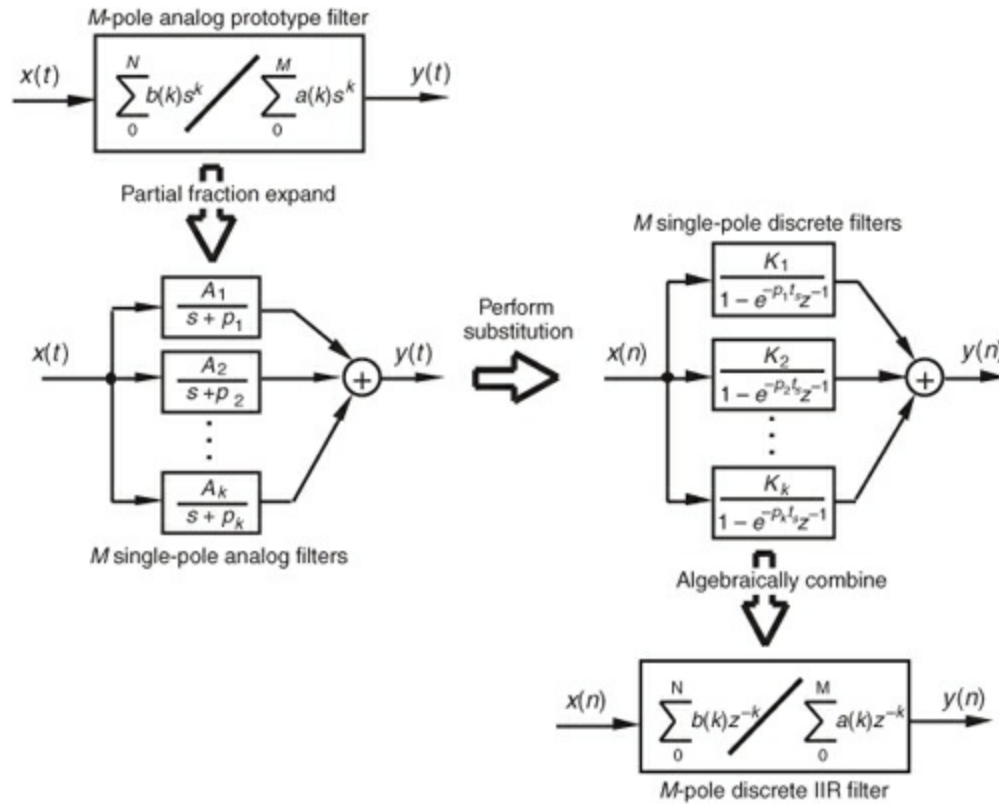
Notice how the signs changed for the $a(k)$ coefficients from [Eqs. \(6-61\)](#) and [\(6-62\)](#) to [Eqs. \(6-63\)](#) and [\(6-64\)](#). These sign changes always seem to cause problems for beginners, so watch out. Also, keep in mind that the time-domain expressions in [Eqs. \(6-63\)](#) and [\(6-64\)](#) apply only to the filter structure in [Figure 6-18](#). The $a(k)$ and $b(k)$, or $t_s \cdot b(k)$, coefficients, however, can be applied to the improved IIR structure shown in [Figure 6-22](#) to complete our design.

Before we go through an actual example of this design process, let's discuss the other impulse invariance design method.

The impulse invariance Design Method 2, also called the standard z -transform method, takes a different approach. It mathematically partitions the prototype analog filter into multiple single-pole

continuous filters and then approximates each one of those by a single-pole digital filter. The set of M single-pole digital filters is then algebraically combined to form an M -pole, M th-ordered IIR filter. This process of breaking the analog filter to discrete filter approximation into manageable pieces is shown in [Figure 6-33](#). The steps necessary to perform an impulse invariance Method 2 design are:

Figure 6-33 Mathematical flow of the impulse invariance design Method 2.



Method 2, Step 1: Obtain the Laplace transfer function $H_c(s)$ for the prototype analog filter in the form of [Eq. \(6-60\)](#). (Same as Method 1, Step 1.)

Method 2, Step 2: Select an appropriate sampling frequency f_s and calculate the sample period as $t_s = 1/f_s$. (Same as Method 1, Step 3.)

Method 2, Step 3: Express the analog filter's Laplace transfer function $H_c(s)$ as the sum of single-pole filters. This requires us to use partial fraction expansion methods to express the ratio of polynomials in [Eq. \(6-60\)](#) in the form of

(6-65)

$$H_c(s) = \frac{b(N)s^N + b(N-1)s^{N-1} + \dots + b(1)s + b(0)}{a(M)s^M + a(M-1)s^{M-1} + \dots + a(1)s + a(0)}$$

$$= \sum_{k=1}^M \frac{A_k}{s + p_k} = \frac{A_1}{s + p_1} + \frac{A_2}{s + p_2} + \dots + \frac{A_M}{s + p_M}$$

where $M > N$, the individual A_k factors are constants, and the k th pole is located at $-p_k$ on the s -plane. We'll denote the k th single-pole analog filter as $H_k(s)$, or

(6-66)

$$H_k(s) = \frac{A_k}{s + p_k}.$$

Method 2, Step 4: Substitute $1 - e^{-p_k t_s} z^{-1}$ for $s + p_k$ in [Eq. \(6-65\)](#). This mapping of each $H_k(s)$ pole, located at $s = -p_k$ on the s -plane, to the $z = e^{-p_k t_s}$ location on the z -plane is how we approximate the impulse response of each single-pole analog filter by a single-pole digital filter. (The reader can find the derivation of this $1 - e^{-p_k t_s} z^{-1}$ substitution, illustrated in our [Figure 6-33](#), in references [\[20\]](#), [\[21\]](#), and [\[26\]](#).) So, the k th analog single-pole filter $H_k(s)$ is approximated by a single-pole digital filter whose z -domain transfer function is

(6-67)

$$H_k(z) = \frac{A_k}{1 - e^{-p_k t_s} z^{-1}}.$$

The final combined discrete filter transfer function $H(z)$ is the sum of the single-poled discrete filters, or

(6-68)

$$H(z) = \sum_{k=1}^M H_k(z) = \sum_{k=1}^M \frac{A_k}{1 - e^{-p_k t_s} z^{-1}}.$$

Keep in mind that the above $H(z)$ is not a function of time. The t_s factor in [Eq. \(6-68\)](#) is a constant equal to the discrete-time sample period.

Method 2, Step 5: Calculate the z -domain transfer function of the sum of the M single-pole digital filters in the form of a ratio of two polynomials in z . Because the $H(z)$ in [Eq. \(6-68\)](#) will be a series of fractions, we'll have to combine those fractions over a common denominator to get a single ratio of polynomials in the familiar form of

(6-69)

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^N b(k)z^{-k}}{1 - \sum_{k=1}^M a(k)z^{-k}}.$$

Method 2, Step 6: Just as in Method 1, Step 6, by inspection, we can express the filter's time-domain equation in the general form of

(6-70)

$$y(n) = b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + \dots + b(N)x(n-N) \\ + a(1)y(n-1) + a(2)y(n-2) + \dots + a(M)y(n-M).$$

Again, notice the $a(k)$ coefficient sign changes from [Eq. \(6-69\)](#) to [Eq. \(6-70\)](#). As described in Method 1, Steps 6 and 7, if we choose to make the digital filter's gain equal to the prototype analog filter's gain by multiplying the $b(k)$ coefficients by the sample period t_s , then the IIR filter's time-domain expression will be in the form

(6-71)

$$y(n) = t_s \cdot [b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + \dots + b(N)x(n-N)] \\ + a(1)y(n-1) + a(2)y(n-2) + \dots + a(M)y(n-M),$$

yielding a final $H(z)$ z-domain transfer function of

(6-71')

$$H(z) = \frac{Y(z)}{X(z)} = \frac{t_s \cdot \sum_{k=0}^N b(k)z^{-k}}{1 - \sum_{k=1}^M a(k)z^{-k}}.$$

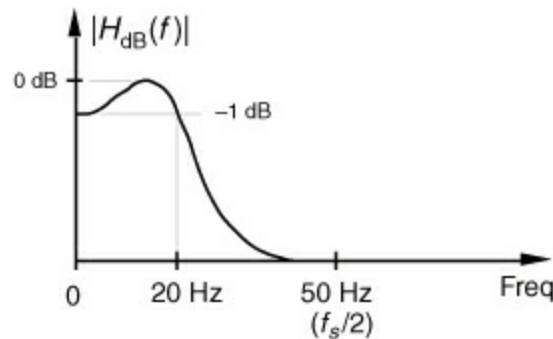
Finally, we can implement the improved IIR structure shown in [Figure 6-22](#) using the $a(k)$ and $b(k)$ coefficients from [Eq. \(6-70\)](#) or the $a(k)$ and $t_s \cdot b(k)$ coefficients from [Eq. \(6-71\)](#).

To provide a more meaningful comparison between the two impulse invariance design methods, let's dive in and go through an IIR filter design example using both methods.

6.10.1 Impulse Invariance Design Method 1 Example

Assume that we need to design an IIR filter that approximates a 2nd-order Chebyshev prototype analog lowpass filter whose passband ripple is 1 dB. Our f_s sampling rate is 100 Hz ($t_s = 0.01$), and the filter's 1 dB cutoff frequency is 20 Hz. Our prototype analog filter will have a frequency magnitude response like that shown in [Figure 6-34](#).

Figure 6-34 Frequency magnitude response of the example prototype analog filter.



Given the above filter requirements, assume that the analog prototype filter design effort results in the $H_c(s)$ Laplace transfer function of

(6-72)

$$H_c(s) = \frac{17410.145}{s^2 + 137.94536s + 17410.145}.$$

It's the transfer function in [Eq. \(6-72\)](#) that we intend to approximate with our discrete IIR filter. To find the analog filter's impulse response, we'd like to get $H_c(s)$ into a form that allows us to use Laplace transform tables to find $h_c(t)$. Searching through systems analysis textbooks, we find the following Laplace transform pair:

(6-73)

$X(s), \text{ Laplace transform of } x(t):$ $\frac{A\omega}{(s + \alpha)^2 + \omega^2}$	$x(t):$ $Ae^{-\alpha t} \cdot \sin(\omega t).$
--	---

Our intent, then, is to modify [Eq. \(6-72\)](#) to get it into the form on the left side of [Eq. \(6-73\)](#). We do

this by realizing that the Laplace transform expression in [Eq. \(6-73\)](#) can be rewritten as

$$(6-74)$$

$$\frac{A\omega}{(s+\alpha)^2 + \omega^2} = \frac{A\omega}{s^2 + 2\alpha s + \alpha^2 + \omega^2}.$$

If we set [Eq. \(6-72\)](#) equal to the right side of [Eq. \(6-74\)](#), we can solve for A , α , and ω . Doing that,

$$(6-75)$$

$$H_c(s) = \frac{17410.145}{s^2 + 137.94536s + 17410.145} = \frac{A\omega}{s^2 + 2\alpha s + \alpha^2 + \omega^2}.$$

Solving [Eq. \(6-75\)](#) for A , α , and ω , we first find

$$(6-76)$$

$$\alpha = \frac{137.94536}{2} = 68.972680;$$

$$(6-77)$$

$$\alpha^2 + \omega^2 = 17410.145,$$

so

$$(6-78)$$

$$\omega = \sqrt{17410.145 - \alpha^2} = 112.485173;$$

and

$$(6-79)$$

$$A = \frac{17410.145}{\omega} = 154.77724.$$

OK, we can now express $H_c(s)$ in the desired form of the left side of [Eq. \(6-74\)](#) as

$$(6-80)$$

$$H_c(s) = \frac{(154.77724)(112.485173)}{(s + 68.972680)^2 + (112.485173)^2}.$$

Using the Laplace transform pair in [Eq. \(6-73\)](#), the time-domain impulse response of the prototype analog filter becomes

$$(6-81)$$

$$h_c(t) = Ae^{-\alpha t} \cdot \sin(\omega t) = 154.77724e^{-68.972680t} \cdot \sin(112.485173t).$$

OK, we're ready to perform Method 1, Step 4, to determine the discrete IIR filter's z -domain transfer function $H(z)$ by performing the z -transform of $h_c(t)$. Again, scanning through digital signal processing textbooks or a good math reference book, we find the following z -transform pair where the time-domain expression is in the same form as [Eq. \(6-81\)](#)'s $h_c(t)$ impulse response:

$$(6-82)$$

$$\begin{array}{ll} x(t): & X(z), z\text{-transform of } x(t): \\ Ce^{-\alpha t} \cdot \sin(\omega t) & \frac{Ce^{-\alpha t} \cdot \sin(\omega t)z^{-1}}{1 - 2[e^{-\alpha t} \cdot \cos(\omega t)]z^{-1} + e^{-2\alpha t} z^{-2}}. \end{array}$$

Remember, now, the α and ω in [Eq. \(6-82\)](#) are generic and are not related to the α and ω values in

[Eqs. \(6-76\)](#) and [\(6-78\)](#). Substituting the constants from [Eq. \(6-81\)](#) into the right side of [Eq. \(6-82\)](#), we get the z-transform of the IIR filter as

(6-83)

$$H(z) = \frac{154.77724e^{-68.972680t} \cdot \sin(112.485173t)z^{-1}}{1 - 2[e^{-68.972680t} \cdot \cos(112.485173t)]z^{-1} + e^{-2 \cdot 68.972680t} z^{-2}}.$$

Performing Method 1, Step 5, we substitute the t_s value of 0.01 for the continuous variable t in [Eq. \(6-83\)](#), yielding the final $H(z)$ transfer function of

(6-84)

$$\begin{aligned} H(z) &= \frac{154.77724e^{-68.972680 \cdot 0.01} \cdot \sin(112.485173 \cdot 0.01)z^{-1}}{1 - 2[e^{-68.972680 \cdot 0.01} \cdot \cos(112.485173 \cdot 0.01)]z^{-1} + e^{-2 \cdot 68.972680 \cdot 0.01} z^{-2}} \\ &= \frac{154.77724e^{-0.68972680} \cdot \sin(1.12485173)z^{-1}}{1 - 2[e^{-0.68972680} \cdot \cos(1.12485173)]z^{-1} + e^{-2 \cdot 0.68972680} z^{-2}} \\ &= \frac{Y(z)}{X(z)} = \frac{70.059517z^{-1}}{1 - 0.43278805z^{-1} + 0.25171605z^{-2}}. \end{aligned}$$

OK, hang in there; we're almost finished. Here are the final steps of Method 1. Because of the transfer function $H(z) = Y(z)/X(z)$, we can cross-multiply the denominators to rewrite the bottom line of [Eq. \(6-84\)](#) as

(6-85)

$$Y(z) \cdot (1 - 0.43278805z^{-1} + 0.25171605z^{-2}) = X(z) \cdot (70.059517z^{-1}),$$

or

$$Y(z) = 70.059517 \cdot X(z)z^{-1} + 0.43278805 \cdot Y(z)z^{-1} - 0.25171605 \cdot Y(z)z^{-2}.$$

By inspection of [Eq. \(6-85\)](#), we can now get the time-domain expression for our IIR filter. Performing Method 1, Steps 6 and 7, we multiply the $x(n-1)$ coefficient by the sample period value of $t_s = 0.01$ to allow for proper scaling as

(6-86)

$$\begin{aligned} y(n) &= 0.01 \cdot 70.059517 \cdot x(n-1) + 0.43278805 \cdot y(n-1) - 0.25171605 \cdot y(n-2) \\ &= 0.70059517 \cdot x(n-1) + 0.43278805 \cdot y(n-1) - 0.25171605 \cdot y(n-2), \end{aligned}$$

and there we (finally) are. The coefficients from [Eq. \(6-86\)](#) are what we use in implementing the improved IIR structure shown in [Figure 6-22](#) to approximate the original 2nd-order Chebyshev analog lowpass filter.

Let's see if we get the same result if we use the impulse invariance design Method 2 to approximate the example prototype analog filter.

6.10.2 Impulse Invariance Design Method 2 Example

Given the original prototype filter's Laplace transfer function as

(6-87)

$$H_c(s) = \frac{17410.145}{s^2 + 137.94536s + 17410.145},$$

and the value of $t_s = 0.01$ for the sample period, we're ready to proceed with Method 2's Step 3. To express $H_c(s)$ as the sum of single-pole filters, we'll have to factor the denominator of [Eq. \(6-87\)](#) and use partial fraction expansion methods. For convenience, let's start by replacing the constants in [Eq. \(6-87\)](#) with variables in the form of

(6-88)

$$H_c(s) = \frac{c}{s^2 + bs + c}$$

where $b = 137.94536$, and $c = 17410.145$. Next, using [Eq. \(6-15\)](#) with $a = 1$, we can factor the quadratic denominator of [Eq. \(6-88\)](#) into

(6-89)

$$H_c(s) = \frac{c}{\left(s + \frac{b}{2} + \sqrt{\frac{b^2 - 4c}{4}}\right) \cdot \left(s + \frac{b}{2} - \sqrt{\frac{b^2 - 4c}{4}}\right)}.$$

If we substitute the values for b and c in [Eq. \(6-89\)](#), we'll find that the quantity under the radical sign is negative. This means that the factors in the denominator of [Eq. \(6-89\)](#) are complex. Because we have lots of algebra ahead of us, let's replace the radicals in [Eq. \(6-89\)](#) with the *imaginary* term jR , where $j = \sqrt{-1}$ and $R = |(b^2 - 4c)/4|$, such that

(6-90)

$$H_c(s) = \frac{c}{(s + b/2 + jR)(s + b/2 - jR)}.$$

OK, partial fraction expansion methods allow us to partition [Eq. \(6-90\)](#) into two separate fractions of the form

(6-91)

$$\begin{aligned} H_c(s) &= \frac{c}{(s + b/2 + jR)(s + b/2 - jR)} \\ &= \frac{K_1}{(s + b/2 + jR)} + \frac{K_2}{(s + b/2 - jR)}, \end{aligned}$$

where the K_1 constant can be found to be equal to $jc/2R$ and constant K_2 is the complex conjugate of K_1 , or $K_2 = -jc/2R$. (To learn the details of partial fraction expansion methods, the interested reader should investigate standard college algebra or engineering mathematics textbooks.) Thus, $H_c(s)$ can be of the form in [Eq. \(6-65\)](#) or

(6-92)

$$H_c(s) = \frac{jc/2R}{(s + b/2 + jR)} + \frac{-jc/2R}{(s + b/2 - jR)}.$$

We can see from [Eq. \(6-92\)](#) that our 2nd-order prototype filter has two poles, one located at $p_1 = -b/2 - jR$ and the other at $p_2 = -b/2 + jR$. We're now ready to map those two poles from the s -plane to the z -plane as called out in Method 2, Step 4. Making our $1 - e^{-p_k t_s} z^{-1}$ substitution for the $s + p_k$

terms in [Eq. \(6-92\)](#), we have the following expression for the z -domain single-pole digital filters:

(6-93)

$$H(z) = \frac{jc / 2R}{1 - e^{-(b/2+jR)t_s} z^{-1}} + \frac{-jc / 2R}{1 - e^{-(b/2-jR)t_s} z^{-1}}.$$

Our objective in Method 2, Step 5, is to massage [Eq. \(6-93\)](#) into the form of [Eq. \(6-69\)](#), so that we can determine the IIR filter's feedforward and feedback coefficients. Putting both fractions in [Eq. \(6-93\)](#) over a common denominator gives us

(6-94)

$$H(z) = \frac{(jc / 2R)(1 - e^{-(b/2-jR)t_s} z^{-1}) - (jc / 2R)(1 - e^{-(b/2+jR)t_s} z^{-1})}{(1 - e^{-(b/2+jR)t_s} z^{-1})(1 - e^{-(b/2-jR)t_s} z^{-1})}.$$

Collecting like terms in the numerator and multiplying out the denominator gives us

(6-95)

$$H(z) = \frac{(jc / 2R)(1 - e^{-(b/2-jR)t_s} z^{-1} - 1 + e^{-(b/2+jR)t_s} z^{-1})}{1 - e^{-(b/2-jR)t_s} z^{-1} - e^{-(b/2+jR)t_s} z^{-1} + e^{-bt_s} z^{-2}}.$$

Factoring the exponentials and collecting like terms of powers of z in [Eq. \(6-95\)](#),

(6-96)

$$H(z) = \frac{(jc / 2R)(e^{-(b/2+jR)t_s} - e^{-(b/2-jR)t_s}) z^{-1}}{1 - (e^{-(b/2-jR)t_s} + e^{-(b/2+jR)t_s}) z^{-1} + e^{-bt_s} z^{-2}}.$$

Continuing to simplify our $H(z)$ expression by factoring out the real part of the exponentials,

(6-97)

$$H(z) = \frac{(jc / 2R)e^{-bt_s/2}(e^{-jRt_s} - e^{jRt_s}) z^{-1}}{1 - e^{-bt_s/2}(e^{jRt_s} + e^{-jRt_s}) z^{-1} + e^{-bt_s} z^{-2}}.$$

We now have $H(z)$ in a form with all the like powers of z combined into single terms, and [Eq. \(6-97\)](#) looks something like the desired form of [Eq. \(6-69\)](#). Knowing that the final coefficients of our IIR filter must be real numbers, the question is “What do we do with those imaginary j terms in [Eq. \(6-97\)](#)?” Once again, Euler to the rescue.[‡] Using Euler's equations for sinusoids, we can eliminate the imaginary exponentials and [Eq. \(6-97\)](#) becomes

[‡] From Euler, we know that $\sin(\theta) = (e^{j\theta} - e^{-j\theta})/2j$, and $\cos(\theta) = (e^{j\theta} + e^{-j\theta})/2$.

(6-98)

$$\begin{aligned} H(z) &= \frac{(jc / 2R)e^{-bt_s/2}[-2j \sin(Rt_s)] z^{-1}}{1 - e^{-bt_s/2}[2 \cos(Rt_s)] z^{-1} + e^{-bt_s} z^{-2}} \\ &= \frac{(c / R)e^{-bt_s/2}[\sin(Rt_s)] z^{-1}}{1 - e^{-bt_s/2}[2 \cos(Rt_s)] z^{-1} + e^{-bt_s} z^{-2}}. \end{aligned}$$

If we plug the values $c = 17410.145$, $b = 137.94536$, $R = 112.48517$, and $t_s = 0.01$ into [Eq. \(6-98\)](#), we get the following IIR filter transfer function:

(6-99)

$$H(z) = \frac{(154.77724)(0.50171312)(0.902203655)z^{-1}}{1 - (0.50171312)(0.86262058)z^{-1} + 0.25171605z^{-2}}$$

$$= \frac{70.059517z^{-1}}{1 - 0.43278805z^{-1} + 0.25171605z^{-2}}.$$

Because the transfer function $H(z) = Y(z)/X(z)$, we can again cross-multiply the denominators to rewrite [Eq. \(6-99\)](#) as

$$(6-100)$$

$$Y(z) \cdot (1 - 0.43278805z^{-1} + 0.25171605z^{-2}) = X(z) \cdot (70.059517z^{-1}),$$

or

$$Y(z) = 70.059517 \cdot X(z)z^{-1} + 0.43278805 \cdot Y(z)z^{-1} - 0.25171605 \cdot Y(z)z^{-2}.$$

Now we take the inverse z -transform of [Eq. \(6-100\)](#), by inspection, to get the time-domain expression for our IIR filter as

$$(6-101)$$

$$y(n) = 70.059517 \cdot x(n-1) + 0.43278805 \cdot y(n-1) - 0.25171605 \cdot y(n-2).$$

One final step remains. To force the IIR filter gain to be equal to the prototype analog filter's gain, we multiply the $x(n-1)$ coefficient by the sample period t_s as suggested in Method 2, Step 6. In this case, there's only one $x(n)$ coefficient, giving us

$$(6-102)$$

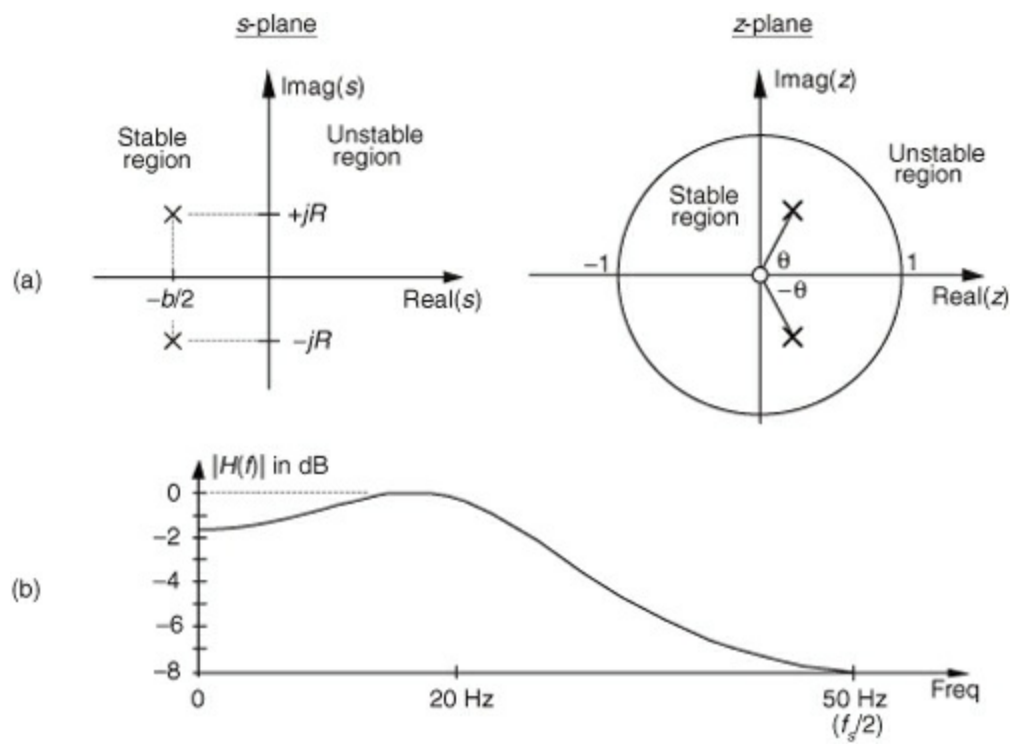
$$y(n) = 0.01 \cdot 70.059517 \cdot x(n-1) + 0.43278805 \cdot y(n-1) - 0.25171605 \cdot y(n-2)$$

$$= 0.70059517 \cdot x(n-1) + 0.43278805 \cdot y(n-1) - 0.25171605 \cdot y(n-2).$$

That compares well with the Method 1 result in [Eq. \(6-86\)](#). (Isn't it comforting to work a problem two different ways and get the same result?)

[Figure 6-35](#) shows, in graphical form, the result of our IIR design example. The s -plane pole locations of the prototype filter and the z -plane poles of the IIR filter are shown in [Figure 6-35\(a\)](#). Because the s -plane poles are to the left of the origin and the z -plane poles are inside the unit circle, both the prototype analog and the discrete IIR filters are stable. We find the prototype filter's s -plane pole locations by evaluating $H_c(s)$ in [Eq. \(6-92\)](#). When $s = -b/2 - jR$, the denominator of the first term in [Eq. \(6-92\)](#) becomes zero and $H_c(s)$ is infinitely large. That $s = -b/2 - jR$ value is the location of the lower s -plane pole in [Figure 6-35\(a\)](#). When $s = -b/2 + jR$, the denominator of the second term in [Eq. \(6-92\)](#) becomes zero and $s = -b/2 + jR$ is the location of the second s -plane pole.

Figure 6-35 Impulse invariance design example filter characteristics: (a) s -plane pole locations of prototype analog filter and z -plane pole locations of discrete IIR filter; (b) frequency magnitude response of the discrete IIR filter.



The IIR filter's z -plane pole locations are found from [Eq. \(6-93\)](#). If we multiply the numerators and denominators of [Eq. \(6-93\)](#) by z ,

(6-103)

$$\begin{aligned} H(z) \cdot \frac{z}{z} &= \frac{z(jc/2R)}{z(1 - e^{-(b/2 + jR)t_s} z^{-1})} + \frac{z(-jc/2R)}{z(1 - e^{-(b/2 - jR)t_s} z^{-1})} \\ &= \frac{(jc/2R)z}{z - e^{-(b/2 + jR)t_s}} + \frac{(-jc/2R)z}{z - e^{-(b/2 - jR)t_s}}. \end{aligned}$$

In [Eq. \(6-103\)](#), when z is set equal to $e^{-(b/2 + jR)t_s}$, the denominator of the first term in [Eq. \(6-103\)](#) becomes zero and $H(z)$ becomes infinitely large. The value of z of

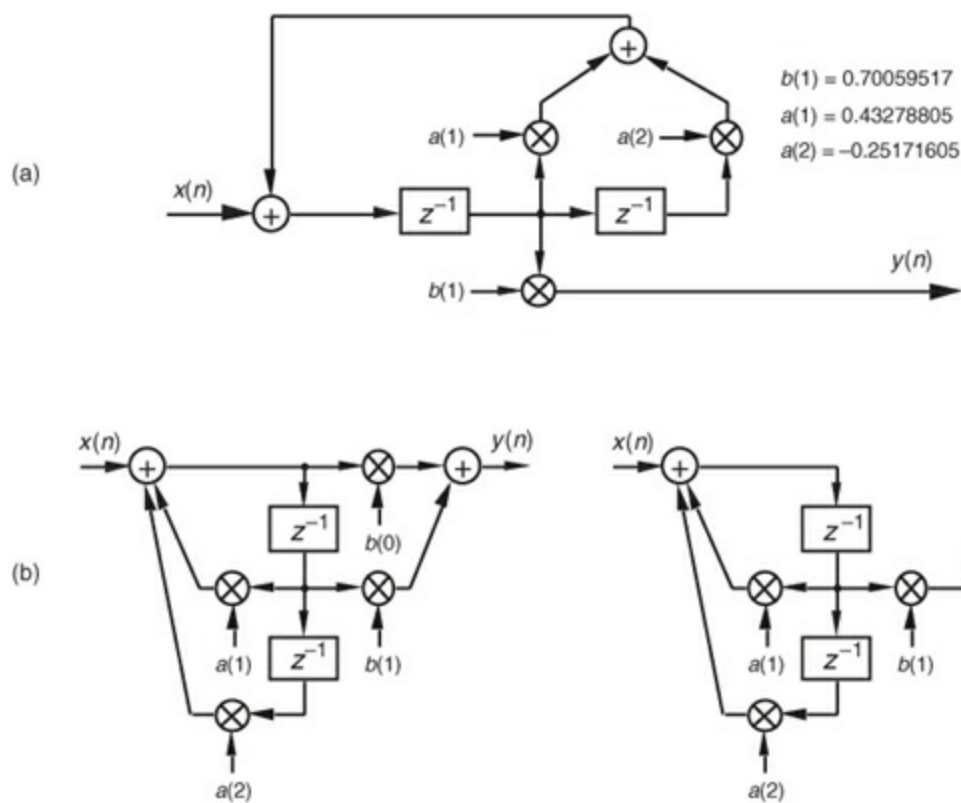
(6-104)

$$z = e^{-(b/2 + jR)t_s} = e^{-bt_s/2} e^{-jRt_s} = e^{-bt_s/2} \angle -Rt_s \text{ radians}$$

defines the location of the lower z -plane pole in [Figure 6-35\(a\)](#). Specifically, this lower pole is located at a distance of $e^{-bt_s/2} = 0.5017$ from the origin, at an angle of $\theta = -Rt_s$ radians, or -64.45° . Being conjugate poles, the upper z -plane pole is located the same distance from the origin at an angle of $\theta = Rt_s$ radians, or $+64.45^\circ$. [Figure 6-35\(b\)](#) illustrates the frequency magnitude response of the IIR filter in Hz.

Two different implementations of our IIR filter are shown in [Figure 6-36](#). [Figure 6-36\(a\)](#) is an implementation of our 2nd-order IIR filter based on the general IIR structure given in [Figure 6-22](#), and [Figure 6-36\(b\)](#) shows the 2nd-order IIR filter implementation based on the alternate structure from [Figure 6-21\(b\)](#). Knowing that the $b(0)$ coefficient on the left side of [Figure 6-36\(b\)](#) is zero, we arrive at the simplified structure on the right side of [Figure 6-36\(b\)](#). Looking carefully at [Figure 6-36\(a\)](#) and the right side of [Figure 6-36\(b\)](#), we can see that they are equivalent.

Figure 6-36 Implementations of the impulse invariance design example filter.

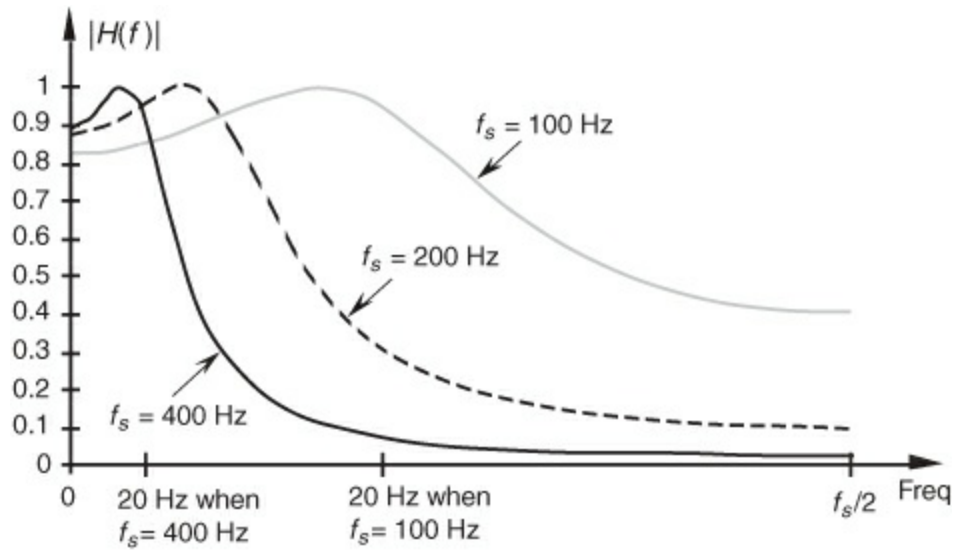


Although both impulse invariance design methods are covered in the literature, we might ask, “Which one is preferred?” There’s no definite answer to that question because it depends on the $H_c(s)$ of the prototype analog filter. Although our Method 2 example above required more algebra than Method 1, if the prototype filter’s s -domain poles were located only on the real axis, Method 2 would have been much simpler because there would be no complex variables to manipulate. In general, Method 2 is more popular for two reasons: (1) the inverse Laplace and z -transformations, although straightforward in our Method 1 example, can be very difficult for higher-order filters, and (2) unlike Method 1, Method 2 can be coded in a software routine or a computer spreadsheet.

Upon examining the frequency magnitude response in [Figure 6-35\(b\)](#), we can see that this 2nd-order IIR filter’s roll-off is not particularly steep. This is, admittedly, a simple low-order filter, but its attenuation slope is so gradual that it doesn’t appear to be of much use as a lowpass filter.[‡] We can also see that the filter’s passband ripple is greater than the desired value of 1 dB in [Figure 6-34](#). What we’ll find is that it’s not the low order of the filter that contributes to its poor performance, but the sampling rate used. That 2nd-order IIR filter response is repeated as the shaded curve in [Figure 6-37](#). If we increased the sampling rate to 200 Hz, we’d get the frequency response shown by the dashed curve in [Figure 6-37](#). Increasing the sampling rate to 400 Hz results in the much improved frequency response indicated by the solid line in the figure. Sampling rate changes do not affect our filter order or implementation structure. Remember, if we change the sampling rate, only the sample period t_s changes in our design equations, resulting in a different set of filter coefficients for each new sampling rate. So we can see that the smaller we make t_s (larger f_s), the better the resulting filter when either impulse invariance design method is used because the replicated spectral overlap indicated in [Figure 6-32\(b\)](#) is reduced due to the larger f_s sampling rate. The bottom line here is that impulse invariance IIR filter design techniques are most appropriate for narrowband filters, that is, lowpass filters whose cutoff frequencies are much smaller than the sampling rate.

[‡] A piece of advice: whenever you encounter any frequency representation (be it a digital filter magnitude response or a signal spectrum) that has nonzero values at $\pm f_s/2$, be suspicious—be very suspicious—that aliasing is taking place.

Figure 6-37 IIR filter frequency magnitude response, on a linear scale, at three separate sampling rates. Notice how the filter's absolute cutoff frequency of 20 Hz shifts relative to the different f_s sampling rates.



The second analytical technique for analog filter approximation, the bilinear transform method, alleviates the impulse invariance method's aliasing problems at the expense of what's called *frequency warping*. Specifically, there's a nonlinear distortion between the prototype analog filter's frequency scale and the frequency scale of the approximating IIR filter designed using the bilinear transform. Let's see why.

6.11 Bilinear Transform IIR Filter Design Method

There's a popular analytical IIR filter design technique known as the *bilinear transform* method. Like the impulse invariance method, this design technique approximates a prototype analog filter defined by the continuous Laplace transfer function $H_c(s)$ with a discrete filter whose transfer function is $H(z)$. However, the bilinear transform method has great utility because

- it allows us simply to substitute a function of z for s in $H_c(s)$ to get $H(z)$, thankfully eliminating the need for Laplace and z -transformations as well as any necessity for partial fraction expansion algebra;
- it maps the entire s -plane to the z -plane, enabling us to completely avoid the frequency-domain aliasing problems we had with the impulse invariance design method; and
- it induces a nonlinear distortion of $H(z)$'s frequency axis, relative to the original prototype analog filter's frequency axis, that sharpens the final roll-off of digital lowpass filters.

Don't worry. We'll explain each one of these characteristics and see exactly what they mean to us as we go about designing an IIR filter.

If the transfer function of a prototype analog filter is $H_c(s)$, then we can obtain the discrete IIR filter z -domain transfer function $H(z)$ by substituting the following for s in $H_c(s)$

$$(6-105)$$

$$s = \frac{2}{t_s} \left(\frac{1 - z^{-1}}{1 + z^{-1}} \right)$$

where, again, t_s is the discrete filter's sampling period ($1/f_s$). Just as in the impulse invariance design method, when using the bilinear transform method, we're interested in where the analog filter's poles end up on the z -plane after the transformation. This s -plane to z -plane mapping behavior is exactly what makes the bilinear transform such an attractive design technique.[‡]

[‡] The bilinear transform is a technique in the theory of complex variables for mapping a function on the complex plane of one variable to the complex plane of another variable. It maps circles and straight lines to straight lines and circles, respectively.

Let's investigate the major characteristics of the bilinear transform's s -plane to z -plane mapping. First we'll show that any pole on the left side of the s -plane will map to the inside of the unit circle in the z -plane. It's easy to show this by solving [Eq. \(6-105\)](#) for z in terms of s . Multiplying [Eq. \(6-105\)](#) by $(t_s/2)(1 + z^{-1})$ and collecting like terms of z leads us to

(6-106)

$$z = \frac{1 + (t_s/2)s}{1 - (t_s/2)s}.$$

If we designate the real and imaginary parts of s as

(6-107)

$$s = \sigma + j\omega_a,$$

where the subscript in the radian frequency ω_a signifies *analog*, [Eq. \(6-106\)](#) becomes

(6-108)

$$z = \frac{1 + \sigma t_s/2 + j\omega_a t_s/2}{1 - \sigma t_s/2 - j\omega_a t_s/2} = \frac{(1 + \sigma t_s/2) + j\omega_a t_s/2}{(1 - \sigma t_s/2) - j\omega_a t_s/2}.$$

We see in [Eq. \(6-108\)](#) that z is complex, comprising the ratio of two complex expressions. As such, if we denote z as a magnitude at an angle in the form of $z = |z|\angle\theta_z$, we know that the magnitude of z is given by

(6-109)

$$|z| = \frac{\text{Mag}_{\text{numerator}}}{\text{Mag}_{\text{denominator}}} = \sqrt{\frac{(1 + \sigma t_s/2)^2 + (\omega_a t_s/2)^2}{(1 - \sigma t_s/2)^2 + (\omega_a t_s/2)^2}}.$$

OK, if σ is negative ($\sigma < 0$), the numerator of the ratio on the right side of [Eq. \(6-109\)](#) will be less than the denominator, and $|z|$ will be less than 1. On the other hand, if σ is positive ($\sigma > 0$), the numerator will be larger than the denominator, and $|z|$ will be greater than 1. This confirms that when using the bilinear transform defined by [Eq. \(6-105\)](#), any pole located on the left side of the s -plane ($\sigma < 0$) will map to a z -plane location inside the unit circle. This characteristic ensures that any stable s -plane pole of a prototype analog filter will map to a stable z -plane pole for our discrete IIR filter. Likewise, any analog filter pole located on the right side of the s -plane ($\sigma > 0$) will map to a z -plane location outside the unit circle when using the bilinear transform. This reinforces our notion that to avoid filter instability, during IIR filter design, we should avoid allowing any z -plane poles to lie outside the unit circle.

Next, let's show that the $j\omega_a$ axis of the s -plane maps to the perimeter of the unit circle in the z -plane. We can do this by setting $\sigma = 0$ in [Eq. \(6-108\)](#) to get

(6-110)

$$z = \frac{1 + j\omega_d t_s / 2}{1 - j\omega_d t_s / 2}.$$

Here, again, we see in [Eq. \(6-110\)](#) that z is a complex number comprising the ratio of two complex numbers, and we know the magnitude of this z is given by

$$(6-111)$$

$$|z|_{\sigma=0} = \frac{\text{Mag}_{\text{numerator}}}{\text{Mag}_{\text{denominator}}} = \sqrt{\frac{(1)^2 + (\omega_d t_s / 2)^2}{(1)^2 + (\omega_d t_s / 2)^2}}.$$

The magnitude of z in [Eq. \(6-111\)](#) is *always* 1. So, as we stated, when using the bilinear transform, the $j\omega_d$ axis of the s -plane maps to the perimeter of the unit circle in the z -plane. However, this frequency mapping from the s -plane to the unit circle in the z -plane is not linear. It's important to know why this frequency nonlinearity, or warping, occurs and to understand its effects. So we shall, by showing the relationship between the s -plane frequency and the z -plane frequency that we'll designate as ω_d .

If we define z on the unit circle in polar form as $z = re^{-j\omega_d}$ as we did for [Figure 6-13](#), where r is 1 and ω_d is the angle, we can substitute $z = e^{j\omega_d}$ in [Eq. \(6-105\)](#) as

$$(6-112)$$

$$s = \frac{2}{t_s} \left(\frac{1 - e^{-j\omega_d}}{1 + e^{-j\omega_d}} \right).$$

If we show s in its rectangular form and partition the ratio in brackets into half-angle expressions,

$$(6-113)$$

$$s = \sigma + j\omega_d = \frac{2}{t_s} \cdot \frac{e^{-j\omega_d/2}(e^{j\omega_d/2} - e^{-j\omega_d/2})}{e^{-j\omega_d/2}(e^{j\omega_d/2} + e^{-j\omega_d/2})}.$$

Using Euler's relationships of $\sin(\theta) = (e^{j\theta} - e^{-j\theta})/2j$ and $\cos(\theta) = (e^{j\theta} + e^{-j\theta})/2$, we can convert the right side of [Eq. \(6-113\)](#) to rectangular form as

$$(6-114)$$

$$\begin{aligned} s = \sigma + j\omega_d &= \frac{2}{t_s} \cdot \frac{e^{-j\omega_d/2}[2j \sin(\omega_d/2)]}{e^{-j\omega_d/2}[2 \cos(\omega_d/2)]} \\ &= \frac{2}{t_s} \cdot \frac{2e^{-j\omega_d/2}}{2e^{-j\omega_d/2}} \cdot \frac{j \sin(\omega_d/2)}{\cos(\omega_d/2)} \\ &= \frac{j2}{t_s} \tan(\omega_d/2). \end{aligned}$$

If we now equate the real and imaginary parts of [Eq. \(6-114\)](#), we see that $\sigma = 0$, and

$$(6-115)$$

$$\omega_d = \frac{2}{t_s} \tan\left(\frac{\omega_d}{2}\right), \quad -\infty \leq \omega_d \leq \infty.$$

The analog frequency ω_a (radians/second) can have any value and its equivalent f_a cyclic frequency is

(6-115')

$$f_a = \frac{\omega_a}{2\pi} \text{ Hz.}$$

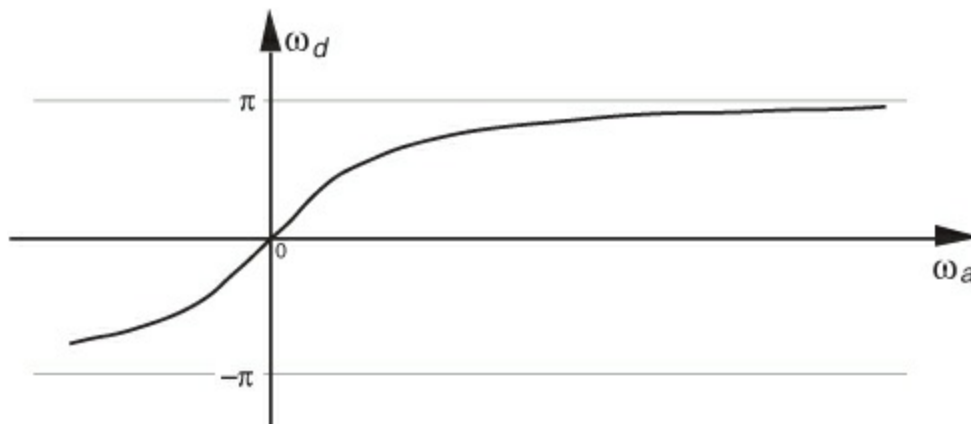
Rearranging [Eq. \(6-115\)](#) to give us the useful expression for the z -domain frequency ω_d , in terms of the s -domain frequency ω_a , we write

(6-116)

$$\omega_d = 2 \tan^{-1} \left(\frac{\omega_a t_s}{2} \right), \quad -\pi \leq \omega_d \leq \pi.$$

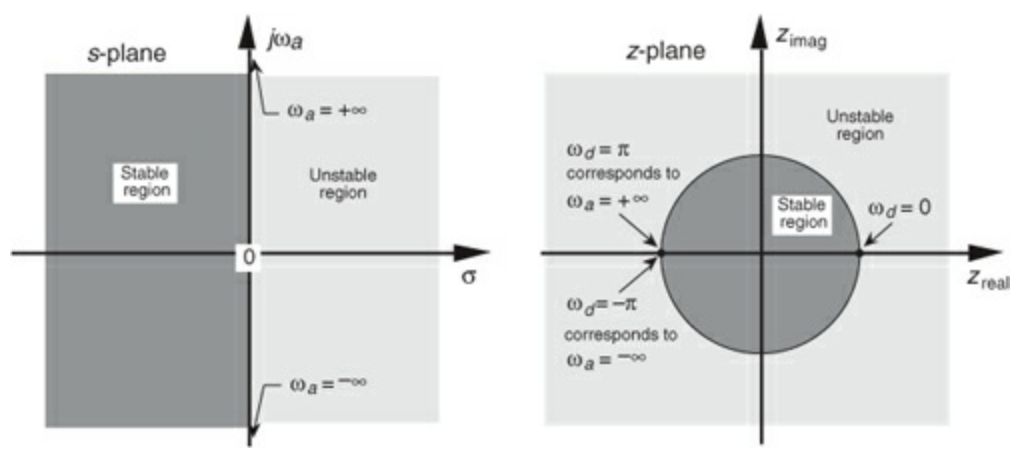
It's critical to notice that the range of ω_d is $\pm\pi$, and the dimensions of digital frequency ω_d are radians/sample (not radians/second). The important relationship in [Eq. \(6-116\)](#), which accounts for the so-called frequency warping due to the bilinear transform, is illustrated in [Figure 6-38](#). Notice that because $\tan^{-1}(\omega_a t_s/2)$ approaches $\pi/2$ as ω_a becomes large, ω_d must then approach twice that value, or π . This means that no matter how large the s -plane's analog ω_a becomes, the z -plane's ω_d will never be greater than π radians/sample ($f_s/2$ Hz).

Figure 6-38 Nonlinear relationship between the z -domain frequency ω_d and the s -domain frequency ω_a .



Remember how we considered [Figure 6-14](#) and stated that only the $-\pi f_s$ to $+\pi f_s$ radians/second frequency range for ω_a can be accounted for on the z -plane? Well, our new mapping from the bilinear transform maps the entire s -plane to the z -plane, and not just the primary strip of the s -plane shown in [Figure 6-14](#). Now, just as a walk along the $j\omega_a$ frequency axis on the s -plane takes us to infinity in either direction, a trip halfway around the unit circle in a counterclockwise direction takes us from $\omega_a = 0$ to $\omega_a = +\infty$ radians/second. As such, the bilinear transform maps the s -plane's entire $j\omega_a$ axis onto the unit circle in the z -plane. We illustrate these bilinear transform mapping properties in [Figure 6-39](#).

Figure 6-39 Bilinear transform mapping of the s -plane to the z -plane.



In an attempt to show the practical implications of this frequency warping, let's relate the s -plane and z -plane frequencies to a more practical measure of frequencies in Hz. Because a ω_d frequency of $\omega_d = \pi$ radians/sample corresponds to a cyclic frequency of $f_s/2$ Hz, we relate ω_d and a digital cyclic frequency f_d using

$$(6-117)$$

$$\omega_d = 2\pi(f_d/f_s) \text{ Hz.}$$

Substituting [Eq. \(6-117\)](#) into [Eq. \(6-115\)](#), and recalling that $\omega_a = 2\pi f_a$, gives us

$$(6-118)$$

$$f_d = \frac{f_s \tan^{-1}(\pi f_a / f_s)}{\pi} \text{ Hz.}$$

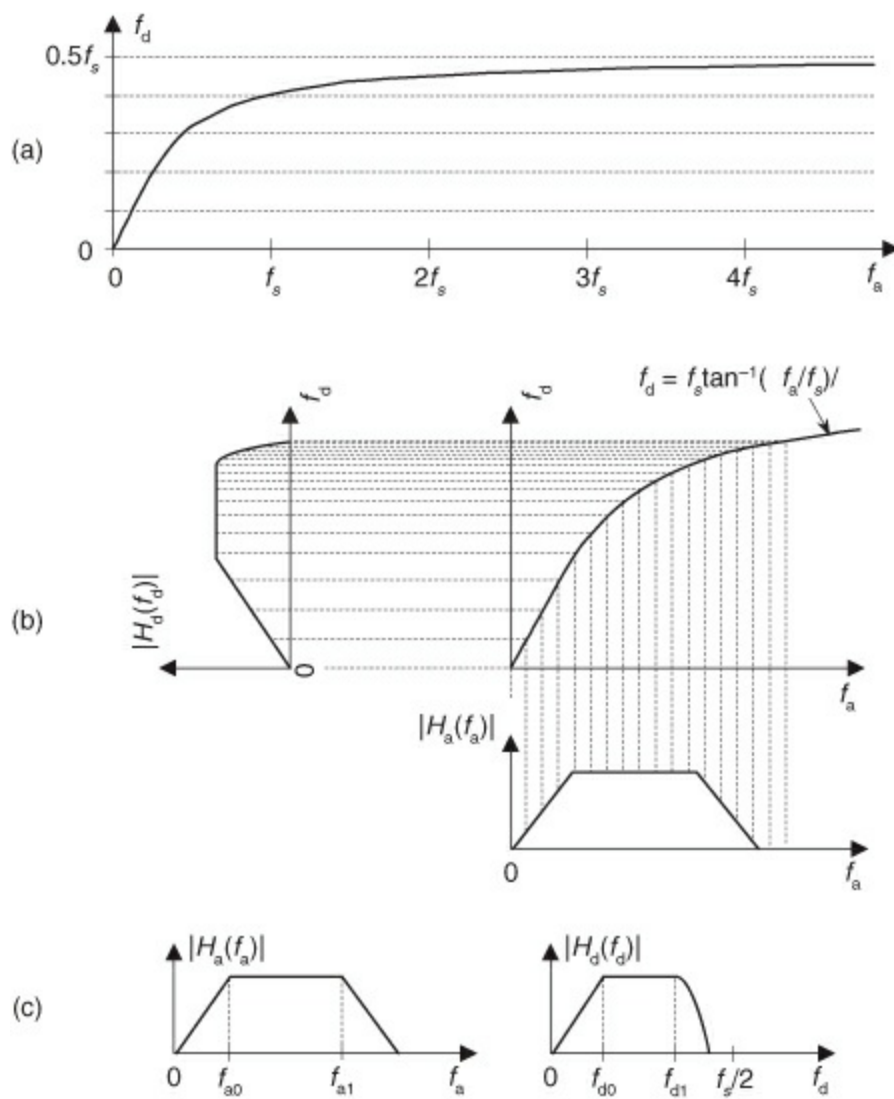
Solving [Eq. \(6-118\)](#) for f_d yields

$$(6-119)$$

$$f_a = \frac{f_s \tan(\pi f_d / f_s)}{\pi} \text{ Hz.}$$

[Equation \(6-119\)](#) is plotted in [Figure 6-40\(a\)](#). [Equations \(6-118\)](#) and [\(6-119\)](#) are very useful! They relate the analog s -plane frequency f_a in Hz to the digital z -plane's warped frequency f_d in Hz. This important nonlinear relationship is plotted in [Figure 6-40\(b\)](#). There we see that the f_d frequency warping (compression) becomes more severe as f_d approaches $f_s/2$.

Figure 6-40 Nonlinear relationship between the f_d and f_a frequencies: (a) frequency warping curve; (b) s -domain frequency response transformation to a z -domain frequency response; (c) example $|H_a(f_a)|$ and transformed $|H_d(f_d)|$.



So what does all this f_a to f_d mapping rigmarole mean? It means two things. First, if a bandpass analog filter's upper cutoff frequency is f_{a1} Hz, a bilinear-transform-designed digital bandpass filter operating at a sample rate of f_s Hz will have an upper cutoff frequency of f_{d1} Hz as shown in [Figure 6-40\(c\)](#). Likewise if a bilinear-transform-designed digital bandpass filter is desired to have an upper cutoff frequency of f_{d1} Hz, then the original prototype analog bandpass filter must be designed (prewarped) to have an upper cutoff frequency of f_{a1} Hz using [Eq. \(6-118\)](#).

Second, no IIR filter response aliasing can occur with the bilinear transform design method. No matter what the shape, or bandwidth, of the $|H_a(f_a)|$ prototype analog filter, none of the $|H_d(f_d)|$ magnitude responses can extend beyond half the sampling rate of $f_s/2$ Hz—and that's what makes the bilinear transform IIR filter design method as popular as it is.

The steps necessary to perform an IIR filter design using the bilinear transform method are as follows:

Step 1: Obtain the Laplace transfer function $H_c(s)$ for the prototype analog filter in the form of [Eq. \(6-43\)](#).

Step 2: Determine the digital filter's equivalent sampling frequency f_s and establish the sample period $t_s = 1/f_s$.

Step 3: In the Laplace $H_c(s)$ transfer function, substitute the expression

(6-120)

$$\frac{2}{t_s} \left(\frac{1 - z^{-1}}{1 + z^{-1}} \right)$$

for the variable s to get the IIR filter's $H(z)$ transfer function.

Step 4: Multiply the numerator and denominator of $H(z)$ by the appropriate power of $(1 + z^{-1})$ and grind through the algebra to collect terms of like powers of z in the form

(6-121)

$$H(z) = \frac{\sum_{k=0}^N b(k)z^{-k}}{1 - \sum_{k=1}^M a(k)z^{-k}}.$$

Step 5: Just as in the impulse invariance design methods, by inspection, we can express the IIR filter's time-domain equation in the general form of

(6-122)

$$y(n) = b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + \dots + b(N)x(n-N) \\ + a(1)y(n-1) + a(2)y(n-2) + \dots + a(M)y(n-M).$$

Although the expression in [Eq. \(6-122\)](#) only applies to the filter structure in [Figure 6-18](#), to complete our design, we can apply the $a(k)$ and $b(k)$ coefficients to the improved IIR structure shown in [Figure 6-22](#).

To show just how straightforward the bilinear transform design method is, let's use it to solve the IIR filter design problem first presented for the impulse invariance design method.

6.11.1 Bilinear Transform Design Example

Again, our goal is to design an IIR filter that approximates the 2nd-order Chebyshev prototype analog lowpass filter, shown in [Figure 6-26](#), whose passband ripple is 1 dB. The f_s sampling rate is 100 Hz ($t_s = 0.01$), and the filter's 1 dB cutoff frequency is 20 Hz. As before, given the original prototype filter's Laplace transfer function as

(6-123)

$$H_c(s) = \frac{17410.145}{s^2 + 137.94536s + 17410.145},$$

and the value of $t_s = 0.01$ for the sample period, we're ready to proceed with Step 3. For convenience, let's replace the constants in [Eq. \(6-123\)](#) with variables in the form of

(6-124)

$$H_c(s) = \frac{c}{s^2 + bs + c}$$

where $b = 137.94536$ and $c = 17410.145$. Performing the substitution of [Eq. \(6-120\)](#) in [Eq. \(6-124\)](#),

(6-125)

$$H(z) = \frac{c}{\left(\frac{2}{t_s} \cdot \frac{1-z^{-1}}{1+z^{-1}}\right)^2 + b \frac{2}{t_s} \left(\frac{1-z^{-1}}{1+z^{-1}}\right) + c}.$$

To simplify our algebra a little, let's substitute the variable a for the fraction $2/t_s$ to give

(6-126)

$$H(z) = \frac{c}{a^2 \left(\frac{1-z^{-1}}{1+z^{-1}}\right)^2 + ab \left(\frac{1-z^{-1}}{1+z^{-1}}\right) + c}.$$

Proceeding with Step 4, we multiply [Eq. \(109\)](#)'s numerator and denominator by $(1+z^{-1})^2$ to yield

(6-127)

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

Multiplying through by the factors in the denominator of [Eq. \(6-127\)](#), and collecting like powers of z ,

(6-128)

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

We're almost there. To get [Eq. \(6-128\)](#) into the form of [Eq. \(6-121\)](#) with a constant term of one in the denominator, we divide [Eq. \(6-128\)](#)'s numerator and denominator by (a^2+ab+c) , giving us

(6-129)

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

We now have $H(z)$ in a form with all the like powers of z combined into single terms, and [Eq. \(6-129\)](#) looks something like the desired form of [Eq. \(6-121\)](#). If we plug the values $a = 2/t_s = 200$, $b = 137.94536$, and $c = 17410.145$ into [Eq. \(6-129\)](#), we get the following IIR filter transfer function:

(6-130)

$$\begin{aligned} H(z) &= \frac{0.20482712(1+2z^{-1}+z^{-2})}{1-0.53153089z^{-1}+0.35083938z^{-2}} \\ &= \frac{0.20482712 + 0.40965424z^{-1} + 0.20482712z^{-2}}{1-0.53153089z^{-1}+0.35083938z^{-2}}, \end{aligned}$$

and there we are. Now, by inspection of [Eq. \(6-130\)](#), we get the time-domain expression for our IIR filter as

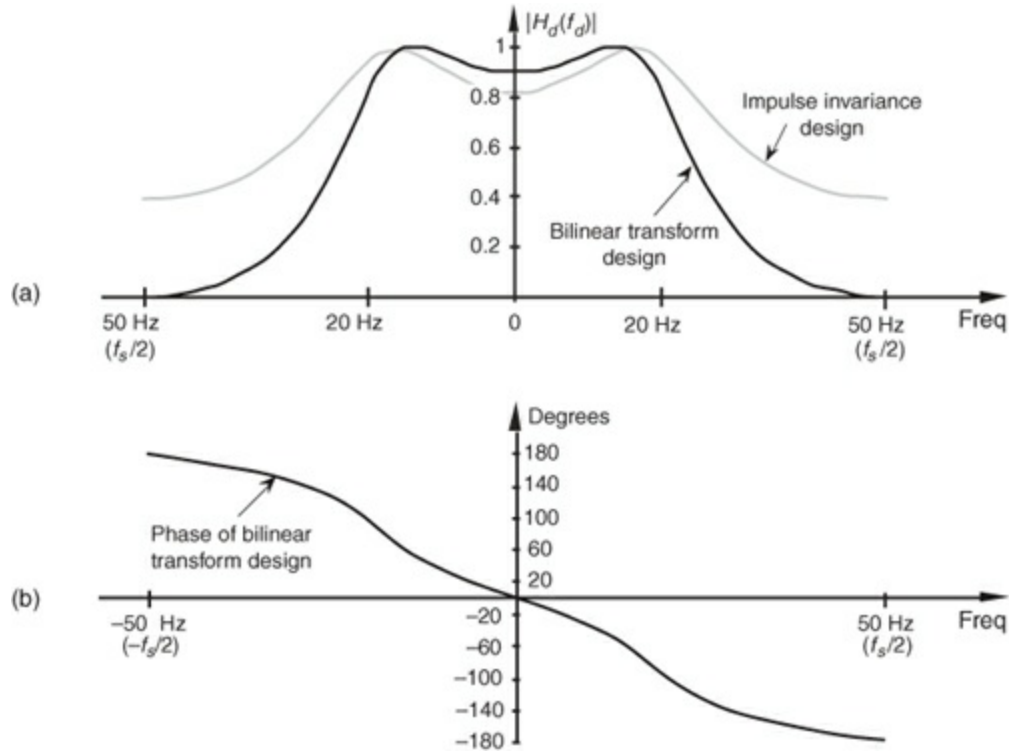
(6-131)

$$\begin{aligned} y(n) &= 0.20482712 \cdot x(n) + 0.40965424 \cdot x(n-1) + 0.20482712 \cdot x(n-2) \\ &\quad + 0.53153089 \cdot y(n-1) - 0.35083938 \cdot y(n-2). \end{aligned}$$

The frequency magnitude response of our bilinear transform IIR design example is shown as the dark

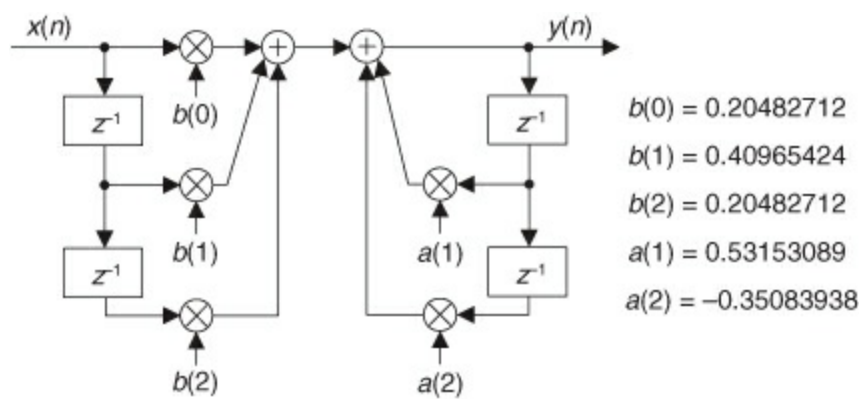
curve in [Figure 6-41\(a\)](#), where, for comparison, we've shown the result of that impulse invariance design example as the shaded curve. Notice how the bilinear-transform-designed filter's magnitude response approaches zero at the folding frequency of $f_s/2 = 50$ Hz. This is as it should be—that's the whole purpose of the bilinear transform design method. [Figure 6-41\(b\)](#) illustrates the nonlinear phase response of the bilinear-transform-designed IIR filter.

Figure 6-41 Comparison of the bilinear transform and impulse invariance design IIR filters: (a) frequency magnitude responses; (b) phase of the bilinear transform IIR filter.



We might be tempted to think that not only is the bilinear transform design method easier to perform than the impulse invariance design method, but that it gives us a much sharper roll-off for our lowpass filter. Well, the frequency warping of the bilinear transform method does compress (sharpen) the roll-off portion of a lowpass filter, as we saw in [Figure 6-40](#), but an additional reason for the improved response is the price we pay in terms of the additional complexity of the implementation of our IIR filter. We see this by examining the implementation of our IIR filter as shown in [Figure 6-42](#). Notice that our new filter requires five multiplications per filter output sample where the impulse invariance design filter in [Figure 6-28\(a\)](#) required only three multiplications per filter output sample. The additional multiplications are, of course, required by the additional feedforward z terms in the numerator of [Eq. \(6-130\)](#). These added $b(k)$ coefficient terms in the $H(z)$ transfer function correspond to zeros in the z -plane created by the bilinear transform that did not occur in the impulse invariance design method.

Figure 6-42 Implementation of the bilinear transform design example filter.



Because our example prototype analog lowpass filter had a cutoff frequency that was $f_s/5$, we don't see a great deal of frequency warping in the bilinear transform curve in [Figure 6-41](#). (In fact, Kaiser has shown that when f_s is large, the impulse invariance and bilinear transform design methods result in essentially identical $H(z)$ transfer functions[18].) Had our cutoff frequency been a larger percentage of f_s , bilinear transform warping would have been more serious, and our resultant $|H_d(f_d)|$ cutoff frequency would have been below the desired value. What the pros do to avoid this is to *prewarp* the prototype analog filter's cutoff frequency requirement before the analog $H_c(s)$ transfer function is derived in Step 1.

In that way, they compensate for the bilinear transform's frequency warping before it happens. We can use [Eq. \(6-115\)](#) to determine the prewarped prototype analog filter lowpass cutoff frequency that we want mapped to the desired IIR lowpass cutoff frequency. We plug the desired IIR cutoff frequency ω_d in [Eq. \(6-115\)](#) to calculate the prototype analog ω_a cutoff frequency used to derive the prototype analog filter's $H_c(s)$ transfer function.

Although we explained how the bilinear transform design method avoids the impulse invariance method's inherent frequency response aliasing, it's important to remember that we still have to avoid filter input data aliasing. No matter what kind of digital filter or filter design method is used, the original input signal data must always be obtained using a sampling scheme that avoids the aliasing described in [Chapter 2](#). If the original input data contains errors due to sample rate aliasing, no filter can remove those errors.

Our introductions to the impulse invariance and bilinear transform design techniques have, by necessity, presented only the essentials of those two design methods. Although rigorous mathematical treatment of the impulse invariance and bilinear transform design methods is inappropriate for an introductory text such as this, more detailed coverage is available to the interested reader[20,21,25,26]. References [25] and [26], by the way, have excellent material on the various prototype analog filter types used as a basis for the analytical IIR filter design methods. Although our examples of IIR filter design using the impulse invariance and bilinear transform techniques approximated analog lowpass filters, it's important to remember that these techniques apply equally well to designing bandpass and highpass IIR filters. To design a highpass IIR filter, for example, we'd merely start our design with a Laplace transfer function for the prototype analog highpass filter. Our IIR digital filter design would then proceed to approximate that prototype highpass filter.

As we have seen, the impulse invariance and bilinear transform design techniques are both powerful and a bit difficult to perform. The mathematics is intricate and the evaluation of the design equations is arduous for all but the simplest filters. As such, we'll introduce a third class of IIR filter design methods based on software routines that take advantage of *iterative optimization* computing

techniques. In this case, the designer defines the desired filter frequency response, and the algorithm begins generating successive approximations until the IIR filter coefficients converge (ideally) to an optimized design.

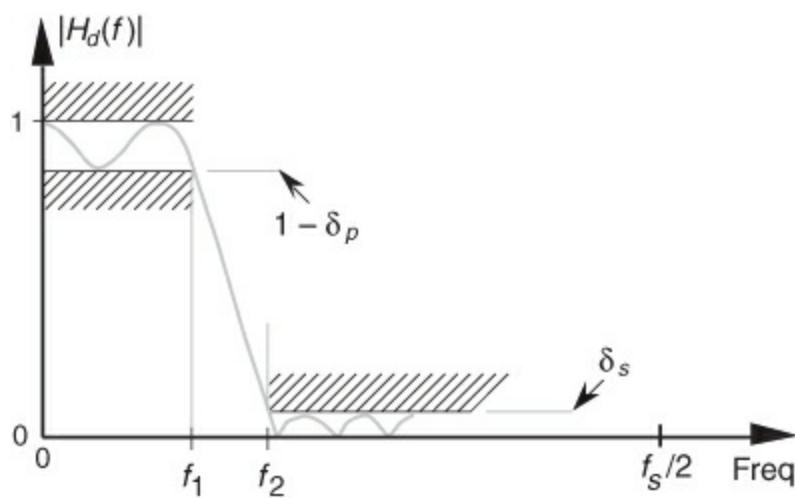
6.12 Optimized IIR Filter Design Method

The final class of IIR filter design methods we'll introduce is broadly categorized as optimization methods. These IIR filter design techniques were developed for the situation when the desired IIR filter frequency response was not of the standard lowpass, bandpass, or highpass form. When the desired response has an arbitrary shape, closed-form expressions for the filter's z -transform do not exist, and we have no explicit equations to work with to determine the IIR filter's coefficients. For this general IIR filter design problem, algorithms were developed to solve sets of linear, or nonlinear, equations on a computer. These software routines mandate that the designer describe, in some way, the desired IIR filter frequency response. The algorithms, then, assume a filter transfer function $H(z)$ as a ratio of polynomials in z and start to calculate the filter's frequency response. Based on some error criteria, the algorithm begins iteratively adjusting the filter's coefficients to minimize the error between the desired and the actual filter frequency response. The process ends when the error cannot be further minimized, or a predefined number of iterations has occurred, and the final filter coefficients are presented to the filter designer. Although these optimization algorithms are too mathematically complex to cover in any detail here, descriptions of the most popular optimization schemes are readily available in the literature [20,21,29–34].

The reader may ask, "If we're not going to cover optimization methods in any detail, why introduce the subject here at all?" The answer is that if we spend much time designing IIR filters, we'll end up using optimization techniques in the form of computer software routines most of the time. The vast majority of commercially available digital signal processing software packages include one or more IIR filter design routines that are based on optimization methods. When a computer-aided design technique is available, filter designers are inclined to use it to design the simpler lowpass, bandpass, or highpass forms even though analytical techniques exist. With all due respect to Laplace, Heaviside, and Kaiser, why plow through all the z -transform design equations when the desired frequency response can be applied to a software routine to yield acceptable filter coefficients in a few seconds?

As it turns out, using commercially available optimized IIR filter design routines is very straightforward. Although they come in several flavors, most optimization routines only require the designer to specify a few key amplitude and frequency values, and the desired order of the IIR filter (the number of feedback taps), and the software computes the final feedforward and feedback coefficients. In specifying a lowpass IIR filter, for example, a software design routine might require us to specify the values for δ_p , δ_s , f_1 , and f_2 shown in [Figure 6-43](#). Some optimization design routines require the user to specify the order of the IIR filter. Those routines then compute the filter coefficients that best approach the required frequency response. Some software routines, on the other hand, don't require the user to specify the filter order. They compute the minimum order of the filter that actually meets the desired frequency response.

Figure 6-43 Example lowpass IIR filter design parameters required for the optimized IIR filter design method.



6.13 A Brief Comparison of IIR and FIR Filters

The question naturally arises as to which filter type, IIR or FIR, is best suited for a given digital filtering application. That's not an easy question to answer, but we can point out a few factors that should be kept in mind. First, we can assume that the differences in the ease of design between the two filter types are unimportant. There are usually more important performance and implementation properties to consider than design difficulty when choosing between an IIR and an FIR filter. One design consideration that may be significant is the IIR filter's ability to simulate a predefined prototype analog filter. FIR filters do not have this design flexibility.

From a hardware standpoint, with so many fundamental differences between IIR and FIR filters, our choice must be based on those filter characteristics that are most and least important to us. For example, if we need a filter with exactly linear phase, then an FIR filter is the only way to go. If, on the other hand, our design requires a filter to accept very high data rates and slight phase nonlinearity is tolerable, we might lean toward IIR filters with their reduced number of necessary multipliers per output sample.

One caveat, though: Just because an FIR filter has, say, three times the number of multiplies per output sample relative to an IIR filter, that does not mean the IIR filter will execute faster on a programmable DSP chip. Typical DSP chips have a *zero-overhead looping* capability whose parallelism speeds the execution of *multiply and accumulate* (MAC) routines, with which FIR filtering is included. The code for IIR filtering has more data/coefficient pointer bookkeeping to accommodate than FIR filter code. So, if you're choosing between an IIR filter requiring K multiplies per output sample and an FIR filter needing $2K$ (or $3K$) multiplies per output sample, code both filters and measure their execution speeds.

[Table 6-1](#) presents a brief comparison of IIR and FIR filters based on several performance and implementation properties.

Table 6-1 IIR and Nonrecursive FIR Filter Characteristics Comparison

Characteristic	IIR	FIR (nonrecursive)
Number of necessary multiplications	Least	Most
Sensitivity to filter coefficient quantization	Can be high.* (24-bit coefficients needed for high-fidelity audio)	Very low (16-bit coefficients satisfy most FIR filter requirements)
Probability of overflow errors	Can be high*	Very low
Stability	Must be designed in	Guaranteed
Linear phase	No	Guaranteed**
Can simulate prototype analog filters	Yes	No
Required coefficient memory	Least	Most
Hardware filter control complexity	Moderate	Simple
Availability of design software	Good	Very good
Ease of design, or complexity of design software	Moderately complicated	Simple
Difficulty of quantization noise analysis	Most complicated	Least complicated
Supports adaptive filtering	With difficulty	Yes

* These problems can be minimized through cascade or parallel implementations.

** Guaranteed so long as the FIR coefficients are symmetrical (or antisymmetrical).

References

- [1] Churchill, R. V. *Modern Operational Mathematics in Engineering*, McGraw-Hill, New York, 1944, pp. 307–334.
- [2] Aseltine, J. A. *Transform Method in Linear System Analysis*, McGraw-Hill, New York, 1958, pp. 287–292.
- [3] Nixon, F. E. *Handbook of Laplace Transformation: Tables and Examples*, Prentice Hall, Englewood Cliffs, New Jersey, 1960.
- [4] Kaiser, J. F. “Digital Filters,” in *System Analysis by Digital Computer*, ed. by F. F. Kuo and J. F. Kaiser, John Wiley and Sons, New York, 1966, pp. 218–277.
- [5] Kaiser, J. F. “Design Methods for Sampled Data Filters,” *Proc. First Annual Allerton Conference on Circuit and System Theory*, 1963, [Chapter 7](#), pp. 221–236.
- [6] Ragazzini, J. R., and Franklin, G. F. *Sampled-Data Control Systems*, McGraw-Hill, New York, 1958, pp. 52–83.
- [7] Milne-Thomson, L. M. *The Calculus of Finite Differences*, Macmillan, London, 1951, pp. 232–251.
- [8] Truxal, J. G. *Automatic Feedback Control System Synthesis*, McGraw-Hill, New York, 1955, p. 283.
- [9] Blackman, R. B. *Linear Data-Smoothing and Prediction in Theory and Practice*, Addison

Wesley, Reading, Massachusetts, 1965, pp. 81–84.

- [10] Oppenheim, A., Schafer, R., and Buck, J. *Discrete-Time Signal Processing*, 2nd ed., Prentice Hall, Upper Saddle River, New Jersey, 1999, pp. 306–307.
- [11] Gold, B., and Jordan, K. L., Jr. “A Note on Digital Filter Synthesis,” *Proceedings of the IEEE*, Vol. 56, October 1968, p. 1717.
- [12] Rabiner, L. R., et al. “Terminology in Digital Signal Processing,” *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-20, No. 5, December 1972, p. 327.
- [13] Jackson, L. B. “On the Interaction of Roundoff Noise and Dynamic Range and Dynamic Range in Digital Filters,” *Bell System Technical Journal*, Vol. 49, February 1970, pp. 159–184.
- [14] Jackson, L. B. “Roundoff Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form,” *IEEE Trans. Audio Electroacoustics*, Vol. AU-18, June 1970, pp. 107–122.
- [15] Sandberg, I. W. “A Theorem Concerning Limit Cycles in Digital Filters,” *Proc. Seventh Annual Allerton Conference on Circuit and System Theory*, Monticello, Illinois, October 1969.
- [16] Ebert, P. M., et al. “Overflow Oscillations in Digital Filters,” *Bell System Technical Journal*, Vol. 48, November 1969, pp. 2999–3020.
- [17] Oppenheim, A. V. “Realization of Digital Filters Using Block Floating Point Arithmetic,” *IEEE Trans. Audio Electroacoustics*, Vol. AU-18, June 1970, pp. 130–136.
- [18] Kaiser, J. F. “Some Practical Considerations in the Realization of Linear Digital Filters,” *Proc. Third Annual Allerton Conference on Circuit and System Theory*, 1965, pp. 621–633.
- [19] Rabiner, L. R., and Rader, C. M., eds. *Digital Signal Processing*, IEEE Press, New York, 1972, p. 361.
- [20] Oppenheim, A. V., and Schafer, R. W. *Discrete Time Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1989, p. 406.
- [21] Rabiner, L. R., and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1975, p. 216.
- [22] Grover, D. “Subject: Re: How to Arrange the (Gain, Pole, Zero) of the Cascaded Biquad Filter.” Usenet group *comp.dsp* post, December 28, 2000.
- [23] Grover, D., and Deller, J. *Digital Signal Processing and the Microcontroller*, Prentice Hall, Upper Saddle River, New Jersey, 1998.
- [24] Stearns, S. D. *Digital Signal Analysis*, Hayden Book Co., Rochelle Park, New Jersey, 1975, p. 114.
- [25] Stanley, W. D., et al. *Digital Signal Processing*, Reston Publishing Co., Reston, Virginia, 1984, p. 191.
- [26] Williams, C. S. *Designing Digital Filters*, Prentice Hall, Englewood Cliffs, New Jersey, 1986, pp. 166–186.
- [27] Johnson, M. “Implement Stable IIR Filters Using Minimal Hardware,” *EDN*, April 14, 1983.
- [28] Oppenheim, A. V., Willsky, A. S., and Young, I. T. *Signals and Systems*, Prentice Hall, Englewood Cliffs, New Jersey, 1983, p. 659.
- [29] Deczky, A. G. “Synthesis of Digital Recursive Filters Using the Minimum P Error Criterion,”

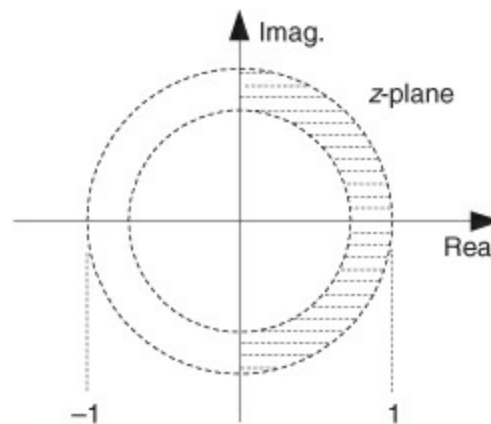
IEEE Trans. on Audio and Electroacoustics, Vol. AU-20, No. 2, October 1972, p. 257.

- [30] Steiglitz, K. "Computer-Aided Design of Recursive Digital Filters," *IEEE Trans. on Audio and Electroacoustics*, Vol. 18, No. 2, 1970, p. 123.
- [31] Richards, M. A. "Application of Deczky's Program for Recursive Filter Design to the Design of Recursive Decimators," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-30, October 1982, p. 811.
- [32] Parks, T. W., and Burrus, C. S. *Digital Filter Design*, John Wiley and Sons, New York, 1987, p. 244.
- [33] Rabiner, L., Graham, Y., and Helms, H. "Linear Programming Design of IIR Digital Filters with Arbitrary Magnitude Functions," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-22, No. 2, April 1974, p. 117.
- [34] Friedlander, B., and Porat, B. "The Modified Yule-Walker Method of ARMA Spectral Estimation," *IEEE Trans. on Aerospace Electronic Systems*, Vol. AES-20, No. 2, March 1984, pp. 158–173.

Chapter 6 Problems

6.1 Review the z -plane depiction in [Figure P6-1](#). Draw a rough sketch of the Laplace s -plane showing a shaded area (on the s -plane) that corresponds to the shaded circular band in [Figure P6-1](#).

Figure P6-1



6.2 Write the $H(z)$ z -domain transfer function equations for the filters described by the following difference equations:

(a) $y(n) = x(n) - y(n-2)$,

(b) $y(n) = x(n) + 3x(n-1) + 2x(n-2) - y(n-3)$,

(c) $y(n) = x(n) + x(n-1) + x(n-3) + x(n-4) - y(n-2)$.

6.3 Knowing the [order](#) of a digital filter is important information. It typically gives us a direct indication of the computational workload (number of additions and multiplications) necessary to compute a single filter output sample. State the order of the filters in Problem 6.2.

6.4 Write the $H(\omega)$ frequency response equations, in both polar and rectangular form, for the filters in Problem 6.2. By "polar form" we mean we want $H(\omega)$ expressed as a ratio of terms using $e^{-jk\omega}$, where k is an integer. By "rectangular form" we mean we want $H(\omega)$ expressed as a ratio in the

form of

$$\frac{a + jb}{c + jd}$$

where a , b , c , and d are cosine and/or sine functions whose arguments are $k\omega$.

(Note: This problem is *not* “busy work.” The rectangular form of $H(\omega)$ is the expression you would model using generic signal processing software to compute and plot a filter’s magnitude and phase response in the frequency domain.)

6.5 Considering the z -domain transfer function associated with a digital filter:

- (a) What does it mean if the filter has one or more poles outside the z -plane’s unit circle?
- (b) What does it mean if the filter has a zero lying exactly on the z -plane’s unit circle?

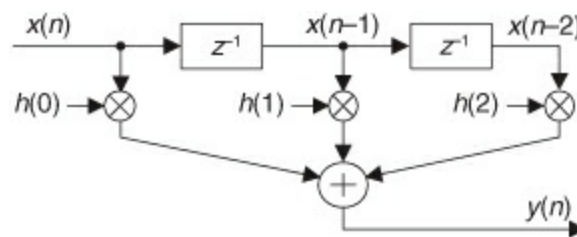
6.6 In the literature of DSP, we usually see filter transfer functions expressed in terms of z where z always has a negative exponent. But sometimes we see transfer functions in terms of z having positive exponents. For example, you might encounter an IIR filter’s transfer function expressed as

$$H(z) = \frac{z^2 + 0.3z + 1}{z^2 + 0.3z + 0.8}.$$

- (a) What is the transfer function expression equivalent to $H(z)$ in terms of z with z having negative-only exponents?
- (b) Is this IIR filter stable? Justify your answer.
- (c) Draw the Direct Form I structure (block diagram), showing the filter’s coefficients.
- (d) Draw the Direct Form II structure, showing the filter’s coefficients.

6.7 Although we didn’t need to use the z -transform to analyze the tapped-delay line (nonrecursive) FIR filters in [Chapter 5](#), we could have done so. Let’s try an FIR filter analysis example using the z -transform. For the filter in [Figure P6-7](#):

Figure P6-7



- (a) Write the time-domain difference equation describing the filter output $y(n)$ in terms of the $x(n)$ input and the $h(k)$ coefficients.
- (b) Write the z -transform of the $y(n)$ difference equation from Part (a).
- (c) Write the z -domain transfer function, $H(z) = Y(z)/X(z)$, of the filter.
- (d) What is the order of this FIR filter?

6.8 Thinking about IIR digital filters:

- (a) Is it true that to determine the frequency response of an IIR filter, we need to know both the filter’s time-domain difference equation *and* the impulse response of that filter? Explain your

answer.

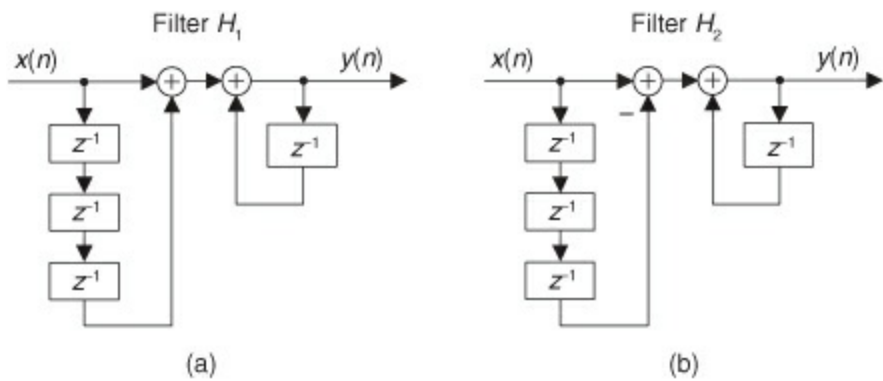
(b) If we know the $H(z)$ z -domain transfer function equation for a digital filter, what must we do to determine the frequency response of that filter?

6.9 Draw the Direct Form I and the Direct Form II block diagrams of the filter represented by the following z -domain transfer function:

$$H_{\text{as}}(z) = \left(\frac{1}{1 - 0.3z^{-1}} \right) \cdot (1 - 4z^{-1} + 2z^{-2}).$$

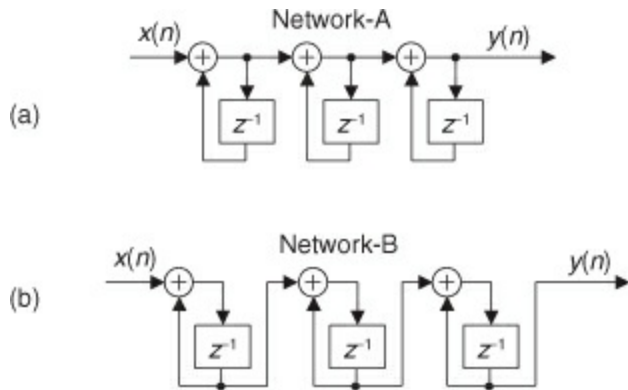
6.10 Consider the two filters in [Figure P6-10](#). (Notice the minus sign at the first adder in [Figure P6-10\(b\)](#).) Determine whether each filter is an IIR or an FIR filter. Justify your answers.

Figure P6-10



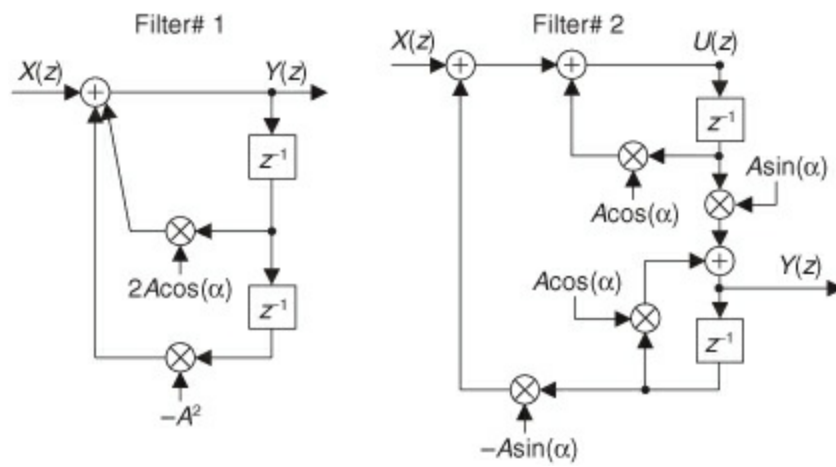
6.11 The author once read a design document describing how an engineer was tasked to implement Network A in [Figure P6-11\(a\)](#), using a programmable DSP chip, as part of a specialized digital filter. The engineer suggested that, due to the chip's internal architecture, for computational speed reasons Network B shown in [Figure P6-11\(b\)](#) should be used instead of Network A. He also stated that the frequency magnitude responses of the two networks are identical. Is that last statement true? Justify your answer.

Figure P6-11



6.12 Prove that the z -plane pole locations for the two filters in [Figure P6-12](#) are identical.

Figure P6-12

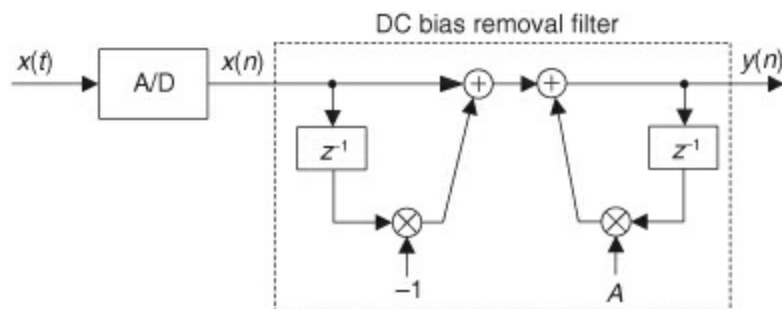


Hint: For Filter #2, write two different equations for $U(z)$ and set those equations equal to each other.

6.13 The discrete-sequence output of commercial analog-to-digital (A/D) converters is often contaminated with a DC bias (a constant-level amplitude offset). Stated in different words, even though the converter's analog $x(t)$ input signal's average value is zero, the converter's $x(n)$ output sequence may have a small nonzero average. As such, depending on the application, A/D converters are sometimes followed by an IIR filter shown in [Figure P6-13](#) that removes the DC bias level from the filter's $x(n)$ input sequence. (The coefficient A is a positive value slightly less than unity.)

- (a) Derive the z -domain transfer function of the DC bias removal filter.
- (b) Prove that the filter has a z -plane zero at $z = 1$, yielding the desired infinite attenuation at the cyclic frequency of zero Hz.
- (c) Draw the block diagram of the Direct Form II version of the DC bias removal filter.

Figure P6-13



6.14 Assume we have the software code to implement a notch filter (a filter that attenuates a very narrow band of frequencies and passes frequencies that are above and below the notch's ω_c center frequency), and the software documentation states the filter is defined by the following transfer function:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1 - 2 \cos(\omega_c) z^{-1} + z^{-2}}{1 - 2R \cos(\omega_c) z^{-1} + R^2 z^{-2}} = \frac{(1 - e^{j\omega_c} z^{-1})(1 - e^{-j\omega_c} z^{-1})}{(1 - R e^{j\omega_c} z^{-1})(1 - R e^{-j\omega_c} z^{-1})}.$$

- (a) If $R = 0.9$, draw the locations of the notch filter's poles and zeros on the z -plane in relation to the notch frequency ω_c .
- (b) Let's say we're processing the signal from a photodiode light sensor in our laboratory and our

signal's time samples are arriving at a sample rate of $f_s = 1.8$ kHz. Assume that 120 Hz *flicker* noise from fluorescent lights is contaminating our photodiode output signal. What would be the correct value for ω_c to use in the notch filter code to attenuate the 120 Hz noise? Show your work.

6.15 Show that for a 2nd-order FIR filter, whose z -domain transfer function is

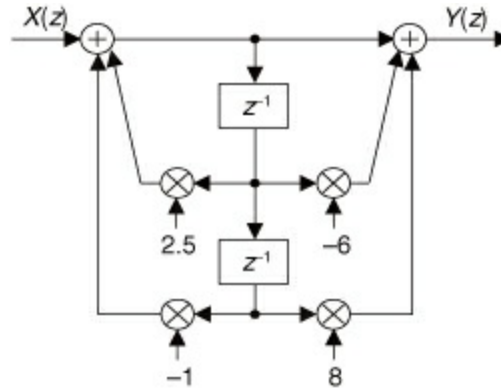
$$H(z) = 1 + Bz^{-1} + z^{-2},$$

the sum of the locations of the filter's two z -plane zeros is equal to $-B$.

6.16 Consider the filter in [Figure P6-16](#).

- Determine the z -domain transfer function, $H(z) = Y(z)/X(z)$, of the filter.
- Draw the z -plane pole/zero diagram of the filter.
- Using the notion of pole-zero cancellation, draw the block diagram of an exact equivalent, but simpler, filter having fewer multipliers than shown in [Figure P6-16](#).

Figure P6-16



6.17 Assume we have a digital filter (having real-valued coefficients) whose complex frequency response is the product of an $M(\omega)$ magnitude response and a $\theta(\omega)$ phase response as

$$H(\omega) = M(\omega)e^{j\theta(\omega)}$$

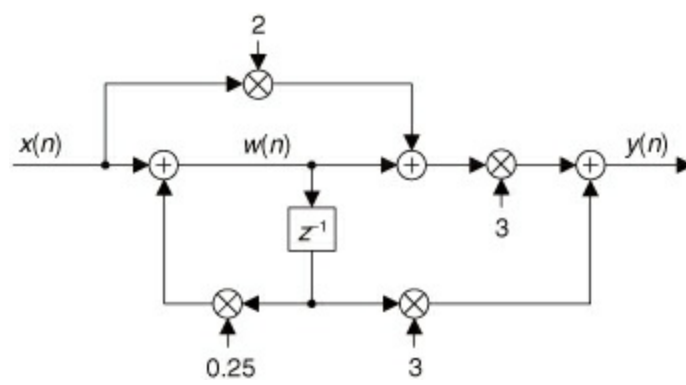
where ω is a normalized frequency variable (in the range of $-\pi$ to π , corresponding to a cyclic frequency range of $-f_s/2$ to $f_s/2$ Hz) measured in radians/sample. Is it possible to have such a real-coefficient filter whose $\theta(\omega)$ phase response is of the form

$$\theta(\omega) = C$$

where C is a nonzero constant? Explain your answer.

6.18 Determine the $H(z)$ transfer function of the recursive network in [Figure P6-18](#).

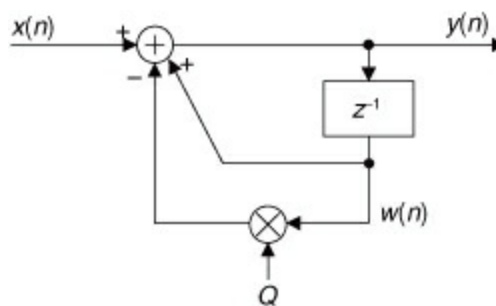
Figure P6-18



6.19 The recursive networks (networks with feedback) that we discussed in this chapter, if they're simple enough, can be analyzed with pencil and paper. This problem gives us practice in such an analysis and prompts us to recall the process of converting a geometric series into a *closed-form* equation.

- (a) Looking at the discrete network in [Figure P6-19](#), show that the $y(n)$ output is equal to D/Q for large values of time index n when the $x(n)$ input samples have a constant amplitude of D . (To keep the system stable, assume that Q is a positive number less than one and the network is “at rest” at time $n = 0$. That is, $w(0) = 0$.)

Figure P6-19



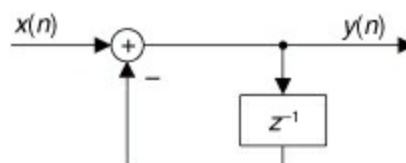
Hint: Write equations for $y(n)$ when $n = 0, 1, 2, 3, \dots$ etc., and develop a general series expression for the $y(n)$ output in terms of D , Q , and n . Next, use [Appendix B](#) to obtain a closed-form (no summation sign) expression for the $y(n)$ when n is a large number.

- (b) When we arrive at a solution to a problem, it's reassuring to verify (double-check) that solution using a different technique. Following this advice, determine the z -domain $H(z)$ transfer function of the network in [Figure P6-19](#) and show that its zero Hz (DC) gain is $1/Q$, verifying your solution to Part (a) of this problem.

- (c) Prove that the recursive network is stable if Q is in the range $0 < Q \leq 1$.

6.20 A discrete system that has at least one pole on the z -plane's unit circle is called a *discrete resonator*, such as the system in [Figure P6-20](#). Such resonators have impulse responses that oscillate indefinitely.

Figure P6-20



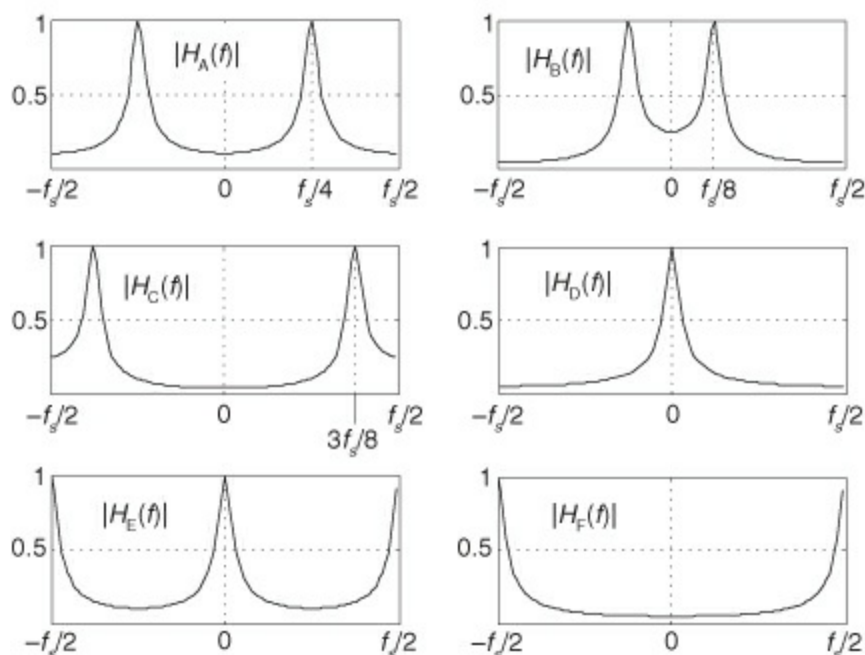
- (a) Draw the z -plane pole/zero diagram of the resonator in the figure.

- (b) At what frequency, measured in terms of the $x(n)$ input f_s sample rate, does the pole of this system reside?
- (c) Draw the time-domain impulse response of the system in [Figure P6-20](#).
- (d) Comment on how the frequency of the oscillating impulse response output samples relates to the system's pole location on the z -plane.

6.21 Given the following six difference equations for various digital filters, determine which equation is associated with which $|H_r(f)|$ filter frequency magnitude response in [Figure P6-21](#). Justify your answers.

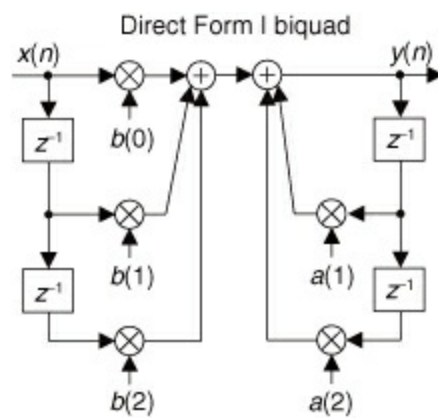
Figure P6-21

$$\begin{aligned}
 y_1(n) &= x(n) + 0.9y_1(n-1) \\
 y_2(n) &= x(n) - 0.9y_2(n-1) \\
 y_3(n) &= x(n) + 0.81y_3(n-2) \\
 y_4(n) &= x(n) - 0.81y_4(n-2) \\
 y_5(n) &= x(n) - 1.8\cos(\pi/4)y_5(n-1) - 0.81y_5(n-2) \\
 y_6(n) &= x(n) + 1.8\cos(\pi/4)y_6(n-1) - 0.81y_6(n-2)
 \end{aligned}$$



6.22 A standard 2nd-order IIR filter (a *biquad*) is shown in its Direct Form I structure in [Figure P6-22](#). Knowing the DC gain (the value $H(\omega)$ at $\omega = 0$ radians/sample) of a filter is critical information when we implement filtering using binary arithmetic. What is the DC gain of the filter in terms of the filter's coefficients?

Figure P6-22



6.23 Review the brief description of [allpass filters](#) in [Appendix F](#).

- (a) Prove that the 1st-order allpass filter, defined by the following $H_{ap}(z)$ transfer function, has an $|H_{ap}(\omega)|$ frequency magnitude response that is unity over its full operating frequency range of $-\pi \leq \omega \leq \pi$ radians/sample ($-f_s/2 \leq f \leq f_s/2$ Hz):

$$H_{ap}(z) = \frac{-K + z^{-1}}{1 - Kz^{-1}}.$$

Variable K is a real-valued scalar constant.

Hint: Rather than prove $|H_{ap}(\omega)| = 1$ for all ω , prove that the frequency magnitude response squared, $|H_{ap}(\omega)|^2$, is equal to unity for all ω .

- (b) Draw the Direct Form I and Direct Form II block diagrams of the $H(z)$ allpass filter.
- (c) Explain why the $H_{ap}(z)$ allpass filter can never have a transfer function *zero* on its z -plane's unit circle.

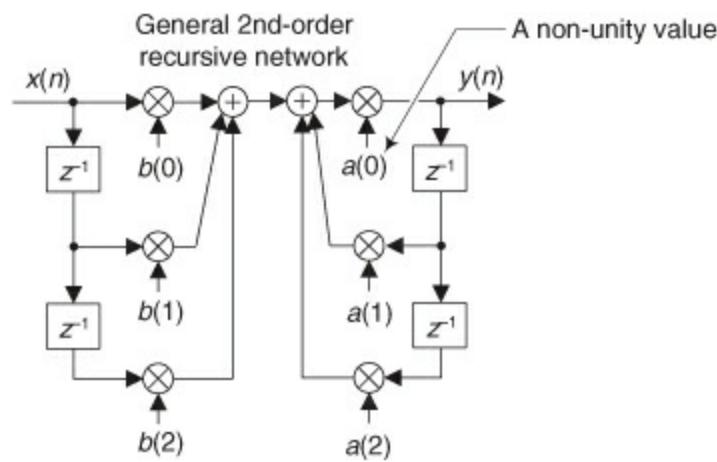
6.24 A simple 1st-order IIR filter, whose z -domain transfer function is

$$H_g(z) = \frac{0.9152 + 0.1889z^{-1}}{1 + 0.1127z^{-1}},$$

has been proposed for use in synthesizing (simulating) guitar music. Is the $H_g(z)$ filter a lowpass or a highpass filter? Justify your answer. [Karjalainen, M., et al. "Towards High-Quality Sound Synthesis of the Guitar and String Instruments," *International Computer Music Conference*, September 10–15, 1993, Tokyo, Japan.]

6.25 There are general 2nd-order recursive networks used in practice, such as that shown in [Figure P6-25](#), where the $a(0)$ coefficient is not unity. Assuming you need to analyze such a network, determine its z -domain transfer function that includes the $a(0)$ coefficient. Show your steps.

Figure P6-25



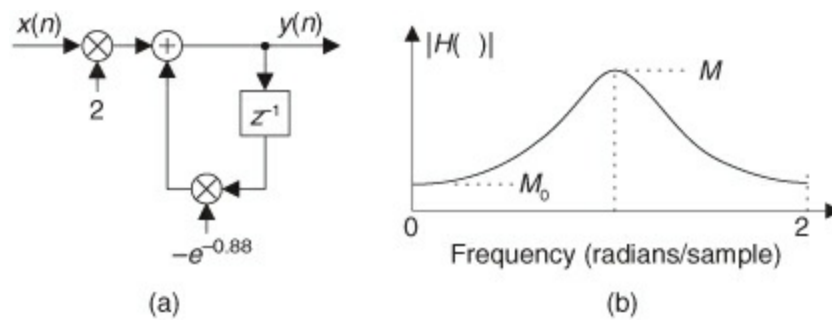
6.26 Consider the recursive highpass filter shown in [Figure P6-26\(a\)](#).

(a) Derive the $H(\omega)$ frequency response equation for the filter.

(b) What is the location of the filter's single z -plane pole?

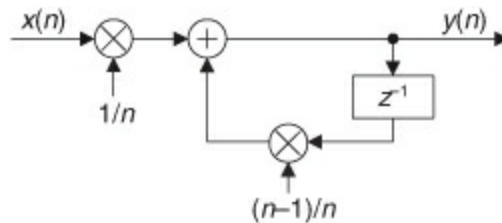
(c) The $|H(\omega)|$ frequency magnitude response of the filter is shown in [Figure P6-26\(b\)](#). What are the values of magnitudes M_0 and M_π ? Show your work.

Figure P6-26



6.27 The recursive network shown in [Figure P6-27](#) can be used to compute the N -point average of N input samples. Although this process works well, it has the disadvantage that as time index n (where $n = 1, 2, 3, 4, \dots$) increases, it requires the real-time computation of both the $1/n$ and $(n-1)/n$ coefficients upon the arrival of each new $x(n)$ input sample.

Figure P6-27



(a) A clever DSP engineer always tries to minimize computations. Show how to modify the network's diagram so that the real-time coefficient-computation workload is reduced.

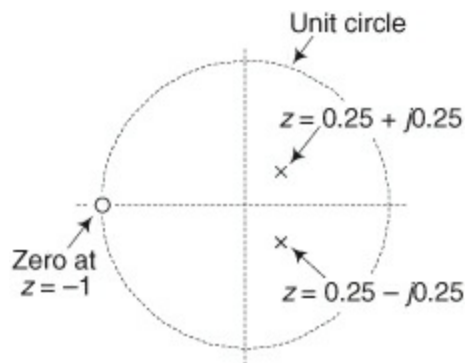
(b) Our N -point averager network has a feedback loop, with possible stability problems. Show how your solution to Part (a) of this problem is a stable network as n increases starting at $n = 1$.

6.28 Given the z -plane pole/zero plot, associated with a 2nd-order IIR digital filter, in [Figure P6-28](#):

(a) What is the $H(z)$ transfer function, in terms of z^{-1} and z^{-2} , of the [Figure P6-28](#) filter having two poles and a single zero on the z -plane? Show how you arrived at your answer.

- (b) Draw the Direct Form I block diagram of the $H(z)$ filter that implements the transfer function arrived at in Part (a) of this problem.
- (c) Draw a new block diagram of the $H(z)$ filter that eliminates one of the multipliers in the Direct Form I block diagram.

Figure P6-28



- 6.29 In the text's [Section 6.5](#) we learned to derive a filter transfer function based on knowing the locations of the filter's poles and zeros. We implied that the roots of polynomial P ,

$$P = z^2 + bz + c,$$

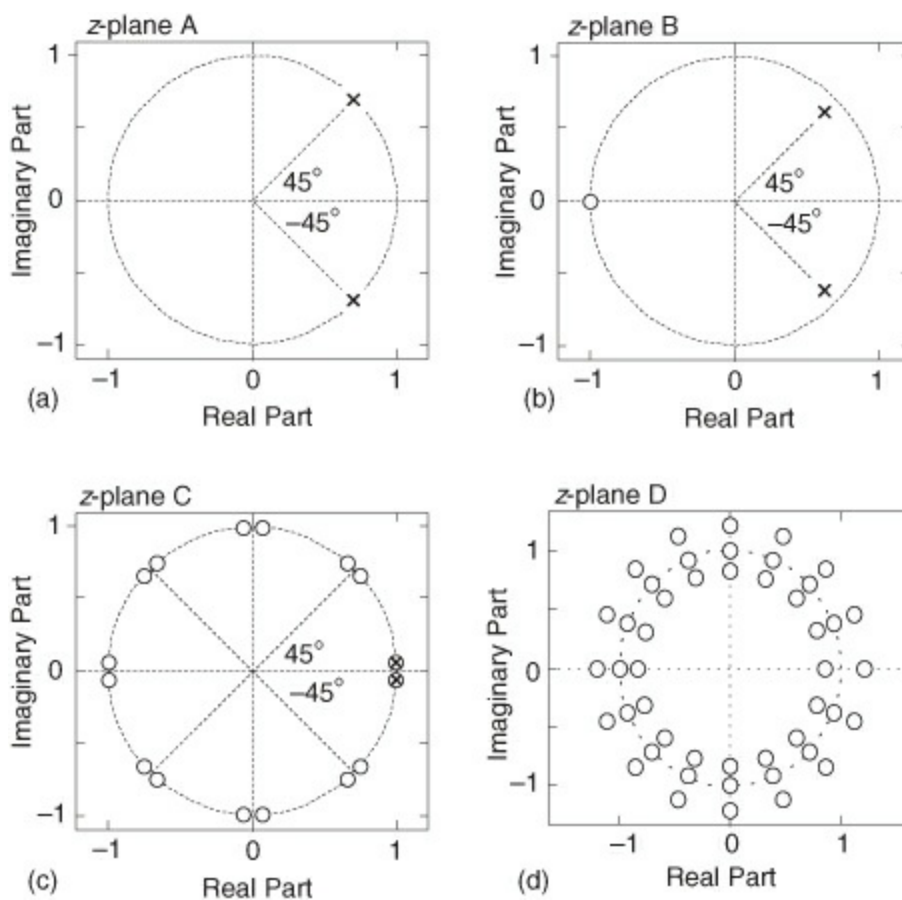
are equal to the roots of polynomial Q , where

$$Q = GP = Gz^2 + Gbz + Gc,$$

with variable G being a real-valued constant. Prove that the roots of P are indeed equal to the roots of Q .

- 6.30 Given the z -plane pole/zero plots in [Figure P6-30](#), associated with the $H(z)$ transfer functions of four digital filters, draw a rough sketch of the four filters' frequency magnitude responses over the frequency range of $-f_s/2$ to $f_s/2$, where f_s is the filter's input signal sample rate.

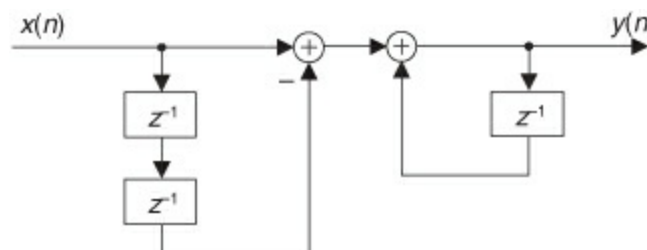
Figure P6-30



Note: The two poles, near $z = 1$ in [Figure P6-30\(c\)](#), are lying exactly on top of two zeros.

- 6.31** Assume that you must implement the lowpass $H(z)$ filter shown in [Figure P6-31](#). Good DSP engineers always simplify their digital networks whenever possible. Show a simplified block diagram of the filter, without changing the filter's frequency response, that has a reduced computational workload and reduced data storage (number of delay elements).

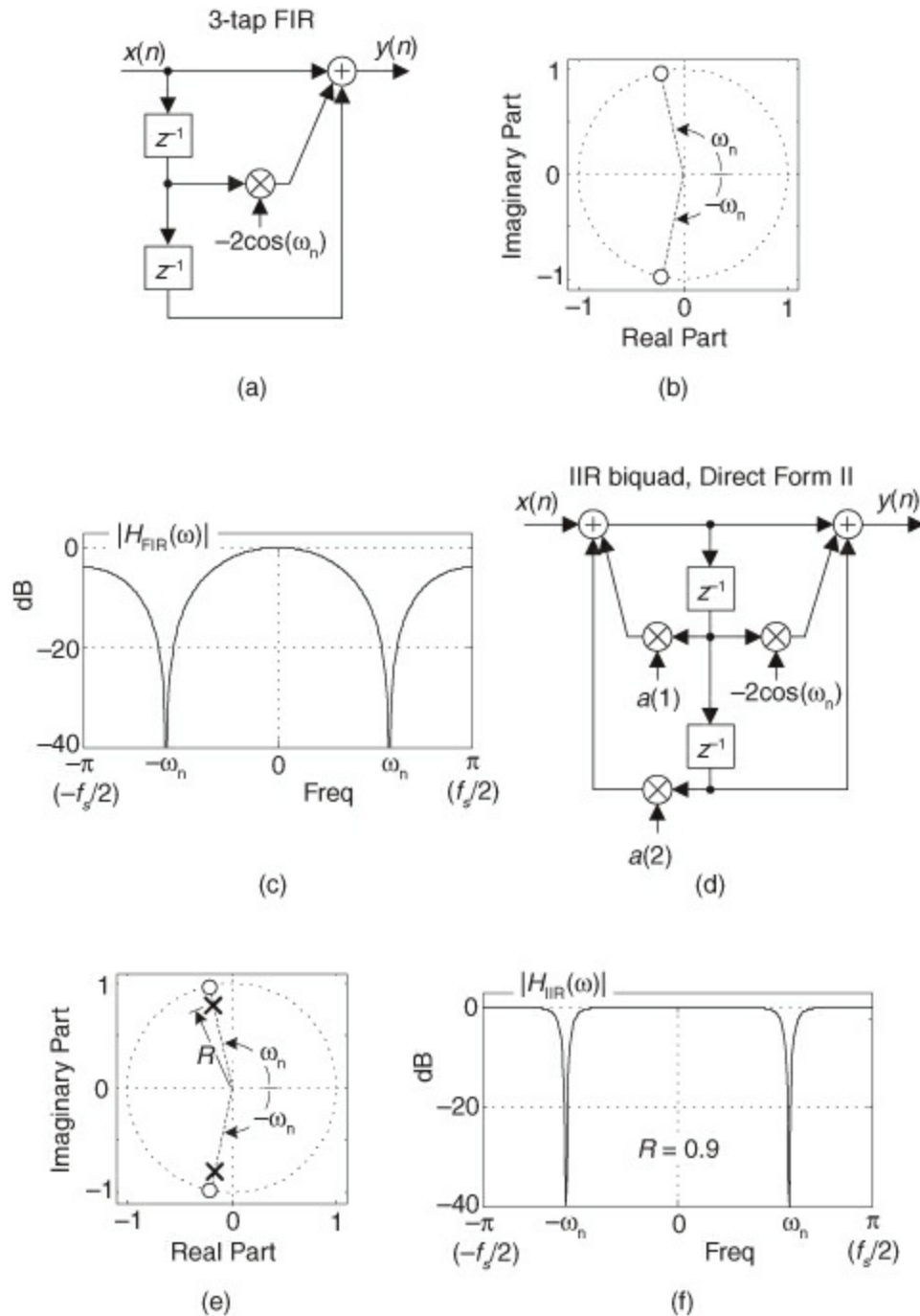
Figure P6-31



Hint: Study the filter's z -plane pole/zero diagram.

- 6.32** In [Chapter 5](#) we had a homework problem whose solution revealed that the 3-tap FIR notch filter in [Figure P6-32\(a\)](#) has complex conjugate z -plane zeros on the unit circle as shown in [Figure P6-32\(b\)](#). That efficient filter, useful for attenuating narrowband noise located at a normalized frequency of ω_n ($-\pi \leq \omega_n \leq \pi$), has a frequency magnitude response shown in [Figure P6-32\(c\)](#). If we want the FIR filter's stopband notches to be narrower, we can implement the 2nd-order IIR filter shown in [Figure P6-32\(d\)](#) that has conjugate z -plane poles at a radius of R just inside the unit circle as shown in [Figure P6-32\(e\)](#). The frequency magnitude response of the IIR notch filter is given in [Figure P6-32\(f\)](#). Here's the problem: Express the [Figure P6-32\(d\)](#) IIR filter's $a(1)$ and $a(2)$ coefficients, in terms of ω_n and R , that will place the z -plane poles as shown in [Figure P6-32\(e\)](#). Show your work.

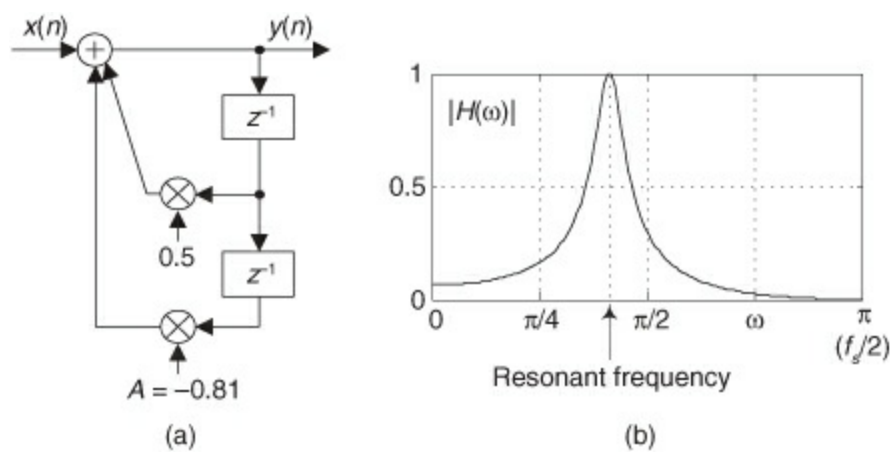
Figure P6-32



Hint: Recall Euler's identity: $2\cos(\theta) = (e^{j\theta} + e^{-j\theta})$.

6.33 Let's exercise our IIR filter analysis skills. Suppose your colleague proposes the 2nd-order IIR filter shown in [Figure P6-33\(a\)](#) to provide narrow passband filtering as shown in [Figure P6-33\(b\)](#). (The $|H(\omega)|$ frequency axis uses the discrete-signal frequency variable ω (radians/sample) with $\omega = \pi$ corresponding to a cyclic frequency of $f_s/2$ Hz.)

Figure P6-33



- (a) Is this 2nd-order IIR filter unconditionally stable?
- (b) Over what range of negative values of the A coefficient will the filter be stable?
- (c) For what negative value of A will the filter be *conditionally* stable (at least one pole on, and no poles outside, the unit circle)?
- (d) What is the resonant frequency (positive frequency) of the filter in terms of the f_s sample rate (in Hz) of the $x(n)$ input?

Hint: If the z -plane's positive-frequency pole is near the unit circle, think about how the angle of that pole is related to the filter's resonant frequency measured in Hz.

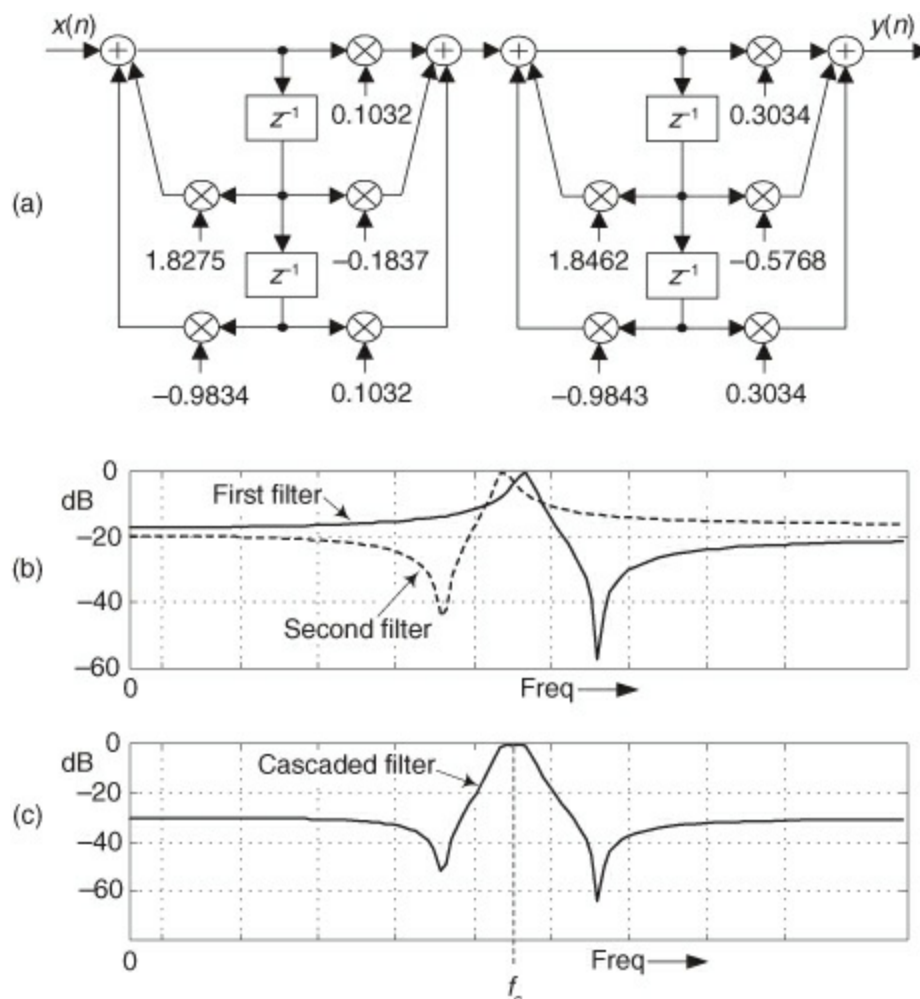
6.34 Think about a 4th-order (5-tap) tapped-delay line finite impulse response (FIR) filter whose z -domain transfer function is

$$H(z) = b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3} + b_4z^{-4}.$$

Assume the filter has real-valued b_k coefficients and that the filter is a linear-phase filter. If one of the filter's z -plane zeros has a value of $z_0 = 0.5657 + j0.5657$, what are the values of the other three z -plane zeros of this filter?

6.35 Here's an interesting problem. As of this writing, in an application note on their website (www.zilog.com), the skilled folks at Zilog Inc. describe a multistage digital bandpass filter used to detect the pitch (frequency) of a musical tone. A two-stage Direct Form II version, where each stage is a 2nd-order IIR filter, of this detection system is the cascaded bandpass filter shown in [Figure P6-35\(a\)](#). The frequency magnitude responses of the first and second filters, over the positive frequency range, are provided in [Figure P6-35\(b\)](#), and the combined (cascaded) frequency magnitude response is provided in [Figure P6-35\(c\)](#).

Figure P6-35



(a) Given that the sample rate of the signal is $f_s = 8000$ samples/second, what musical note will the [Figure P6-35\(a\)](#) two-stage bandpass filter detect? That is, what musical note is closest to the f_c center frequency of the two-stage filter's passband in [Figure P6-35\(c\)](#)? Explain how you arrived at your answer. For your convenience, the frequencies of several musical notes of an equal-tempered scale are provided in the following table.

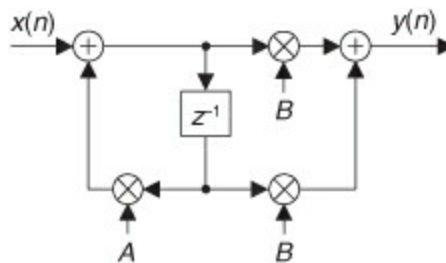
Table P6-1 Musical Note Frequencies

Musical note	Frequency (Hz)	Musical note	Frequency (Hz)
C4	261.626 (Middle C)	G#4	415.305
C#4	277.183	A4	440.0 (Concert A)
D4	293.665	A#4	466.164
D#4	311.127	B4	493.883
E4	329.628	C5	523.251
F4	349.228	C#5	554.365
F#4	369.994	D5	587.330
G4	391.995	D#5	622.254

(b) Finally, are the two 2nd-order IIR filters stable? Explain how you arrived at your answer.

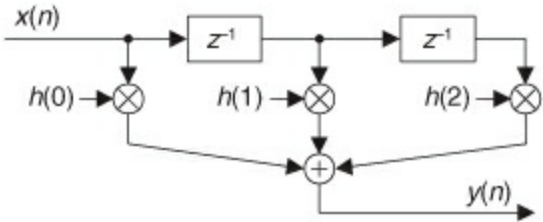
6.36 Consider the Direct Form II IIR filter shown in [Figure P6-36](#), which requires three multiplies per filter output sample. Smart DSP engineers reduce computations wherever possible. Draw a block diagram of a filter equivalent to that in [Figure P6-36](#) that requires fewer than three multiplies per filter output sample.

Figure P6-36



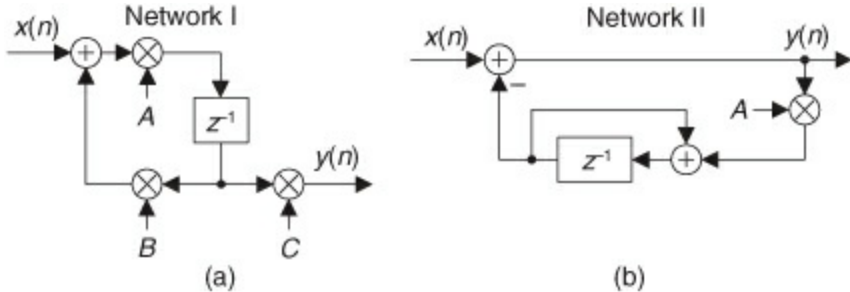
6.37 In high-speed, hardware-only, linear-phase filtering, the transposed structure of a tapped-delay line FIR filter is often preferred over a traditional tapped-delay line FIR filter. That's because the parallel structure of transposed FIR filters reduces the time required to perform multiple addition operations. Draw the transposed structure of the traditional FIR filter in [Figure P6-37](#). In your solution, make sure the $x(n)$ input is on the left-hand side.

Figure P6-37



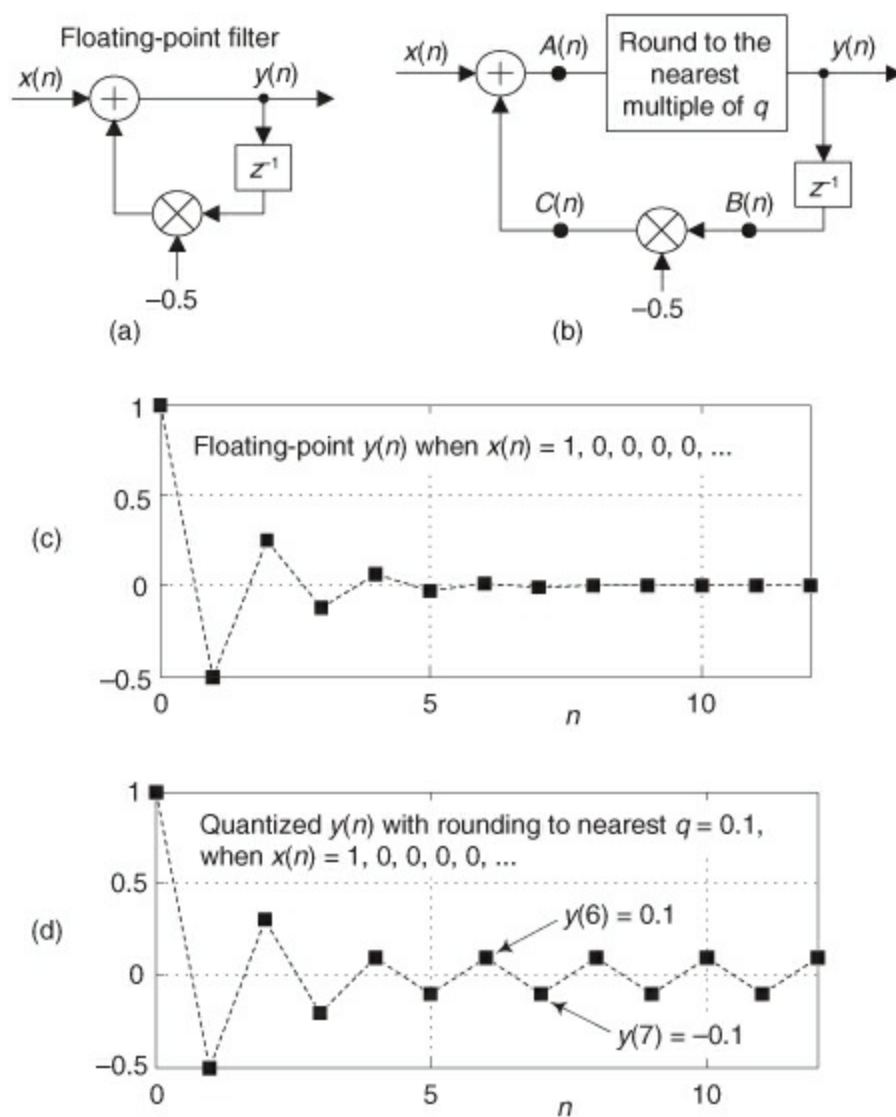
6.38 Draw the transposed structures of the networks in [Figure P6-38](#). In your solutions, make sure the $x(n)$ inputs are on the left-hand side.

Figure P6-38



6.39 In the text we discussed the problem of *limit cycles* in IIR filters when finite-precision values (finite binary word width) are used to represent data values. To reiterate that concept, the unit-sample impulse response of the 1st-order IIR filter in [Figure P6-39\(a\)](#) is shown in [Figure P6-39\(c\)](#). That impulse response was computed using the very high precision of a 64-bit floating-point binary number system within the filter. In [Figure P6-39\(c\)](#) we see that this stable IIR filter's $y(n)$ impulse response properly decays toward zero amplitude as time advances.

Figure P6-39



In fixed-point binary filter implementations, if rounding is used to limit the binary word width (the precision of data sample values) at the output of the filter's adder, the ill effects of *limit cycles* may occur. This rounding operation is shown in [Figure P6-39\(b\)](#) where the $y(n]$ output is rounded to a value that is a multiple of a rounding precision factor whose value is q . If rounding to the nearest $q = 0.1$ value is implemented, the filter's impulse response exhibits unwanted limit cycles as shown in [Figure P6-39\(d\)](#), where the $y(n]$ impulse response continually oscillates between ± 0.1 as time advances.

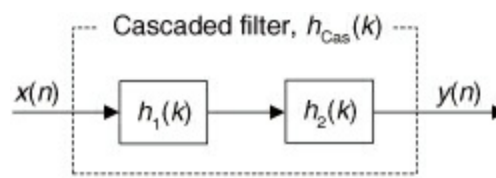
- (a) Reducing the value of the rounding precision factor q is supposed to help reduce the level of the unwanted limit cycle oscillations. Plot the unit-sample impulse response of the quantizing filter in [Figure P6-39\(b\)](#) when $q = 0.05$.

Note: If an $A(n]$ data value is exactly between two multiples of q , round away from zero.

- (b) Comparing [Figure P6-39\(c\)](#), [Figure P6-39\(d\)](#), and your solution from the above Part (a), make a statement regarding how the peak-to-peak amplitude of the quantizing filter's limit cycle behavior is related to the value of the rounding precision factor q .

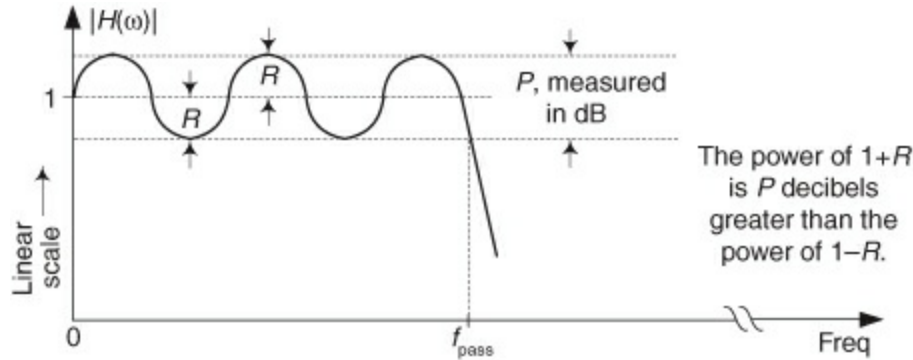
6.40 Given the $h_1(k]$ and $h_2(k]$ impulse responses of the two filters in [Figure P6-40](#), what is the impulse response of the $h_{\text{Cas}}(k]$ cascaded combination filter?

Figure P6-40



6.41 Here's a problem whose solution may, someday, be useful to the reader. Many commercial digital filter design software packages require the user to specify a *desired* filter's maximum passband ripple, in terms of a linear *peak deviation* parameter represented by R , for a lowpass filter magnitude response in [Figure P6-41](#).

Figure P6-41

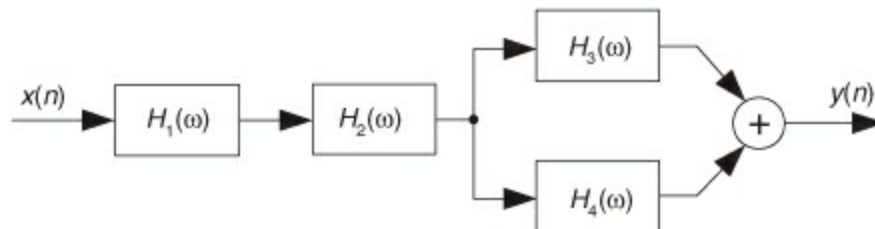


(a) Let's say that in a lowpass filter design effort, we only know the desired passband ripple specified in terms of a peak-peak logarithmic (dB) parameter P shown in [Figure P6-41](#). If $P = 2$ dB, what is R ? Stated in different words, if we only have the $P = 2$ dB desired passband ripple value available to us, what R value must we specify in our filter design software? Show how you arrived at your solution.

(b) Given your solution to the above Part (a), now derive a general equation that defines the linear R deviation parameter in terms of the logarithmic (dB) peak-peak passband ripple parameter P .

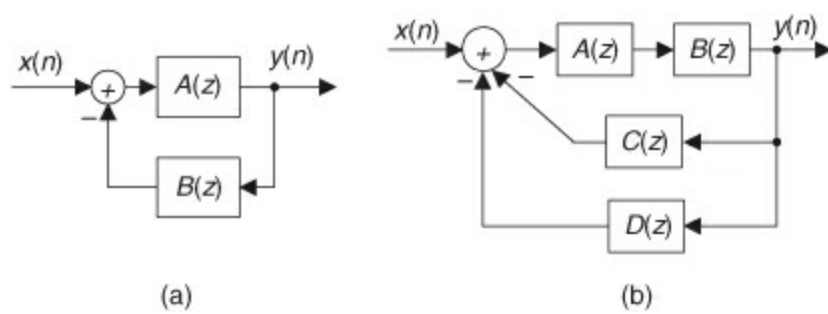
6.42 Many digital filters are implemented as both cascaded (series) and parallel combinations of subfilters. Given the four individual $H_k(\omega)$ subfilter frequency responses in [Figure P6-42](#), what is the equation for the overall frequency response of this combination of subfilters in terms of $H_1(\omega)$, $H_2(\omega)$, $H_3(\omega)$, and $H_4(\omega)$?

Figure P6-42



6.43 Many feedback systems can be reduced to the form of the *generic feedback* system shown in [Figure P6-43\(a\)](#).

Figure P6-43



- (a) Prove that the z -domain transfer function of the feedback system in [Figure P6-43\(a\)](#) is the following expression:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{A(z)}{1 + A(z)B(z)}.$$

Note: The above $H(z)$ expression is well known, particularly in the field of digital control systems, because it is encountered so often in practice.

- (b) If we replace the z variable in $H(z)$ with $e^{j\omega}$, we obtain an $H(\omega)$ equation, describing the frequency response of the system in [Figure P6-43\(a\)](#), whose generic form is

$$H(\omega) = \frac{Y(\omega)}{X(\omega)} = \frac{A(\omega)}{1 + A(\omega)B(\omega)}.$$

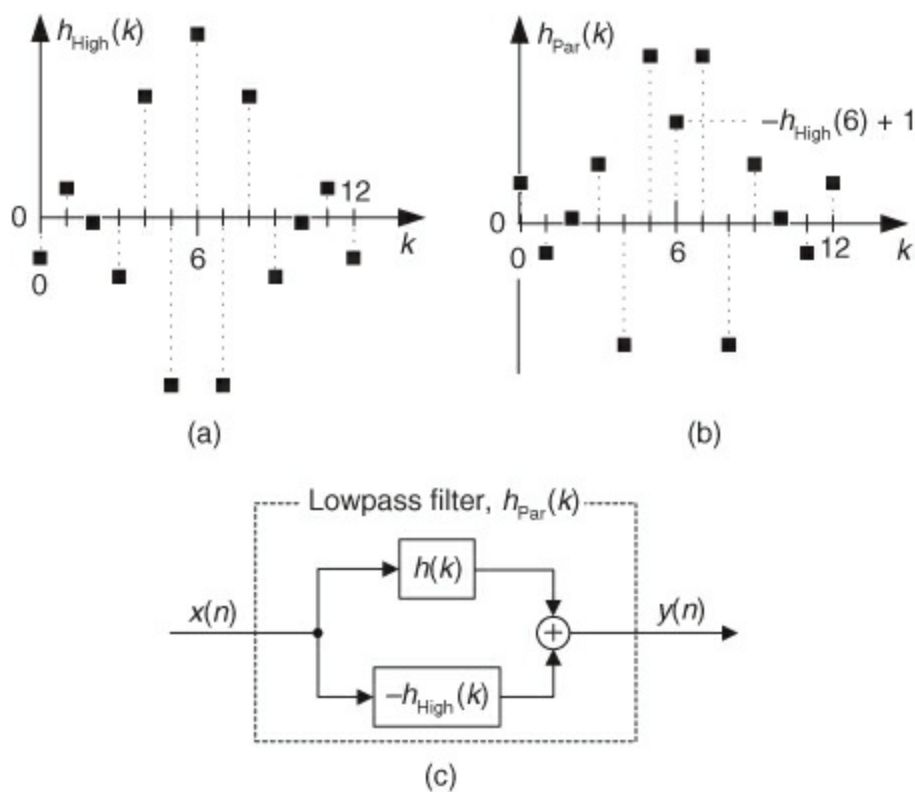
(Notice that we don't use the $e^{j\omega}$ term, for example $H(e^{j\omega})$, in our notation for a frequency response. We use the standard $H(\omega)$ notation instead.) With the above $H(\omega)$ equation in mind, what is the expression for the $H(\omega)$ frequency response of the system shown in [Figure P6-43\(b\)](#)?

Hint: Use the principles of cascaded and parallel subsystems to obtain a simplified network structure.

- 6.44** In the text we discussed the analysis of digital filters comprising the parallel combination of two subfilters. Using a highpass filter whose impulse response is the $h_{\text{High}}(k)$ samples in [Figure P6-44\(a\)](#), we can implement a lowpass filter if we're able to build a parallel network whose impulse response is the $h_{\text{Par}}(k)$ samples in [Figure P6-44\(b\)](#). The parallel network's $h_{\text{Par}}(k)$ samples are defined by

$$h_{\text{Par}}(k) = \begin{cases} -h_{\text{High}}(k), & \text{when } k \neq 6 \\ -h_{\text{High}}(k) + 1, & \text{when } k = 6. \end{cases}$$

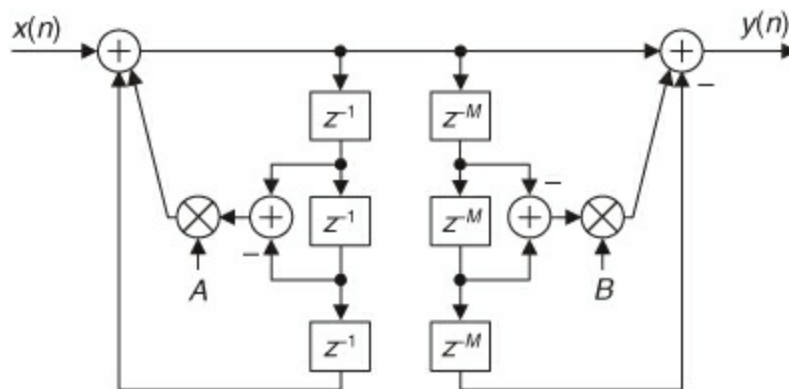
Figure P6-44



- (a) If the parallel lowpass filter network is that shown in [Figure P6-44\(c\)](#), what is the impulse response of the $h(k)$ subfilter?
- (b) Draw the parallel lowpass filter network showing what processing elements are in the $h(k)$ subfilter block.

6.45 Assume we are given the lowpass filter shown in [Figure P6-45](#) and, based on the IIR discussion in the text's [Section 6.9](#), we must *scale* the filter to reduce its passband gain without changing its frequency response shape. Draw a block diagram of the scaled filter.

Figure P6-45



6.46 You're working on a project to upgrade an analog temperature-sensing and processing system. Your job is to design a digital integrator, to replace an analog integrator whose Laplace s -domain transfer function is

$$H(s) = \frac{1}{s},$$

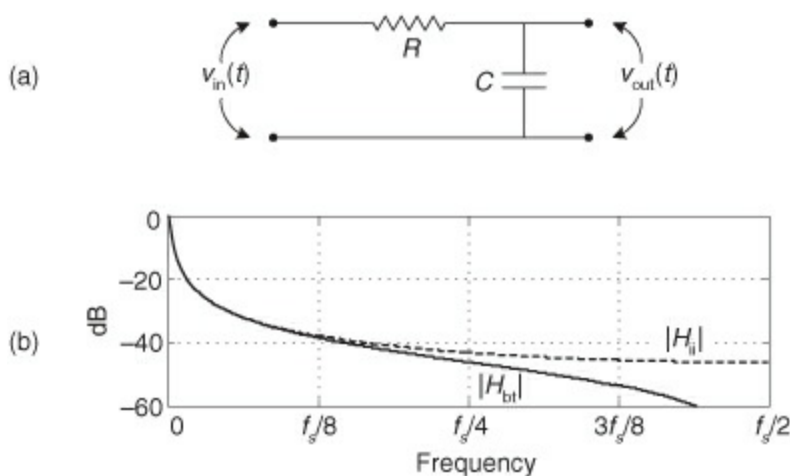
that will receive a new temperature sample once every 2 seconds. Because ideal integrators have a frequency magnitude response of zero at high frequencies, your digital integrator must have a frequency magnitude response less than 0.01 at $f_s/2$.

- (a) What is the z -domain transfer function of a digital integrator replacement for $H(s)$ designed using the impulse invariance Method 2 design technique?
- (b) What is the z -domain transfer function of a digital integrator designed using the bilinear transform design technique?
- (c) Verify that each of your digital integrators has a z -plane pole at the same frequency at which the $H(s)$ analog integrator had an s -plane pole.
- (d) Which of the two digital integrators, from Part (a) and Part (b), will you submit as your final design, and why?

6.47 Due to its simplicity, the 1st-order analog lowpass filter shown in [Figure P6-47\(a\)](#) is often used to attenuate high-frequency noise in a $v_{in}(t)$ input signal voltage. This lowpass filter's s -domain transfer function is

$$H(s) = \frac{1}{1 + RCs}.$$

Figure P6-47



- (a) Determine a digital filter's $H_{ii}(z)$ z -domain transfer function that simulates $H(s)$, using the impulse invariance Method 2 process. Draw the digital filter's Direct Form II block diagram (structure) where the coefficients are in terms of R and C . For simplicity, assume that $t_s = 1$.
- (b) Determine a digital filter's $H_{bt}(z)$ z -domain transfer function that simulates $H(s)$, using the bilinear transform process. Draw the digital filter's Direct Form II block diagram where the coefficients are in terms of R and C . Again, assume that $t_s = 1$.
- (c) When properly designed, the filters' normalized frequency magnitude responses, $|H_{ii}|$ and $|H_{bt}|$, are those shown in [Figure P6-47\(b\)](#) (plotted on a logarithmic vertical scale). Why does the $|H_{bt}|$ response have such large attenuation at high frequencies?

6.48 A 1st-order analog highpass filter's s -domain transfer function is

$$H(s) = \frac{s}{s + \omega_0}.$$

Determine a digital filter's $H(z)$ z -domain transfer function that simulates $H(s)$ using the bilinear transform process. Given that frequency $\omega_0 = 62.832$ radians/second, assume that the digital filter's sample rate is $f_s = 100$ Hz. Manipulate your final $H(z)$ expression so that it is in the following

form:

$$H(z) = \frac{A + Bz^{-1}}{1 + Cz^{-1}}$$

where A , B , and C are constants. The above $H(z)$ form enables convenient modeling of the digital filter's transfer function using commercial signal processing software.

- 6.49** Let's plow through the algebra to design a 2nd-order digital IIR filter that approximates an analog lowpass filter. Assume the filter's s -domain transfer function is

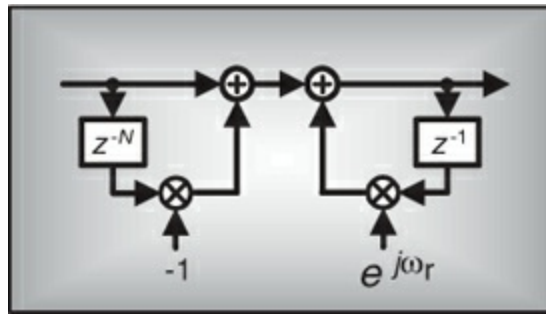
$$H(s) = \frac{5}{s(s - 0.8)}$$

and the digital filter's sample rate is 1000 samples/second. Derive, using the bilinear transform, the z -domain transfer function equation of the discrete filter that simulates the above $H(s)$ continuous lowpass filter.

- 6.50** Let's say that your colleague has designed a prototype analog lowpass filter whose *cutoff* frequency is 3.8 kHz. (By "cutoff frequency" we mean the frequency where the lowpass filter's magnitude response is 3 dB below its average passband magnitude response.) Next, assume your colleague wants you to use the bilinear transform method to design a digital filter whose performance is equivalent to that of the analog filter when the sample rate is $f_s = 11$ kHz.

- (a) Given that the analog lowpass filter's f_a cutoff frequency is 3.8 kHz, what will be the f_d cutoff frequency of the digital lowpass filter in Hz?
- (b) Given that we want the digital lowpass filter's cutoff frequency to be *exactly* 3.8 kHz, the prototype analog filter will have to be redesigned. What should be the f_a cutoff frequency of the new analog lowpass filter?

Chapter Seven. Specialized Digital Networks and Filters



We begin this chapter by presenting three useful digital networks—differentiators, integrators, and matched filters—that are common in the world of DSP. Beyond generic applications that require derivatives to be computed, differentiators are a key component of FM (frequency modulation) demodulation. A common application of integration is computing the integral of stock market prices over some period of days to determine trends in stock price data. Matched filters are used to detect the arrival of a specific discrete signal sequence, such as a radar return signal.

Later in this chapter we introduce two specialized implementations of finite impulse response (FIR) filters: interpolated lowpass FIR filters and frequency sampling filters. The common thread between these two FIR filter types is that they're lean mean filtering machines. They wring every last drop of computational efficiency from a guaranteed-stable linear-phase filter. In many lowpass filtering applications these FIR filter types can attain greatly reduced computational workloads compared to the traditional Parks-McClellan-designed FIR filters discussed in [Chapter 5](#).

We discuss this chapter's specialized digital networks and FIR filters now because their behavior will be easier to understand using the z -transform concepts introduced in the last chapter.

7.1 Differentiators

This section focuses on simple tapped-delay line (FIR) differentiators. The idea of differentiation is well defined in the world of continuous (analog) signals, but the notion of derivatives is not strictly defined for discrete signals. However, fortunately we *can* approximate the calculus of a derivative operation in DSP. To briefly review the notion of differentiation, think about a continuous sinewave, whose frequency is ω radians/second, represented by

(7-1)

$$x(t) = \sin(2\pi ft) = \sin(\omega t).$$

The derivative of that sinewave is

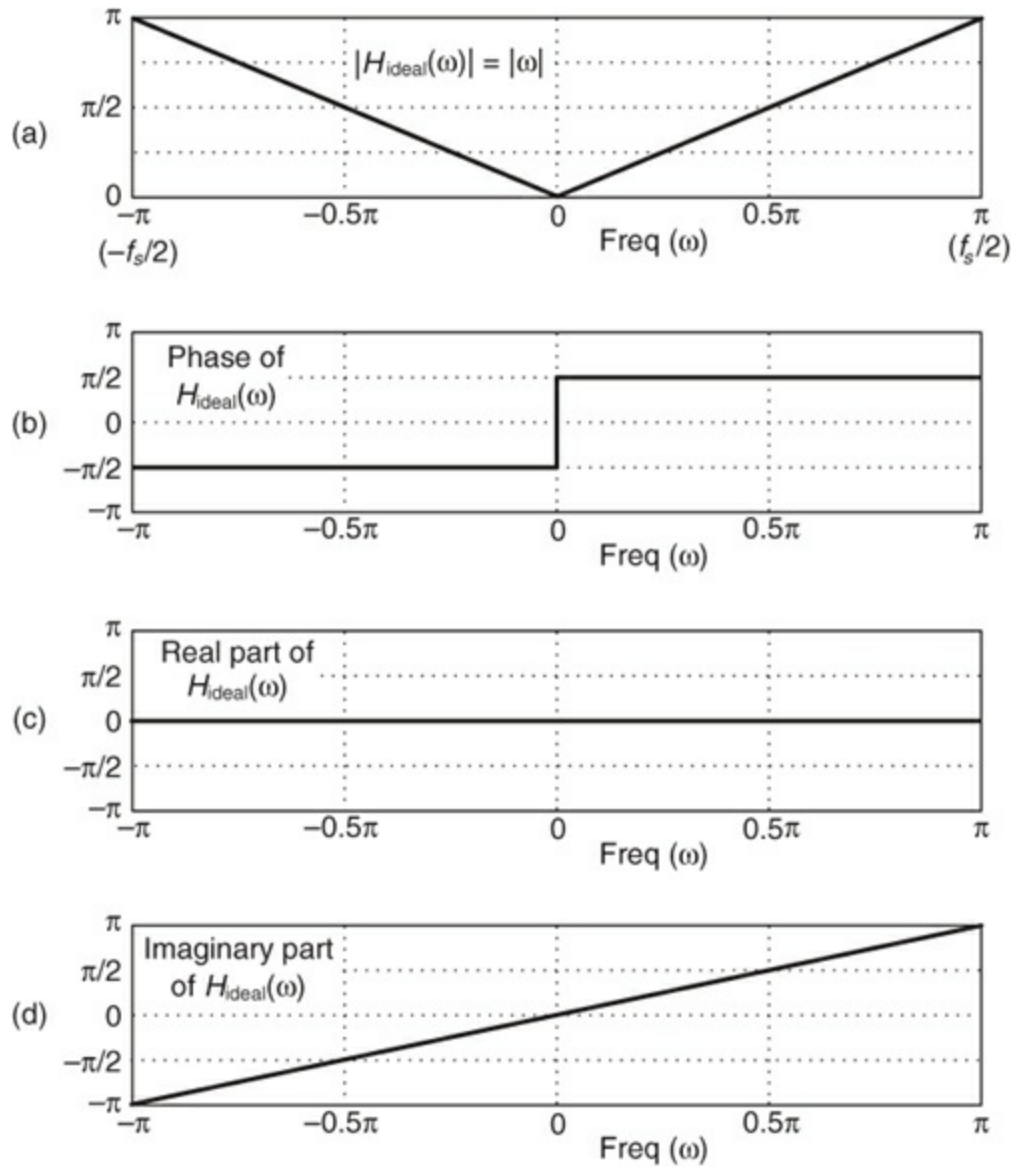
(7-1')

$$\frac{dx(t)}{dt} = \omega \cos(\omega t).$$

So the derivative of a sinewave is a cosine wave whose amplitude is proportional to the original sinewave's frequency. [Equation \(7-1\)](#) tells us that an ideal digital differentiator's frequency magnitude response is a straight line linearly increasing with frequency ω as shown in [Figure 7-1\(a\)](#). The differentiator's phase is that shown in [Figure 7-1\(b\)](#), where the digital frequency $\omega = \pi$ radians/sample is equivalent to half the signal data sample rate in Hz ($f_s/2$).

Figure 7-1 Ideal differentiator frequency response: (a) magnitude; (b) phase in radians; (c) real part;

(d) imaginary part.



Given the magnitude and phase response of our ideal digital differentiator, we can draw the real and imaginary parts of its frequency response as shown in [Figures 7-1\(c\)](#) and [7-1\(d\)](#). (The real part of the response is identically zero.) What we can say is that our ideal differentiator has the simple frequency response described, in rectangular form, by

(7-2)

$$H_{ideal}(\omega) = j\omega.$$

With these thoughts in mind, let's see how we can build a digital differentiator. We start by exploring two simple discrete-time FIR (nonrecursive) differentiators: a *first-difference* and a *central-difference* differentiator. They are computationally simple schemes for approximating the derivative of an $x(n)$ time-domain sequence with respect to time.

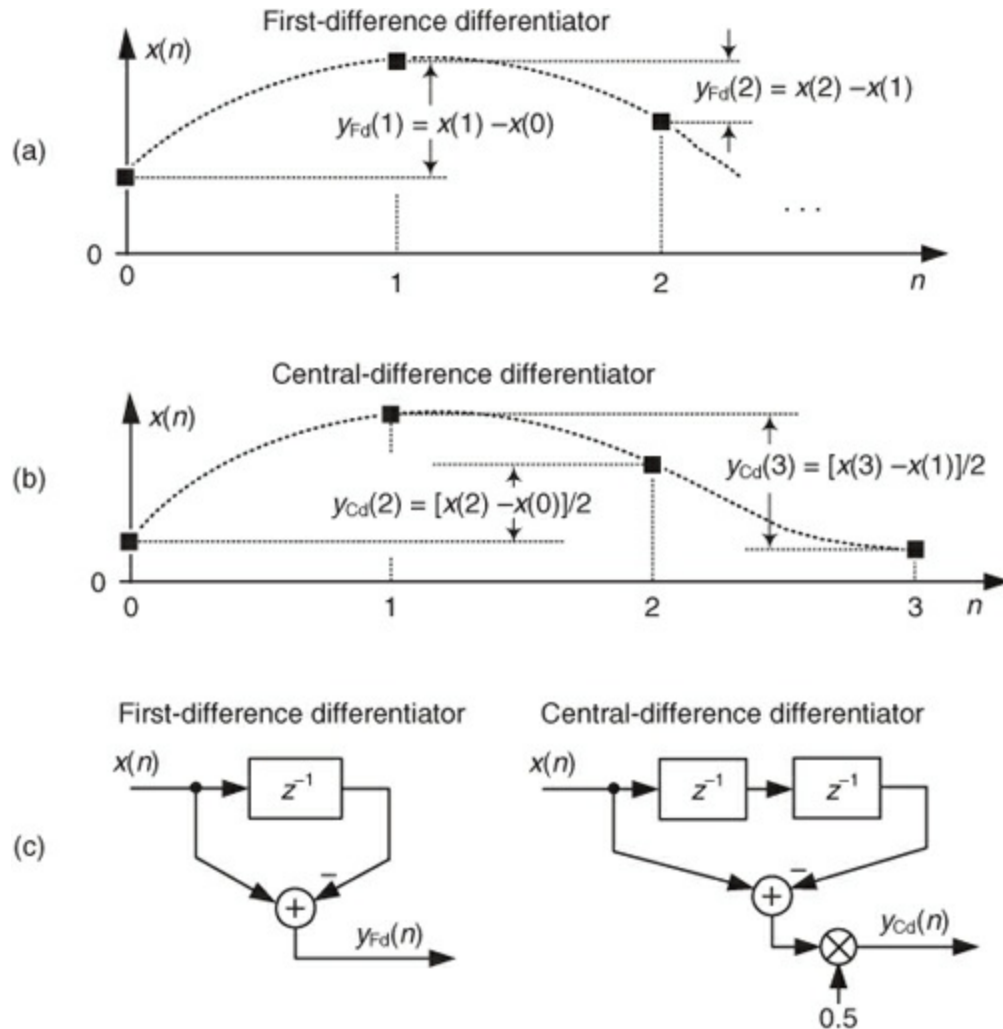
7.1.1 Simple Differentiators

With respect to the $x(n)$ samples in [Figure 7-2\(a\)](#), the *first-difference* differentiator is simply the process of computing the difference between successive $x(n)$ samples. (While DSP purists prefer to use the terminology *digital differencer*, we'll use the popular term *differentiator* for our purposes.) If we call $y_{Fd}(n)$ the output of a first-difference differentiator, then $y_{Fd}(n)$ is

(7-3)

$$y_{Fd}(n) = x(n) - x(n-1).$$

Figure 7-2 Simple differentiators.



For the $x(n)$ samples in [Figure 7-2\(b\)](#), the *central-difference* differentiator is the process of computing the average difference between alternate pairs of $x(n)$ samples. If we call $y_{Cd}(n)$ the output of a central-difference differentiator, then $y_{Cd}(n)$ is

(7-4)

$$\begin{aligned} y_{Cd}(n) &= \frac{[x(n) - x(n-1)] + [x(n-1) - x(n-2)]}{2} \\ &= [x(n) - x(n-2)]/2. \end{aligned}$$

The two simple differentiators are implemented with tapped-delay line structures, just like our standard FIR filters in [Chapter 5](#), as shown in [Figure 7-2\(c\)](#). (In fact, the two differentiators are merely two different forms of a comb filter, as discussed in detail in [Section 7.5.1](#), and this is why differentiators are often called *differentiating filters*.) So what's the difference (no pun intended) between these two simple differentiators? They are different with respect to their frequency responses, which we now investigate.

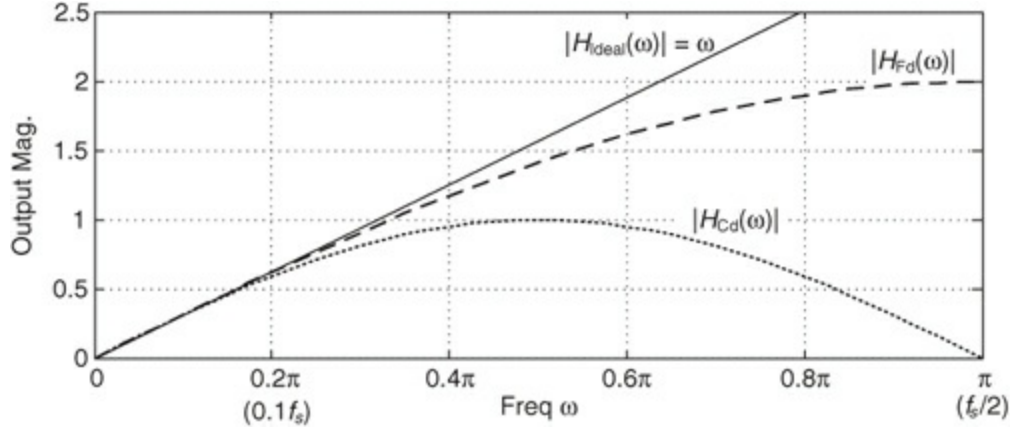
The first-difference differentiator is the most fundamental notion of digital differentiation, i.e., computing the difference between successive samples of a discrete sequence. The problem with this differentiator is that many real-world signals have high-frequency spectral components consisting of noise, and the first-difference differentiator amplifies that noise. The frequency magnitude response of a first-difference differentiator is

(7-5)

$$|H_{Fd}(\omega)| = 2 |\sin(\omega/2)|$$

as shown by the dashed curve in [Figure 7-3](#), where it has the characteristic of a highpass filter. (For comparison, we show an ideal differentiator's straight-line $|H_{Ideal}(\omega)| = \omega$ magnitude response in [Figure 7-3](#).) Looking at that dashed curve, we see how the first-difference differentiator tends to amplify high-frequency spectral components, and this may be detrimental because real-world signals often contain high-frequency noise.

Figure 7-3 Frequency magnitude responses of simple differentiators.



The central-difference differentiator's $|H_{Cd}(\omega)|$ frequency magnitude response, on the other hand, is

(7-6)

$$|H_{Cd}(\omega)| = |\sin(\omega)|$$

as shown by the dotted curve in [Figure 7-3](#), and this differentiator can be useful in that it tends to attenuate high-frequency (noise) spectral components. Looking at the $|H_{Cd}(\omega)|$ curve, we see that the price we pay for that high-frequency attenuation is a reduction in the frequency range over which the central-difference differentiator approaches an ideal differentiator's linear $|H_{Ideal}(\omega)|$. The central-difference differentiator's linear range is from 0 to only, say, 0.2π radians/sample ($0.1f_s$ Hz). The *useful* operating frequency ranges of the first-difference and central-difference differentiators are fairly narrow. This means the differentiators are only accurate when the spectral content of the input signal is low in frequency with respect to the input signal's f_s sample rate.

Another dissimilarity between the [Figure 7-2\(c\)](#) differentiators is their group delay. Because the impulse response (coefficients) of these tapped-delay line networks are antisymmetrical, both differentiators have linear phase responses, and thus both networks have a constant time delay (delay between the input and output, also called [group delay](#)). Like the tapped-delay line FIR filters in [Chapter 5](#), antisymmetrical-coefficient differentiators have a group delay (measured in samples) determined by

(7-7)

$$G_{diff} = \frac{D}{2} \text{ samples}$$

where D is the number of unit-delay elements in their tapped-delay lines. (D can also be viewed as one less than the length of the impulse response of a differentiator.) Hence the first-difference differentiator, where $D = 1$, has an input-to-output delay of $1/2 = 0.5$ samples. The central-difference

differentiator, where $D = 2$, has a group delay of $2/2 = 1$ sample. Whether or not a differentiator's time delay is an integer number of samples is very important in applications where multiple-signal sequences must be aligned (synchronized) in time. (An example of this integer-delay differentiation issue is the FM demodulator discussion in [Section 13.22](#).)

DSP folk have improved, in certain respects, upon the above two computationally simple differentiators in an attempt to (1) extend the linear operating frequency range, (2) continue to attenuate high-frequency spectral components, and (3) keep the number of arithmetic computations as low as possible. It is to those specialized differentiators that we now turn our attention.

7.1.2 Specialized Narrowband Differentiators

DSP pioneer Richard Hamming provided the following

(7-8)

$$h(k) = \frac{-3k}{M(M+1)(2M+1)}$$

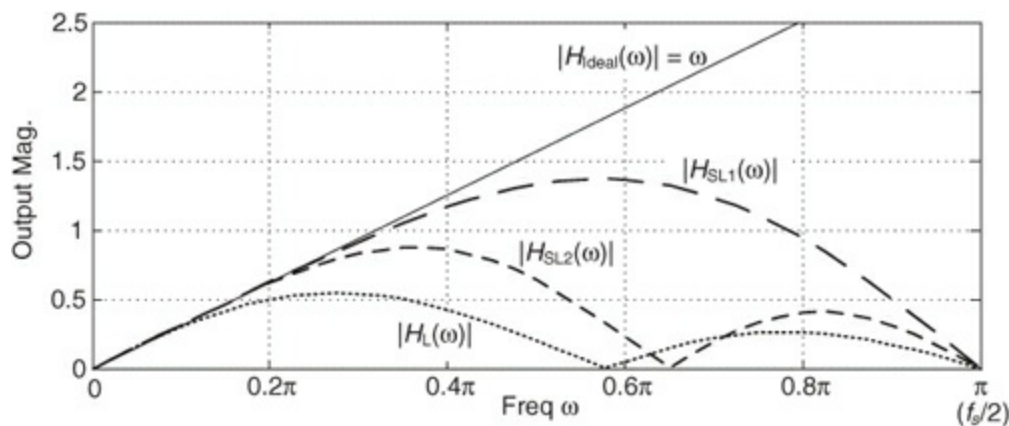
as an expression to compute the coefficients of what he called “low-noise Lanczos,” differentiating filters having $2M+1$ coefficients[\[1\]](#). Variable k , the integer index of those coefficients, ranges from $-M$ to M . If we set $M = 1$ in [Eq. \(7-8\)](#), we obtain the coefficients of the standard central-difference differentiator in [Figure 7-2\(c\)](#). Assigning $M = 2$ to [Eq. \(7-8\)](#) yields the coefficients

(7-9)

$$h_L(k) = 0.2, 0.1, 0, -0.1, -0.2$$

for a five-coefficient differentiator whose $|H_L(\omega)|$ magnitude response is the dotted curve in [Figure 7-4](#). The $h_L(k)$ differentiator in [Eq. \(7-9\)](#) is of interest because if we're willing to multiply those coefficients by 10, we have a high-gain differentiator requiring only two multiplies per output sample. (Happily, those multiplications can be implemented with a binary arithmetic left shift, thus eliminating the multiplications altogether.) The disadvantage of this $h_L(k)$ differentiator is that its linear operating frequency range is the smallest of any differentiator we've considered so far.

Figure 7-4 Frequency magnitude responses of Lanczos differentiators.



Hamming presented two expressions for what he called “super Lanczos low-noise differentiators.” The first expression yielded the five-coefficient differentiator defined by

(7-10)

$$h_{SL1}(k) = \frac{-1}{6}, \frac{8}{6}, 0, \frac{-8}{6}, \frac{1}{6}$$

whose normalized $|H_{\text{SL1}}(\omega)|$ magnitude response is the long-dash curve in [Figure 7-4](#). The $h_{\text{SL1}}(k)$ differentiator has a wider linear operating frequency range than the $h_{\text{L}}(k)$ differentiator, but at the expense of degraded high-frequency attenuation. However, $h_{\text{SL1}}(k)$ is also of interest because if we're willing to multiply the coefficients by 6, we again have a high-gain differentiator requiring only two multiplies per output sample. (Again, those multiplications by ± 8 can be implemented with binary arithmetic left shifts to eliminate the multiplication operations.)

Hamming's second expression for a super Lanczos low-noise differentiator generated the seven-coefficient differentiator defined by

$$(7-11)$$

$$h_{\text{SL2}}(k) = \frac{-22}{126}, \frac{67}{126}, \frac{58}{126}, 0, \frac{-58}{126}, \frac{-67}{126}, \frac{22}{126}$$

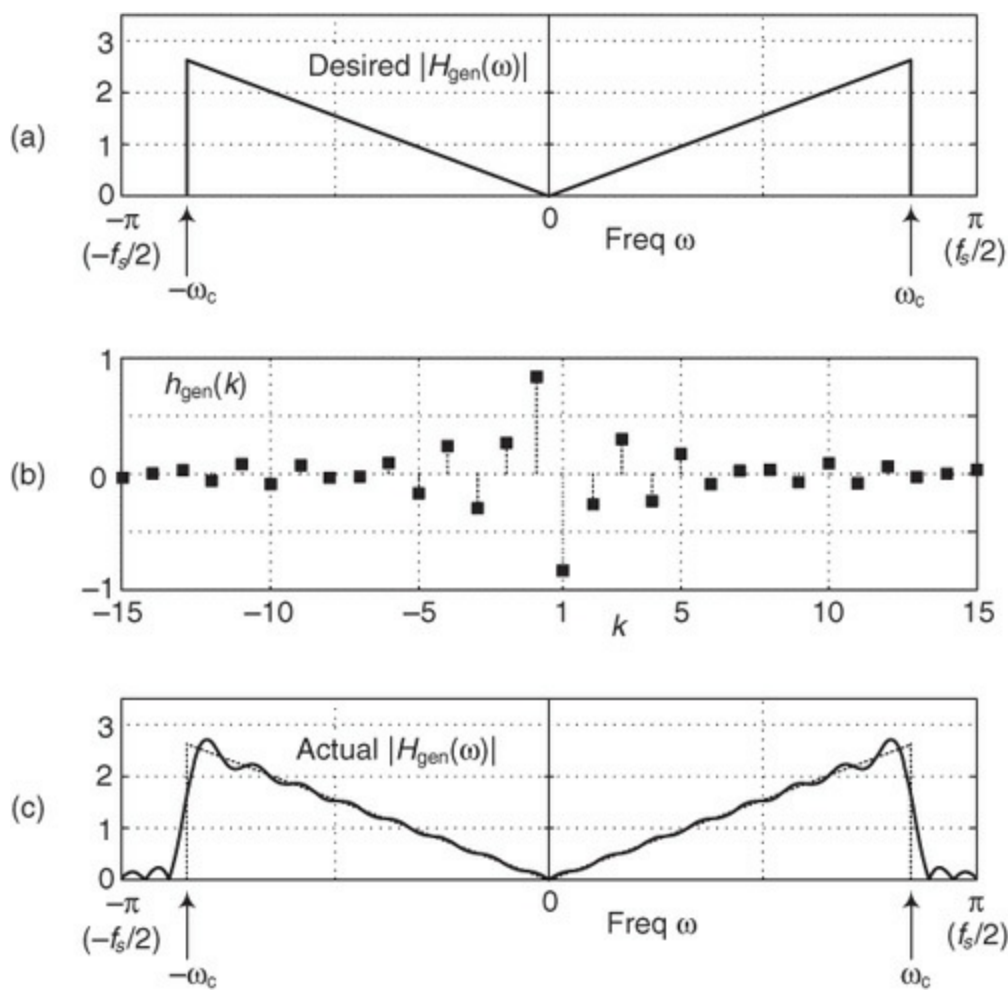
whose normalized $|H_{\text{SL2}}(\omega)|$ magnitude response is the short-dash curve in [Figure 7-4](#). In terms of linear operating frequency range and high-frequency attenuation, the $h_{\text{SL2}}(k)$ differentiator is a reasonable compromise between the $h_{\text{L}}(k)$ and $h_{\text{SL1}}(k)$ differentiators. Notice how the $h_{\text{SL2}}(k)$ differentiator has a good high-frequency noise attenuation characteristic. Then again, in one respect, the $h_{\text{SL2}}(k)$ differentiator is not all that *super* because it requires six multiplies per output sample. (We can do better. [Section 13.38](#) presents a very computationally efficient narrowband differentiator whose linear operating frequency range exceeds that of the $h_{\text{SL1}}(k)$ differentiator.)

With the exception of the first-difference differentiator, after accounting for their constant integer group delays, all of the above differentiators achieve the ideal $H_{\text{ideal}}(\omega)$ phase response in [Figure 7-1\(b\)](#). In the next section we introduce high-performance wideband differentiators.

7.1.3 Wideband Differentiators

Nonrecursive discrete-time differentiators having wider linear operating frequency ranges than the above simple differentiators can be built. All we must do is find the coefficients of a general *wideband* differentiator whose frequency magnitude response is shown in [Figure 7-5\(a\)](#), having a cutoff frequency of ω_c .

Figure 7-5 Frequency response of a *general* wideband differentiator: (a) desired magnitude response; (b) 30 $h_{\text{gen}}(k)$ coefficients; (c) actual magnitude response.



We can derive an equation defining the $h_{\text{gen}}(k)$ coefficients of a general wideband differentiator by defining those coefficients to be the inverse Fourier transform of our desired frequency response from [Eq. \(7-2\)](#) of $H_{\text{ideal}}(\omega) = j\omega$ for continuous ω defined over the range of $-\omega_c \leq \omega \leq \omega_c$. Following this strategy, the coefficients of our general differentiator are given by

$$(7-12)$$

$$h_{\text{gen}}(k) = \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} H_{\text{ideal}}(\omega) e^{j\omega k} d\omega = \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} j\omega e^{j\omega k} d\omega = \frac{j}{2\pi} \int_{-\omega_c}^{\omega_c} \omega e^{j\omega k} d\omega.$$

We can perform the integration in [Eq. \(7-12\)](#) using the dreaded (but useful) integration by parts method, or by searching our math reference books for a closed-form integral expression in the form of [Eq. \(7-12\)\[2\]](#). Being successful in this second approach, we find that the integral of $(\omega e^{j\omega k})d\omega$ is $(e^{j\omega k}) (j\omega k - 1)/(jk)^2$. Using this information, we can write

$$(7-13)$$

$$\begin{aligned} h_{\text{gen}}(k) &= \frac{j}{2\pi} \left[\frac{(e^{j\omega k})(j\omega k - 1)}{(jk)^2} \right]_{-\omega_c}^{\omega_c} = \frac{j}{2\pi} \left[\frac{e^{j\omega_c k}}{k^2} - \frac{j\omega_c e^{j\omega_c k}}{k} - \frac{e^{-j\omega_c k}}{k^2} - \frac{j\omega_c e^{-j\omega_c k}}{k} \right] \\ &= \frac{j}{2\pi} \left[\frac{j2\sin(\omega_c k)}{k^2} - \frac{j2\omega_c \cos(\omega_c k)}{k} \right] = \frac{\omega_c \cos(\omega_c k)}{\pi k} - \frac{\sin(\omega_c k)}{\pi k^2} \end{aligned}$$

where integer index k is $-\infty \leq k \leq \infty$, and $k \neq 0$.

The real-valued $h_{\text{gen}}(k)$ in [Eq. \(7-13\)](#) can be used to compute the coefficients of a tapped-delay line

digital differentiator. This expression, however, is based on the notion that we need an infinite number of differentiator coefficients to achieve the desired response in [Figure 7-5\(a\)](#). Because implementing an infinite-tap differentiator is not possible in our universe, [Figure 7-5\(b\)](#) shows [Eq. \(7-13\)](#) limited (truncated) to a manageable 30 coefficients, and [Figure 7-5\(c\)](#) provides the frequency magnitude response of that 30-tap differentiator with $\omega_c = 0.85\pi$. (The ripples in that magnitude response are to be expected once we think about it. Truncation in one domain causes ripples in the other domain, right?)

As a brief aside, if we set $\omega_c = \pi$ in [Eq. \(7-13\)](#), the coefficients of an N -coefficient differentiator become

(7-14)

$$h_{\omega_c=\pi}(k) = \frac{(-1)^k}{k}$$

where $-(N-1)/2 \leq k \leq (N-1)/2$, and $k \neq 0$. When index $k = 0$, $h_{\omega_c=\pi}(0)$ is set to zero. [Equation \(7-14\)](#) is by far the most popular form given in the standard DSP textbooks for computing digital differentiator coefficients. Using [Eq. \(7-14\)](#), however, is only valid for even-order (N is odd) differentiators, and it is applicable only when the cutoff frequency is $\omega_c = \pi$ ($f_s/2$ Hz).

So where do we stand regarding these wideband differentiators? We've obtained [Eq. \(7-13\)](#) for computing the coefficients of a general wideband differentiator. Unfortunately that expression has a time-domain index (k) having negative values, which can be inconvenient to model using commercial signal processing software. We've discussed the widely disseminated [Eq. \(7-14\)](#) and mentioned its limitations. Again, we can do better.

For a more useful form of an $h_{\text{gen}}(k)$ expression for an arbitrary-length N -tap differentiator we propose the following:

(7-15)

$$h_{\text{gen}}(k) = \frac{\omega_c \cos(\omega_c [k - M])}{\pi[k - M]} - \frac{\sin(\omega_c [k - M])}{\pi[k - M]^2}$$

where $M = (N-1)/2$, and $0 \leq k \leq N-1$. For odd N we set $h_{\text{gen}}((N-1)/2)$, the center coefficient, to zero. [Eq. \(7-15\)](#) looks a bit messy, but it's quite practical because

- the differentiator passband width, ω_c , is a design variable, and not fixed at $\omega_c = \pi$ as in [Eq. \(7-14\)](#);
- the number of taps, N , can be odd or even; and
- the coefficient index k is never negative.

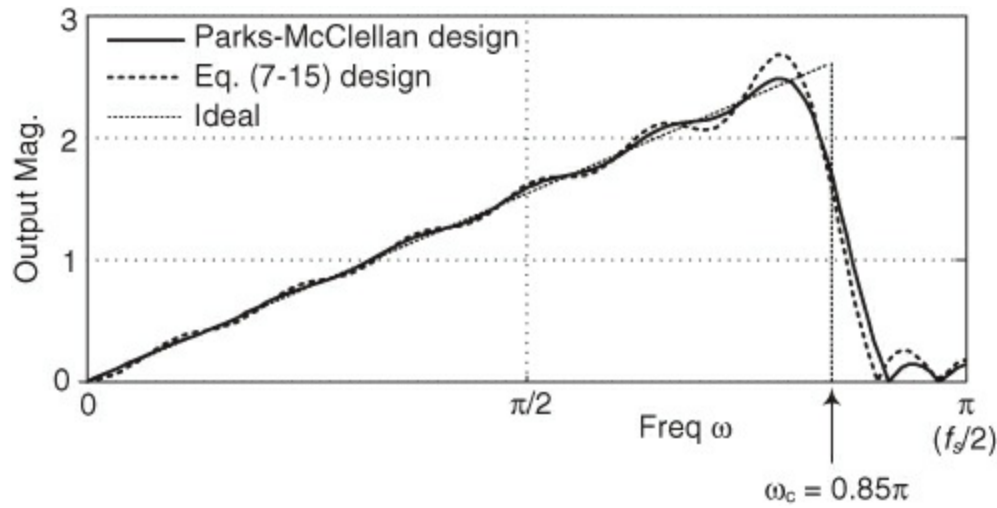
Fortunately, because of the range of index k , [Eq. \(7-15\)](#) is straightforward to model using commercially available signal processing software.

7.1.4 Optimized Wideband Differentiators

For completeness, we point out that the widely available Parks-McClellan algorithm can be used to design wideband digital differentiators whose performance is superior to those produced by [Eq. \(7-15\)](#) when the number of taps N is greater than, say, 25. That behavior is illustrated in [Figure 7-6](#),

where the solid curve shows the frequency magnitude response of an $N = 30$ Parks-McClellan-designed differentiator for $\omega_c = 0.85\pi$, and the bold dashed curve is an $N = 30$ differentiator designed using [Eq. \(7-15\)](#).

Figure 7-6 Frequency magnitude responses of 30-tap wideband differentiators.



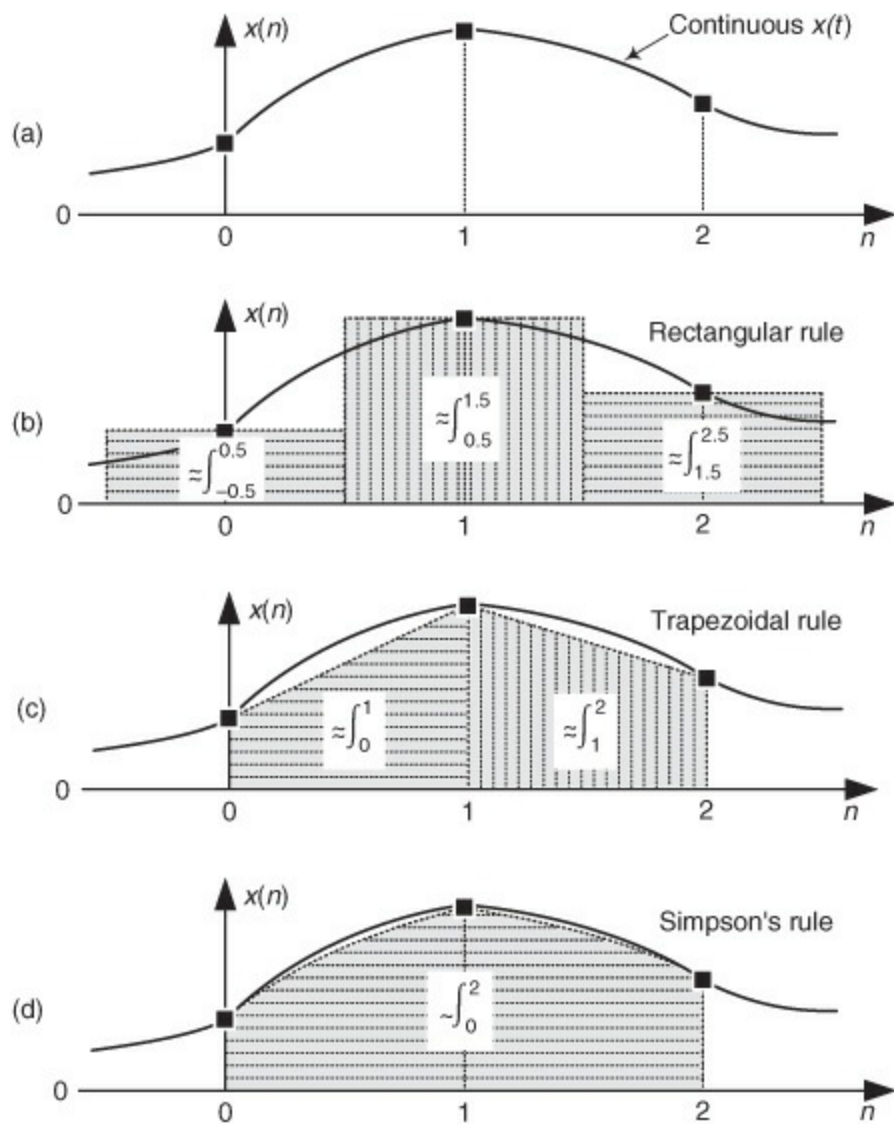
What the DSP pioneers found, in studying the Parks-McClellan algorithm, is that it computes coefficients that provide more accurate differentiation when N is even as opposed to when N is odd. (However, we must keep in mind that the group delay through an even-tap differentiator is not an integer number of samples, and this could be troublesome in systems that require time synchronization among multiple signals.) Design curves showing the relative error for various-length even- and odd- N Parks-McClellan differentiators versus ω_c are available[3,4].

Of course, windowing a wideband differentiator's coefficients, using one of the common window sequences described in [Chapters 3](#) and [5](#), will greatly reduce the ripples in a differentiator's magnitude response. (Windowing in one domain reduces ripples in the other domain, right?) Improved magnitude response linearity, through time-domain windowing, comes at the expense of degrading the sharpness of the response's transition region near ω_c .

7.2 Integrators

The idea of integration is well defined in the domain of continuous (analog) signals, but not so clearly defined in the world of discrete signals. With that said, here we discuss approximating continuous integration by using digital filters that perform numerical integration of sampled signals. We'll discuss digital integration networks whose outputs estimate the area under a continuous curve such as the $x(t)$ function shown in [Figure 7-7\(a\)](#).

Figure 7-7 Integrator areas of summation.



7.2.1 Rectangular Rule Integrator

One simple way to estimate, to approximate, the area under the $x(t)$ curve is to merely sum the $x(n)$ samples. Such a *rectangular rule* integrator computes the sum of the shaded rectangles shown in [Figure 7-7\(b\)](#). In the time domain we define the rectangular rule integrator, a running summation, as

$$(7-16)$$

$$y_{\text{Re}}(n) = x(n) + y_{\text{Re}}(n-1),$$

where the current sum, $y_{\text{Re}}(n)$, is the previous $y_{\text{Re}}(n-1)$ sum plus the current input sample $x(n)$. When $n = 2$, for example, [Eq. \(7-16\)](#) adds the area under the right-side shaded rectangle shown in [Figure 7-7\(b\)](#) to the previous sum $y_{\text{Re}}(1)$ to compute $y_{\text{Re}}(2)$. The height and width of that right-side shaded rectangle are $x(2)$ and one, respectively.

The frequency response of this rectangular rule integrator is

$$(7-16')$$

$$H_{\text{Re}}(\omega) = \frac{1}{1 - e^{-j\omega}}.$$

7.2.2 Trapezoidal Rule Integrator

A useful area integration estimation scheme is the *trapezoidal rule* defined by

(7-17)

$$y_{Tr}(n) = 0.5x(n) + 0.5x(n-1) + y_{Tr}(n-1).$$

When $n = 2$, for example, [Eq. \(7-17\)](#) computes the area (the average of $x(2) + x(1)$) under the right-side shaded trapezoid shown in [Figure 7-7\(c\)](#) and adds that value to the previous $y_{Tr}(1)$ to compute $y_{Tr}(2)$. The frequency response of the trapezoidal rule integrator is

(7-17')

$$H_{Tr}(\omega) = \frac{0.5 + 0.5e^{-j\omega}}{1 - e^{-j\omega}}.$$

7.2.3 Simpson's Rule Integrator

A popular discrete-time integration scheme is *Simpson's rule* defined by

(7-18)

$$y_{Si}(n) = [x(n) + 4x(n-1) + x(n-2)]/3 + y_{Si}(n-2),$$

where three samples are used to compute the area under the single shaded curve in [Figure 7-7\(d\)](#). The frequency response of the Simpson's rule integrator is

(7-18')

$$H_{Si}(\omega) = \frac{1 + 4e^{-j\omega} + e^{-j2\omega}}{3(1 - e^{-j2\omega})}.$$

(Simpson's rule is named after the eighteenth-century English mathematician Thomas Simpson. Oddly enough, Simpson's rule was actually developed by Sir Isaac Newton rather than Simpson. But don't hold that against Simpson because the famous iterative method for finding the roots of a polynomial, called *Newton's method*, was developed by Simpson!)

The above three time-domain integration approximations were developed using the principles of *polynomial curve fitting* where Simpson's rule fits three signal samples to a second-order polynomial in x , the trapezoidal rule fits two samples to a first-order polynomial in x , and the rectangular rule uses a single sample in a zero-order polynomial in x .

7.2.4 Tick's Rule Integrator

For completeness, we point out an integration approximation similar to Simpson's rule that you may encounter in the literature of DSP called *Tick's rule*. It's defined as

(7-19)

$$y_{Ti}(n) = 0.3584x(n) + 1.2832x(n-1) + 0.3584x(n-2) + y_{Ti}(n-2)$$

having a frequency response given by

(7-19')

$$H_{Ti}(\omega) = \frac{0.3584 + 1.2832e^{-j\omega} + 0.3584e^{-j2\omega}}{1 - e^{-j2\omega}}.$$

The Tick's rule integrator was designed to be especially accurate over the low-frequency range of $0 \leq \omega \leq \pi/2$ radians/sample (zero to $f_s/4$ Hz) with little concern for its accuracy at higher frequencies[\[5\]](#).

7.2.5 Integrator Performance Comparison

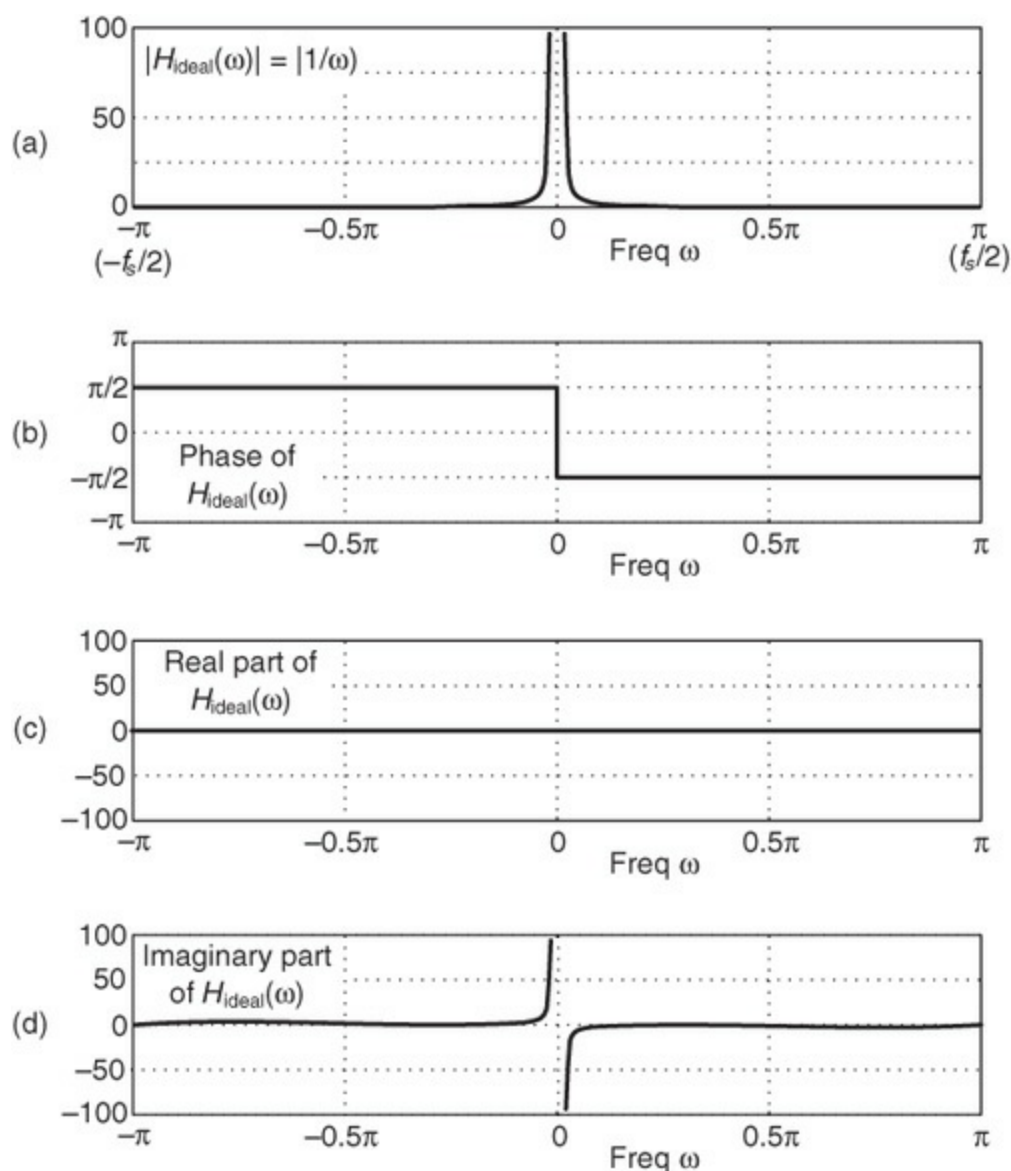
OK, so how well do the above discrete integrators perform? We can measure their performance by comparing their behavior to an ideal continuous (analog) integrator. Doing so, we first recall that the integral of the continuous function $\cos(\omega t)$ is

(7-20)

$$\int \cos(\omega t) dt = \frac{\sin(\omega t)}{\omega},$$

telling us that if we apply a sinusoid to an ideal integrator, the output of the integrator will be a sinusoid, phase-shifted by $-\pi/2$ radians (-90°), whose amplitude is reduced by a factor of $1/\omega$. Thus the frequency magnitude response of an ideal integrator is $|H_{\text{ideal}}| = |1/\omega|$ as shown in [Figure 7-8\(a\)](#), and the integrator's phase is that shown in [Figure 7-8\(b\)](#), where the digital frequency $\omega = \pi$ radians/sample is equivalent to half the signal data sample rate in Hz ($f_s/2$).

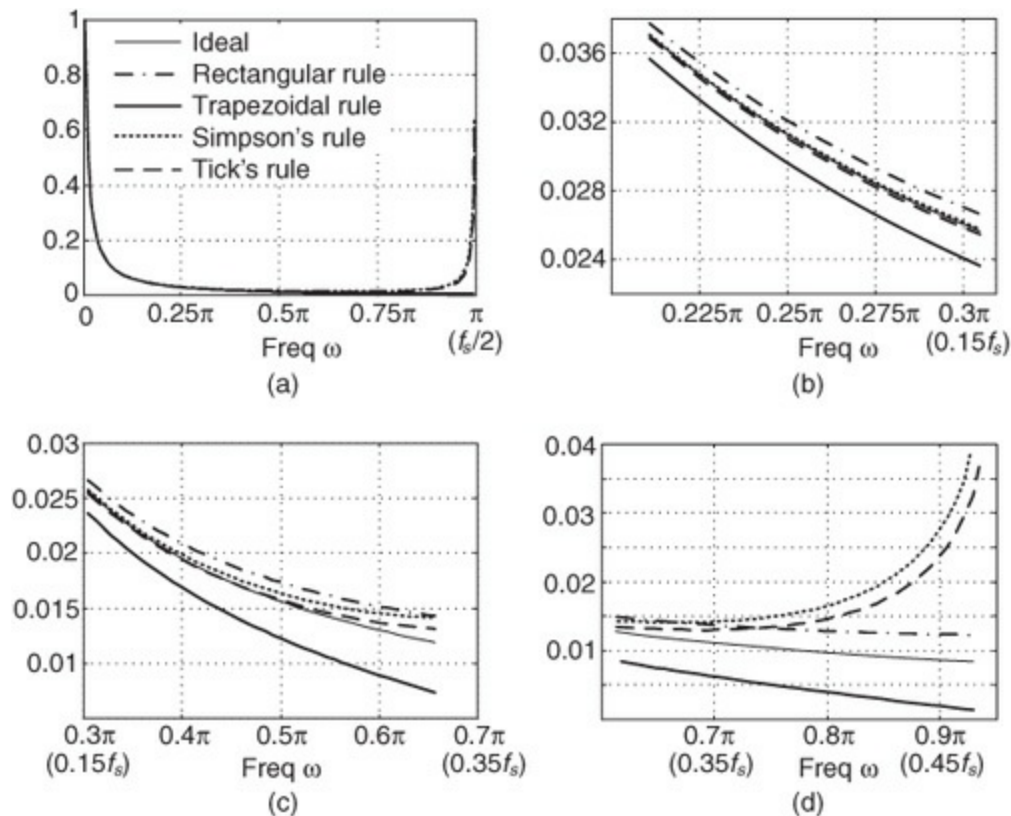
Figure 7-8 Ideal integrator frequency response: (a) magnitude; (b) phase in radians; (c) real part; (d) imaginary part.



The frequency magnitude responses of an ideal integrator and our four digital integrators are shown in [Figure 7-9](#) over various bandwidths in the positive-frequency range of $0 \leq \omega \leq \pi$. For ease of comparison, the magnitude curves are all normalized so that their peak values, in the vicinity of $\omega =$

0, are unity. (Note that the ideal integrator's response curve in [Figure 7-9\(b\)](#) is obscured by the Simpson's rule and Tick's rule curves.) What we see from [Figure 7-9](#) is that the various integrators have very little difference over the ω frequency range of 0 to $\pi/2$ radians/sample (zero to $f_s/4$ Hz), but above that range there are meaningful differences that we'll discuss in a moment.

Figure 7-9 Normalized frequency magnitude responses of four integrators.

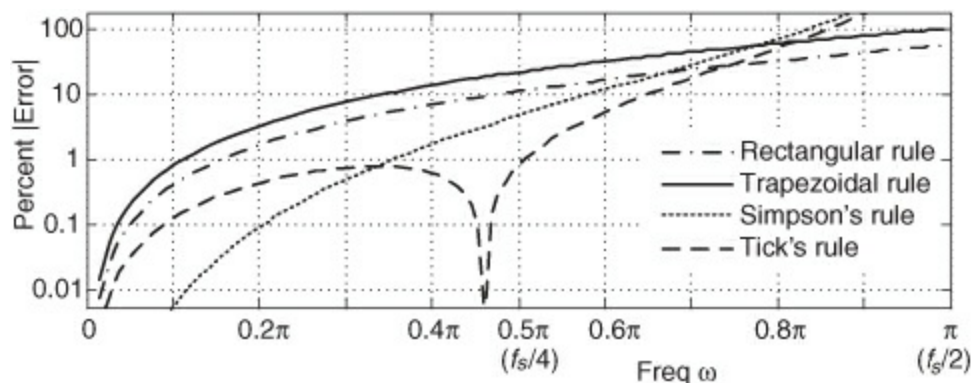


The magnitude response curves in [Figure 7-9](#) are a bit difficult to interpret when a linear magnitude axis is used. With that thought in mind, [Figure 7-10](#) shows the various integrators' percent absolute errors using logarithmic axis scaling. We defined the percent absolute error as

(7-21)

$$\text{Percent absolute error} = 100 \left| \frac{|H_{\text{integrator}}(\omega)| - |H_{\text{ideal}}(\omega)|}{|H_{\text{ideal}}(\omega)|} \right|.$$

Figure 7-10 Integrator absolute errors in percent.



Looking at the error curves in [Figure 7-10](#) might cause you to think, "These integrators aren't very accurate. For example, the Simpson's rule integrator has roughly a 7 percent error at $\omega = 0.5\pi$ ($f_s/4$ Hz)." Well, the situation is not as bad as it first appears. Looking at the ideal integrator's response in

[Figure 7-9\(a\)](#), we must realize that a 7 percent error of the small magnitude response values near $\omega = 0.5\pi$ is not nearly as significant as a 7 percent error for the larger magnitude response values below $\omega = 0.1\pi$. So this means our simple integrators are quite accurate at low frequencies where we most need high accuracy.

What we learn from [Figure 7-10](#) is that all of the digital integrators have good accuracy at low frequencies, with the Tick's rule and Simpson's rule integrators being the most accurate. (The phrase "low frequencies" means that the spectral components of the function, the signal, we are trying to integrate are low in frequency relative to the f_s sample rate.) However, if the integrators' input signals have appreciable noise spectral components near $f_s/2$ Hz, the Tick's rule and Simpson's rule integrators will amplify that noise because those integrators have z -domain transfer function poles (infinite gain) at $z = -1$, corresponding to a cyclic frequency of $f_s/2$ Hz. In such high-frequency noise scenarios the rectangular or trapezoidal rule integrators should be used because they provide improved attenuation of spectral components in the vicinity of $f_s/2$ Hz.

The integrators that we've discussed are interesting because they are recursive networks and they all have linear phase. However, only the trapezoidal, Simpson's, and Tick's rule integrators achieve the ideal $H_{\text{ideal}}(\omega)$ phase response in [Figure 7-8\(b\)](#).

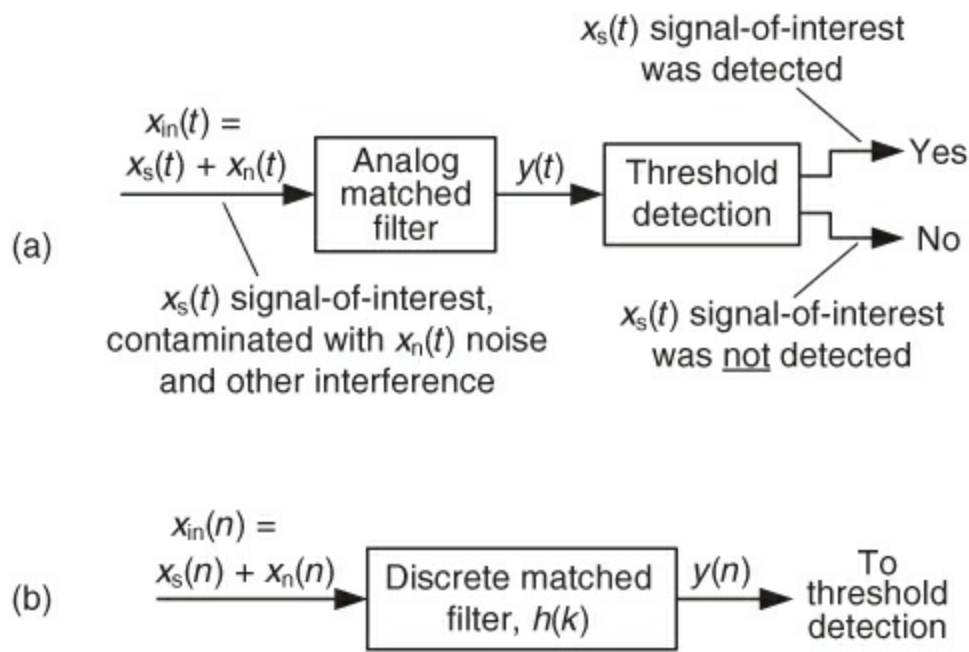
The above integrators all have z -domain transfer function poles at $z = 1$, corresponding to a cyclic frequency of zero Hz, and this has an important consequence when we implement integrators in digital hardware. Those poles force us to ensure that the numerical format of our integrator hardware can accommodate summation results when the $x(n)$ input sequence has a nonzero average value (a constant amplitude bias). Stated in different words, the widths of our binary data registers must be large enough to guarantee that any nonzero amplitude bias on $x(n)$ will not cause numerical overflow and corrupt the data within an integrator. [Chapter 10](#)'s discussion of cascaded integrator-comb (CIC) filters elaborates on this integrator data register width issue.

7.3 Matched Filters

In this section we introduce a signal processing operation known as *matched filtering*. A matched filter is a process that maximizes the signal-power-to-noise-power ratio (SNR) of its $y(t)$ output when a specified $x_s(t)$ signal of interest arrives at its input. Such a process is widely used in radar, sonar, oil exploration, digital communications systems, and frequency-domain processing of two-dimensional images. Those systems are designed to reliably detect (recognize) if, and at what instant in time or position in space, a well-defined $s(t)$ signal of interest arrived at their inputs.

Matched filtering, for continuous signals, is depicted in [Figure 7-11\(a\)](#). In that figure the system's $x_{\text{in}}(t)$ input signal is an $x_s(t)$ signal of interest, which may be a radar signal or perhaps a small portion of a digital communications signal, contaminated with an $x_n(t)$ noise signal. The matched filter's task is to maximize the SNR of the $y(t)$ signal so that reliable detection of $x_s(t)$ can be performed.

Figure 7-11 Matched filtering: (a) continuous signal implementation; (b) digital implementation with $h(k)$ impulse response.



7.3.1 Matched Filter Properties

So the question is “What should the frequency response of the matched filter be to maximize our chances of detecting the presence of $x_s(t)$ in the noisy $x_{in}(t)$ signal?” The answer can be found in most communications textbooks[6], and here’s how we interpret that answer. Given the $S(f)$ spectrum of $x_s(t)$, the desire to maximize the SNR of $y(t)$, lots of calculus, plus an application of Schwarz’s inequality, the optimum $H(f)$ frequency response of the matched filter can be shown to be

(7-22)

$$H(f) = S^*(f)e^{-j2\pi fT}$$

where T is the time duration of the $x_s(t)$ signal measured in seconds, and $*$ signifies conjugation. Equation (7-22) tells us that the optimum $H(f)$ frequency response of the continuous matched filter should be the complex conjugate of the spectrum of our signal of interest multiplied by a phase shift that’s a linear function of frequency. Stated in different words, the time-domain impulse response of the optimum matched filter is the inverse Fourier transform of $S^*(f)$ shifted in the negative-time direction by T seconds. We now provide a physical meaning of all of this as we determine how to implement a matched filter in discrete-time sampled systems.

To show how to build a digital (discrete-time) matched filter, as shown in Figure 7-11(b), first we need to determine the $h(k)$ impulse response of the filter. Let’s make the following assumptions:

- Our discrete signal of interest is an N -sample $x_s(n)$ sequence.
- $S(m)$ is the N -point discrete Fourier transform (DFT) of $x_s(n)$.
- m is the discrete frequency index $0 \leq m \leq N-1$.
- The $x_{in}(n)$ data input sample rate is f_s samples/second.

Under these assumptions we convert the continuous frequency f in Eq. (7-22) to the DFT’s discrete frequencies of mf_s/N to express the digital matched filter’s $H(m)$ discrete frequency response as

(7-23)

$$H(m) = S^*(m)e^{-j2\pi mf_s T/N}$$

where T is the time duration of the $x_s(n)$ signal. OK, our next step is to define T such that the inverse DFT of $H(m)$, our desired $h(k)$, is the inverse DFT of $S^*(m)$ shifted in the negative-time direction by an amount equal to the time duration of $x_s(n)$. This sounds a bit complicated, but it's really not so bad, as we shall see.

To determine T in [Eq. \(7-23\)](#) we need to know how many sample periods (with $1/f_s$ being one period) comprise the time duration of an N -sample $x_s(n)$ sequence. The answer is: The time duration of an N -sample discrete sequence is $N-1$ sample periods. (Readers should convince themselves that this is true.) So T in [Eq. \(7-23\)](#) is $(N-1)/f_s$ seconds, and we write the discrete frequency response of our discrete matched filter as

$$(7-24)$$

$$H(m) = S^*(m)e^{-j2\pi m(N-1)/N}.$$

Finally, our discrete matched filter's $h(k)$ impulse response is the N -point inverse DFT of $H(m)$, which, from [Appendix C](#), is merely the straight time reversal (left-to-right flip) of $x_s(n)$. And there you have it—we express our optimum $h(k)$ as

$$(7-25)$$

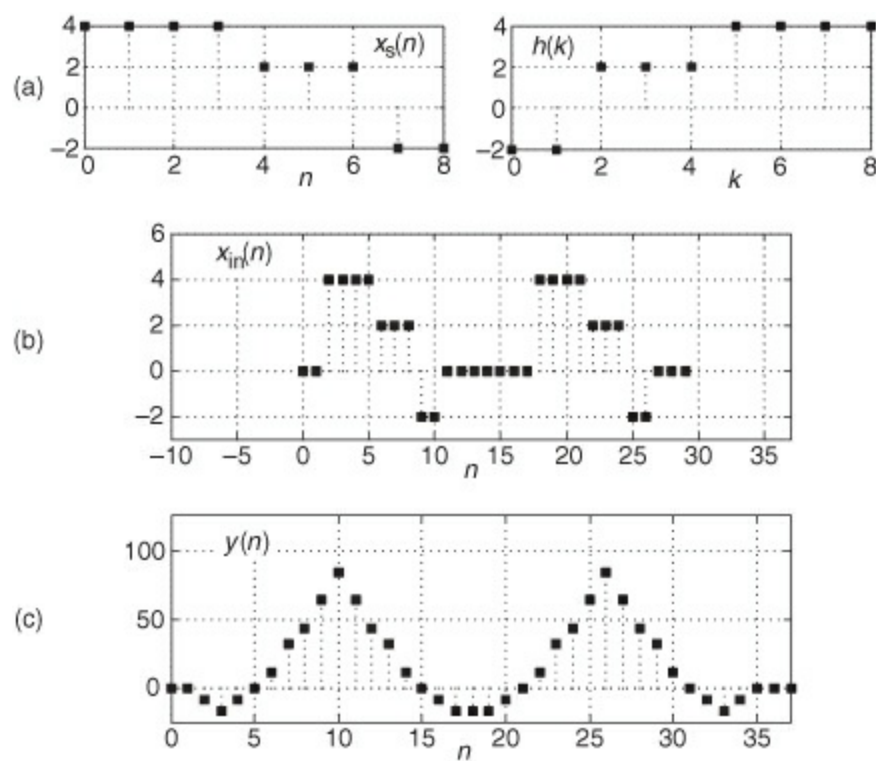
$$h(k) = x_s(N-k-1)$$

where k ranges from 0 to $N-1$. We struggled through the above process of developing [Eq. \(7-25\)](#) so the reader would understand the origin of our expression for $h(k)$.

7.3.2 Matched Filter Example

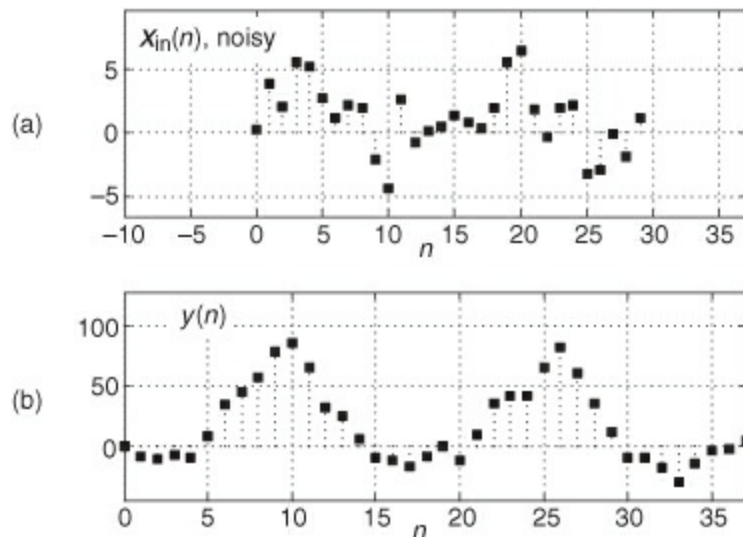
As an example of matched filtering, [Figure 7-12\(a\)](#) shows an $N = 9$ sample $x_s(n)$ signal-of-interest sequence and the optimum $h(k)$ matched filter impulse response. If the matched filter's $x_{in}(n)$ input contains two occurrences of $x_s(n)$, as shown in [Figure 7-12\(b\)](#), the filter's $y(n)$ output will be the two pulses (each one symmetrical) shown in [Figure 7-12\(c\)](#). Our signal recognition process is then making sure the threshold detection process in [Figure 7-11\(a\)](#) detects the high-level peaks in $y(n)$. It's useful to remind ourselves that the $x_{in}(n)$ sequence enters the filter in a reversed order from that shown in [Figure 7-12\(b\)](#). That is, sample $x_{in}(0)$ enters the filter first, followed by the $x_{in}(1)$ sample, and so on. So the $x_s(n)$ sequences within $x_{in}(n)$, arriving at the filter's input, are in the same left-right orientation as the filter's $h(k)$ impulse response.

Figure 7-12 Matched filtering example: (a) signal of interest $x_s(n)$ and $h(k)$; (b) filter $x_{in}(n)$ input; (c) filter $y(n)$ output.



To show the value of matched filtering, [Figure 7-13\(a\)](#) shows an $x_{in}(n)$ input sequence, having two occurrences of the previous $x_s(n)$, but this time badly contaminated with random noise. It's very difficult to see the two $x_s(n)$ sequences in $x_{in}(n)$. In this noisy-signal case the filter's $y(n)$ output, shown in [Figure 7-13\(b\)](#), still distinctly exhibits the two peaks similar to the noise-free example in [Figure 7-12\(c\)](#). Actually, we should call the two peaks in [Figure 7-12\(c\)](#) “correlation peaks” because our matched filter is performing a correlation between the $x_{in}(n)$ input signal and the predefined $x_s(n)$ signal of interest. The $y(n)$ output is not the $x_s(n)$ signal of interest— $y(n)$ is a quantitative measure of the similarity between the $x_{in}(n)$ input signal and $x_s(n)$.

Figure 7-13 Matched filtering example: (a) filter $x_{in}(n)$ input contaminated with noise; (b) filter $y(n)$ output.



7.3.3 Matched Filter Implementation Considerations

There are a number of important topics to consider when implementing matched filters. They are:

- Matched filters are most conveniently implemented with tapped-delay line FIR filters like those

we studied in [Chapter 5](#). The $h(k)$ sequence merely becomes the coefficients of the FIR filter. Again, our digital matched filter performs convolution of $x_{\text{in}}(n)$ and $h(k)$, which is equivalent to performing correlation between $x_{\text{in}}(n)$ and $x_s(n)$. In the 1980s the TRW LSI Products organization produced an integrated circuit that contained a 32-tap FIR architecture used for matched filtering. The chip was justifiably called a *digital correlator*.

- As we discussed in [Section 5.9.2](#), time-domain convolution can be implemented by way of frequency-domain multiplication. When the lengths of $x_s(n)$ and $h(k)$ are large (say, $N > 80$) and forward and inverse FFTs are used, frequency-domain multiplication may be more computationally efficient than traditional time-domain convolution.
- The $H(m)$ frequency response given in [Eq. \(7-23\)](#) is based on two assumptions: (1) that the $x_n(n)$ noise is truly random, having a flat-level broadband power spectrum, which means there is no correlation between one $x_n(n)$ noise sample and any subsequent $x_n(n+k)$ sample in [Figure 7-11\(b\)](#); and (2) the $x_n(n)$ noise's probability density function (PDF) is Gaussian in shape. Such noise is referred to as *additive white noise* (AWN). If the $x_n(n)$ noise is not AWN, for example, when $x_s(n)$ is a radio signal and $x_n(n)$ is a high-level intentional jamming signal, or when $x_s(n)$ is a single data symbol of a digital communications signal contaminated with some previous-in-time data symbol, then [Eq. \(7-22\)](#) must be modified. References [\[7–9\]](#) provide additional information regarding this non-AWN scenario.
- Matched filtering is easy to perform. However, the detection threshold operation in [Figure 7-11](#), to detect the peaks in [Figure 7-13\(b\)](#), can become difficult to implement reliably depending on the nature of $x_s(n)$, $x_n(n)$, and the SNR of $x_{\text{in}}(n)$. If we set the threshold too high, then we reduce our probability of detection by risking failure to detect $x_s(n)$. If we set the threshold too low, then we increase our probability of false alarm by incorrectly identifying a noise spike in $y(n)$ as an occurrence of our desired $x_s(n)$. Advanced signal processing textbooks, by way of statistics and an abundance of probability theory, cover these topics. Representative descriptions of these concepts are provided in references [\[10,11\]](#).

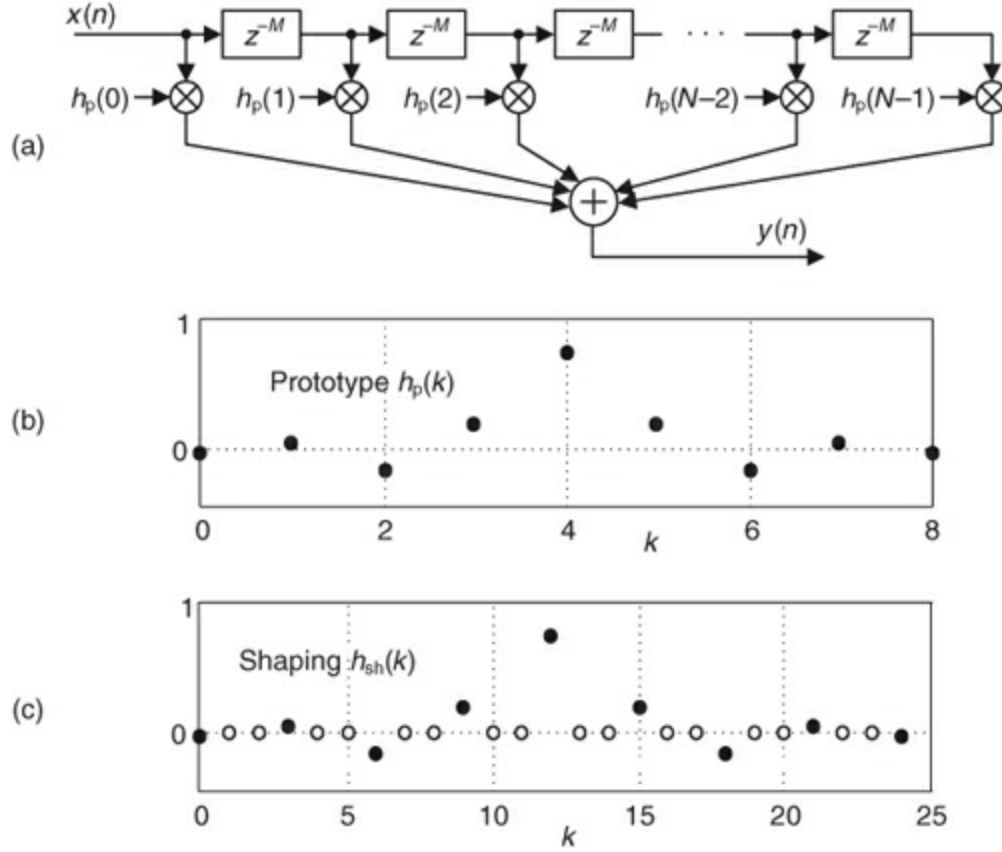
7.4 Interpolated Lowpass FIR Filters

In this section we cover a class of digital filters, called *interpolated FIR filters*, used to build narrowband lowpass FIR filters that can be more computationally efficient than the traditional Parks-McClellan-designed tapped-delay line FIR filters that we studied in [Chapter 5](#). Interpolated FIR filters can reduce traditional narrowband lowpass FIR filter computational workloads by more than 80 percent. In their description, we'll introduce interpolated FIR filters with a simple example, discuss how filter parameter selection is made, provide filter performance curves, and go through a simple lowpass filter design example showing their computational savings over traditional FIR filters [\[12,13\]](#).

Interpolated FIR (IFIR) filters are based upon the behavior of an N -tap nonrecursive linear-phase FIR filter when each of its unit delays is replaced with M unit delays, with the *expansion factor* M being an integer, as shown in [Figure 7-14\(a\)](#). If the $h_p(k)$ impulse response of a 9-tap FIR filter is that shown in [Figure 7-14\(b\)](#), the impulse response of an *expanded* FIR filter, where for example $M = 3$, is the $h_{\text{sh}}(k)$ in [Figure 7-14\(c\)](#). The M unit delays result in the zero-valued samples, the white dots, in

the $h_{\text{sh}}(k)$ impulse response. Our variable k is merely an integer time-domain index where $0 \leq k \leq N-1$. To define our terminology, we'll call the original FIR filter the *prototype* filter—that's why we used the subscript "p" in $h_p(k)$ —and we'll call the filter with expanded delays the *shaping* subfilter. Soon we'll see why this terminology is sensible.

Figure 7-14 Filter relationships: (a) shaping FIR filter with M unit delays between the taps; (b) impulse response of a prototype FIR filter; (c) impulse response of an expanded-delay shaping FIR filter with $M = 3$.



We can express a prototype FIR filter's z -domain transfer function as

(7-26)

$$H_p(z) = \sum_{k=0}^{N_p-1} h_p(k) z^{-k}$$

where N_p is the length of h_p . The transfer function of a general shaping FIR filter, with z in [Eq. \(7-26\)](#) replaced with z^M , is

(7-27)

$$H_{\text{sh}}(z^M) = \sum_{k=0}^{N_p-1} h_p(k) z^{-kM}.$$

Later we'll see why we chose to provide [Eqs. \(7-26\)](#) and [\(7-27\)](#). If the number of coefficients in the prototype filter is N_p , the shaping filter has N_p nonzero coefficients and an expanded impulse response length of

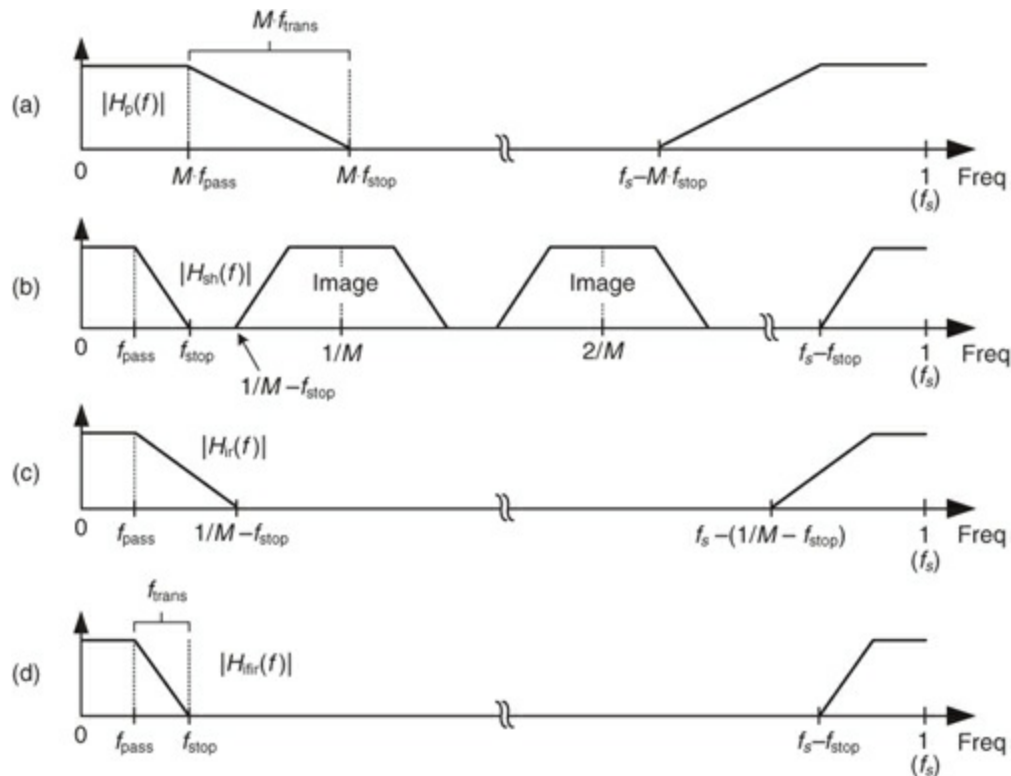
(7-28)

$$N_{\text{sh}} = M(N_p - 1) + 1.$$

Later we'll see how N_{sh} has an important effect on the implementation of IFIR filters.

The frequency-domain effect of those M unit delays is shown in [Figure 7-15](#). As we should expect, an M -fold expansion of the time-domain filter impulse response causes an M -fold compression (and repetition) of the frequency-domain $|H_p(f)|$ magnitude response as in [Figure 7-15\(b\)](#). While $H_p(f)$ has a single passband, $H_{\text{sh}}(f)$ has M passbands. (The frequency axis of these curves is normalized to the f_s filter input signal sample rate. For example, the normalized frequency f_{pass} is equivalent to a frequency of $f_{\text{pass}}f_s$ Hz.) Those repetitive passbands in $|H_{\text{sh}}(f)|$ centered about integer multiples of $1/M$ (f_s/M Hz) are called *images*, and on them we now focus our attention.

Figure 7-15 IFIR filter magnitude responses: (a) the prototype filter; (b) shaping subfilter; (c) image-reject subfilter; (d) final IFIR filter.



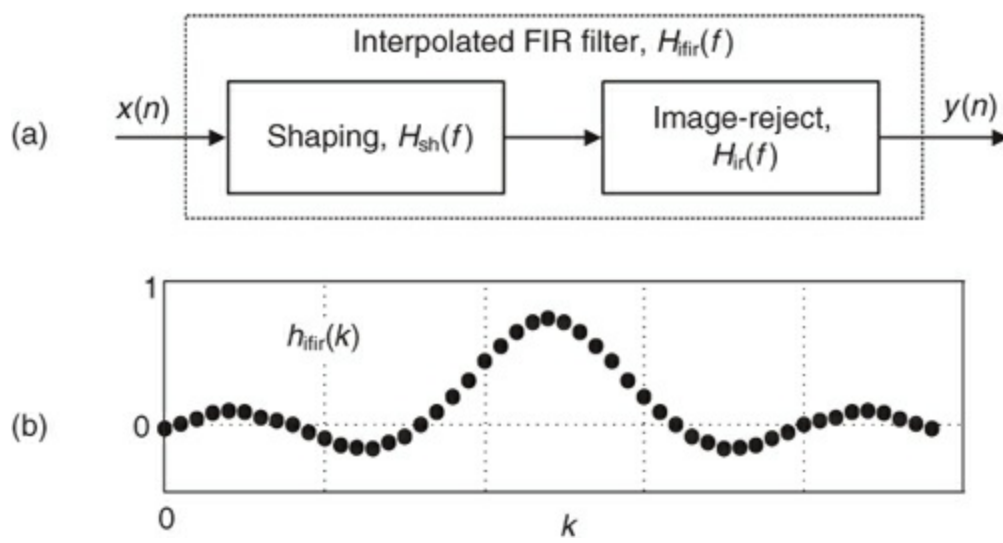
If we follow the shaping subfilter with a lowpass *image-reject* subfilter, [Figure 7-15\(c\)](#), whose task is to attenuate the image passbands, we can realize a multistage filter whose frequency response is shown in [Figure 7-15\(d\)](#). The resultant $|H_{\text{ifir}}(f)|$ frequency magnitude response is, of course, the product

$$(7-29)$$

$$|H_{\text{ifir}}(f)| = |H_{\text{sh}}(f)| \cdot |H_{\text{ir}}(f)|.$$

The structure of the cascaded subfilters is the so-called IFIR filter shown in [Figure 7-16\(a\)](#), with its interpolated impulse response given in [Figure 7-16\(b\)](#).

Figure 7-16 Lowpass interpolated FIR filter: (a) cascade structure; (b) resultant impulse response.



If the original desired lowpass filter's passband width is f_{pass} , its stopband begins at f_{stop} , and the transition region width is $f_{\text{trans}} = f_{\text{stop}} - f_{\text{pass}}$, then the prototype subfilter's normalized frequency parameters are defined as

$$(7-30)$$

$$f_{\text{p-pass}} = M f_{\text{pass}}.$$

$$(7-30')$$

$$f_{\text{p-stop}} = M f_{\text{stop}}.$$

$$(7-30'')$$

$$f_{\text{p-trans}} = M f_{\text{trans}} = M(f_{\text{stop}} - f_{\text{pass}}).$$

The image-reject subfilter's frequency parameters are

$$(7-31)$$

$$f_{\text{ir-pass}} = f_{\text{pass}}.$$

$$(7-31')$$

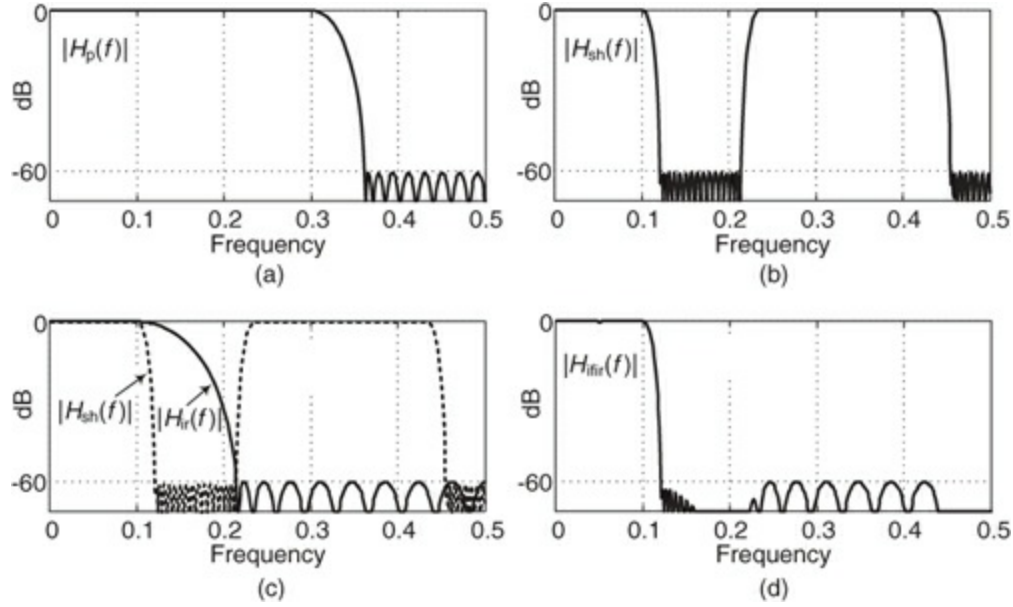
$$f_{\text{ir-stop}} = \frac{1}{M} - f_{\text{stop}}.$$

The stopband attenuations of the prototype filter and image-reject subfilter are identical and set equal to the desired IFIR filter stopband attenuation. The word *interpolated* in the acronym IFIR is used because the image-reject subfilter interpolates samples in the prototype filter's $h_p(k)$ impulse response, making the overall IFIR filter's impulse response equal to the $h_{\text{Ifir}}(k)$ sequence in [Figure 7-34\(b\)](#). Note that $h_{\text{Ifir}}(k)$ does not represent the coefficients used in any FIR subfilter filter. Sequence $h_{\text{Ifir}}(k)$ is the convolution of the shaping and image-reject subfilters' impulse responses (coefficients). Some authors emphasize this attribute by referring to the image-reject subfilter as an *interpolator*. The f_s sample rate remains unchanged within an IFIR filter, so no actual signal interpolation takes place.

To give you an incentive to continue reading, the following example shows the terrific computational advantage of using IFIR filters. Consider the design of a *desired* linear-phase FIR filter whose

normalized passband width is $f_{\text{pass}} = 0.1$, its passband ripple is 0.1 dB, the transition region width is $f_{\text{trans}} = 0.02$, and the stopband attenuation is 60 dB. (The passband ripple is a peak-peak specification measured in dB.) With an expansion factor of $M = 3$, the $|H_p(f)|$ frequency magnitude response of the prototype filter is shown in [Figure 7-17\(a\)](#). The normalized frequency axis for these curves is such that a value of 0.5 on the abscissa represents the cyclic frequency $f_s/2$ Hz, half the sample rate. The frequency response of the shaping subfilter, for $M = 3$, is provided in [Figure 7-17\(b\)](#) with an image passband centered about $(1/M)$ Hz. The response of the image-reject subfilter is the solid curve in [Figure 7-17\(c\)](#), and the response of the overall IFIR filter is provided in [Figure 7-17\(d\)](#).

Figure 7-17 Example lowpass IFIR filter magnitude responses: (a) the prototype filter; (b) shaping subfilter; (c) image-reject subfilter; (d) final IFIR filter.



Satisfying the original desired filter specifications in [Figure 7-17\(d\)](#) would require a traditional tapped-delay FIR filter with $N_{\text{tfir}} = 137$ taps, where the “tfir” subscript means *traditional FIR*. In our IFIR filter, the shaping and the image-reject subfilters require $N_p = 45$ and $N_{\text{ir}} = 25$ taps respectively, for a total of $N_{\text{ifir}} = 70$ taps. We can define the percent reduction in computational workload (number of multiplies per filter output sample) of an IFIR filter, over a traditional tapped-delay line FIR filter, as

$$(7-32)$$

$$\% \text{ computation reduction} = 100 \cdot \frac{N_{\text{tfir}} - N_p - N_{\text{ir}}}{N_{\text{tfir}}}.$$

As such, the above example IFIR filter has achieved a multiplication computational workload reduction, over a traditional FIR filter, of

$$(7-32')$$

$$\% \text{ computation reduction} = 100 \cdot \frac{137 - 45 - 25}{137} = 49\%.$$

[Figure 7-17](#) shows how the transition region width (the shape) of $|H_{\text{ifir}}(f)|$ is determined by the transition region width of $|H_{\text{sh}}(f)|$, and this justifies the decision to call $h_{\text{sh}}(k)$ the *shaping* subfilter.

7.4.1 Choosing the Optimum Expansion Factor M

The expansion factor M deserves our attention because it can have a profound effect on the computational efficiency of IFIR filters. To show this, had we used $M = 2$ in our [Figure 7-17](#) example, we would have realized an IFIR filter described by the $M = 2$ row in [Table 7-1](#). In that case the computation reduction over a conventional FIR filter is 43 percent. With $M = 2$, a reduced amount of frequency-domain compression occurred in $H_{\text{sh}}(f)$, which mandated more taps in $h_{\text{sh}}(k)$ than were needed in the $M = 3$ case.

Table 7-1 IFIR Filter Computation Reduction versus M

Expansion factor M	Number of taps			Computation reduction
	$h_{\text{sh}}(k)$	$h_{\text{ir}}(k)$	IFIR total	
2	69	8	77	43%
3	45	25	70	49%
4	35	95	130	8%

Now had $M = 4$ been used, the computation reduction, over a single traditional tapped-delay line FIR filter, would only be 8 percent as shown in [Table 7-1](#). This is because the $H_{\text{sh}}(f)$ passband images would be so close together that a high-performance (increased number of taps) image-reject subfilter would be required. As so often happens in signal processing designs, there is a trade-off to be made. We would like to use a large value for M to compress the $H_{\text{sh}}(f)$'s transition region width as much as possible, but a large M reduces the transition region width of the image-reject subfilter, which increases the number of taps in $h_{\text{ir}}(k)$ and its computational workload. In our [Figure 7-17](#) IFIR filter example an expansion factor of $M = 3$ is optimum because it yields the greatest computation reduction over a traditional tapped-delay line FIR filter.

The optimum IFIR filter expansion factor was found by Mehrnia and Willson[\[14\]](#) to be

$$(7-33)$$

$$M_{\text{opt}} = \frac{1}{f_{\text{pass}} + f_{\text{stop}} + \sqrt{f_{\text{stop}} - f_{\text{pass}}}}.$$

We'll explore the meaning, and effects, of [Eq. \(7-33\)](#) in the next few pages, but first let's determine the percent computation reduction afforded to us by IFIR filters.

7.4.2 Estimating the Number of FIR Filter Taps

To estimate the computation reduction achieved by using IFIR filters, an algorithm is needed to compute the number of taps, N_{tfir} , in a traditional tapped-delay line FIR filter. Several authors have proposed empirical relationships for estimating N_{tfir} for traditional tapped-delay line FIR filters based on passband ripple, stopband attenuation, and transition region width[\[15–17\]](#). A particularly simple expression for N_{tfir} , giving results consistent with other estimates for passband ripple values near 0.1 dB, is

$$(7-34)$$

$$N_{\text{tfir}} \approx \frac{\text{Atten}}{22(f_{\text{stop}} - f_{\text{pass}})}$$

where $Atten$ is the stopband attenuation measured in dB, and f_{pass} and f_{stop} are the normalized frequencies in [Figure 7-15\(d\)\[17\]](#). (Again, by “normalized” we mean that the f_{pass} and f_{stop} frequency values are normalized to the filter input sample rate, f_s , in Hz. For example, $f_{\text{pass}} = 0.1$ is equivalent to a continuous-time frequency of $f_{\text{pass}} = 0.1f_s$ Hz.) Likewise, the number of taps in the prototype and image-reject subfilters can be estimated using

$$(7-34')$$

$$N_p \approx \frac{Atten}{22(M)(f_{\text{stop}} - f_{\text{pass}})}$$

$$(7-34'')$$

$$N_{\text{ir}} \approx \frac{Atten}{22(1/M - f_{\text{stop}} - f_{\text{pass}})}.$$

7.4.3 Modeling IFIR Filter Performance

As it turns out, IFIR filter computational workload reduction depends on the expansion factor M , the passband width, and the transition region width of the desired IFIR filter. To show this, we substitute the above expressions for N_{tfir} , N_p , and N_{ir} into [Eq. \(7-32\)](#) and write

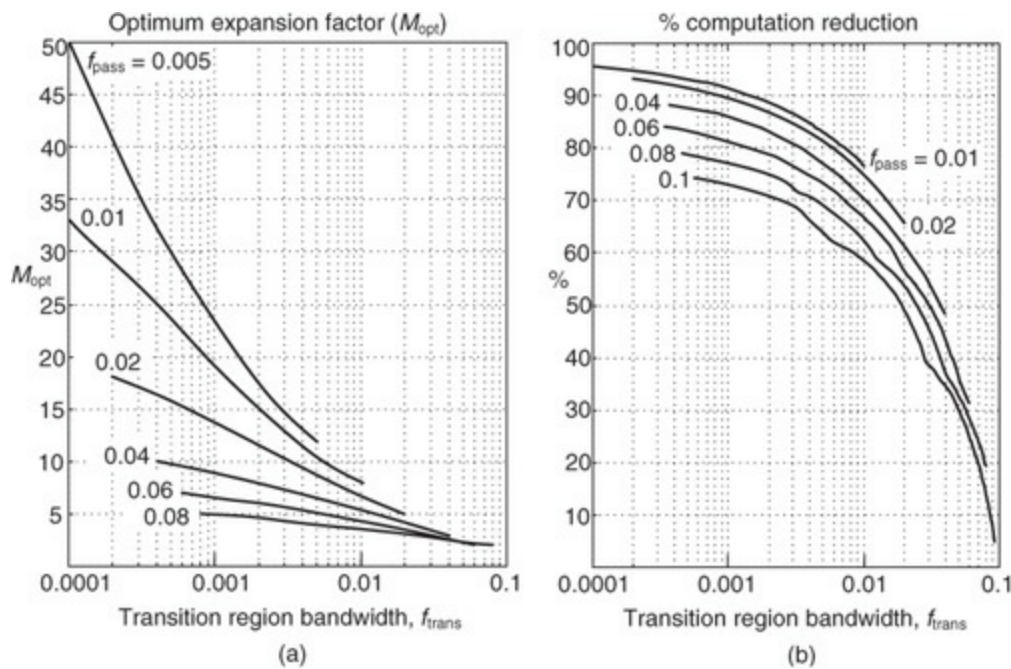
$$(7-35)$$

$$\% \text{ computation reduction} = 100 \cdot \left[\frac{M-1}{M} - \frac{Mf_{\text{trans}}}{1 - Mf_{\text{trans}} - 2Mf_{\text{pass}}} \right],$$

where $f_{\text{trans}} = f_{\text{stop}} - f_{\text{pass}}$.

Having [Eqs. \(7-33\)](#) and [\(7-35\)](#) available to us, we can now see the performance of IFIR filters. The optimum expansion factor curves from [Eq. \(7-33\)](#) are plotted, versus desired IFIR filter transition region width, for various values of passband width in [Figure 7-18\(a\)](#). When various optimum expansion factors are used in an IFIR filter design, the percent computation reduction, when an M_{opt} value is plugged into [Eq. \(7-35\)](#), is that shown in [Figure 7-18\(b\)](#).

Figure 7-18 IFIR filter performance versus desired transition region width for various passband widths: (a) optimum expansion factors; (b) percent computation reduction.



So in IFIR filter design, we use our desired filter transition region width and passband width values to determine the M_{opt} optimum expansion factor using either [Eq. \(7-33\)](#) or the curves in [Figure 7-18\(a\)](#). Given that M_{opt} value, we estimate our IFIR filter's percent computation reduction from either [Eq. \(7-35\)](#) or the curves in [Figure 7-18\(b\)](#). We'll go through an IFIR filter design example shortly.

7.4.4 IFIR Filter Implementation Issues

The computation reduction of IFIR filters is based on the assumption that they are implemented as two separate subfilters as in [Figure 7-16](#). We have resisted the temptation to combine the two subfilters into a single filter whose coefficients are the convolution of the subfilters' impulse responses. Such a maneuver would eliminate the zero-valued coefficients of the shaping subfilter, and we'd lose all our desired computation reduction.

The curves in [Figure 7-18\(a\)](#) indicate an important implementation issue when using IFIR filters. With decreasing IFIR filter passband width, larger expansion factors, M , can be used. When using programmable DSP chips, larger values of M require that a larger block of hardware *data memory*, in the form of a *circular buffer*, be allocated to hold a sufficient number of input $x(n)$ samples for the shaping subfilter. The size of this data memory must be equal to at least N_{sh} as indicated in [Eq. \(7-28\)](#). Some authors refer to this data memory allocation requirement, to accommodate all the stuffed zeros in the $h_{sh}(k)$ impulse response, as a disadvantage of IFIR filters. This is a misleading viewpoint because, as it turns out, the N_{sh} length of $h_{sh}(k)$ is only a few percent larger than the length of the impulse response of a traditional FIR filter having the same performance as an IFIR filter. So from a data storage standpoint the price we pay to use IFIR filters is a slight increase in the size of memory to accommodate N_{sh} , plus the data memory of size K_{ir} needed for the image-reject subfilter. In practice, for narrowband lowpass IFIR filters, K_{ir} is typically less than 10 percent of N_{sh} .

When implementing an IFIR filter with a programmable DSP chip, the filter's computation reduction gain can only be realized if the chip's architecture enables *zero-overhead looping* through the circular data memory using an increment equal to the expansion factor M . That looping capability ensures that only the nonzero-valued coefficients of $h_{sh}(k)$ are used in the shaping subfilter computations.

In practice the shaping and image-reject subfilters should be implemented with a *folded* tapped-delay line FIR structure, exploiting their impulse response symmetry, to reduce the number of necessary multiplications by a factor of two. (See [Section 13.7](#).) Using a folded structure does not alter the performance curves provided in [Figure 7-18](#). Regarding an IFIR filter's implementation in fixed-point hardware, its sensitivity to coefficient quantization errors is no greater than the errors exhibited by traditional FIR filters[\[12\]](#).

7.4.5 IFIR Filter Design Example

The design of practical lowpass IFIR filters is straightforward and comprises four steps:

1. Define the desired lowpass filter performance requirements.
2. Determine a candidate value for the expansion factor M .
3. Design and evaluate the shaping and image-reject subfilters.
4. Investigate IFIR performance for alternate expansion factors near the initial M value.

As a design example, refer to [Figure 7-15\(d\)](#) and assume we want to build a lowpass IFIR filter with $f_{\text{pass}} = 0.02$, a peak-peak passband ripple of 0.5 dB, a transition region bandwidth of $f_{\text{trans}} = 0.01$ (thus $f_{\text{stop}} = 0.03$), and 50 dB of stopband attenuation. First, we find the $f_{\text{trans}} = 0.01$ point on the abscissa of [Figure 7-18\(a\)](#) and follow it up to the point where it intersects the $f_{\text{pass}} = 0.02$ curve. This intersection indicates that we should start our design with an expansion factor of $M = 7$. (The same intersection point in [Figure 7-18\(b\)](#) suggests that we can achieve a computational workload reduction of roughly 75 percent.)

With $M = 7$, and applying [Eq. \(7-30\)](#), we use our favorite traditional FIR filter design software to design a linear-phase prototype FIR filter with the following parameters:

$$\begin{aligned} f_{\text{p-pass}} &= M(0.02) = 0.14, \\ \text{passband ripple} &= (0.5)/2 \text{ dB} = 0.25 \text{ dB}, \\ f_{\text{p-stop}} &= M(0.03) = 0.21, \text{ and} \\ \text{stopband attenuation} &= 50 \text{ dB}. \end{aligned}$$

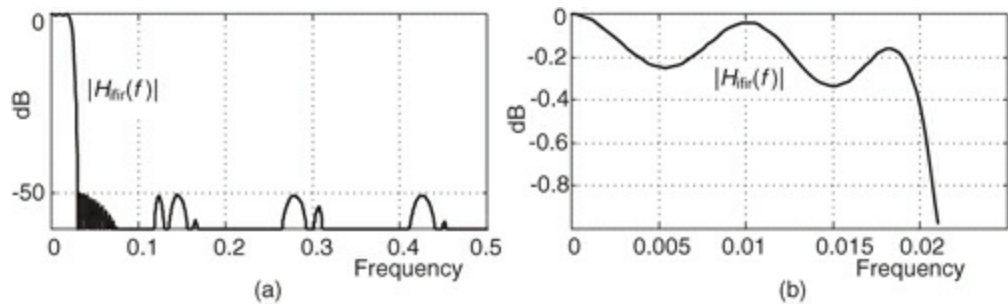
(Notice how we used our cascaded filters' passband ripple rule of thumb from [Section 6.8.1](#) to specify the prototype filter's passband ripple to be half our final desired ripple, and we'll do the same for the image-reject subfilter.) Such a prototype FIR filter will have $N_{\text{p}} = 33$ taps and, from [Eq. \(7-28\)](#), when expanded by $M = 7$ the shaping subfilter will have an impulse response length of $N_{\text{sh}} = 225$ samples.

Next, using [Eq. \(7-31\)](#), we design an image-reject subfilter having the following parameters:

$$\begin{aligned} f_{\text{ir-pass}} &= f_{\text{pass}} = 0.02, \\ \text{passband ripple} &= (0.5)/2 \text{ dB} = 0.25 \text{ dB}, \\ f_{\text{ir-stop}} &= 1/M - f_{\text{stop}} = 1/7 - 0.03 = 0.113, \text{ and} \\ \text{stopband attenuation} &= 50 \text{ dB}. \end{aligned}$$

This image-reject subfilter will have $N_{\text{ir}} = 27$ taps and when cascaded with the shaping subfilter will yield an IFIR filter requiring 60 multiplications per filter output sample. The frequency response of the IFIR filter is shown in [Figure 7-19\(a\)](#), with passband response detail provided in [Figure 7-19\(b\)](#).

Figure 7-19 IFIR filter design example magnitude responses: (a) full response; (b) passband response detail.



A traditional FIR filter satisfying our design example specifications would require approximately $N_{\text{tfir}} = 240$ taps. Because the IFIR filter requires only 60 multiplications per output sample, using [Eq. \(7-32\)](#), we have realized a computational workload reduction of 75 percent. The final IFIR filter design step is to sit back and enjoy a job well done.

Further modeling of our design example for alternate expansion factors yields the IFIR filter performance results in [Table 7-2](#). There we see how the M expansion factors of 5 through 8 provide very similar computational reductions and N_{sh} -sized data storage requirements for the shaping subfilter.

Table 7-2 Design Example Computation Reduction versus M

Expansion factor M	Number of taps			N_{sh} data storage	Computation reduction
	$h_{\text{sh}}(k)$	$h_{\text{ir}}(k)$	IFIR total		
3	76	8	84	226	65%
4	58	12	70	229	71%
5	46	17	63	226	74%
6	39	22	61	229	75%
7	33	27	60	225	75%
8	29	33	62	225	74%
9	26	41	67	226	72%
10	24	49	73	231	70%
11	21	60	81	221	66%

IFIR filters are suitable whenever narrowband lowpass linear-phase filtering is required, for example, the filtering prior to decimation for narrowband channel selection within wireless communications receivers, or in digital television. IFIR filters are essential components in sharp-transition wideband frequency-response masking FIR filters[\[18,19\]](#). In addition, IFIR filters can also be employed in narrowband two-dimensional filtering applications.

Additional, and more complicated, IFIR design methods have been described in the literature. Improved computational workload reduction, on the order of 30 to 40 percent beyond that presented here, has been reported using an intricate design scheme when the [Figure 7-16](#) image-reject subfilter is replaced with multiple stages of filtering[\[20\]](#).

If you “feel the need for speed,” there are additional ways to reduce the computational workload of IFIR filters. Those techniques are available in references [\[21\]](#) and [\[22\]](#). We will revisit IFIR filters in [Chapter 10](#) to see how they are used in sample rate conversion (decimation or interpolation)

applications.

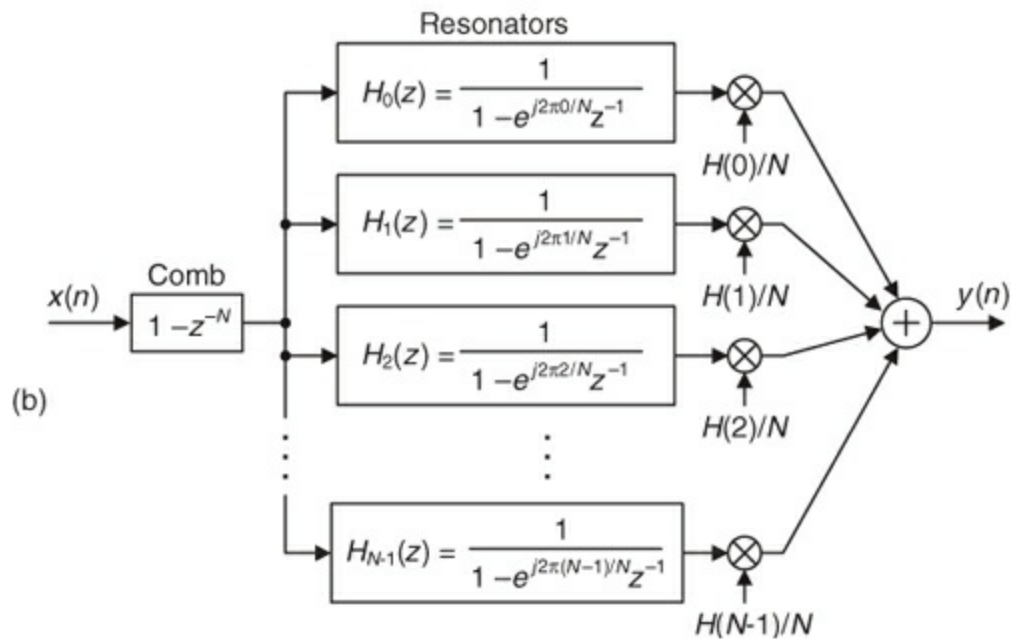
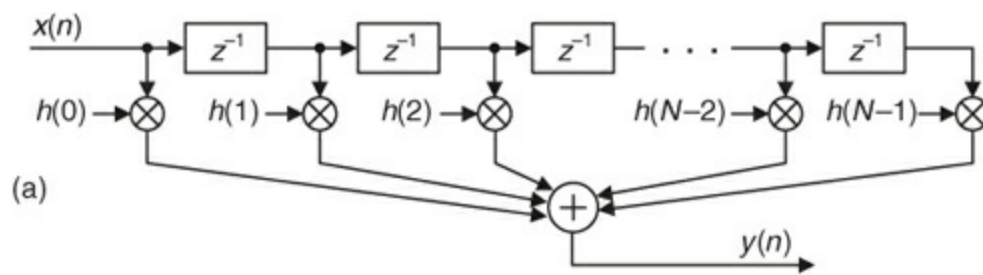
To conclude our linear-phase narrowband IFIR filter material, we reiterate that they can achieve significant computational workload reduction (as large as 90 percent) relative to traditional tapped-delay line FIR filters, at the cost of less than a 10 percent increase in hardware data memory requirements. Happily, IFIR implementation is a straightforward cascade of filters designed using readily available traditional FIR filter design software.

7.5 Frequency Sampling Filters: The Lost Art

This section describes a class of digital filters, called *frequency sampling filters*, used to implement linear-phase FIR filter designs. Although frequency sampling filters were developed over 35 years ago, the advent of the powerful Parks-McClellan tapped-delay line FIR filter design method has driven them to near obscurity. Thus in the 1970s frequency sampling filter implementations lost favor to the point where their coverage in today's DSP classrooms and textbooks ranges from very brief to nonexistent. However, we'll show how frequency sampling filters remain *more computationally efficient* than Parks-McClellan-designed filters for certain applications where the desired passband width is less than roughly one-fifth the sample rate. The purpose of this material is to introduce the DSP practitioner to the structure, performance, and design of frequency sampling filters, and to present a detailed comparison between a proposed high-performance frequency sampling filter implementation and its tapped-delay line FIR filter equivalent. In addition, we'll clarify and expand the literature of frequency sampling filters concerning the practical issues of phase linearity, filter stability, gain normalization, and computational workload using design examples.

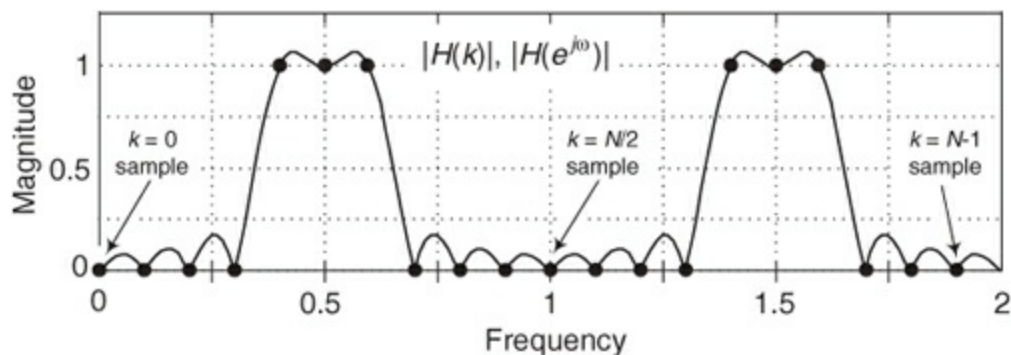
Frequency sampling filters were founded upon the fact that a traditional N -tap nonrecursive tapped-delay line (direct convolution) FIR filter as shown in [Figure 7-20\(a\)](#) can be implemented as a comb filter in cascade with a bank of N complex resonators as shown in [Figure 7-20\(b\)](#). We call the filter in [Figure 7-20\(b\)](#) a general *frequency sampling filter* (FSF), and its equivalence to the nonrecursive FIR filter has been verified[23–25]. While the $h(k)$ coefficients, where $0 < k < N-1$, of N -tap nonrecursive FIR filters are typically real-valued, in general they can be complex, and that's the initial assumption made in equating the two filters in [Figure 7-20](#). The $H(k)$ gain factors, the discrete Fourier transform of the $h(k)$ time-domain coefficients, are, in the general case, complex values represented by $|H(k)|e^{j\phi(k)}$.

Figure 7-20 FIR filters: (a) N -tap nonrecursive tapped-delay line; (b) equivalent N -section frequency sampling filter.



The basis of FSF design is the definition of a desired FIR filter frequency response in the form of $H(k)$ frequency-domain samples, whose magnitudes are depicted as dots in [Figure 7-21](#). Next, those complex $H(k)$ sample values are used as gain factors following the resonators in the FSF structure (block diagram). If you haven't seen it before, please don't be intimidated by this apparently complicated FSF structure. We'll soon understand every part of it, and how those parts work together.

Figure 7-21 Defining a desired filter response by frequency sampling.



Later we'll develop the math to determine the interpolated (actual) frequency magnitude response $|H(e^{j\omega})|$ of an FSF shown by the continuous curve in [Figure 7-21](#). In this figure, the frequency axis labeling convention is a normalized angle measured in π radians/sample with the depicted ω frequency range covering 0 to 2π radians/sample, corresponding to a cyclic frequency range of 0 to f_s , where f_s is the sample rate in Hz.

To avoid confusion, we remind the reader that there is a popular nonrecursive FIR filter design technique known as the *frequency sampling design method* described in the DSP literature. That

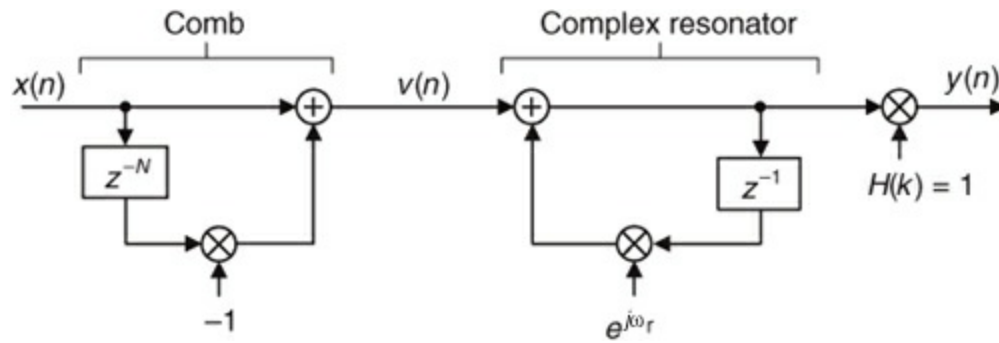
design scheme begins (in a manner similar to an FSF design) with the definition of desired $H(k)$ frequency response samples, then an inverse discrete Fourier transform is performed on those samples to obtain a time-domain impulse response sequence that's used as the $h(k)$ coefficients in the nonrecursive N -tap FIR structure of [Figure 7-20\(a\)](#). In the FSF design method described here, the desired frequency-domain $H(k)$ sample values are the coefficients used in the FSF structure of [Figure 7-20\(b\)](#) which is typically called the *frequency sampling implementation* of an FIR filter.

Although more complicated than nonrecursive FIR filters, FSFs deserve study because in many narrowband filtering situations they can implement a linear-phase FIR filter at a reduced computational workload relative to an N -tap nonrecursive FIR filter. The computation reduction occurs because, while all of the $h(k)$ coefficients are used in the nonrecursive FIR filter implementation, most of the $H(k)$ values will be zero-valued, corresponding to the stopband, and need not be implemented. To understand the function and benefits of FSFs, we start by considering the behavior of the comb filter and then review the performance of a single digital resonator.

7.5.1 Comb Filter and Complex Resonator in Cascade

A single section of a complex FSF is a comb filter followed by a single complex digital resonator as shown in [Figure 7-22](#).

Figure 7-22 A single section of a complex FSF.



The $1/N$ gain factor following a resonator in [Figure 7-20\(b\)](#) is omitted, for simplicity, from the single-section complex FSF. (The effect of including the $1/N$ factor will be discussed later.) To understand the single-section FSF's operation, we first review the characteristics of the nonrecursive comb filter whose time-domain difference equation is

$$(7-36)$$

$$v(n) = x(n) - x(n-N),$$

with its output equal to the input sequence minus the input delayed by N samples. The comb filter's z -domain transfer function is

$$(7-37)$$

$$H_{\text{comb}}(z) = \frac{V(z)}{X(z)} = 1 - z^{-N}.$$

The frequency response of a comb filter, derived in [Section G.1](#) of [Appendix G](#), is

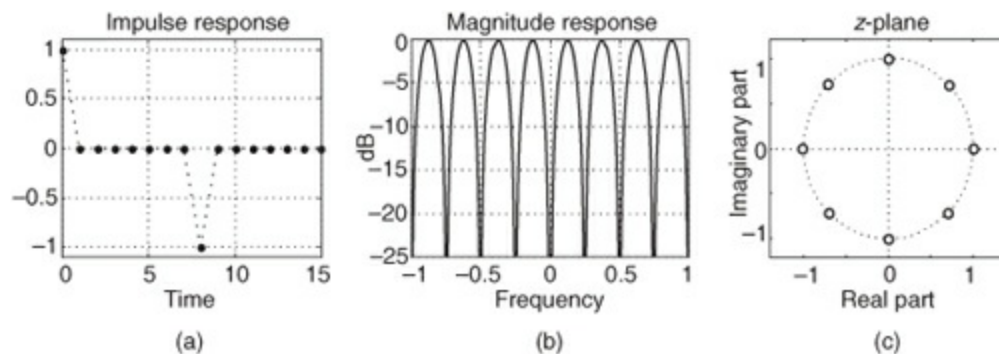
$$(7-38)$$

$$H_{\text{comb}}(e^{j\omega}) = e^{-j(\omega N - \pi)/2} 2\sin(\omega N/2),$$

with a magnitude response of $|H_{\text{comb}}(e^{j\omega})| = 2|\sin(\omega N/2)|$ whose maximum value is 2. It's meaningful

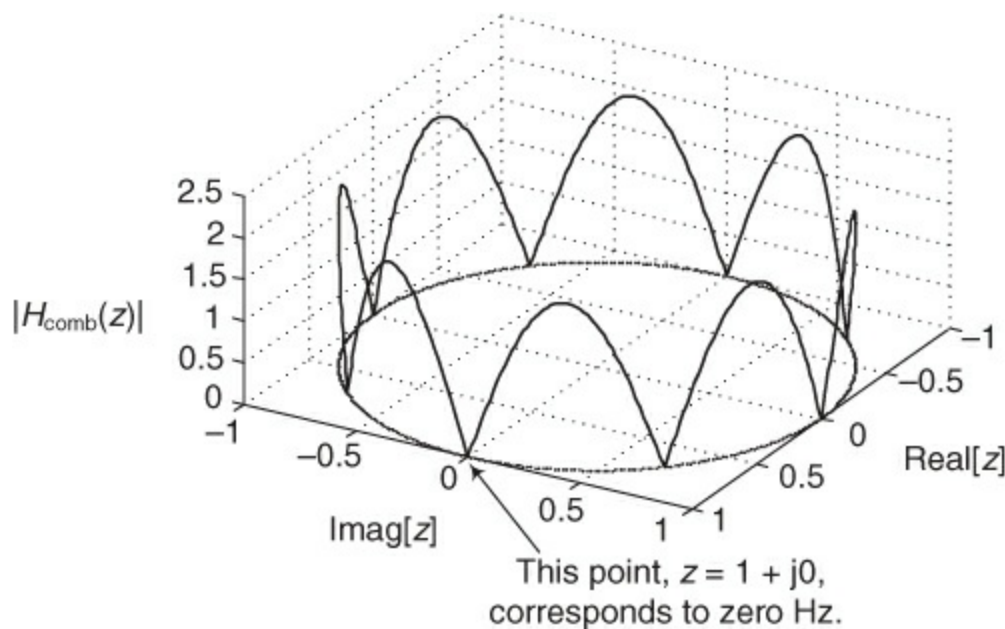
to view the comb filter's time-domain impulse response and frequency-domain magnitude response as shown in [Figure 7-23](#) for $N = 8$. The magnitude response makes it clear why the term *comb* is used.

Figure 7-23 Time- and frequency-domain characteristics of an $N = 8$ comb filter.



[Equation \(7-37\)](#) leads to a key feature of this comb filter; its transfer function has N periodically spaced zeros around the z -plane's unit circle as shown in [Figure 7-23\(c\)](#). Each of those zeros, located at $z(k) = e^{j2\pi k/N}$, where $k = 0, 1, 2, \dots, N-1$, corresponds to a magnitude null in [Figure 7-23\(b\)](#), where the normalized frequency axis is labeled from $-\pi$ to $+\pi$ radians/sample. Those $z(k)$ values are the N roots of unity when we set [Eq. \(7-37\)](#) equal to zero, yielding $z(k)^N = (e^{j2\pi k/N})^N = 1$. We can combine the magnitude response (on a linear scale) and z -plane information in the three-dimensional z -plane depiction shown in [Figure 7-24](#), where we see the intersection of the $|H_{\text{comb}}(z)|$ surface and the unit circle. Breaking the curve at the $z = -1$ point, and laying it flat, corresponds to the magnitude curve in [Figure 7-23\(b\)](#).

Figure 7-24 The z -plane frequency magnitude response of the $N = 8$ comb filter.



To preview where we're going, soon we'll build an FSF by cascading the comb filter with a digital resonator having a transfer function pole lying on top of one of the comb's z -plane zeros, resulting in a linear-phase bandpass filter. With this thought in mind, let's characterize the digital resonator in [Figure 7-22](#).

The complex resonator's time-domain difference equation is

$$(7-39)$$

$$y(n) = v(n) + e^{j\omega_r} y(n-1),$$

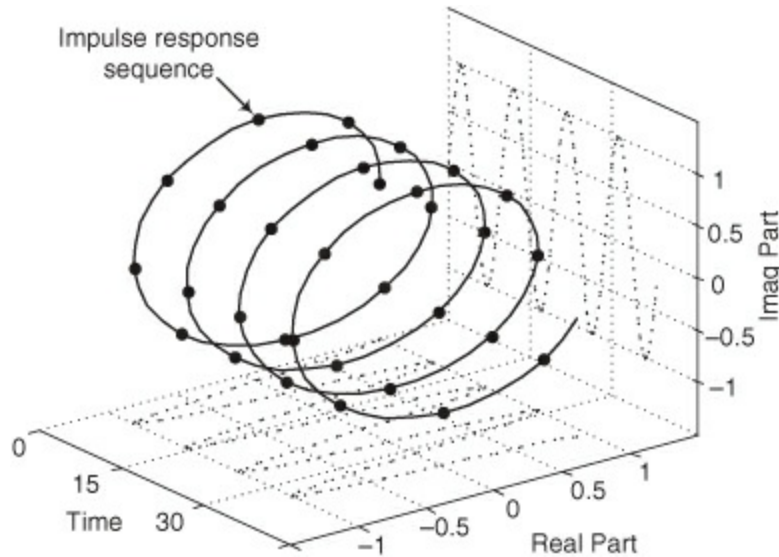
where the angle ω_r , $-\pi \leq \omega_r \leq \pi$ determines the resonant frequency of our resonator. We show this by considering the resonator's z -domain transfer function

(7-40)

$$H_{\text{res}}(z) = \frac{Y(z)}{V(z)} = \frac{1}{1 - e^{j\omega_r} z^{-1}}$$

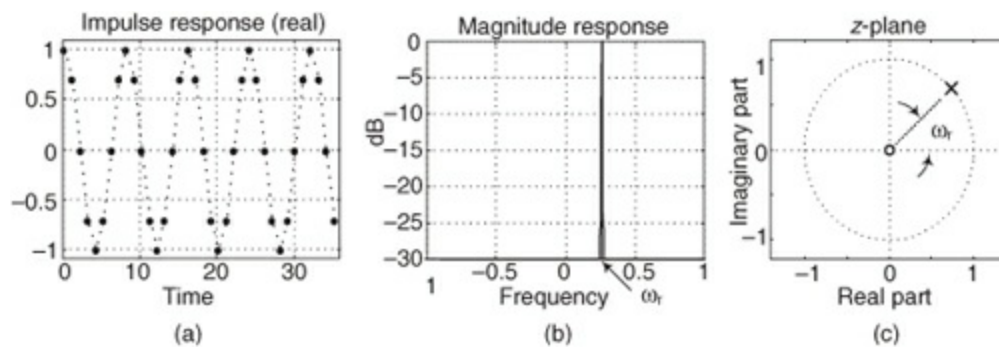
and the resonator's complex time-domain impulse response, for $\omega_r = \pi/4$, in [Figure 7-25](#).

Figure 7-25 Single complex digital resonator impulse response with $\omega_r = \pi/4$.



The $\omega_r = \pi/4$ resonator's impulse response is a complex sinusoid, the real part (a cosine sequence) of which is plotted in [Figure 7-26\(a\)](#), and can be considered infinite in duration. (The imaginary part of the impulse response is, as we would expect, a sinewave sequence.) The frequency magnitude response is very narrow and centered at ω_r . The resonator's $H_{\text{res}}(z)$ has a single zero at $z = 0$, but what concerns us most is its pole, at $z = e^{j\omega_r}$, on the unit circle at an angle of ω_r as shown in [Figure 7-26\(c\)](#). We can think of the resonator as an infinite impulse response (IIR) filter that's conditionally stable because its pole is neither inside nor outside the unit circle.

Figure 7-26 Time- and frequency-domain characteristics of a single complex digital resonator with $\omega_r = \pi/4$.



We now analyze the single-section complex FSF in [Figure 7-22](#). The z -domain transfer function of this FSF is the product of the individual transfer functions and $H(k)$, or

(7-41)

$$H(z) = H_{\text{comb}}(z) H_{\text{res}}(z) H(k) = (1 - z^{-N}) \frac{H(k)}{1 - e^{j\omega_r} z^{-1}}.$$

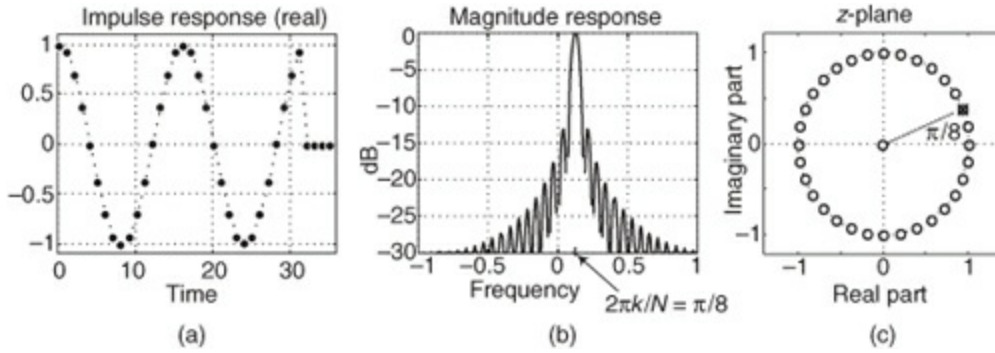
If we restrict the resonator's resonant frequency ω_r to be $2\pi k/N$, where $k = 0, 1, 2, \dots, N-1$, then the resonator's z -domain pole will be located atop one of the comb's zeros and we'll have an FSF transfer function of

(7-42)

$$H_{\text{ss}}(z) = (1 - z^{-N}) \frac{H(k)}{1 - e^{j2\pi k/N} z^{-1}},$$

where the “ss” subscript means a single-section complex FSF. We can understand a single-section FSF by reviewing its time- and frequency-domain behavior for $N = 32$, $k = 2$, and $H(2) = 1$ as shown in [Figure 7-27](#).

Figure 7-27 Time- and frequency-domain characteristics of a single-section complex FSF where $N = 32$, $k = 2$, and $H(2) = 1$.



[Figure 7-27](#) is rich in information. We see that the complex FSF's impulse response is a truncated complex sinusoid whose real part is shown in [Figure 7-27\(a\)](#). The positive impulse from the comb filter started the resonator oscillation at zero time. Then at just the right sample, $N = 32$ samples later, which is $k = 2$ cycles of the sinusoid, the negative impulse from the comb arrives at the resonator to cancel all further oscillation. The frequency magnitude response, being the Fourier transform of the truncated sinusoidal impulse response, takes the form of a $\sin(x)/x$ -like function. In the z -plane plot of [Figure 7-27](#), the resonator's pole is indeed located atop the comb filter's $k = 2$ zero on the unit circle, canceling the frequency magnitude response null at $2\pi k/N = \pi/8$ radians. (Let's remind ourselves that a normalized angular frequency of $2\pi k/N$ radians/sample corresponds to a cyclic frequency of $k f_s/N$, where f_s is the sample rate in Hz. Thus the filter in [Figure 7-27](#) resonates at $f_s/16$ Hz.)

We can determine the FSF's interpolated frequency response by evaluating the $H_{\text{ss}}(z)$ transfer function on the unit circle. Substituting $e^{j\omega}$ for z in $H_{\text{ss}}(z)$ in [Eq. \(7-42\)](#), as detailed in [Appendix G, Section G.2](#), we obtain an $H_{\text{ss}}(e^{j\omega})$ frequency response of

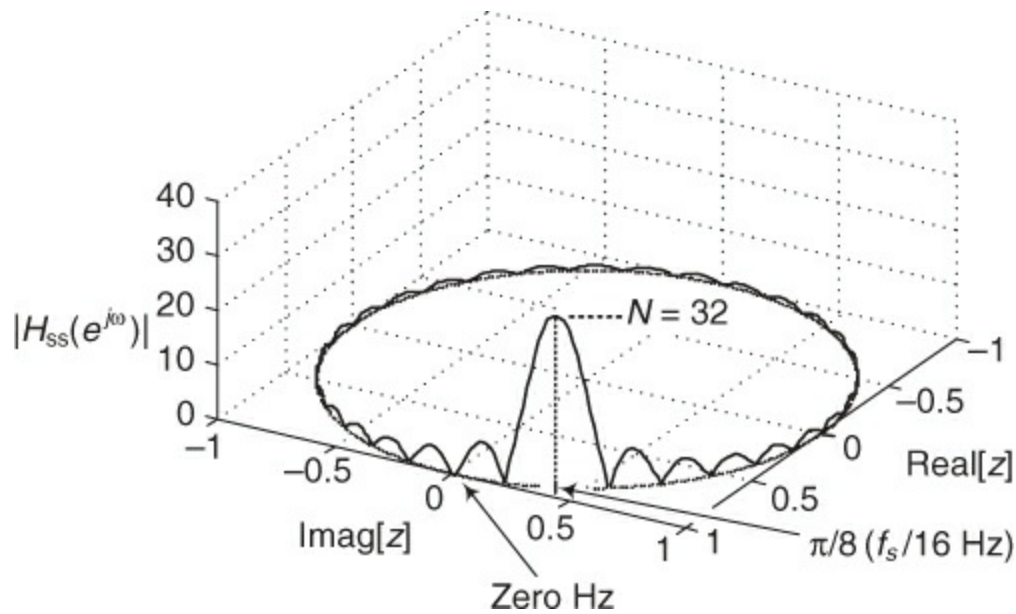
(7-43)

$$H_{\text{ss}}(e^{j\omega}) = H_{\text{ss}}(z)|_{z=e^{j\omega}} = e^{-j\omega(N-1)/2} e^{-j\pi k/N} H(k) \frac{\sin(\omega N/2)}{\sin(\omega/2 - \pi k/N)}.$$

Evaluating $|H_{\text{ss}}(e^{j\omega})|$ over the frequency range of $-\pi < \omega < \pi$ yields the curve in [Figure 7-27\(b\)](#). Our single-section FSF has linear phase because the $e^{-j\pi k/N}$ term in [Eq. \(7-43\)](#) is a fixed phase angle

based on constants N and k , the angle of $H(k)$ is fixed, and the $e^{-j\omega(N-1)/2}$ phase term is a linear function of frequency (ω). As derived in [Appendix G, Section G.2](#), the maximum magnitude response of a single-section complex FSF is N when $|H(k)| = 1$, and we illustrate this fact in [Figure 7-28](#).

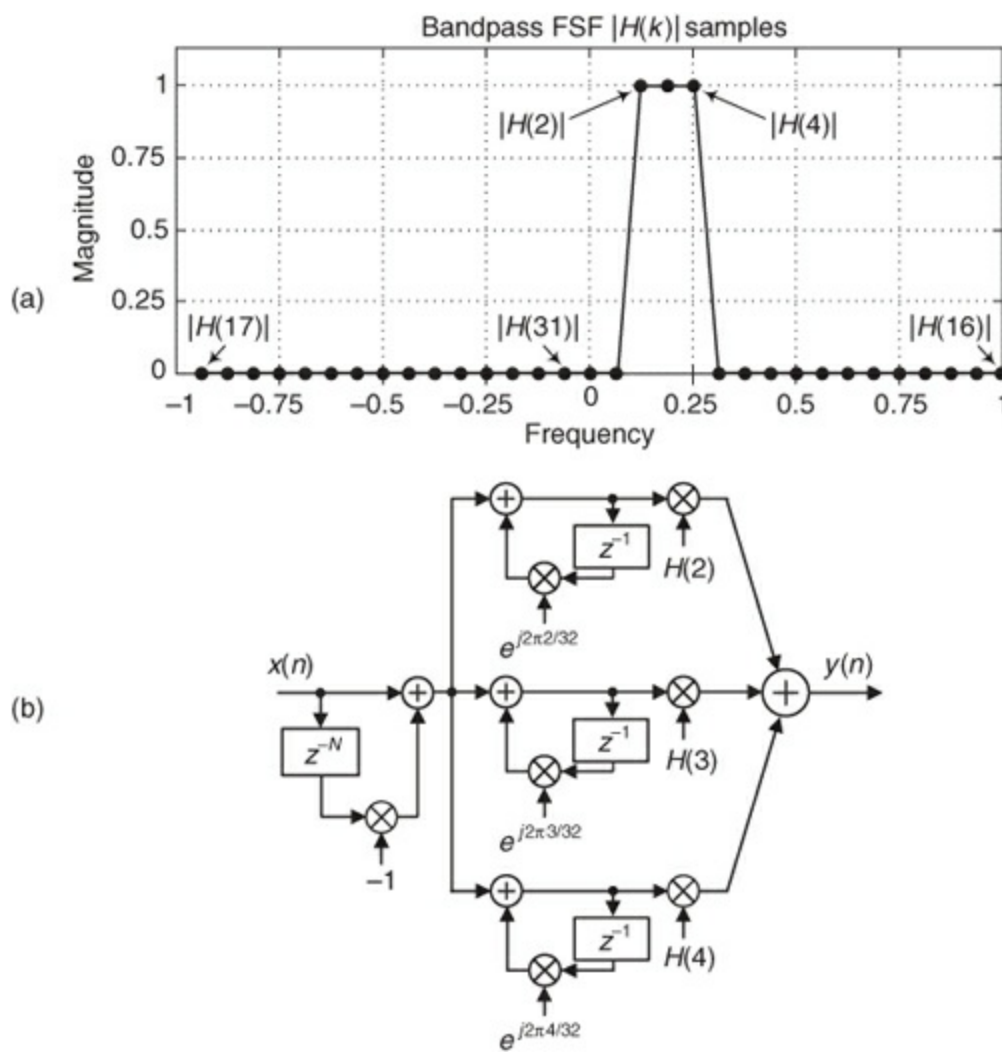
Figure 7-28 The z -plane frequency magnitude response of a single-section complex FSF with $N = 32$ and $k = 2$.



7.5.2 Multisection Complex FSFs

In order to build useful FSFs we use multiple resonator sections, as indicated in [Figure 7-20\(b\)](#), to provide bandpass FIR filtering. For example, let's build a three-section complex bandpass FSF by establishing the following parameters: $N = 32$, and the nonzero frequency samples are $H(2)$, $H(3)$, and $H(4)$. The desired frequency magnitude response is shown in [Figure 7-29\(a\)](#) with the bandpass FSF structure provided in [Figure 7-29\(b\)](#).

Figure 7-29 Three-section $N = 32$ complex FSF: (a) desired frequency magnitude response; (b) implementation.



Exploring this scenario, recall that the z -domain transfer function of parallel filters is the sum of the individual transfer functions. So, the transfer function of an N -section complex FSF from [Eq. \(7-42\)](#) is

(7-44)

$$H_{\text{cplx}}(z) = (1 - z^{-N}) \sum_{k=0}^{N-1} \frac{H(k)}{1 - e^{j2\pi k/N} z^{-1}},$$

where the subscript “cplx” means a complex multisection FSF.

Let’s pause for a moment to understand [Eq. \(7-44\)](#); the first factor on the right side represents a comb filter, and the comb is in cascade (multiplication) with the sum of ratio terms. The summation of the ratios (each ratio is a resonator) means those resonators are connected in parallel. Recall from [Section 6.8.1](#) that the combined transfer function of filters connected in parallel is the sum of the individual transfer functions. It’s important to be comfortable with the form of [Eq. \(7-44\)](#) because we’ll be seeing many similar expressions in the material to come.

So a comb filter is driving a bank of resonators. For an $N = 32$ complex FSF we could have up to 32 resonators, but in practice only a few resonators are needed for narrowband filters. In [Figure 7-29](#), we used only three resonators. That’s the beauty of FSFs; most of the $H(k)$ gain values in [Eq. \(7-44\)](#) are zero-valued and those resonators are not implemented, keeping the FSF computationally efficient.

Using the same steps as in [Appendix G, Section G.2](#), we can write the frequency response of a multisection complex FSF, such as in [Figure 7-29](#), as

(7-45)

$$H_{\text{cplx}}(e^{j\omega}) = e^{-j\omega(N-1)/2} \sum_{k=0}^{N-1} \frac{H(k)e^{-j\pi k/N} \sin(\omega N/2)}{\sin(\omega/2 - \pi k/N)}.$$

The designer of a multisection complex FSF can achieve any desired filter phase response by specifying the $\phi(k)$ phase angle value of each nonzero complex $H(k) = |H(k)|e^{j\phi(k)}$ gain factor. However, to build a linear-phase complex FSF, the designer must (1) specify the $\phi(k)$ phase values to be a linear function of frequency, and (2) define the $\phi(k)$ phase sequence so its slope is $-(N-1)/2$. This second condition forces the FSF to have a positive time delay of $(N-1)/2$ samples, as would the N -tap nonrecursive FIR filter in [Figure 7-20\(a\)](#). The following expressions for $\phi(k)$, with N being even, satisfy those two conditions.

(7-46)

$$\phi(k) = k \frac{2\pi}{N} \frac{-(N-1)}{2} = \frac{-k\pi(N-1)}{N}, \quad k = 0, 1, 2, \dots, \frac{N}{2} - 1.$$

(7-46')

$$\phi(N/2) = 0.$$

(7-46'')

$$\phi(k) = (N-k) \frac{2\pi}{N} \frac{(N-1)}{2} = \frac{\pi(N-k)(N-1)}{N}, \quad k = \frac{N}{2} + 1, \dots, N-1.$$

If N is odd, the linear-phase $H(k)$ phase values are

(7-47)

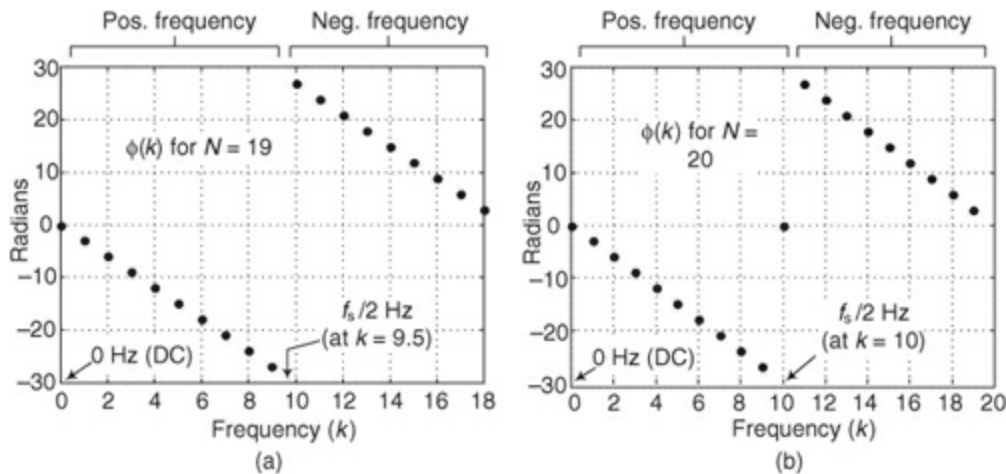
$$\phi(k) = k \frac{2\pi}{N} \frac{-(N-1)}{2} = \frac{-k\pi(N-1)}{N}, \quad k = 0, 1, 2, \dots, \frac{N-1}{2}.$$

(7-47')

$$\phi(k) = (N-k) \frac{2\pi}{N} \frac{(N-1)}{2} = \frac{\pi(N-k)(N-1)}{N}, \quad k = \frac{N+1}{2}, \dots, N-1.$$

Two example linear-phase $\phi(k)$ sequences, for $N = 19$ and $N = 20$, are shown in [Figure 7-30](#). The $\phi(0) = 0$ values set the phase to be zero at zero Hz, and the $\phi(N/2) = 0$, at the cyclic frequency of $f_s/2$ in [Figure 7-30\(b\)](#), ensures a symmetrical time-domain impulse response.

Figure 7-30 Linear phase of $H(k)$ for a single-section FSF: (a) $N = 19$; (b) $N = 20$.



Assigning the appropriate phase for the nonzero $H(k)$ gain factors is, however, only half the story in building a multisection FSF. There's good news to be told. Examination of the frequency response in [Eq. \(7-45\)](#) shows us a simple way to achieve phase linearity in practice. Substituting $|H(k)|e^{j\phi(k)}$, with

$\phi(k)$ defined by [Eq. \(7-46\)](#) above, for $H(k)$ in [Eq. \(7-45\)](#) provides the expression for the frequency response of an even- N multisection linear-phase complex FSF,

(7-48)

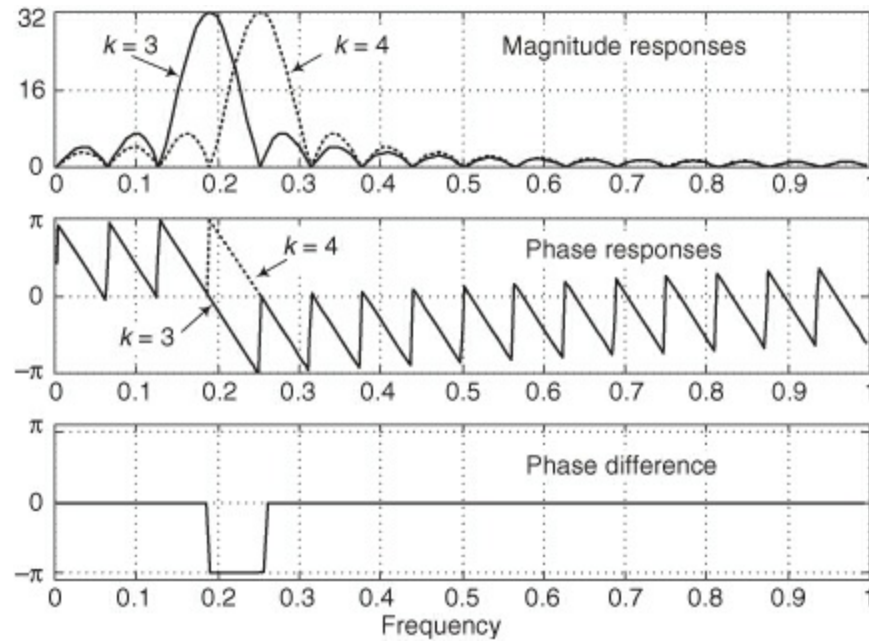
$$H_{\text{cplx,lp}}(e^{j\omega}) = e^{-j\omega(N-1)/2} \sin(\omega N/2) \times \left[\frac{|H(N/2)| e^{-j\pi/2}}{\sin(\omega/2 - \pi/2)} + \sum_{k=0}^{(N/2)-1} \frac{|H(k)| (-1)^k}{\sin(\omega/2 - \pi k/N)} - \sum_{k=(N/2)+1}^{N-1} \frac{|H(k)| (-1)^k}{\sin(\omega/2 - \pi k/N)} \right],$$

where the “lp” subscript indicates linear phase.

[Equation \(7-48\)](#) is not as complicated as it looks. It merely says the total FSF frequency response is the sum of individual resonators’ $\sin(x)/x$ -like frequency responses. The first term within the brackets represents the resonator centered at $k = N/2$ ($f_s/2$). The first summation is the positive-frequency resonators and the second summation represents the negative-frequency resonators.

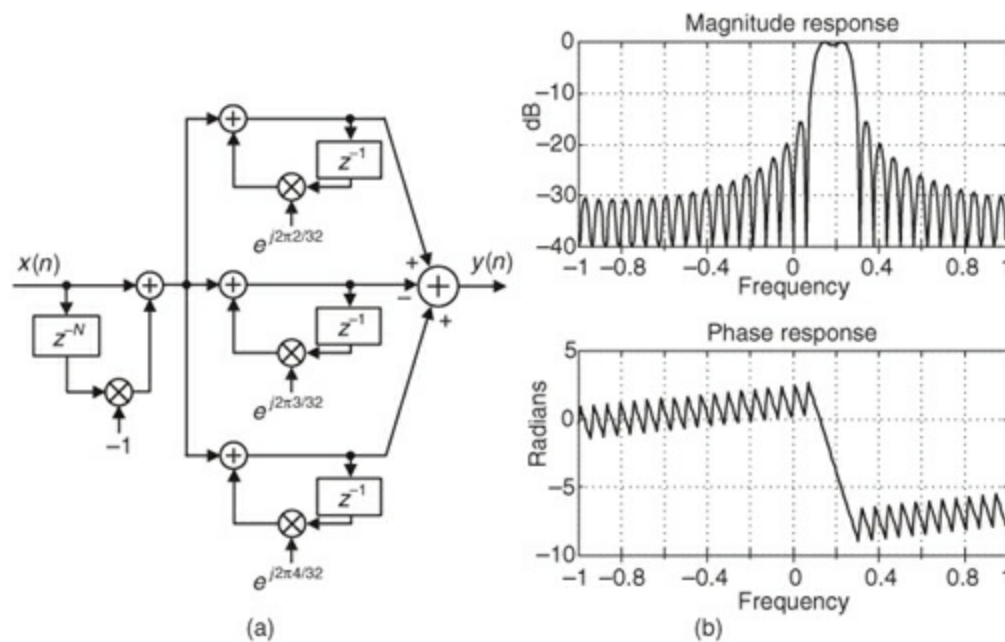
The $(-1)^k$ terms in the numerators of [Eq. \(7-48\)](#) deserve our attention because they are an alternating sequence of plus and minus ones. Thus a single-section frequency response will be 180° out of phase relative to its neighboring section. That is, the outputs of neighboring single-section FSFs will have a fixed π -radians phase difference over the passband common to both filters as shown in [Figure 7-31](#). (The occurrence of the $(-1)^k$ factors in [Eq. \(7-48\)](#) is established in [Appendix G, Section G.3](#).)

Figure 7-31 Comparison of the magnitude and phase responses, and phase difference, between the $k = 3$ and the $k = 4$ FSFs, when $N = 32$.



The effect of those $(-1)^k$ factors is profound and not emphasized nearly enough in the literature of FSFs. Rather than defining each nonzero complex $H(k)$ gain factor with its linearly increasing phase angles $\phi(k)$, we can build a linear-phase multisection FSF by using just the $|H(k)|$ magnitude values and incorporating the alternating signs for those real-valued gain factors. In addition, if the nonzero $|H(k)|$ gain factors are all equal to one, we avoid [Figure 7-29](#)’s gain factor multiplications altogether as shown in [Figure 7-32\(a\)](#).

Figure 7-32 Simplified $N = 32$ three-section linear-phase complex bandpass FSF: (a) implementation; (b) frequency response.



The unity-valued $|H(k)|$ gain factors and the alternating-signed summation allow the complex gain multiplies in [Figure 7-29\(b\)](#) to be replaced by simple adds and subtracts as in [Figure 7-32\(a\)](#). We add the even- k and subtract the odd- k resonator outputs. [Figure 7-32\(b\)](#) confirms the linear phase, with phase discontinuities at the magnitude nulls, of these multisection complex FSFs. The transfer function of the simplified complex linear-phase FSF is

(7-49)

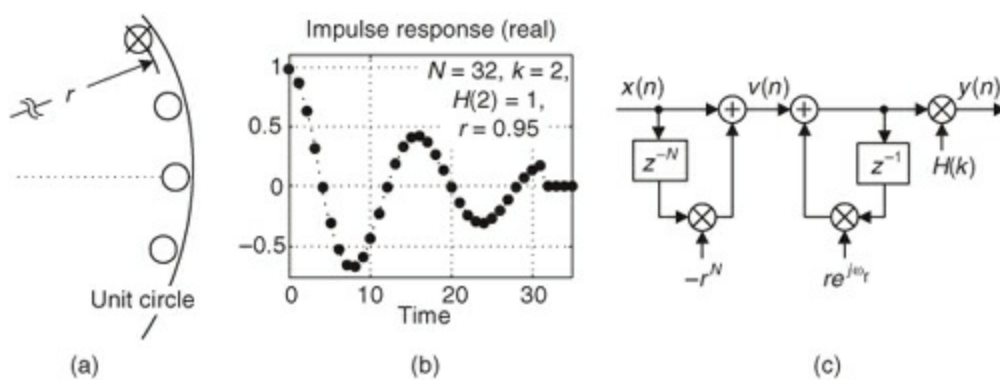
$$H_{\text{cplx}}(z) = (1 - z^{-N}) \sum_{k=0}^{N-1} \frac{(-1)^k}{1 - e^{j2\pi k/N} z^{-1}}.$$

(We didn't use the "lp" subscript, meaning linear phase, in [Eq. \(7-49\)](#) because, from here on, all our complex FSFs will be linear phase.)

7.5.3 Ensuring FSF Stability

So far we've discussed complex FSFs with pole/zero cancellation on the unit circle. However, in practice exact cancellation requires infinite-precision arithmetic, and real-world binary word quantization errors in the FSF's coefficients can make the filter poles lie outside the unit circle. The result would be an unstable filter, whose impulse response is no longer finite in duration, which must be avoided. (This is a beautiful example of the time-honored axiom "In theory, there's no difference between theory and practice. In practice, sometimes the *theory* doesn't work.") Even if a pole is located only very slightly outside the unit circle, roundoff noise will grow as time increases, corrupting the output samples of the filter. We prevent this problem by moving the comb filter's zeros and the resonators' poles just inside the unit circle as depicted in [Figure 7-33\(a\)](#). Now the zeros and a pole are located on a circle of radius r , where the damping factor r is just slightly less than 1.

Figure 7-33 Ensuring FSF stability: (a) poles and zeros are inside the unit circle; (b) real part of a stable single-section FSF impulse response; (c) FSF structure.



We call r the damping factor because a single-stage FSF impulse response becomes a damped sinusoid. For example, the real part of the impulse response of a single-stage complex FSF, where $N = 32$, $k = 2$, $H(2) = 2$, and $r = 0.95$, is shown in [Figure 7-33\(b\)](#). Compare that impulse response to [Figure 7-27\(a\)](#). The structure of a single-section FSF with zeros and a pole inside the unit circle is shown in [Figure 7-33\(c\)](#).

The comb filter's feedforward coefficient is $-r^N$ because the new z -domain transfer function of this comb filter is

$$(7-50)$$

$$H_{\text{comb}, r < 1}(z) = \frac{V(z)}{X(z)} = 1 - r^N z^{-N},$$

with the N zeros for this comb being located at $z_{r < 1}(k) = re^{j2\pi k/N}$, where $k = 0, 1, 2, \dots, N-1$.

Those $z_{r < 1}(k)$ values are the N roots of r^N when we set [Eq. \(7-50\)](#) equal to zero, yielding $z_{r < 1}(k)^N = (re^{j2\pi k/N})^N = r^N$. The z -domain transfer function of the resonator in [Figure 7-33\(b\)](#), with its pole located at a radius of r , at an angle of $2\pi k/N$, is

$$(7-51)$$

$$H_{\text{res}, r < 1}(z) = \frac{1}{1 - re^{j2\pi k/N} z^{-1}},$$

leading us to the transfer function of a guaranteed-stable single-section complex FSF of

$$(7-52)$$

$$H_{\text{gs,ss}}(z) = H_{\text{comb}, r < 1}(z) H_{\text{res}, r < 1}(z) H(k) = (1 - r^N z^{-N}) \frac{H(k)}{1 - [re^{j2\pi k/N}] z^{-1}}$$

whose implementation is shown in [Figure 7-33\(c\)](#). The subscript “gs,ss” means a guaranteed-stable single-section FSF. The z -domain transfer function of a guaranteed-stable N -section complex FSF is

$$(7-53)$$

$$H_{\text{gs,cplx}}(z) = (1 - r^N z^{-N}) \sum_{k=0}^{N-1} \frac{H(k)}{1 - [re^{j2\pi k/N}] z^{-1}}$$

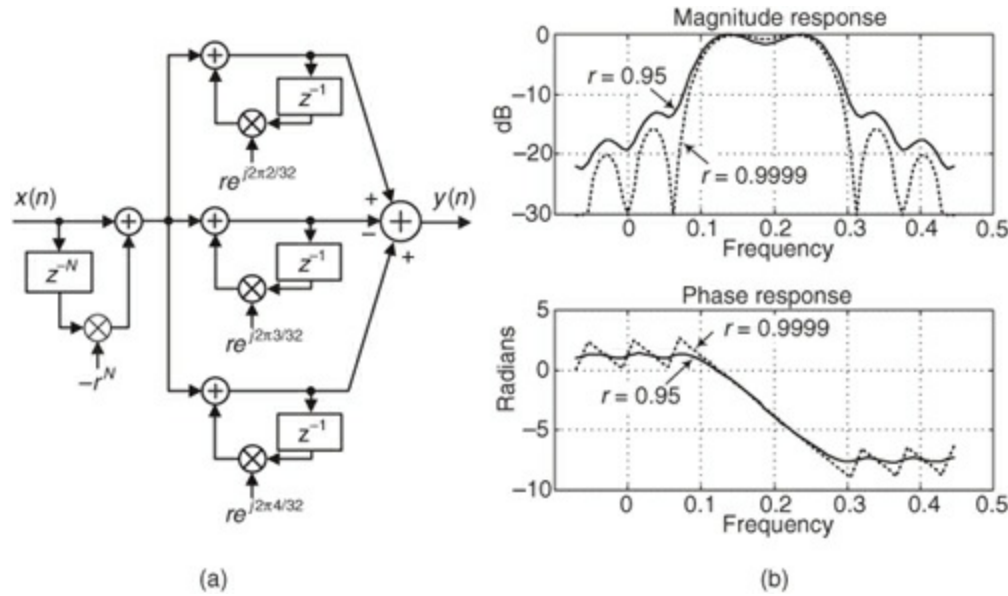
where the subscript “gs,cplx” means a guaranteed-stable complex multisection FSF. The frequency response of a guaranteed-stable multisection complex FSF (derived in [Appendix G, Section G.4](#)) is

$$(7-54)$$

$$H_{\text{gs, cplx}}(e^{j\omega}) = \sqrt{r^{N-1}} e^{-j\omega(N-1)/2} \sum_{k=0}^{N-1} \frac{H(k) e^{-j\pi k/N} \sinh[N \ln(r)/2 - jN\omega/2]}{\sinh[\ln(r)/2 - j(\omega - 2\pi k/N)/2]}.$$

If we modify the bandpass FSF structure in [Figure 7-32\(a\)](#) to force the zeros and poles inside the unit circle, we have the structure shown in [Figure 7-34\(a\)](#). The frequency-domain effects of zeros and poles inside the unit circle are significant, as can be seen in [Figure 7-34\(b\)](#) for the two cases where $r = 0.95$ and $r = 0.9999$.

Figure 7-34 Guaranteed-stable $N = 32$ three-section linear-phase complex bandpass FSF: (a) implementation; (b) frequency response for two values of damping factor r .



[Figure 7-34\(b\)](#) shows how a value of $r = 0.95$ badly corrupts our complex bandpass FSF performance; the stopband attenuation is degraded, and significant phase nonlinearity is apparent. Damping factor r values of less than unity cause phase nonlinearity because the filter is in a *nonreciprocal zero* condition. Recall a key characteristic of FIR filters: To maintain linear phase, any z -plane zero located inside the unit circle at $z = z_{r<1}(k)$, where $z_{r<1}(k)$ is not equal to 0, must be accompanied by a zero at a reciprocal location, namely, $z = 1/z_{r<1}(k)$ outside the unit circle. We do not satisfy this condition here, leading to phase nonlinearity. (The reader should have anticipated nonlinear phase due to the asymmetrical impulse response in [Figure 7-33\(b\)](#).) The closer we can place the zeros to the unit circle, the more linear the phase response. So the recommendation is to define r to be as close to unity as your binary number format allows[\[26\]](#). If integer arithmetic is used, set $r = 1 - 1/2^B$, where B is the number of bits used to represent a filter coefficient magnitude.

Another stabilization method worth considering is decrementing the largest component (either real or imaginary) of a resonator's $e^{j2\pi k/N}$ feedback coefficient by one least significant bit. This technique can be applied selectively to problematic resonators and is effective in combating instability due to rounding errors, which results in finite-precision $e^{j2\pi k/N}$ coefficients having magnitudes greater than unity.

Thus far we've reviewed FSFs with complex coefficients and frequency magnitude responses not symmetrical about zero Hz. Next we explore FSFs with real-only coefficients having conjugate-symmetric frequency magnitude and phase responses.

7.5.4 Multisection Real-Valued FSFs

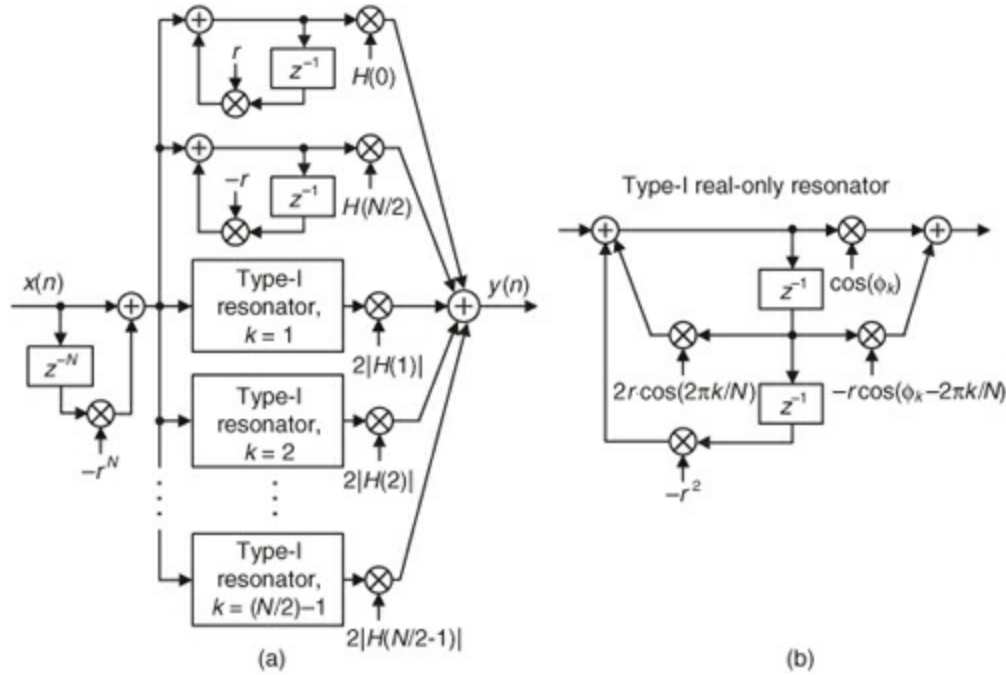
We can obtain real-FSF structures (real-valued coefficients) by forcing our complex N -section FSF, where N is even, to have conjugate poles, by ensuring that all nonzero $H(k)$ gain factors are accompanied by conjugate $H(N-k)$ gain factors, so that $H(N-k) = H^*(k)$. That is, we can build real-valued FSFs if we use conjugate pole pairs located at angles of $\pm 2\pi k/N$ radians. The transfer function of such an FSF (derived in [Appendix G, Section G.5](#)) is

(7-55)

$$H_{\text{gs,real}}(z) = (1 - r^N z^{-N}) \left[\frac{|H(0)|}{1 - rz^{-1}} + \frac{|H(N/2)|}{1 + rz^{-1}} \right] + \sum_{k=1}^{N/2-1} \frac{2 |H(k)| [\cos(\phi_k) - r \cos(\phi_k - 2\pi k/N) z^{-1}]}{1 - [2r \cos(2\pi k/N)] z^{-1} + r^2 z^{-2}},$$

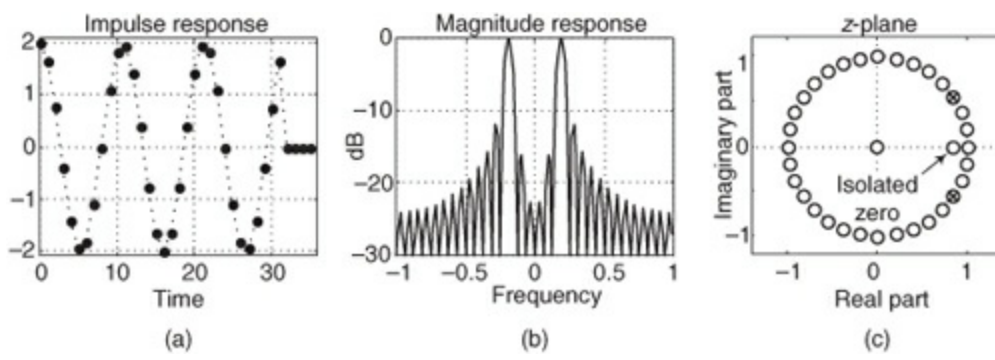
where the subscript “gs,real” means a guaranteed-stable real-valued multisection FSF, and ϕ_k is the desired phase angle of the k th section. [Eq. \(7-55\)](#) defines the structure of a *Type-I real FSF* to be as shown in [Figure 7-35\(a\)](#), requiring five multiplies per resonator output sample. The implementation of a real pole-pair resonator, using real-only arithmetic, is shown in [Figure 7-35\(b\)](#).

Figure 7-35 Guaranteed-stable, even- N , Type-I real FSF: (a) structure; (b) using real-only resonator coefficients.



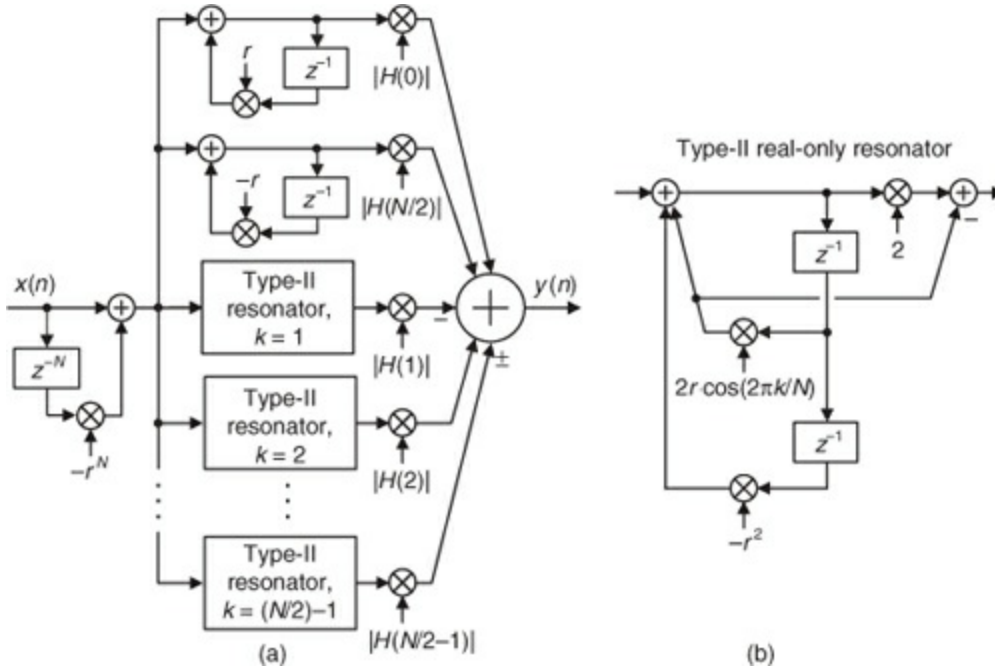
Of course, for lowpass FSFs the stage associated with the $H(N/2)$ gain factor in [Figure 7-35](#) would not be implemented, and for bandpass FSFs neither stage associated with the $H(0)$ and $H(N/2)$ gain factors would be implemented. The behavior of a single-section Type-I real FSF with $N = 32$, $k = 3$, $H(3) = 1$, $r = 0.99999$, and $\phi_3 = 0$ is provided in [Figure 7-36](#).

Figure 7-36 Time- and frequency-domain characteristics of a single-section Type-I FSF when $N = 32$, $k = 3$, $H(3) = 1$, $r = 0.99999$, and $\phi_3 = 0$.



An alternate version of the Type-I FSF, with a simplified resonator structure, can be developed by setting all ϕ_k values equal to zero and moving the gain factor of two inside the resonators. Next we incorporate the alternating signs in the final summation as shown in [Figure 7-37](#) to achieve linear phase, just as we did to arrive at the linear-phase multisection complex FSF in [Figure 7-32\(a\)](#), by adding the even- k and subtracting the odd- k resonator outputs. The “ \pm ” symbol in [Figure 7-37\(a\)](#) warns us that when N is even, $k = (N/2) - 1$ can be odd or even.

Figure 7-37 Linear-phase, even- N , Type-II real FSF: (a) structure; (b) real-only resonator implementation.



If the nonzero $|H(k)|$ gain factors are unity, this Type-II real FSF requires only three multiplies per section output sample. When a resonator's multiply by 2 can be performed by a hardware binary arithmetic left shift, only two multiplies are needed per output sample. The transfer function of this real-valued FSF is

(7-56)

$$H_{\text{Type-II}}(z) = (1 - r^N z^{-N}) \left[\frac{|H(0)|}{1 - rz^{-1}} - \frac{|H(N/2)|}{1 + rz^{-1}} + \sum_{k=1}^{N/2-1} \frac{(-1)^k |H(k)| [2 - 2r \cos(2\pi k/N) z^{-1}]}{1 - [2r \cos(2\pi k/N)] z^{-1} + r^2 z^{-2}} \right]$$

Neither the Type-I nor the Type-II FSF has exactly linear phase. While the phase nonlinearity is

relatively small, their passband group delays can have a peak-to-peak fluctuation of up to two sample periods ($2/f_s$) when used in multisection applications. This phase nonlinearity is unavoidable because those FSFs have isolated zeros located at $z = r\cos(2\pi k/N)$, when $\phi_k = 0$, as shown in [Figure 7-36\(c\)](#). Because the isolated zeros inside the unit circle have no accompanying reciprocal zeros located outside the unit circle at $z = 1/[r\cos(2\pi k/N)]$, sadly, this causes phase nonlinearity.

While the Type-I and -II FSFs are the most common types described in the literature of FSFs, their inability to yield exact linear phase has not received sufficient attention or analysis. In the next section we take steps to obtain linear phase by repositioning the isolated zero.

7.5.5 Linear-Phase Multisection Real-Valued FSFs

We can achieve exact real-FSF phase linearity by modifying the Type-I real FSF resonator's feedforward coefficients, in [Figure 7-35\(b\)](#), moving the isolated zero on top of the comb filter's zero located at $z = r$. We do this by setting $\phi_k = \pi k/N$. The numerator of the transfer function of one section of the real FSF, from [Eq. \(7-55\)](#), is

$$\cos(\phi_k) - r\cos(\phi_k - 2\pi k/N)z^{-1}.$$

If we set this expression equal to zero, and let $\phi_k = \pi k/N$, we find that the shifted isolated zero location z_0 is

$$\cos(\phi_k) - r\cos(\phi_k - 2\pi k/N)z_0^{-1} = \cos(\pi k/N) - r\cos(\pi k/N - 2\pi k/N)z_0^{-1} = 0$$

or

$$z_0 = \frac{r \cos(\pi k/N)}{\cos(\pi k/N)} = r.$$

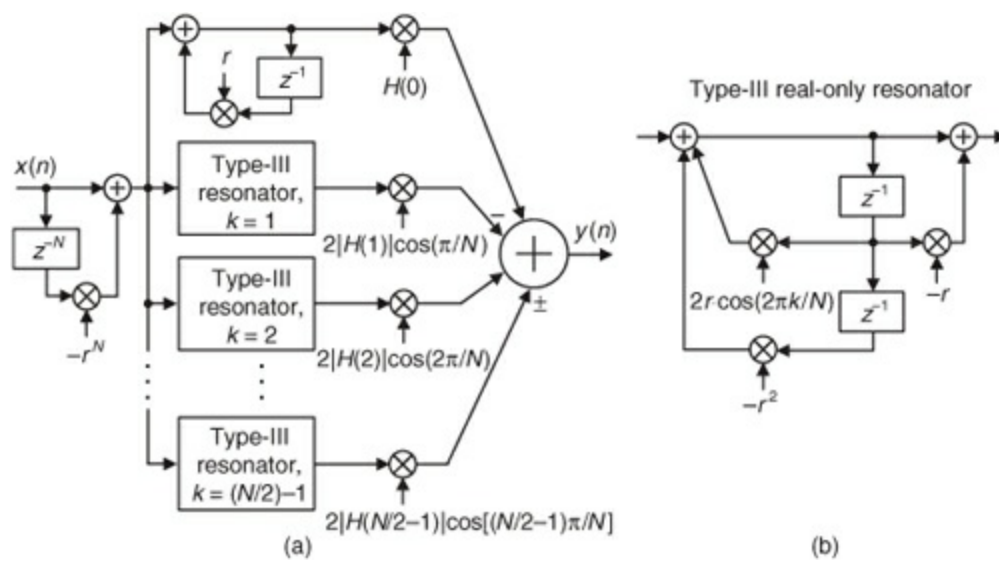
Substituting $\pi k/N$ for ϕ_k in [Eq. \(7-55\)](#) yields the transfer function of a linear-phase Type-III real FSF as

(7-57)

$$H_{\text{TypeIII}}(z) = (1 - r^N z^{-N}) \left[\frac{|H(0)|}{1 - rz^{-1}} + \sum_{k=1}^{N/2-1} \frac{2(-1)^k |H(k)| \cos(\pi k/N) [1 - rz^{-1}]}{1 - [2r \cos(2\pi k/N)]z^{-1} + r^2 z^{-2}} \right].$$

The implementation of the linear-phase Type-III real FSF is shown in [Figure 7-38](#), requiring four multiplies per section output sample.

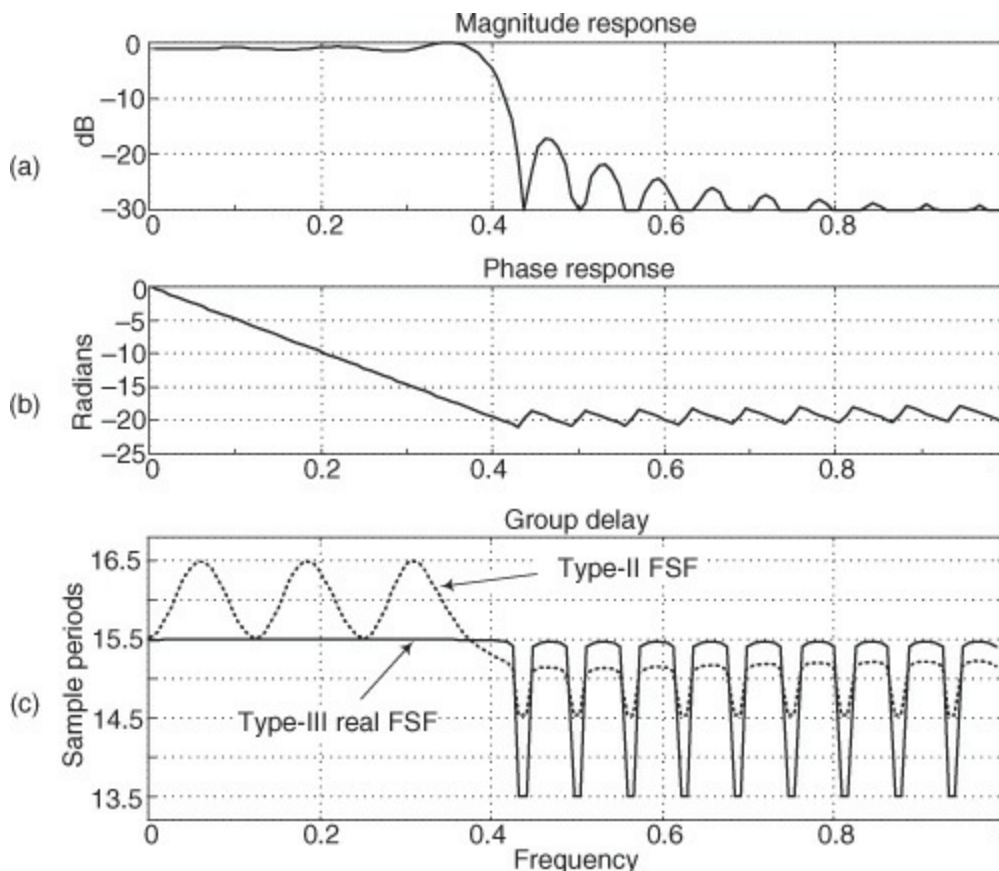
Figure 7-38 Even- N , linear-phase, Type-III real FSF: (a) structure; (b) real-only resonator implementation.



Notice that the $H(N/2), f_s/2$, section is absent in [Figure 7-38\(a\)](#). We justify this as follows: The even- N Type-I, -II, and -III real-FSF sections have impulse responses comprising N nonzero samples. As such, their $k = N/2$ sections' impulse responses, comprising even-length sequences of alternating plus and minus ones, are not symmetrical. This asymmetry property would corrupt the exact linear phase should a $k = N/2$ section be included. Consequently, as with even-length nonrecursive FIR filters, even- N Type-I, -II, and -III real FSFs cannot be used to implement linear-phase highpass filters.

[Figure 7-39](#) shows the frequency-domain performance of an eight-section Type-III FSF, for $N = 32$ where the eight sections begin at DC ($0 \leq k \leq 7$). [Figure 7-39\(c\)](#) provides a group delay comparison between the Type-III FSF and an equivalent eight-section Type-II FSF showing the improved Type-III phase linearity having a constant group delay of $(N-1)/2$ samples over the passband.

Figure 7-39 Interpolated frequency-domain response of a Type-III FSF having eight sections with $N = 32$: (a) magnitude response; (b) phase response; (c) group delay compared with an equivalent Type-II FSF.



7.5.6 Where We've Been and Where We're Going with FSFs

We've reviewed the structure and behavior of a complex FSF whose complex resonator stages had poles residing atop the zeros of a comb filter, resulting in a recursive FIR filter. Next, to ensure filter implementation stability, we forced the pole/zero locations to be just inside the unit circle. We examined a guaranteed-stable even- N Type-I real FSF having resonators with conjugate pole pairs, resulting in an FSF with real-valued coefficients. Next, we modified the Type-I real-FSF structure, yielding the more computationally efficient but only moderately linear-phase Type-II real FSF. Finally, we modified the coefficients of the Type-I real FSF and added post-resonator gain factors, resulting in the exact linear-phase Type-III real FSF. During this development, we realized that the even- N Type-I, -II, and -III real FSFs cannot be used to implement linear-phase highpass filters.

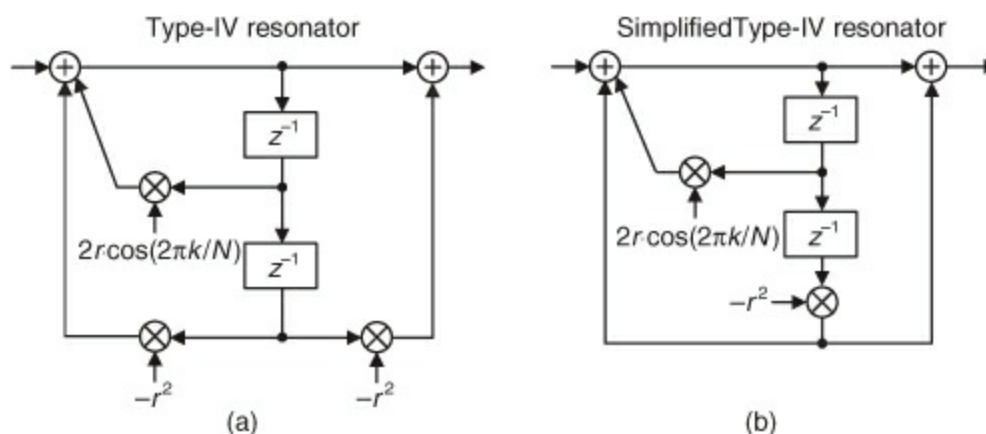
In the remainder of this section we introduce a proposed resonator structure that provides superior filtering properties compared to the Type-I, -II, and -III resonators. Next we'll examine the use of nonzero transition band filter sections to improve overall FSF passband ripple and stopband attenuation, followed by a discussion of several issues regarding modeling and designing FSFs. We'll compare the performance of real FSFs to their equivalent Parks-McClellan-designed N -tap nonrecursive FIR filters. Finally, a detailed FSF design procedure is presented.

7.5.7 An Efficient Real-Valued FSF

There are many real-valued resonators that can be used in FSFs, but of particular interest to us is the Type-IV resonator presented in [Figure 7-40\(a\)](#). This resonator deserves attention because it

- is guaranteed stable,
- exhibits highly linear phase,
- uses real-valued coefficients,
- is computationally efficient,
- can implement highpass FIR filters, and
- yields stopband attenuation performance superior to the Type-I, -II, and -III FSFs.

Figure 7-40 Type-IV resonator: (a) original structure; (b) simplified version.



From here on, the Type IV will be our FSF of choice.

Cascading Type-IV resonators with a comb filter provide a Type-IV real-only FSF with a transfer function of

$$H_{\text{Type-IV}}(z) = (1 - r^N z^{-N}) \sum_{k=0}^{N/2} \frac{(-1)^k |H(k)| (1 - r^2 z^{-2})}{1 - 2r \cos(2\pi k/N) z^{-1} + r^2 z^{-2}}$$

where N is even. (Note: When N is odd, $k = N/2$ is not an integer and the $|H(N/2)|$ term does not exist.) As derived in [Appendix G, Section G.6](#), the Type-IV FSF frequency response is

(7-59)

$$H_{\text{Type-IV}}(e^{j\omega}) = e^{-j\omega N/2} \sum_{k=0}^{N/2} \frac{(-1)^k |H(k)| [\cos(\omega N/2 - \omega) - \cos(\omega N/2 + \omega)]}{\cos(2\pi k/N) - \cos(\omega)}.$$

The even- N frequency and resonance magnitude responses for a single Type-IV FSF section are

(7-60)

$$H_{\text{Type-IV}}(e^{j\omega}) = e^{-j\omega N/2} \frac{\cos(\omega N/2 - \omega) - \cos(\omega N/2 + \omega)}{\cos(2\pi k/N) - \cos(\omega)}$$

and

$$(7-61)$$

$$|H_{\text{Type-IV}}(e^{j\omega})|_{\omega=2\pi k/N} = N, \quad k = 1, 2, \dots, \frac{N}{2} - 1,$$

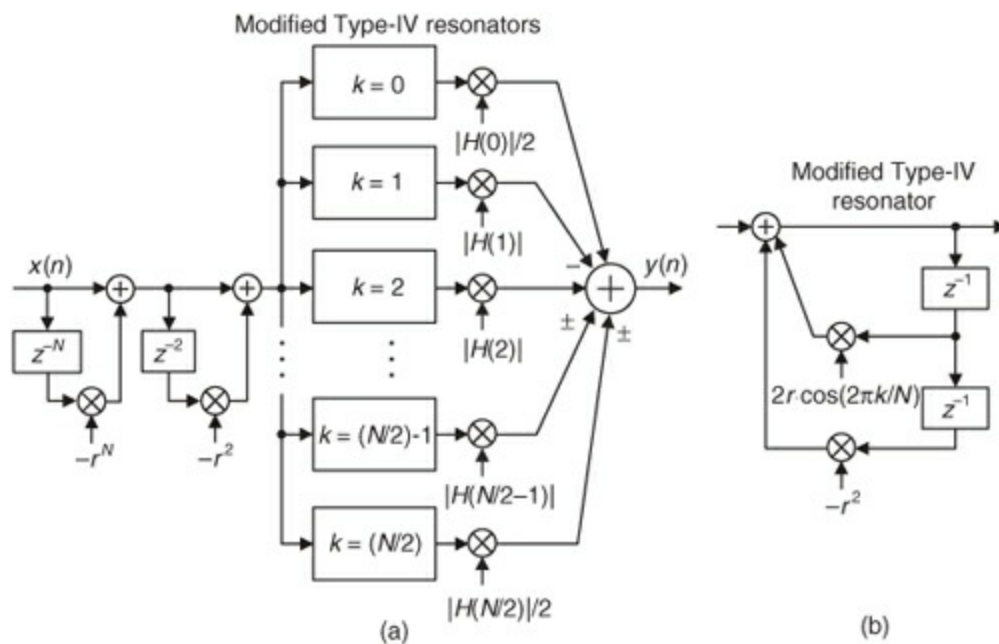
and its resonant magnitude gains at $k = 0$ (DC), and $k = N/2$ ($f_s/2$), are

$$(7-62)$$

$$|H_{\text{Type-IV}}(e^{j\omega})|_{\omega=0} = |H_{\text{Type-IV}}(e^{j\omega})|_{\omega=\pi} = 2N.$$

To reduce the number of resonator multiply operations, it is tempting to combine the dual $-r^2$ factors in [Figure 7-40\(a\)](#) to simplify the Type-IV resonator as shown in [Figure 7-40\(b\)](#). However, a modification reducing both the number of multiply and addition operations in each FSF resonator is to implement the $(1 - r^2 z^{-2})$ term in the numerator of [Eq. \(7-58\)](#) as a second-order comb filter as shown in [Figure 7-41\(a\)\[27\]](#). This transforms the Type-IV resonator to that shown in [Figure 7-41\(b\)](#). The $|H(0)|/2$ and $|H(N/2)|/2$ gain factors compensate for the gain of $2N$ for a Type-IV resonator in [Eq. \(7-62\)](#).

Figure 7-41 Type-IV, even- N , real FSF: (a) structure; (b) its modified resonator.



The cascaded-comb subfilter combination has N zeros spaced around the unit circle with dual z -plane zeros at $z = 1$ (0 Hz) and dual zeros at $z = -1$ ($f_s/2$ Hz). However, the $k = 0$ and $k = N/2$ Type-IV resonators have dual poles at $z = \pm 1$, respectively, enabling the necessary pole/zero cancellation.

The “ \pm ” symbols in [Figure 7-41\(a\)](#) remind us, again, that when N is even, $k = N/2$ could be odd or even. This FSF has the very agreeable characteristic of having an impulse response whose length is $N+1$ samples. Thus, unlike the Type-I, -II, and -III FSFs, an even- N Type-IV FSF’s $k = N/2$ section impulse response (alternating ± 1 s) is symmetrical, and Type-IV FSFs can be used to build linear-phase highpass filters. When N is odd, the $k = N/2$ section is absent in [Figure 7-41](#), and the odd- N Type-IV transfer function is identical to [Eq. \(7-58\)](#) with the summation’s upper limit being $(N-1)/2$ instead of $N/2$.

We’ve covered a lot of ground thus far, so [Table 7-3](#) summarizes what we’ve discussed concerning real-valued FSFs. The table lists the average passband group delay measured in sample periods, the number of multiplies and additions necessary per output sample for a single FSF section, and a few remarks regarding the behavior of the various real FSFs. The section gains at their resonant frequencies, assuming the desired $|H(k)|$ gain factors are unity, is N for all four of the real-valued FSFs.

Table 7-3 Summary of Even- N Real FSF Properties

Real FSF type	Group delay	Multiplies	Adds	Remarks
Type I (Figure 7-35)	$\frac{N}{2}$	5	3	Real-coefficient FSF. Phase is only moderately linear.
Type II (Figure 7-37)	$\frac{N}{2}$	3	3	Modified, and more efficient, version of the Type-I FSF. Phase is only moderately linear. A binary left shift may eliminate one resonator multiply.
Type III (Figure 7-38)	$\frac{N-1}{2}$	4	3	Very linear phase. Cannot implement a linear-phase highpass filter.
Type IV (Figure 7-41)	$\frac{N}{2}$	2	2	Very linear phase. Improved stopband attenuation. Usable for lowpass, bandpass, or highpass filtering.

7.5.8 Modeling FSFs

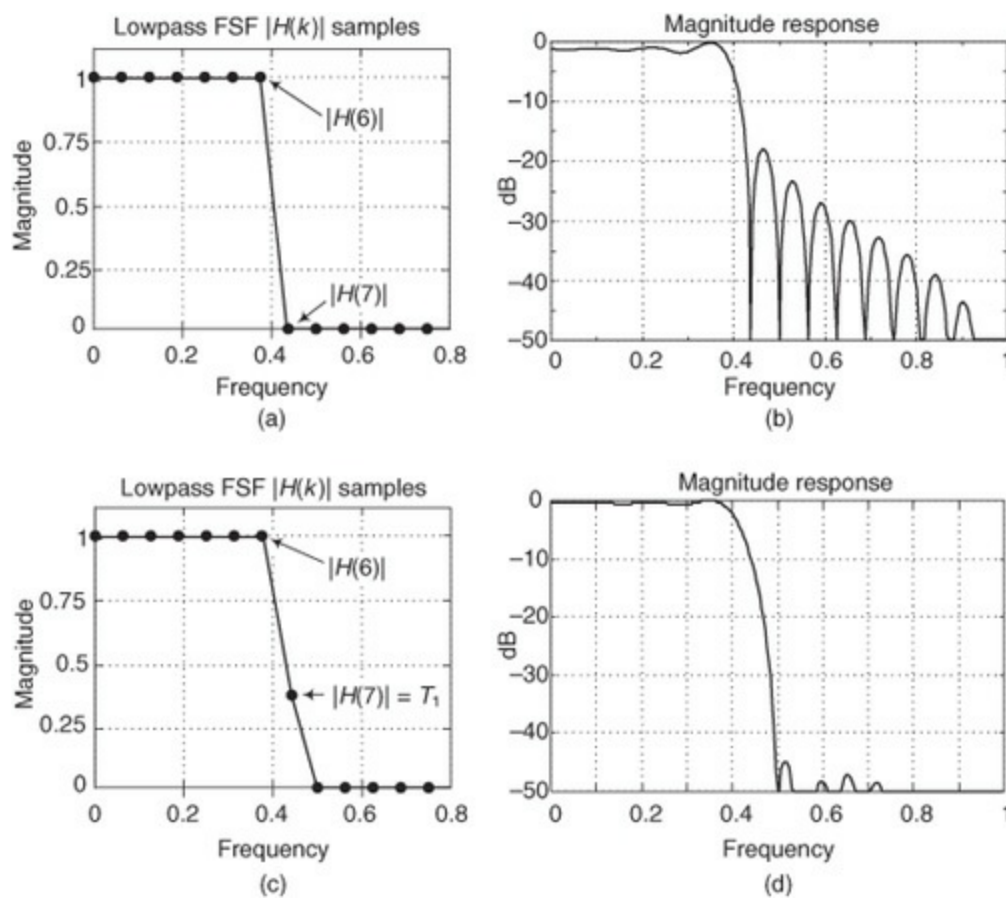
We derived several different $H(e^{j\omega})$ frequency response equations here, not so much to use them for modeling FSF performance but to examine them in order to help define the structure of our FSF block diagrams. For example, it was analyzing the properties of $H_{\text{Type-IV}}(e^{j\omega})$ that motivated the use of the $1/2$ gain factors in the Type-IV FSF structure.

When modeling the FSFs, fortunately it's not necessary to write code to calculate frequency-domain responses using the various $H(e^{j\omega})$ equations provided here. All that's necessary is code to compute the time-domain impulse response of the FSF being modeled, pad the impulse response with enough zeros so the padded sequence is 10 to 20 times the length of the original, perform a discrete Fourier transform (DFT) on the padded sequence, and plot the resulting interpolated frequency-domain magnitude and phase. Of course, forcing your padded sequence length to be an integer power of two enables the use of the efficient FFT to calculate frequency-domain responses. Alternatively, many commercial signal processing software packages have built-in functions to calculate filter frequency response curves based solely on the filter's transfer function coefficients.

7.5.9 Improving Performance with Transition Band Coefficients

We can increase FSF stopband attenuation if we carefully define $|H(k)|$ magnitude samples in the transition region, between the passband and stopband. For example, consider a lowpass Type-IV FSF having seven sections, of unity gain, with $N = 32$, whose desired performance is shown in [Figure 7-42\(a\)](#), and whose interpolated (actual) frequency magnitude response is provided in [Figure 7-42\(b\)](#).

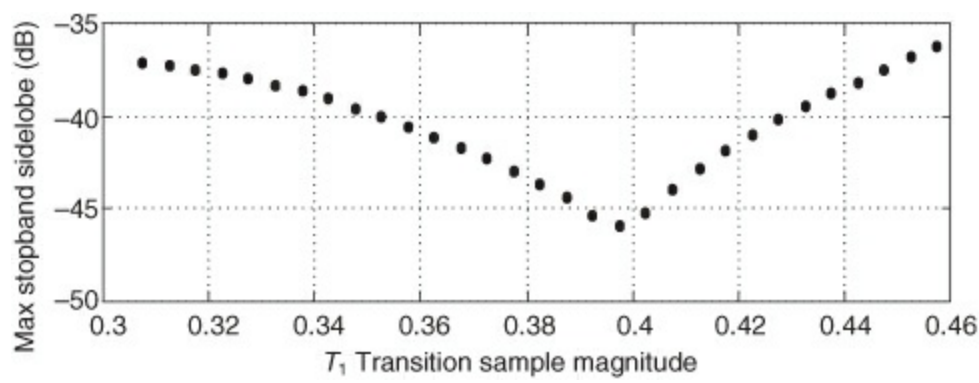
Figure 7-42 Seven-section, $N = 32$, Type-IV FSF: (a) desired frequency response; (b) actual performance; (c) using one transition sample; (d) improved stopband attenuation.



The assignment of a transition magnitude sample, coefficient T_1 whose value is between 0 and 1 as in [Figure 7-42\(c\)](#), reduces the abruptness in the transition region between the passband and the stopband of our desired magnitude response. Setting $T_1 = 0.389$ results in reduced passband ripple and the improved stopband sidelobe suppression shown in [Figure 7-42\(d\)](#). The price we pay for the improved performance is the computational cost of an additional FSF section and increased width of the transition region.

Assigning a coefficient value of $T_1 = 0.389$ was not arbitrary or magic. Measuring the maximum stopband sidelobe level for various values of $0 \leq T_1 \leq 1$ reveals the existence of an optimum value for T_1 . [Figure 7-43](#) shows that the maximum stopband sidelobe level is minimized when $T_1 = 0.389$. The minimum sidelobe level of -46 dB (when $T_1 = 0.389$) corresponds to the height of the maximum stopband sidelobe in [Figure 7-42\(d\)](#). This venerable and well-known method of employing a transition region coefficient to reduce passband ripple and minimize stopband sidelobe levels also applies to bandpass FSF design where the transition sample is used just before and just after the filter's passband unity-magnitude $|H(k)|$ samples.

Figure 7-43 Maximum stopband sidelobe level as a function of the transition region coefficient value for a seven-section Type-IV real FSF when $N = 32$.



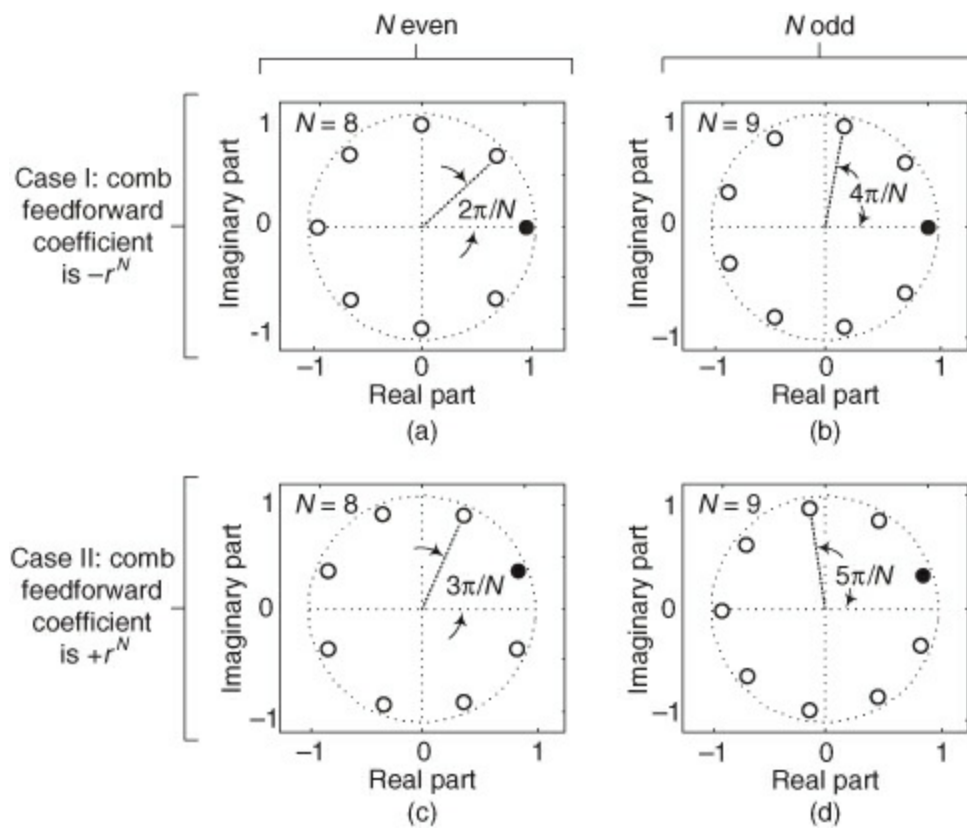
Further stopband sidelobe level suppression is possible if two transition coefficients, T_1 and T_2 , are used such that $0 \leq T_2 \leq T_1 \leq 1$. (Note: For lowpass filters, if T_1 is the $|H(k)|$ sample, then T_2 is the $|H(k+1)|$ sample.) Each additional transition coefficient used improves the stopband attenuation by approximately 25 dB. However, finding the optimum values for the T coefficients is a daunting task. Optimum transition region coefficient values depend on the number of unity-gain FSF sections, the value of N , and the number of coefficients used; and unfortunately there's no closed-form equation available to calculate optimum transition coefficient values. We must search for them empirically.

If one coefficient (T_1) is used, finding its optimum value can be considered a one-dimensional search. If two coefficients are used, the search becomes two-dimensional, and so on. Fortunately, descriptions of linear algebra search techniques are available[23,28–30], and commercial mathematical software packages have built-in *optimization* functions to facilitate this computationally intensive search. For Type-I/II FSFs, tables of optimum transition region coefficients for various N , and number of FSF sections used, have been published[23], and a subset of those tables is provided as an appendix in a textbook[24]. For the higher-performance Type-IV FSFs, tables of optimum coefficient values have been compiled by the author and are provided in [Appendix H](#). With this good news in mind, shortly we'll look at a real-world FSF design example to appreciate the benefits of FSFs.

7.5.10 Alternate FSF Structures

With the primary comb filter in [Figure 7-41](#) having a feedforward coefficient of $-r^N$, its even- N transfer function zeros are equally spaced around and inside the unit circle, as shown in [Figures 7-23\(c\)](#) and [7-44\(a\)](#), so the $k = 1$ zero is at an angle of $2\pi/N$ radians. In this Case I, if N is odd, the comb's zeros are spaced as shown in [Figure 7-44\(b\)](#). An alternate situation, Case II, exists when the comb filter's feedforward coefficient is $+r^N$. In this mode, an even- N comb filter's zeros are rotated counterclockwise on the unit circle by π/N radians as shown in [Figure 7-44\(c\)](#), where the comb's $k = 1$ zero is at an angle of $3\pi/N$ radians[27]. The $k = 0$ zeros are shown as solid dots.

Figure 7-44 Four possible orientations of comb filter zeros near the unit circle.



The structure of a Case II FSF is identical to that of a Case I FSF; however, the Case II resonators' coefficients must be altered to rotate the poles by π/N radians to ensure pole/zero cancellation. As such, the transfer function of a Case II, even- N , Type-IV FSF is

(7-63)

$$H_{\text{Case II, Type-IV}}(z) = (1 + r^N z^{-N}) \sum_{k=0}^{N/2} \frac{(-1)^k |H(k)| (1 - r^2 z^{-2})}{1 - 2r \cos[2\pi(k + 1/2)/N] z^{-1} + r^2 z^{-2}}.$$

Because a Case II FSF cannot have a pole or a zero at $z = 1$ (DC), these FSFs are unable to build lowpass filters, in which case the z^{-2} -delay comb filter in [Figure 7-41\(a\)](#) is not implemented. [Table 7-4](#) lists the filtering capabilities of the various Case I and Case II Type-IV FSFs.

Table 7-4 Type-IV FSF Modes of Operation

Type-IV FSF	Lowpass	Bandpass	Highpass
Case I, N even	Yes	Yes	Yes
Case I, N odd	Yes	Yes	No
Case II, N even	No	Yes	No
Case II, N odd	No	Yes	Yes

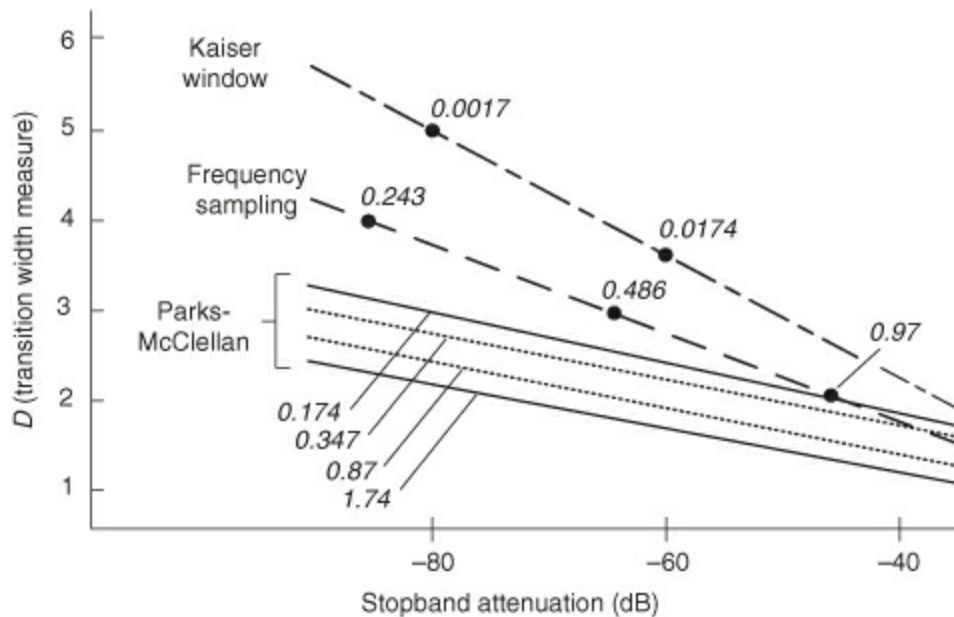
Their inability to implement lowpass filtering limits the utility of Case II FSFs, but their resonators' shifted center frequencies do provide some additional flexibility in defining the cutoff frequencies of bandpass and highpass filters. The remainder of this material considers only the Case I Type-IV FSFs.

7.5.11 The Merits of FSFs

During the formative years of FIR filter theory (early 1970s) a computer program was made available for designing nonrecursive FIR filters using the Parks-McClellan (PM) method[\[31\]](#). This filter design technique provided complete control in defining desired passband ripple, stopband attenuation, and transition region width. (FIR filters designed using the PM method are often called *optimal FIR*, *Remez Exchange*, or *equiripple* filters.) Many of the early descriptions of the PM design method

included a graphical comparison of various lowpass FIR filter design methods similar to that redrawn in [Figure 7-45](#)[29,32,33]. In this figure, a given filter's normalized (to impulse response length N , and the sample rate f_s) transition bandwidth is plotted as a function of its minimum stopband attenuation. (Passband peak-to-peak ripple values, measured in dB, are provided numerically within the figure.)

Figure 7-45 Comparison of the Kaiser window, frequency sampling, and Parks-McClellan-designed FIR filters.



Because the normalized transition bandwidth measure D in [Figure 7-45](#) is roughly independent of N , this diagram is considered to be a valid performance comparison of the three FIR filter design methods. For a given passband ripple and stopband attenuation, the PM-designed FIR filter could exhibit the narrowest transition bandwidth, thus the highest performance filter. The wide dissemination of [Figure 7-45](#) justifiably biased filter designers toward the PM design method. (During his discussion of [Figure 7-45](#), one author boldly declared, “The smaller is D , the better is the filter”[32].) Given its flexibility, improved performance, and ease of use, the PM method quickly became the predominant FIR filter design technique. Consequently, in the 1970s FSF implementations lost favor to the point where they’re rarely mentioned in today’s DSP classrooms or textbooks.

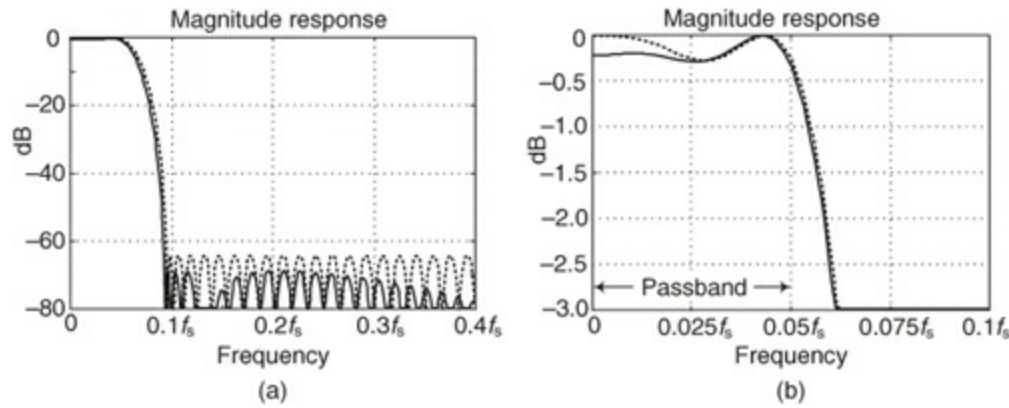
However, from a computational workload standpoint, [Figure 7-45](#) is like modern swimwear: what it shows is important; what it does not show is vital. That design comparison does not account for those situations where both frequency sampling and PM filters meet the filter performance requirements but the frequency sampling implementation is more computationally efficient. The following FIR filter design example illustrates this issue and shows why FSFs should be included in our bag of FIR filter design tools.

7.5.12 Type-IV FSF Example

Consider the design of a linear-phase lowpass FIR filter whose cutoff frequency is 0.05 times the sample rate f_s , the stopband must begin at 0.095 times f_s , the maximum passband ripple is 0.3 dB peak to peak, and the minimum stopband attenuation must be 65 dB. If we designed a six-section Type-IV FSF with $N = 62$ and $r = 0.99999$, its frequency-domain performance satisfies our requirements and is the solid curve in [Figure 7-46\(a\)](#).

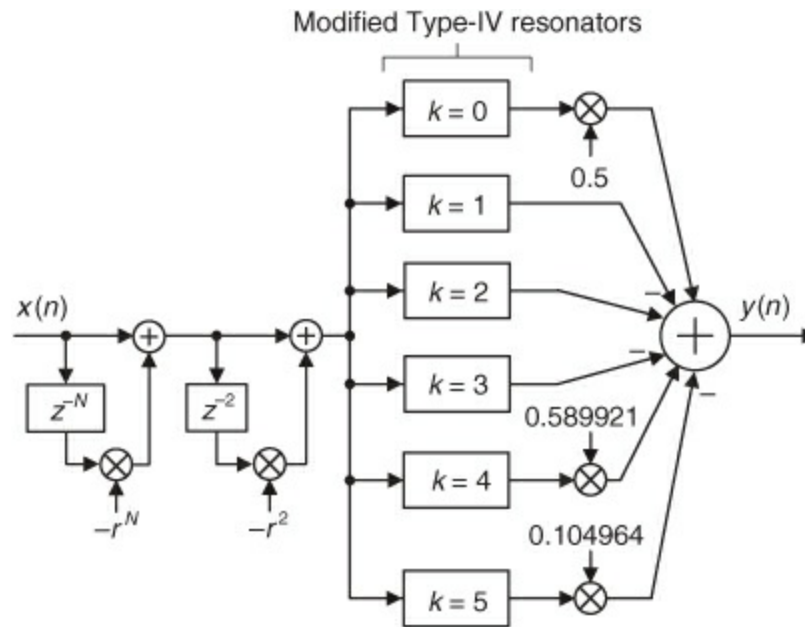
Figure 7-46 An $N = 62$, six-section Type-IV real FSF (solid) versus a 60-tap PM-designed filter

(dashed): (a) frequency magnitude response; (b) passband response detail.



For this FSF, two transition region coefficients of $|H(4)| = T_1 = 0.589921$ and $|H(5)| = T_2 = 0.104964$ were used. Included in Figure 7-46(a), the dashed curve, is the magnitude response of a PM-designed 60-tap nonrecursive FIR filter. Both filters meet the performance requirements and have linear phase. The structure of the Type-IV FSF is provided in Figure 7-47. A PM-designed filter implemented using a *folded* nonrecursive FIR structure (as discussed in Section 13.7), exploiting its impulse response symmetry to halve the number of multipliers, requires 30 multiplies and 59 adds per output sample. We see the computational savings of the Type-IV FSF, which requires only 17 multiplies and 19 adds per output sample. (Note: The FSF's $H(k)$ gain factors are all zero for $6 \leq k \leq 31$.)

Figure 7-47 Structure of the $N = 62$, six-section, lowpass, Type-IV FSF with two transition region coefficients.



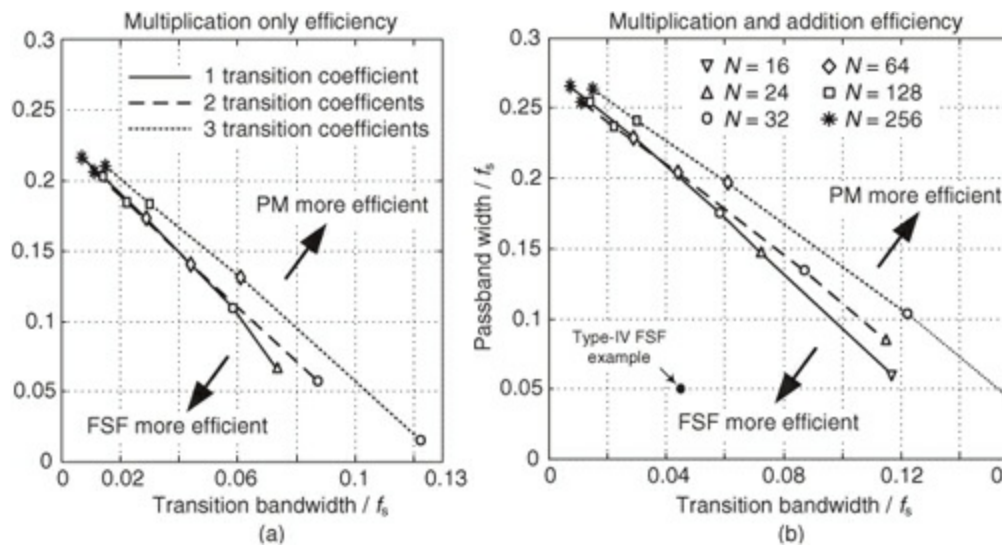
7.5.13 When to Use an FSF

In this section we answer the burning question “When is it smarter (more computationally efficient) to use a Type-IV FSF instead of a PM-designed FIR filter?” The computational workload of a PM-designed FIR filter is directly related to the length of its time-domain impulse response, which in turn is inversely proportional to the filter transition bandwidth. The narrower the transition bandwidth, the greater the nonrecursive FIR filter’s computational workload measured in arithmetic operations per output sample. On the other hand, the computational workload of an FSF is roughly proportional to its passband width. The more FSF sections needed for wider bandwidths and the more transition region coefficients used, the higher the FSF’s computational workload. So in comparing the computational

workload of FSFs to PM-designed FIR filters, both transition bandwidth and passband width must be considered.

[Figure 7-48](#) provides a computational comparison between even- N Type-IV FSFs and PM-designed nonrecursive lowpass FIR filters. The curves represent desired filter performance parameters where a Type-IV FSF and a PM-designed filter have approximately equal computational workloads per output sample. (The bandwidth values in the figure axis are normalized to the sample rate, so, for example, a frequency value of 0.1 equals $f_s/10$.) If the desired FIR filter transition region width and passband width combination (a point) lies beneath the curves in [Figure 7-48](#), then a Type-IV FSF is computationally more efficient than a PM-designed filter. [Figure 7-48\(a\)](#) is a comparison accounting only for multiply operations. For filter implementations where multiplies and adds require equal processor clock cycles, [Figure 7-48\(b\)](#) provides the appropriate comparison. The solitary black dot in the figure indicates the comparison curves location for the above [Figure 7-46](#) Type-IV FSF filter example. (A Type-IV FSF design example using the curves in [Figure 7-48](#) will be presented later.)

Figure 7-48 Computational workload comparison between even- N lowpass Type-IV FSFs and nonrecursive PM FIR filters: (a) only multiplications considered; (b) multiplications and additions considered. (No $1/N$ gain factor used.)



The assumptions made in creating [Figure 7-48](#) are an even- N Type-IV FSF, damping factor $r = 0.99999$, and no $1/N$ gain scaling from [Figure 7-20\(b\)](#). The PM filter implementation used in the comparison assumes a symmetrical impulse response, an M -tap folded structure requiring $M/2$ multiplies, and $M-1$ additions. The performance criteria levied on the PM filter are typical Type-IV FSF properties (when floating-point coefficients are used) given in [Table 7-5](#).

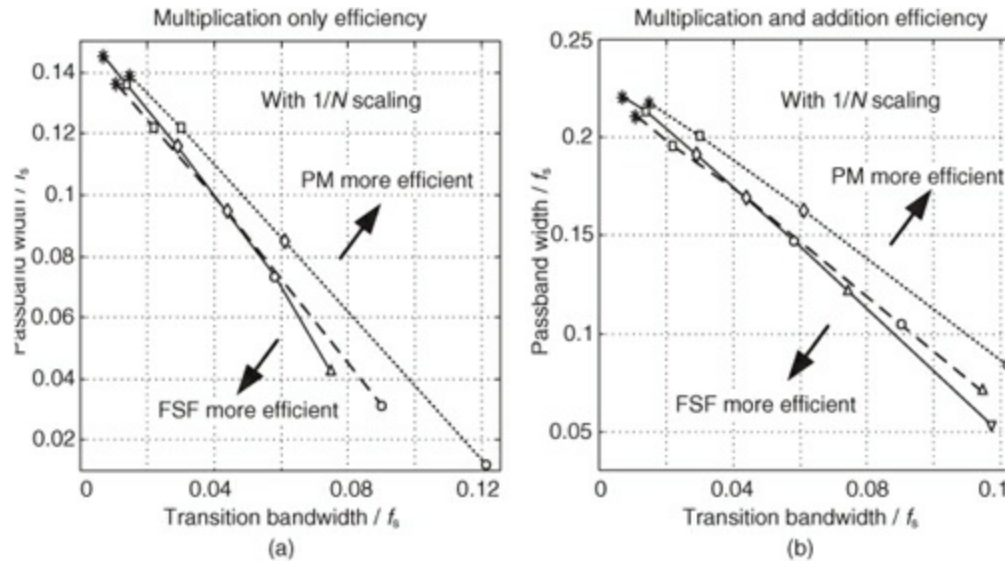
Table 7-5 Typical Even- N Type-IV FSF Properties

Parameter	1-coefficient	2-coefficient	3-coefficient
Passband peak-peak ripple (dB)	0.7	0.35	0.16
Minimum stopband attenuation (dB)	-45	-68	-95

The Type-IV resonators have a gain of N , so the subsequent gain of the FSF in [Figure 7-47](#) is N . For FSF implementations using floating-point numbers, this gain may not be a major design issue. In filters implemented with fixed-point numbers, the gain of N could cause overflow errors, and a gain reduction by a scaling factor of $1/N$ may be necessary. Occasionally, in the literature, a $1/N$ scaling

factor is shown as a single multiply just prior to an FSF's comb filter[24,34]. This scale factor implementation is likely to cause an intolerable increase in the input signal quantization noise. A more practical approach is to apply the $1/N$ scaling at the output of each FSF section, prior to the final summation, as indicated in [Figure 7-20\(b\)](#). Thus every FSF resonator output is multiplied by a non-unity value, increasing the computational workload of the FSF. In this situation, the computational comparison between even- N Type-IV FSFs and PM filters becomes as shown in [Figure 7-49](#).

Figure 7-49 Computational workload comparison between even- N lowpass Type-IV FSFs, with resonator output $1/N$ scaling included, and nonrecursive PM FIR filters: (a) only multiplications considered; (b) multiplications and additions considered.



Of course, if N is an integer power of two, some hardware implementations may enable resonator output $1/N$ scaling with hardwired binary arithmetic right shifts to avoid explicit multiply operations. In this situation, the computational workload comparison in [Figure 7-48](#) applies.

As of this writing, programmable DSP chips typically cannot take advantage of the folded FIR filter structure assumed in the creation of [Figures 7-29](#) and [7-30](#). In this case, an M -tap direct convolution FIR filter has the disadvantage that it must perform the full M multiplies per output sample. However, DSP chips have the advantage of *zero-overhead looping* and single-clock-cycle *multiply and accumulate* (MAC) capabilities, making them more efficient for direct convolution FIR filtering than for filtering using the recursive FSF structures. Thus, a DSP chip implementation's disadvantage of more necessary computations and its advantage of faster execution of those computations roughly cancel each other. With those thoughts in mind, we're safe to use [Figures 7-48](#) and [7-49](#) as guides in deciding whether to use a Type-IV FSF or a PM-designed FIR filter in a programmable DSP chip filtering application.

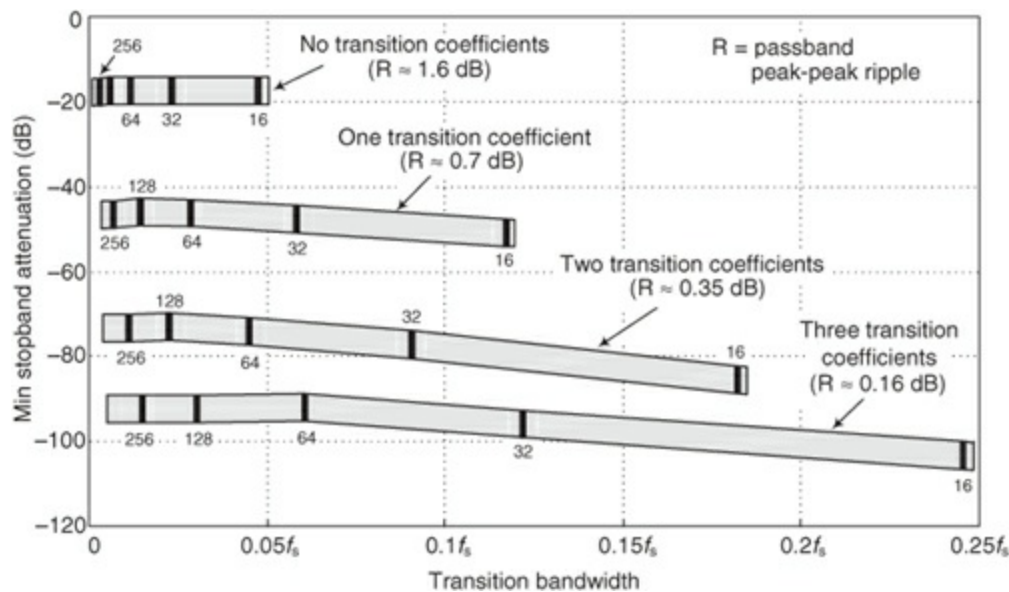
Finally, we conclude, from the last two figures, that Type-IV FSFs are more computationally efficient than PM-designed FIR filters in lowpass applications where the passband is less than approximately $f_s/5$ and the transition bandwidth is less than roughly $f_s/8$.

7.5.14 Designing FSFs

The design of a practical Type-IV FSF comprises three stages: (1) determine if an FSF can meet the desired filter performance requirements, (2) determine if a candidate FSF is more, or less, computationally efficient than an equivalent PM-designed filter, and (3) design the FSF and verify its performance. To aid in the first stage of the design, [Figure 7-50](#) shows the typical minimum stopband

attenuation of Type-IV FSFs, as a function of transition bandwidth, for various numbers of transition region coefficients. (Various values for N , and typical passband ripple performance, are given within [Figure 7-50](#) as general reference parameters.)

Figure 7-50 Typical Type-IV lowpass FSF stopband attenuation performance as a function of transition bandwidth.



In designing an FSF, we find the value N to be a function of the desired filter transition bandwidth, and the number of FSF sections required is determined by both N and the desired filter bandwidth. Specifically, given a linear-phase FIR filter's desired passband width, passband ripple, transition bandwidth, and minimum stopband attenuation, the design of a linear-phase lowpass Type-IV FSF proceeds as follows:

1. Using [Figure 7-50](#), determine which shaded performance band in the figure satisfies the desired minimum stopband attenuation. This step defines the number of transition coefficients necessary to meet stopband attenuation requirements.
2. Ensure that your desired transition bandwidth value resides within the chosen shaded band. (If the transition bandwidth value lies to the right of a shaded band, a PM-designed filter will be more computationally efficient than a Type-IV FSF and the FSF design proceeds no further.)
3. Determine if the typical passband ripple performance, provided in [Figure 7-50](#), for the candidate shaded band is acceptable. If so, a Type-IV FSF can meet the desired lowpass filter performance requirements. If not, a PM design should be investigated.
4. Based on arithmetic implementation priorities, determine the appropriate computational workload comparison chart to be used from [Figure 7-48](#) or [Figure 7-49](#) in determining if an FSF is more efficient than a PM-designed nonrecursive filter.
5. Using the desired filter's transition bandwidth and passband width as coordinates of a point on the selected computational workload comparison chart, determine if that point lies beneath the appropriate curve. If so, an FSF can meet the filter performance requirements with fewer computations per filter output sample than a PM-designed filter. (If the transition bandwidth/passband point lies above the appropriate curve, a PM design should be investigated and the FSF design proceeds no further.)
6. Assuming the designer has reached this step of the evaluation process, the design of a Type-IV

FSF proceeds by selecting the filter order N . The value of N depends on the desired transition bandwidth and the number of transition coefficients from Step 1 and, when the transition bandwidth is normalized to f_s , can be estimated using

(7-64)

$$N \approx \frac{f_s(\text{number of transition coefficients} + 1)}{\text{transition bandwidth}}.$$

7. The required number of unity-gain FSF sections, integer M , is roughly proportional to the desired normalized double-sided passband width divided by the FSF's frequency resolution (f_s/N) and is estimated using

(7-65)

$$M \approx \frac{2N(\text{passband width})}{f_s}.$$

8. Given the initial integer values for N and M , find the appropriate optimized transition coefficient gain values from the compiled tables in [Appendix H](#). If the tables do not contain optimized coefficients for the given N and M values, the designer may calculate approximate coefficients by linear interpolation of the tabulated values. Alternatively, an optimization software program may be used to find optimized transition coefficients.
9. Using the values for N , M , and the optimized transition coefficients, determine the interpolated (actual) frequency response of the filter as a function of those filter parameters. This frequency response can be obtained by evaluating [Eq. \(7-59\)](#). More conveniently, the frequency response can be determined with a commercial signal processing software package to perform the DFT of the FSF's impulse response sequence, or by using the transfer function coefficients in [Eq. \(7-58\)](#).
10. Next the fun begins as the values for N and M are modified slightly, and Steps 8 and 9 are repeated, as the design process converges to a minimum value of M to minimize the computational workload, and optimized transition coefficients maximizing stopband attenuation.
11. When optimized filter parameters are obtained, they are then used in a Type-IV FSF implementation, as in [Figure 7-47](#).
12. The final FSF design step is to sit back and enjoy a job well done.

The Type-IV FSF example presented in [Figures 7-46](#) and [7-47](#) provides an illustration of Steps 6 and

7. The initial design estimates for N and M are

$$N_{\text{init.}} \approx \frac{f_s(2+1)}{(0.095-0.05)f_s} \approx 66, \text{ and } M_{\text{init.}} \approx \frac{(2)(62)(0.05f_s)}{f_s} \approx 6.2.$$

Repeated iterations of Steps 8 through 11 converge to the parameters of $N = 62$ and $M = 6$ that satisfy the desired filter performance specifications.

7.5.15 FSF Summary

We've introduced the structure, performance, and design of frequency sampling FIR filters. Special emphasis was given to the practical issues of phase linearity, stability, and computational workload of these filters. In addition, we presented a detailed comparison of the high-performance Type-IV FSF and its nonrecursive FIR equivalent. Performance curves were presented to aid the designer in choosing between a Type-IV FSF and a Parks-McClellan-designed FIR filter for a given narrowband

linear-phase filtering application. We found that

- Type-IV FSFs are more computationally efficient, for certain stopband attenuation levels, than Parks-McClellan-designed nonrecursive FIR filters in lowpass (or highpass) applications where the passband is less than $f_s/5$ and the transition bandwidth is less than $f_s/8$ (see [Figures 7-29](#) and [7-30](#));
- FSFs are modular; their components (sections) are computationally identical and well understood;
- tables of optimum transition region coefficients, used to improve Type-IV FSF performance, can be generated (as was done in [Appendix H](#)); and
- although FSFs use recursive structures, they can be designed to be guaranteed stable and have linear phase.

References

- [1] Hamming, R. *Digital Filters*, 3rd ed., Dover, Mineola, New York, 1998, pp. 133–137.
- [2] Selby, S. *CRC Standard Mathematical Tables*, 21st ed., CRC Press, Boca Raton, Florida, 1973, p. 453.
- [3] Proakis, J., and Manolakis, D. *Digital Signal Processing: Principles, Algorithms, and Applications*, 3rd ed., Prentice Hall, Upper Saddle River, New Jersey, 1996, pp. 656–657.
- [4] Rabiner, L., and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice Hall, Upper Saddle River, New Jersey, 1975, p. 169.
- [5] Hamming, R. *Numerical Methods for Scientists and Engineers*, 2nd ed., Dover, Mineola, New York, 1986, pp. 590–591.
- [6] Sklar, B. *Digital Communications: Fundamentals and Applications*, 2nd ed., Prentice Hall, Englewood Cliffs, New Jersey, 1988, pp. 122–127.
- [7] Engelberg, S. *Random Signals and Noise: A Mathematical Introduction*, CRC Press, Boca Raton, Florida, 2007, [Chapter 6](#).
- [8] Shanmugam, K. *Digital and Analog Communication Systems*, John Wiley and Sons, New York, 1979, [Chapter 8](#).
- [9] Mahafza, B. *Introduction to Radar Analysis*, CRC Press, Boca Raton, Florida, 1998, [Chapter 6](#).
- [10] Poor, H. Vincent. *An Introduction to Signal Detection and Estimation*, Springer Publishing, New York, 1998.
- [11] Van Trees, H. *Detection, Estimation, and Modulation Theory, Part I*, Wiley-Interscience Publishing, Malden, Massachusetts, 2001.
- [12] Mitra, S. K., et al. “Interpolated Finite Impulse Response Filters,” *IEEE Trans. Acoust. Speech, and Signal Proc.*, Vol. ASSP-32, June 1984, pp. 563–570.
- [13] Vaidyanathan, P. *Multirate Systems and Filter Banks*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- [14] Mehrnia, A., and Willson, A., Jr. “On Optimal IFIR Filter Design,” *Proc. of the 2004 International Symp. on Circuits and Systems (ISCAS)*, Vol. 3, May 23–26, 2004, pp. 133–136.
- [15] Crochiere, R., and Rabiner, L. “Interpolation and Decimation of Digital Signals—A Tutorial

Review,” *Proceedings of the IEEE*, Vol. 69, No. 3, March 1981, pp. 300–331.

- [16] Kaiser, J. “Nonrecursive Digital Filter Design Using Io-Sinh Window Function,” *Proc. 1974 IEEE Int. Symp. Circuits Systems*, April 1974, pp. 20–23.
- [17] Harris, F. “Digital Signal Processing for Digital Modems,” DSP World Spring Design Conference, Santa Clara, California, April 1999.
- [18] Lim, Y. “Frequency-Response Masking Approach for the Synthesis of Sharp Linear Phase Digital Filters,” *IEEE Trans. Circuits Syst.*, Vol. 33, April 1986, pp. 357–364.
- [19] Yang, R., et al. “A New Structure of Sharp Transition FIR Filters Using Frequency-Response Masking,” *IEEE Trans. Circuits Syst.*, Vol. 35, August 1988, pp. 955–966.
- [20] Saramaki, T., et al. “Design of Computationally Efficient Interpolated FIR Filters,” *IEEE Trans. Circuits Syst.*, Vol. 35, January 1988, pp. 70–88.
- [21] Lyons, R. “Turbocharging Interpolated FIR Filters,” *IEEE Signal Processing Magazine*, “DSP Tips and Tricks” column, Vol. 24, No. 5, September 2007.
- [22] Lyons, R. *Streamlining Digital Signal Processing*, IEEE Press, Piscataway, New Jersey, 2007, pp. 73–84.
- [23] Rabiner, L., et al. “An Approach to the Approximation Problem for Nonrecursive Digital Filters,” *IEEE Trans. Audio Electroacoust.*, Vol. AU-18, June 1970, pp. 83–106.
- [24] Proakis, J., and Manolakis, D. *Digital Signal Processing: Principles, Algorithms, and Applications*, 3rd ed., Prentice Hall, Upper Saddle River, New Jersey, 1996, pp. 506–507.
- [25] Taylor, F., and Mellott, J. *Hands-on Digital Signal Processing*, McGraw-Hill, New York, 1998, pp. 325–331.
- [26] Rader, C., and Gold, B. “Digital Filter Design Techniques in the Frequency Domain,” *Proceedings of the IEEE*, Vol. 55, February 1967, pp. 149–171.
- [27] Rabiner, L., and Schafer, R. “Recursive and Nonrecursive Realizations of Digital Filters Designed by Frequency Sampling Techniques,” *IEEE Trans. Audio Electroacoust.*, Vol. AU-19, September 1971, pp. 200–207.
- [28] Gold, B., and Jordan, K. “A Direct Search Procedure for Designing Finite Duration Impulse Response Filters,” *IEEE Trans. Audio Electroacoust.*, Vol. AU-17, March 1969, pp. 33–36.
- [29] Rabiner, L., and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice Hall, Upper Saddle River, New Jersey, 1975, pp. 105–112.
- [30] Rorabaugh, C. *DSP Primer*, McGraw-Hill, New York, 1999, pp. 278–279.
- [31] Parks, T., and McClellan, J. “A Program for the Design of Linear Phase Finite Impulse Response Digital Filters,” *IEEE Trans. Audio Electroacoust.*, Vol. AU-20, August 1972, pp. 195–199.
- [32] Herrmann, O. “Design of Nonrecursive Digital Filters with Linear Phase,” *Electron. Lett.*, Vol. 6, May 28, 1970, pp. 328–329.
- [33] Rabiner, L. “Techniques for Designing Finite-Duration-Impulse-Response Digital Filters,” *IEEE Trans. on Communication Technology*, Vol. COM-19, April 1971, pp. 188–195.
- [34] Ingle, V., and Proakis, J. *Digital Signal Processing Using MATLAB*, Brookes/Cole Publishing,

Chapter 7 Problems

7.1 Prove that the frequency magnitude responses of the first-difference and central-difference differentiators can be represented by

$$|H_{Fd}(\omega)| = 2|\sin(\omega/2)|$$

and

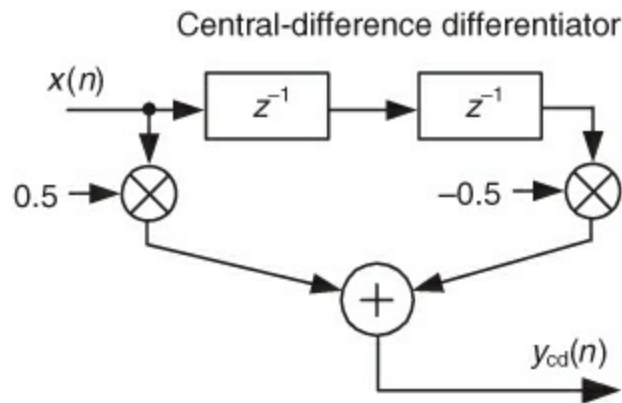
$$|H_{Cd}(\omega)| = |\sin(\omega)|$$

respectively. The ω frequency-domain variable has a range of $-\pi \leq \omega \leq \pi$, corresponding to a cyclic frequency range of $-f_s/2$ to $f_s/2$, where f_s is the differentiator's input signal sample rate in Hz.

Hint: Keep a list of trigonometric identities close at hand.

7.2 Redraw the block diagram of the central-difference differentiator, shown in [Figure P7-2](#), giving a new reduced-computation structure having only one multiplier.

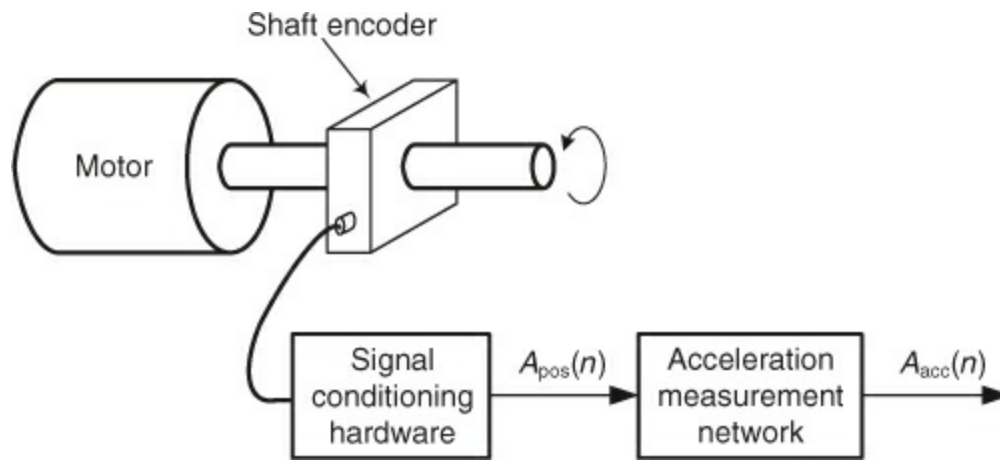
Figure P7-2



7.3 I once encountered an Internet web page that stated: “The central-difference differentiator is equivalent to two first-difference differentiators in series (cascaded).” Upon his reading that statement, if Rocky Balboa said, “This is very true,” would he be correct? Show how you arrived at your answer.

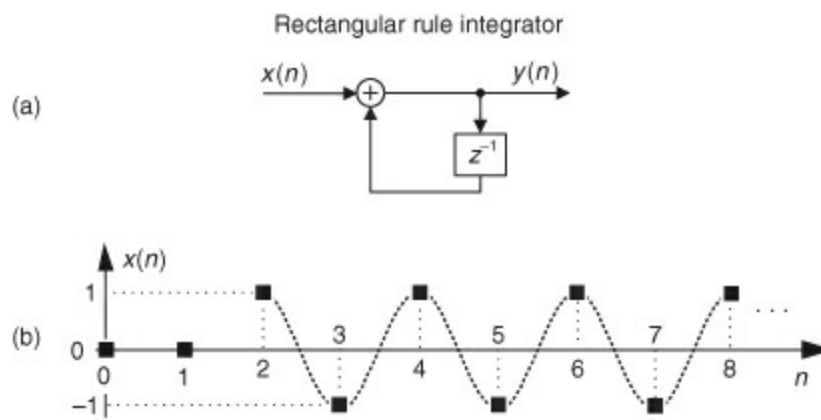
7.4 Assume you are working on a system that contains a servomotor and the motor's angular position, $A_{\text{pos}}(n)$, is measured by a shaft encoder coupled directly to the motor shaft as shown in [Figure P7-4](#). With the $A_{\text{pos}}(n)$ discrete-signal sequence representing the motor shaft angular position, assume that the $A_{\text{pos}}(n)$ sequence is contaminated with high-frequency noise and that the acceleration measurement network is restricted to performing no more than one multiplication per $A_{\text{pos}}(n)$ input sample. What is the block diagram of the acceleration measurement network that will generate the $A_{\text{acc}}(n)$ signal representing the acceleration of the motor shaft? Explain how you arrived at your solution.

Figure P7-4



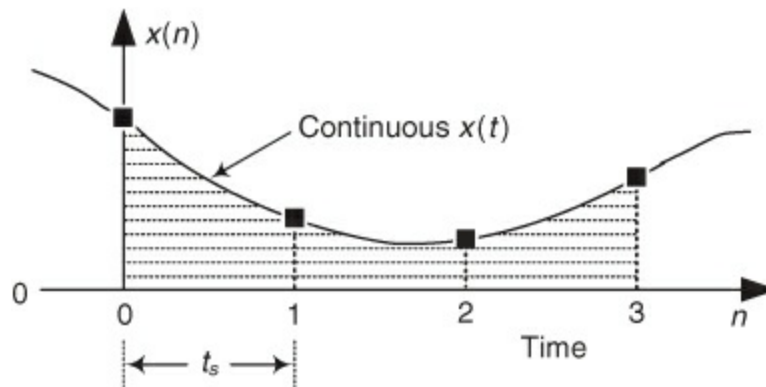
- 7.5** Here's a differentiator design problem. Compute the coefficients of an $N = 7$ -tap wideband differentiator whose cutoff frequency is $\omega_c = \pi$.
- 7.6** Differentiators are often used in digital receivers to perform FM (frequency modulation) demodulation. For a differentiator to be useful for FM demodulation its group delay should be an integer number of samples. One proposed differentiating filter has the following difference equation:
- $$y_{\text{diff}}(n) = \frac{-x(n)}{16} + x(n-2) - x(n-4) + \frac{x(n-6)}{16}$$
- where $y_{\text{diff}}(n)$ is the differentiator's output. Prove that this differentiator's group delay is an integer number of samples.
- 7.7** This author once read, on the Internet, the surprising statement that “the real part of the frequency response of the rectangular rule integrator is 0.5 for all input frequencies.” Prove the validity of that statement.
- 7.8** In the text we introduced the rectangular rule integrator whose block diagram is shown in [Figure P7-8\(a\)](#).
- (a)** Assume that, at time index $n = 2$, we apply an $x(n)$ cosine sequence whose frequency is half the input signal sample rate ($f_s/2$), shown in [Figure P7-8\(b\)](#), to a rectangular rule integrator. Given this $x(n)$ input sequence, draw the $y(n)$ output sequence of the integrator.
- (b)** Based on the $y(n)$ sequence from Part (a), by what factor does the rectangular rule integrator have an amplitude gain, or an amplitude loss, when the input is a sinusoid whose frequency is $f_s/2$ Hz?
- (c)** Use the text's equation for the frequency response of the integrator to verify your answer to Part (b) of this problem.

Figure P7-8



- 7.9** Examine the $x(n)$ samples obtained from digitizing the continuous $x(t)$ signal shown in [Figure P7-9](#), and assume we must estimate the shaded area, A , under the $x(t)$ curve during the time interval of 0 to $3t_s$ seconds. If we use a trapezoidal rule integrator for our computations, will our estimate of the shaded area A be larger or smaller than the true area A ? Justify your answer.

Figure P7-9



- 7.10** In the continuous-signal domain an integrator is defined by the following Laplace-domain transfer function expression:

$$H_{\text{int}}(s) = \frac{1}{s}.$$

- (a) Using the bilinear transform method ([Chapter 6](#)) for converting analog s -domain transfer functions to discrete z -domain transfer functions, what is the $H(\omega)$ frequency response expression for the bilinear transform-designed integrator corresponding to the $H(s)$ above?
- (b) Which of the integrator types discussed in this chapter's text (rectangular rule, trapezoidal rule, Simpson's rule, and Tick's rule) is most similar to the bilinear transform-designed integrator obtained in Part (a)?
- 7.11** Considering the four discrete integrators discussed in the text, rectangular rule, trapezoidal rule, Simpson's rule, and Tick's rule:
- (a) Which integrator type is the most computationally simple?
- (b) Which integrator has a transfer function pole (infinite gain) at $z = -1$? Show how you arrived at your answer.
- (c) Which integrator has a frequency magnitude response exactly equal to zero (infinite attenuation) at $f_s/2$ Hz? Show how you arrived at your answer.

- 7.12** Consider the following four-sample $s(n)$ sequence to be a signal of interest that we want to

detect using a tapped-delay line FIR matched filter:

$$s(n) = [2, 4, -1, 1].$$

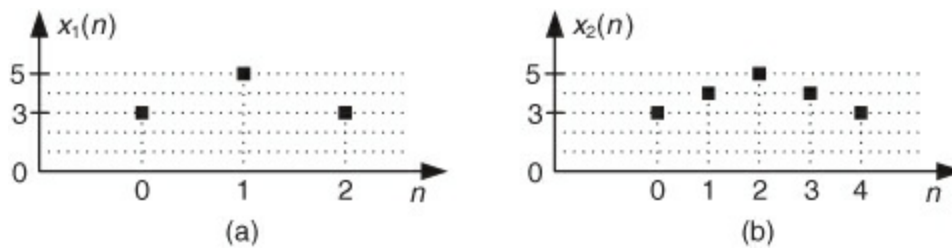
(a) Draw the block diagram of the FIR matched filter showing the filter coefficients.

(b) What is the $y(n)$ output sequence of the filter as the $s(n)$ sequence passes completely through the delay line of the matched filter?

7.13 If $x_s(n)$ is an N -sample, noise-free signal of interest we wish to detect using a tapped-delay line matched filter, what is a simple algebraic expression for the maximum filter output sample value?

7.14 Considering the signal-to-noise ratio (SNR) discussion at the end of the text's matched filter material, think about a signal of interest represented by the three-sample, triangular-like $x_1(n)$ shown in [Figure P7-14\(a\)](#).

Figure P7-14



Next, consider representing the $x_1(n)$ signal's triangular envelope with five samples like the $x_2(n)$ sequence shown in [Figure P7-14\(b\)](#). What is the improvement in a matched filter's $y_2(n)$ output SNR, measured in decibels (dB), by using the $x_2(n)$ sequence as our signal of interest over the $y_1(n)$ output SNR using the $x_1(n)$ sequence as our signal of interest? Justify your answer.

7.15 In designing lowpass IFIR filters we prefer to use large values for the M expansion factor because large M reduces the number of taps in the shaping subfilter. How do large values for M affect the number of taps in the associated image-reject subfilter?

7.16 Assume we're designing a lowpass IFIR filter with an expansion factor of $M = 5$ and our desired IFIR filter has a passband from zero to 4 kHz ($f_{\text{pass}} = 4$ kHz).

(a) What is the passband width, in Hz, of the *prototype* FIR filter used to start our IFIR filter design effort?

(b) How many passband images will reside between zero and $f_s/2$ Hz in the IFIR *shaping* subfilter?

7.17 Assume we're tasked to design a lowpass IFIR filter, operating at a sampling rate of $f_s = 3$ MHz, having the following properties:

- Passband width of 60 kHz
- Maximum passband peak-to-peak ripple of 0.2 dB
- Stopband beginning at 61.2 kHz

(a) What is the value of the optimum M expansion factor for the shaping subfilter?

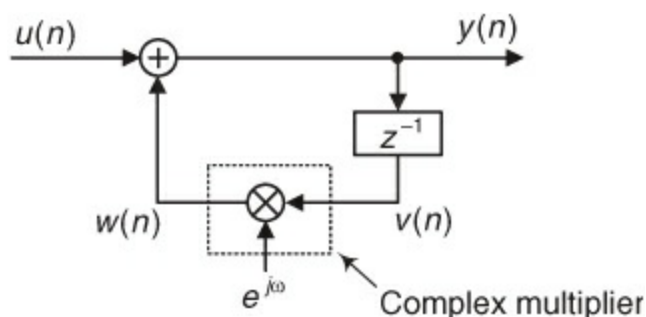
(b) What is the maximum passband peak-to-peak ripple requirement, in dB, for the prototype filter?

(c) Using the text's [Figure 7-18\(b\)](#), what percent computational reduction should we expect from the IFIR filter relative to a Parks-McClellan-designed tapped-delay line FIR filter?

7.18 In studying frequency sampling filters (FSFs) we encountered networks containing complex multipliers like that shown in [Figure P7-18](#). Let's now explore the details of such a multiplier.

- Write the time-domain difference equation of the complex $w(n)$ output of the complex multiplier in terms of $v(n) = v_R(n) + jv_I(n)$ and $e^{j\omega}$ where all the variables are in rectangular form. Variable ω is a fixed radian angle in the range $-\pi \leq \omega \leq \pi$.
- Based on the difference equation from Part (a), draw the block diagram of the complex multiplier where the complex $v(n)$ and $w(n)$ discrete sequences, and the constant complex $e^{j\omega}$ multiplicand, are shown in rectangular (real-only variables) form.
- How many real multiplies and how many real additions are required to perform a single complex multiply?

Figure P7-18



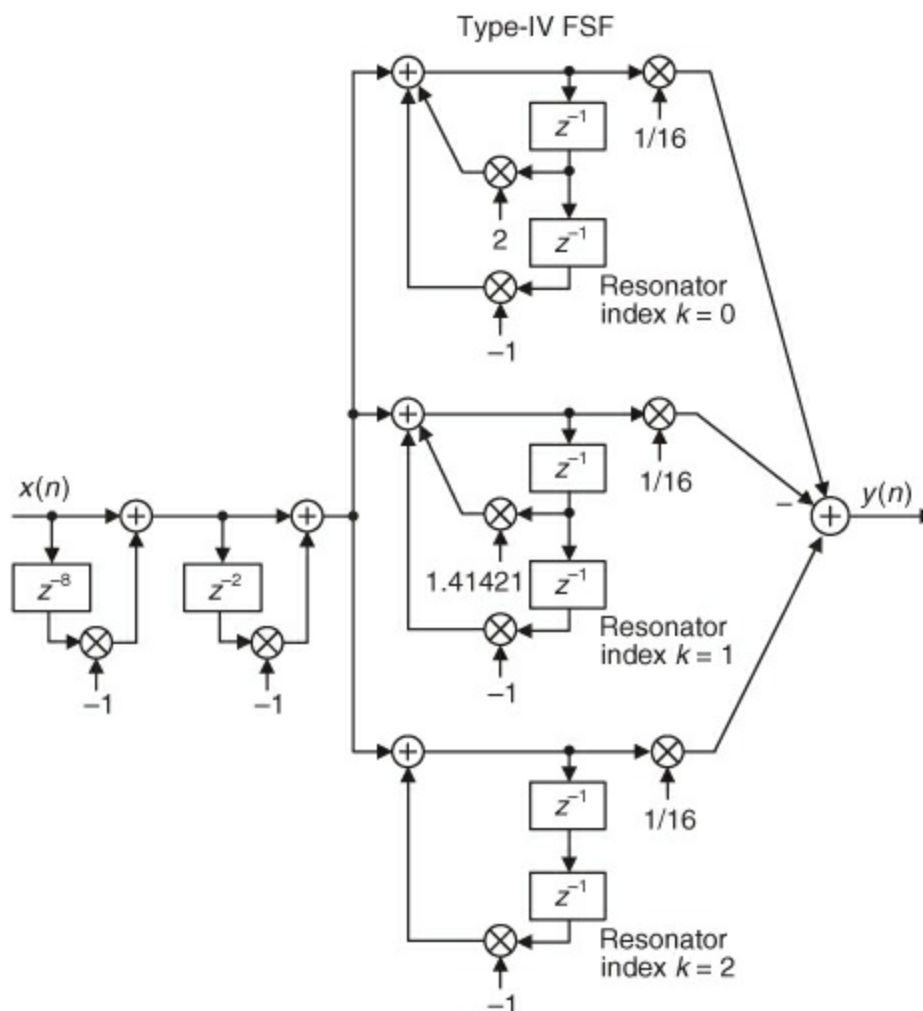
7.19 To exercise your frequency sampling filter (FSF) analysis skills, consider the three-stage Type-IV FSF in [Figure P7-19](#).

- Draw the FSF's pole and zero locations on the z -plane.

Hint: From [Figure P7-19](#), determine the filter's N and r parameters to find the zeros' locations on the unit circle, and use each resonator's k index value to find the FSF's pole locations on the unit circle.

- Is this filter a lowpass, bandpass, or highpass filter? Justify your answer.
- Draw a rough sketch of the filter's frequency magnitude response over the frequency range of $-\pi \leq \omega \leq \pi$ radians/sample ($-f_s/2 \leq f \leq f_s/2$ Hz).

Figure P7-19



7.20 Concerning cascaded subfilters used in frequency sampling filters (FSFs):

(a) What is the impulse response of the “two cascaded comb filters” combination in a Type-IV FSF? (See the text’s [Figure 7-41](#).) Assume $N = 8$, and the damping factor $r = 1$.

(b) Will this cascaded combination of comb filters have linear phase? Explain how you arrived at your answer.

Hint: Recall what is an important characteristic of the impulse response of a linear-phase digital filter.

(c) If we set $r = 0.9$, will the combination of comb filters have linear phase? Explain how you arrived at your answer.

7.21 What is the range, roughly, of stopband attenuation that can be achieved with a Type-IV frequency sampling filter (FSF) having a single non-unity transition band coefficient?

7.22 If a linear-phase lowpass filter needs to be designed to have a minimum of 85 dB of stopband attenuation, is it possible to design a Type-IV frequency sampling filter (FSF) that satisfies this attenuation requirement?

7.23 Assume you’ve designed a six-section, $N = 23$, lowpass, Type-IV frequency sampling filter (FSF) (such as that in [Figure 7-41](#)) having three low-frequency passband sections, and the remaining three sections will use [transition](#) coefficients to achieve your desired stopband attenuation.

(a) What are the optimum values for those three coefficients? Show how you arrived at your solution.

(b) Approximately what level of stopband attenuation (in dB) should we expect from our six-section, $N = 23$ FSF? Show how you arrived at your solution.

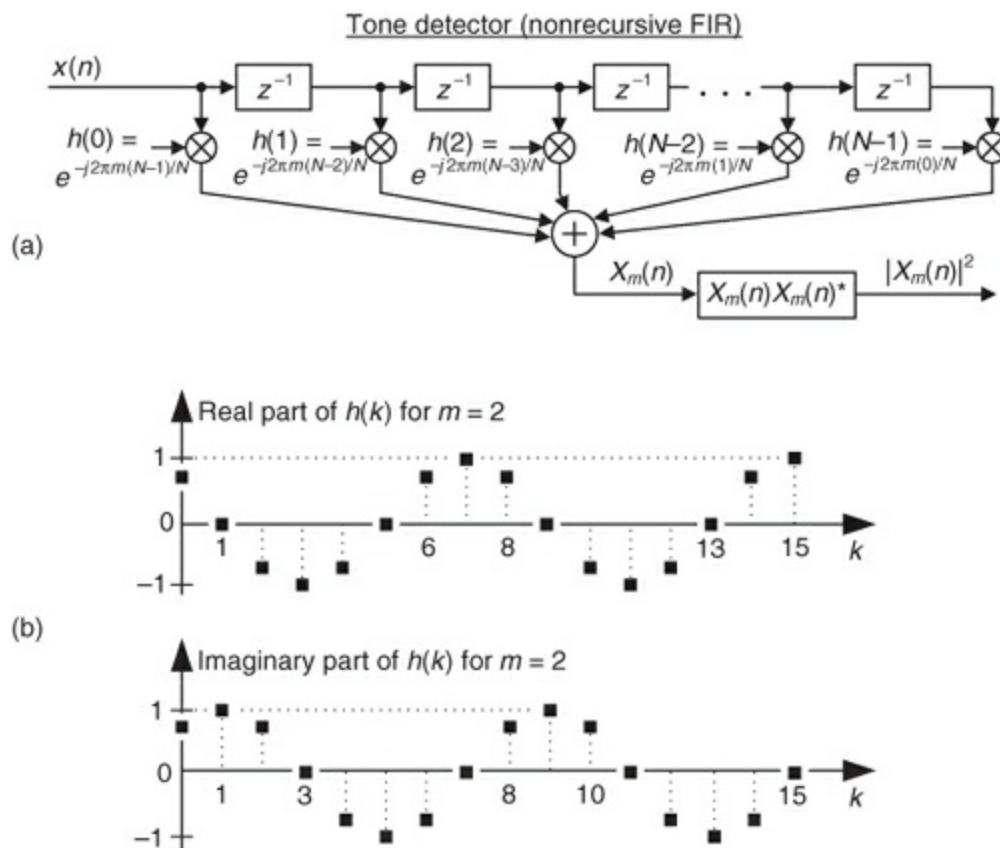
(c) Draw the block diagram (the [structure](#)) of your FSF. No resonator details need be drawn. Just show a resonator as a block labeled with its appropriate k value.

7.24 Here is a frequency sampling filter (FSF) problem whose solution is a network useful for computing a single-bin discrete Fourier transform (DFT) output magnitude sample in a way that avoids having to perform an entire DFT. Such a network can be used for sinusoidal tone detection.

Think about building a tapped-delay line FIR filter, with the complex coefficients shown in [Figure P7-24-I\(a\)](#). After the arrival of the $x(N-1)$ input sample the system's output is equal to the m th DFT bin's power sample of an N -point DFT of the most recent N input samples. What we're saying is that we can build a real-time spectral power measurement system for computing successive single-bin DFT power values (the m th DFT bin power samples over time). The tapped-delay line filter's $h(k)$ coefficients are

$$h(k) = e^{-j2\pi m(N-1-k)/N}$$

Figure P7-24-I



where filter coefficient index k is $0 \leq k \leq N-1$. The output term $|X_m(n)|^2$ in [Figure P7-24-I\(a\)](#) represents the value of the DFT's m th bin power at time instant n . (Power samples are computed by multiplying $X_m(n)$ samples by their conjugates $X_m(n)^*$.)

As an example, if we needed to detect the power of the $m = 2$ bin of an $N = 16$ -point DFT, the complex $h(k)$ coefficients of the tapped-delay line filter would be $h(k) = e^{-j2\pi 2(15-k)/16}$ as shown in [Figure P7-24-I\(b\)](#). In this case our real-time spectral power detector would be the filter shown in [Figure P7-24-II\(a\)](#). (Notice how those coefficients are a flipped-in-time version of an $e^{-j2\pi 2k/16}$ sequence. That's necessary because the earlier-in-time $x(n)$ samples are toward the right in the

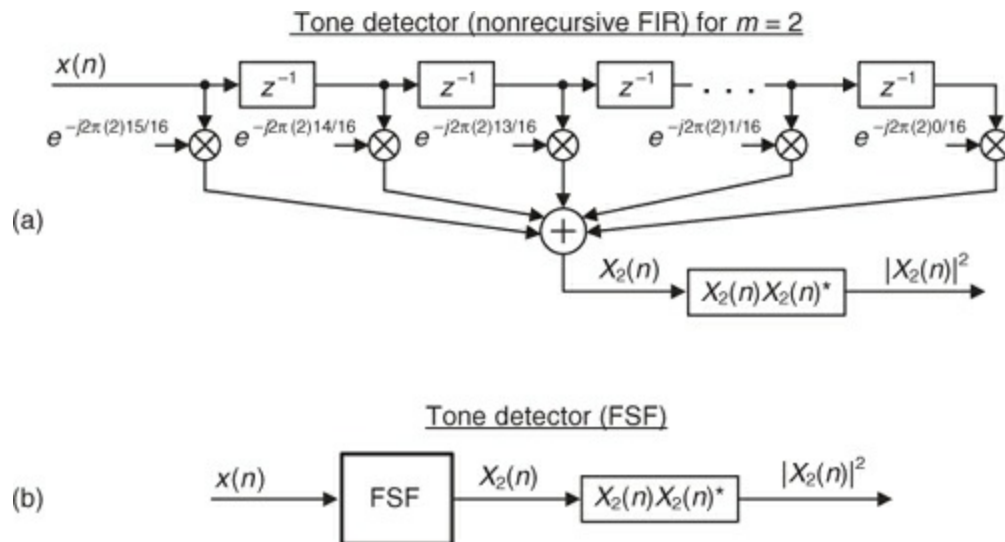
tapped-delay line filter.)

- (a) Reviewing the equivalency of a tapped-delay line FIR filter and a complex FSF in the text's [Figure 7-20](#), draw the FSF block diagram that enables the real-time computation of the single $m = 2$ bin power for an $N = 16$ -point DFT function performed in [Figure P7-24-II\(a\)](#). That is, show what would be the network inside the FSF block in [Figure P7-24-II\(b\)](#), including the numerical values for the $H(k)/N$ gain factors following the FSF resonators.

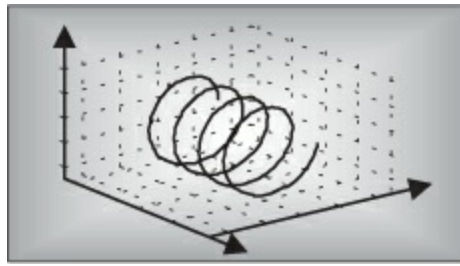
Hint: There are two ways to solve this problem. You could (1) perform a customary DFT analysis on the tapped-delay line filter's $h(k)$ coefficients to obtain the FSF's $H(k)/N$ gain factors, or (2) you could determine the $H(z)$ z -domain transfer function of the FSF to determine its block diagram. (Pick your poison.)

- (b) If the f_s sample rate of the $x(n)$ inputs in [Figure P7-24-II](#) is 200 kHz, what is the center frequency of the $m = 2$ DFT bin of our real-time single-bin 16-point DFT power measurement system?

Figure P7-24-II



Chapter Eight. Quadrature Signals



Quadrature signals are based on the notion of complex numbers. Perhaps no other topic causes more heartache for newcomers to DSP than these numbers and their strange terminology of *j-operator*, *complex*, *analytic*, *imaginary*, *real*, and *orthogonal*. If you're a little unsure of the physical meaning of complex numbers and the $j = \sqrt{-1}$ operator, don't feel bad because you're in good company. Sixteenth-century Italian mathematician Girolamo Cardano described an imaginary number "as subtle as it is useless." In the seventeenth century Gottfried Leibniz described imaginary numbers as being "*amphibian*, halfway between existence and nonexistence." (The term *imaginary* was first used by the brilliant mathematician/philosopher René Descartes in the seventeenth century and was meant to be derogatory. That's because not only was the notion of the square root of a negative number dubious at best, surprisingly there was no consensus at that time as to the true meaning of negative real numbers.) Even Karl Gauss, one of the world's greatest mathematicians, called the *j-operator* the "shadow of shadows." Here we'll shine some light on that shadow so you'll never have to call the *Quadrature Psychic Hotline* for help.

Quadrature signals, represented by complex numbers, are used in just about every field of science and engineering.[‡] Quadrature signals are of interest to us because they describe the effects of Fourier analysis as well as the quadrature processing and implementations that take place in modern digital communications systems. In this chapter we'll review the fundamentals of complex numbers and get comfortable with how they're used to represent quadrature signals. Next we'll examine the notion of negative frequency as it relates to quadrature signal algebraic notation and learn to speak the language of quadrature processing. In addition, we'll use three-dimensional time- and frequency-domain plots to clarify and give physical meaning to quadrature signals.

[‡] That's because complex sinusoids are solutions to those second-order linear differential equations used to describe so much of nature.

8.1 Why Care about Quadrature Signals?

Quadrature signal formats, also called *complex signals*, are used in many digital signal processing applications, such as

- digital communications systems,
- radar systems,
- time difference of arrival processing in radio direction-finding schemes,
- coherent pulse measurement systems,
- antenna beamforming applications, and
- single sideband modulators.

These applications fall into the general category known as *quadrature processing*, and they provide

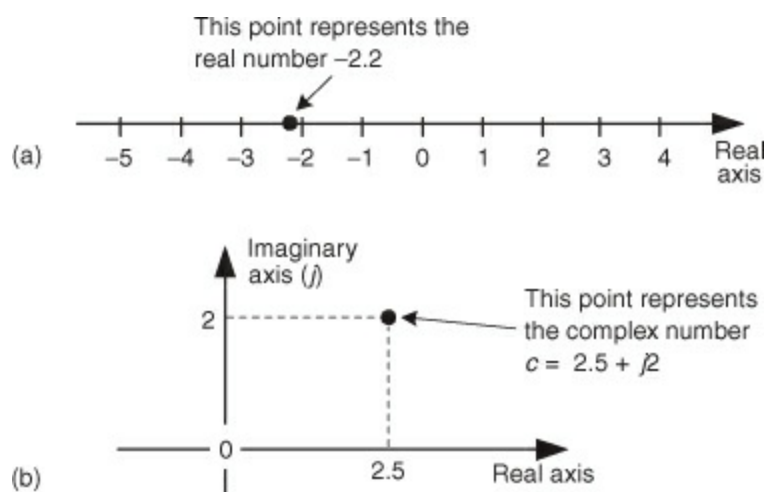
additional processing power through the coherent measurement of the phase of sinusoidal signals.

A quadrature signal is a two-dimensional signal whose value at some instant in time can be specified by a single complex number having two parts: what we call the *real part* and the *imaginary part*. (The words *real* and *imaginary*, although traditional, are unfortunate because of their meanings in our everyday speech. Communications engineers use the terms *in-phase* and *quadrature phase*. More on that later.) Let's review the mathematical notation of these complex numbers.

8.2 The Notation of Complex Numbers

To establish our terminology, we define real numbers to be those numbers we use in everyday life, like a voltage, a temperature on the Fahrenheit scale, or the balance of your checking account. These one-dimensional numbers can be either positive or negative, as depicted in [Figure 8-1\(a\)](#). In that figure we show a one-dimensional axis and say a single real number can be represented by a point on the axis. Out of tradition, let's call this axis the *real axis*.

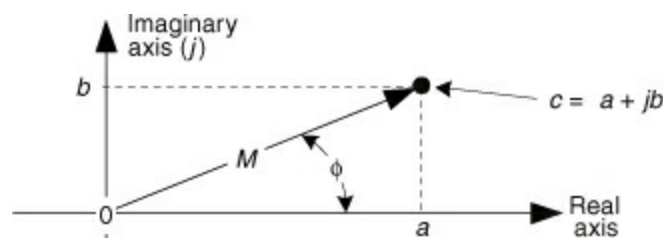
Figure 8-1 Graphical interpretations: (a) a real number; (b) a complex number.



A complex number c is shown in [Figure 8-1\(b\)](#) where it's also represented as a point. Complex numbers are not restricted to lying on a one-dimensional line but can reside anywhere on a two-dimensional plane. That plane is called the *complex plane* (some mathematicians like to call it an *Argand diagram*), and it enables us to represent complex numbers having both real and imaginary parts. For example, in [Figure 8-1\(b\)](#), the complex number $c = 2.5 + j2$ is a point lying on the complex plane on neither the real nor the imaginary axis. We locate point c by going +2.5 units along the real axis and up +2 units along the imaginary axis. Think of those real and imaginary axes exactly as you think of the east-west and north-south directions on a road map.

We'll use a geometric viewpoint to help us understand some of the arithmetic of complex numbers. Taking a look at [Figure 8-2](#), we can use the trigonometry of right triangles to define several different ways of representing the complex number c .

Figure 8-2 The phasor representation of complex number $c = a + jb$ on the complex plane.



Our complex number c is represented in a number of different ways in the literature, as shown in

Table 8-1 Complex Number Notation

Notation name	Math expression	Remarks
Rectangular form	$c = a + jb$	Used for explanatory purposes. (8-1) Easiest to understand. (Also called the <i>Cartesian form</i> .)
Trigonometric form	$c = M[\cos(\varnothing) + j\sin(\varnothing)]$	Commonly used to describe quadrature signals in communications systems. (8-2)
Polar form	$c = Me^{j\varnothing}$	Most puzzling, but the primary form used in math equations. (8-3) (Also called the <i>exponential form</i> . Sometimes written as $M\exp(j\varnothing)$.)
Magnitude-angle form	$c = M\angle\varnothing$	Used for descriptive purposes, but too cumbersome for use in algebraic equations. (Essentially a shorthand version of Eq. (8-3).) (8-4)

Eqs. (8-3) and (8-4) remind us that c can also be considered the tip of a phasor on the complex plane, with magnitude M , in the direction of \varnothing degrees relative to the positive real axis as shown in Figure 8-2. Keep in mind that c is a complex number and the variables a , b , M , and \varnothing are all real numbers. The magnitude of c , sometimes called the modulus of c , is

(8-5)

$$M = |c| = \sqrt{a^2 + b^2}.$$

The phase angle \varnothing , the *argument* of c , is the arctangent of the ratio of the imaginary part over the real part, or

(8-6)

$$\varnothing = \tan^{-1} \left(\frac{b}{a} \right) \text{ radians.}$$

If we set Eq. (8-3) equal to Eq. (8-2), $Me^{j\varnothing} = M[\cos(\varnothing) + j\sin(\varnothing)]$, we can state what's named in his honor and now called one of Euler's identities as

(8-7)

$$e^{j\varnothing} = \cos(\varnothing) + j\sin(\varnothing).$$

The suspicious reader should now be asking, "Why is it valid to represent a complex number using that strange expression of the base of the natural logarithms, e , raised to an imaginary power?" We can validate Eq. (8-7) as did Europe's wizard of infinite series, Leonhard Euler, by plugging $j\varnothing$ in for z in the series expansion definition of e^z in the top line of Figure 8-3.[‡] That substitution is shown on the second line. Next we evaluate the higher orders of j to arrive at the series in the third line in the figure. Those of you with elevated math skills like Euler (or who check some math reference book) will recognize that the alternating terms in the third line are the series expansion definitions of the cosine and sine functions.

[‡] Leonhard Euler, born in Switzerland in 1707, is considered by many historians to be the world's greatest mathematician. By the way, the name Euler is pronounced 'oy-ler.

Figure 8-3 One derivation of Euler's equation using series expansions for e^z , $\cos(\theta)$, and $\sin(\theta)$.

$$\begin{aligned}
 e^z &= 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \frac{z^4}{4!} + \dots \\
 e^{j\phi} &= 1 + j\phi + \frac{(j\phi)^2}{2!} + \frac{(j\phi)^3}{3!} + \frac{(j\phi)^4}{4!} + \frac{(j\phi)^5}{5!} + \frac{(j\phi)^6}{6!} + \dots \\
 e^{j\phi} &= 1 + j\phi - \frac{\phi^2}{2!} - j\frac{\phi^3}{3!} + \frac{\phi^4}{4!} + j\frac{\phi^5}{5!} - \frac{\phi^6}{6!} + \dots
 \end{aligned}$$

$$e^{j\phi} = \cos(\phi) + j\sin(\phi)$$

Figure 8-3 verifies Eq. (8-7) and justifies our representation of a complex number using the Eq. (8-3) polar form: $Me^{j\theta}$. If we substitute $-j\theta$ for z in the top line of Figure 8-3, we end up with a slightly different, and very useful, form of Euler's identity:

$$(8-8)$$

$$e^{-j\theta} = \cos(\theta) - j\sin(\theta).$$

The polar form of Eqs. (8-7) and (8-8) benefits us because:

- It simplifies mathematical derivations and analysis, turning trigonometric equations into the simple algebra of exponents. Math operations on complex numbers follow exactly the same rules as real numbers.
- It makes adding signals merely the addition of complex numbers (vector addition).
- It's the most concise notation.
- It's indicative of how digital communications system are implemented and described in the literature.

Here's a quick example of how the polar form of complex numbers can simplify a mathematical analysis. Let's say we wanted to understand the process of multiplying complex number $c_1 = \cos(\theta) + j\sin(\theta)$ by another complex number, $c_2 = \cos(2\theta) - j\sin(2\theta)$, whose angle is the negative of twice c_1 's angle. The product is

$$(8-9)$$

$$\begin{aligned}
 c_1 c_2 &= [\cos(\theta) + j\sin(\theta)][\cos(2\theta) - j\sin(2\theta)] \\
 &= \cos(\theta)\cos(2\theta) + \sin(\theta)\sin(2\theta) + j[\sin(\theta)\cos(2\theta) - \cos(\theta)\sin(2\theta)].
 \end{aligned}$$

Using the trigonometric *function product* identities, we can write Eq. (8-9) as

$$(8-10)$$

$$\begin{aligned}
 c_1 c_2 &= (1/2)[\cos(-\theta) + \cos(3\theta) + \cos(-\theta) - \cos(3\theta)] \\
 &\quad + j(1/2)[\sin(3\theta) + \sin(-\theta) - \sin(3\theta) + \sin(-\theta)] \\
 &= \cos(-\theta) + j\sin(-\theta) = \cos(\theta) - j\sin(\theta).
 \end{aligned}$$

So the $c_1 c_2$ product yields the complex conjugate of c_1 . That's not too thrilling, but what is interesting is how trivial a polar form $c_1 c_2$ product analysis turns out to be. We can complete our polar form

analysis in one brief line:

(8-11)

$$c_1 c_2 = e^{j\theta} e^{-j2\theta} = e^{-j\theta},$$

which is equivalent to [Eq. \(8-10\)](#). For math analysis, polar form is usually the notation of choice.

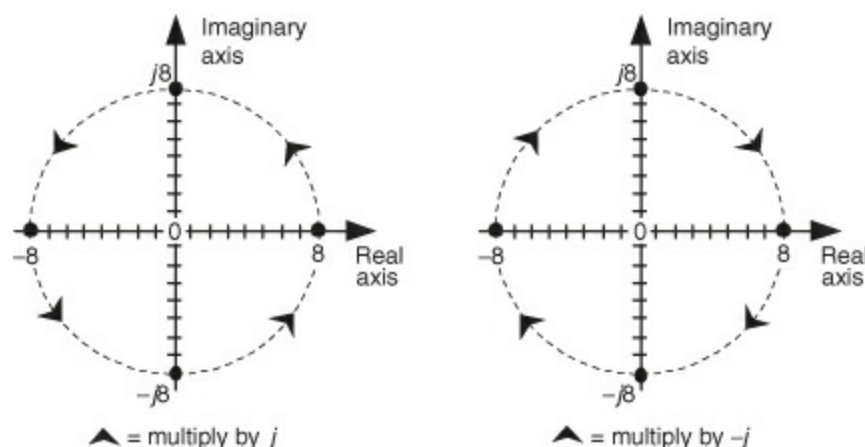
Back to quadrature signals. We'll be using [Eqs. \(8-7\)](#) and [\(8-8\)](#) to understand the nature of time-domain quadrature signals. But first let's take a deep breath and enter the Twilight Zone of the j operator.

You've seen the definition $j = \sqrt{-1}$ before. Stated in words, we say that j represents a number that when multiplied by itself results in negative one. Well, this definition causes difficulty for the beginner because we all know that any number multiplied by itself always results in a positive number. (Unfortunately, engineering textbooks often define j and then, with justified haste, swiftly carry on with all the ways the j operator can be used to analyze sinusoidal signals. Readers soon forget about the question "What does $j = \sqrt{-1}$ actually mean?") Well, $\sqrt{-1}$ had been on the mathematical scene for some time but wasn't taken seriously until it had to be used to solve cubic polynomial equations in the sixteenth century[1,2]. Mathematicians reluctantly began to accept the abstract concept of $\sqrt{-1}$ without having to visualize it, because its mathematical properties were consistent with the arithmetic of normal real numbers.

It was Euler's equating complex numbers to real sines and cosines, and Gauss's brilliant introduction of the complex plane, that finally legitimized the notion of $\sqrt{-1}$ to Europe's mathematicians in the eighteenth century. Euler, going beyond the province of real numbers, showed that complex numbers had a clean, consistent relationship to the well-known real trigonometric functions of sines and cosines. As Einstein showed the equivalence of mass and energy, Euler showed the equivalence of real sines and cosines to complex numbers. Just as modern-day physicists don't know what an electron is but they understand its properties, we'll not worry about what j is and be satisfied with understanding its behavior. We'll treat j not as a number, but as an operation performed on a number, as we do with negation or multiplication. For our purposes, the j -operator means rotate a complex number by 90 degrees counterclockwise. (For our friends in the UK, counterclockwise means your anti-clockwise.) Let's see why.

We'll get comfortable with the complex plane representation of imaginary numbers by examining the mathematical properties of the $j = \sqrt{-1}$ operator as shown in [Figure 8-4](#).

Figure 8-4 What happens to the real number 8 when multiplied by j and $-j$.



Multiplying any number on the real axis by j results in an imaginary product lying on the imaginary

axis. The example on the left in [Figure 8-4](#) shows that if $+8$ is represented by the dot lying on the positive real axis, multiplying $+8$ by j results in an imaginary number, $+j8$, whose position has been rotated 90 degrees counterclockwise (from $+8$), putting it on the positive imaginary axis. Similarly, multiplying $+j8$ by j results in another 90-degree rotation, yielding the -8 lying on the negative real axis because $j^2 = -1$. Multiplying -8 by j results in a further 90-degree rotation, giving the $-j8$ lying on the negative imaginary axis. Whenever any number represented by a dot is multiplied by j , the result is a counterclockwise rotation of 90 degrees. (Conversely, multiplication by $-j$ results in a clockwise rotation of -90 degrees on the complex plane.)

If we let $\varnothing = \pi/2$ in [Eq. 8-7](#), we can say

$$(8-12)$$

$$e^{j\pi/2} = \cos(\pi/2) + j\sin(\pi/2) = 0 + j1, \text{ or}$$

$$e^{j\pi/2} = j.$$

Here's the point to remember. If you have a single complex number, represented by a point on the complex plane, multiplying that number by j or by $e^{j\pi/2}$ will result in a new complex number rotated 90 degrees counterclockwise (CCW) on the complex plane. Don't forget this, as it will be useful as you begin reading the literature of quadrature processing systems!

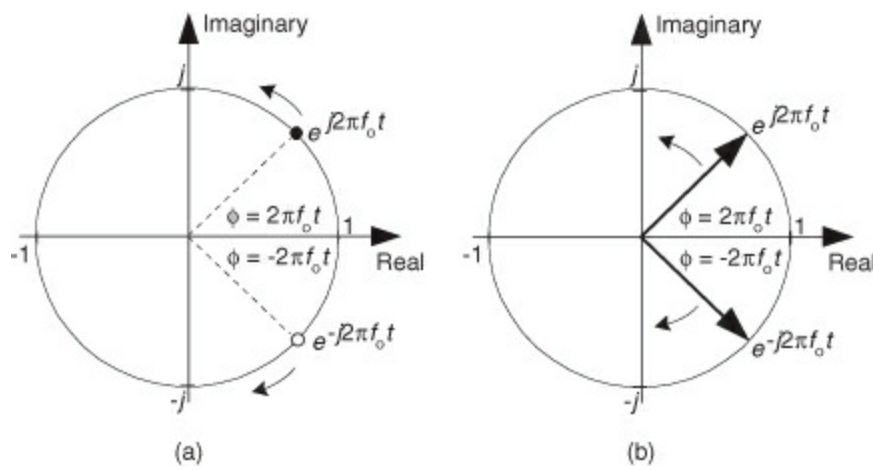
Let's pause for a moment here to catch our breath. Don't worry if the ideas of imaginary numbers and the complex plane seem a little mysterious. It's that way for everyone at first—you'll get comfortable with them the more you use them. (Remember, the j -operator puzzled Europe's heavyweight mathematicians for many years.) Granted, not only is the mathematics of complex numbers a bit strange at first, but the terminology is almost bizarre. While the term *imaginary* is an unfortunate one to use, the term *complex* is downright weird. When first encountered, the phrase "complex numbers" makes us think *complicated numbers*. This is regrettable because the concept of complex numbers is not really so complicated.[‡] Just know that the purpose of the above mathematical rigmarole was to validate [Eqs. \(8-2\)](#), [\(8-3\)](#), [\(8-7\)](#), and [\(8-8\)](#). Now, let's (finally!) talk about time-domain signals.

[‡] The brilliant American engineer Charles P. Steinmetz, who pioneered the use of real and imaginary numbers in electrical circuit analysis in the early twentieth century, refrained from using the term *complex numbers*—he called them *general numbers*.

8.3 Representing Real Signals Using Complex Phasors

We now turn our attention to a complex number that is a function of time. Consider a number whose magnitude is one, and whose phase angle increases with time. That complex number is the $e^{j2\pi f_0 t}$ point shown in [Figure 8-5\(a\)](#). (Here the $2\pi f_0$ term is frequency in radians/second, and it corresponds to a frequency of f_0 cycles/second where f_0 is measured in Hz.) As time t gets larger, the complex number's phase angle increases and our number orbits the origin of the complex plane in a CCW direction. [Figure 8-5\(a\)](#) shows the number, represented by the solid dot, frozen at some arbitrary instant in time. If, say, the frequency $f_0 = 2$ Hz, then the dot would rotate around the circle two times per second. We can also think of another complex number $e^{-j2\pi f_0 t}$ (the white dot) orbiting in a clockwise direction because its phase angle gets more negative as time increases.

Figure 8-5 A snapshot, in time, of two complex numbers whose exponents change with time: (a) numbers shown as dots; (b) numbers shown as phasors.

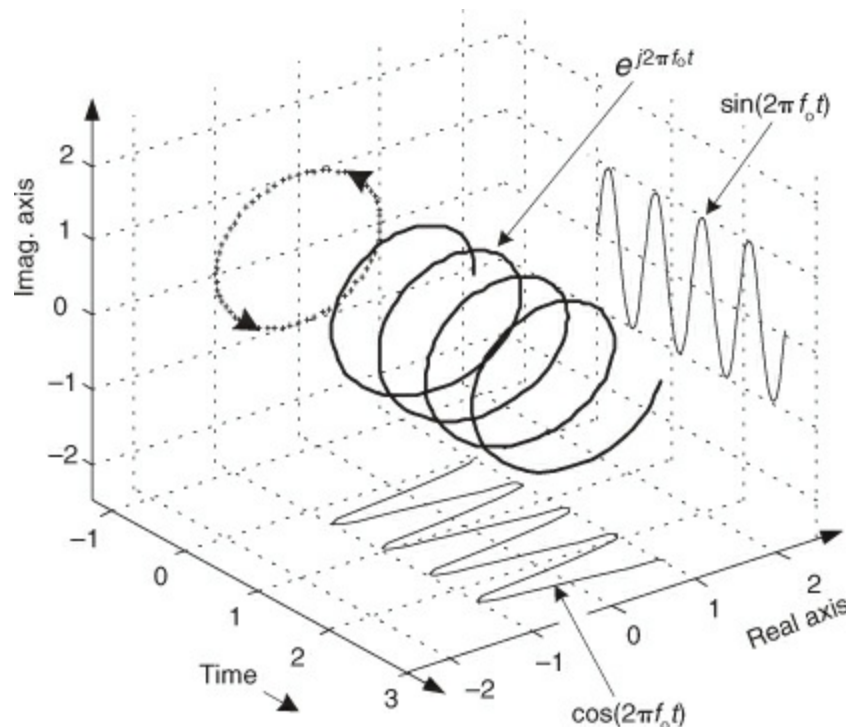


Let's now call our two complex expressions, $e^{j2\pi f_0 t}$ and $e^{-j2\pi f_0 t}$, quadrature signals. Each has both real and imaginary parts, and they are both functions of time. Those $e^{j2\pi f_0 t}$ and $e^{-j2\pi f_0 t}$ expressions are often called *complex exponentials* in the literature.

We can also think of those two quadrature signals, $e^{j2\pi f_0 t}$ and $e^{-j2\pi f_0 t}$, as the tips of two phasors rotating in opposite directions, as shown in [Figure 8-5\(b\)](#). We're going to stick with this phasor notation for now because it'll allow us to achieve our goal of representing real sinusoids in the context of the complex plane. Don't touch that dial!

To ensure that we understand the behavior of a simple quadrature signal, [Figure 8-6](#) shows the three-dimensional path of the $e^{j2\pi f_0 t}$ signal as time passes. We've added the time axis, coming out of the page, to show how $e^{j2\pi f_0 t}$ follows a corkscrew path spiraling along, and centered about, the time axis. The real and imaginary parts of $e^{j2\pi f_0 t}$ are shown as the sine and cosine projections in [Figure 8-6](#) and give us additional insight into [Eq. 8-7](#).

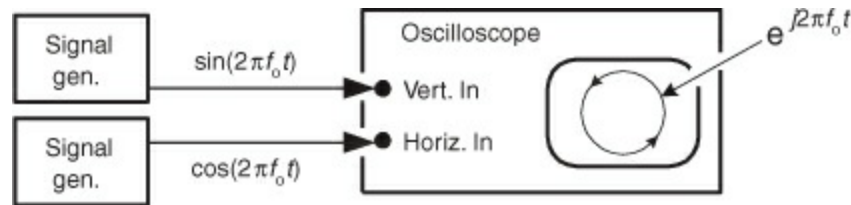
Figure 8-6 The motion of the $e^{j2\pi f_0 t}$ complex signal as time increases.



To appreciate the physical meaning of our discussion here, let's remember that a continuous quadrature signal $e^{j2\pi f_0 t} = \cos(2\pi f_0 t) + j\sin(2\pi f_0 t)$ is not just mathematical mumbo jumbo. We can generate $e^{j2\pi f_0 t}$ in our laboratory and transmit it to another lab down the hall. All we need is two

sinusoidal signal generators, set to the same frequency f_0 . (However, somehow we have to synchronize those two hardware generators so their relative phase shift is fixed at 90 degrees.) Next we connect coax cables to the generators' output connectors and run those two cables, labeled *cos* for the cosine signal and *sin* for the sinewave signal, to their destination as shown in [Figure 8-7](#).

Figure 8-7 Displaying a quadrature signal using an oscilloscope.



Now for a two-question pop quiz. First question: In the other lab, what would we see on the screen of an oscilloscope if the continuous real $\cos(2\pi f_0 t)$ and $\sin(2\pi f_0 t)$ signals were connected to the horizontal and vertical input channels, respectively, of the scope (remembering, of course, to set the scope's horizontal sweep control to the External position)? That's right. We'd see the scope's electron beam rotating counterclockwise in a circle on the scope's screen.

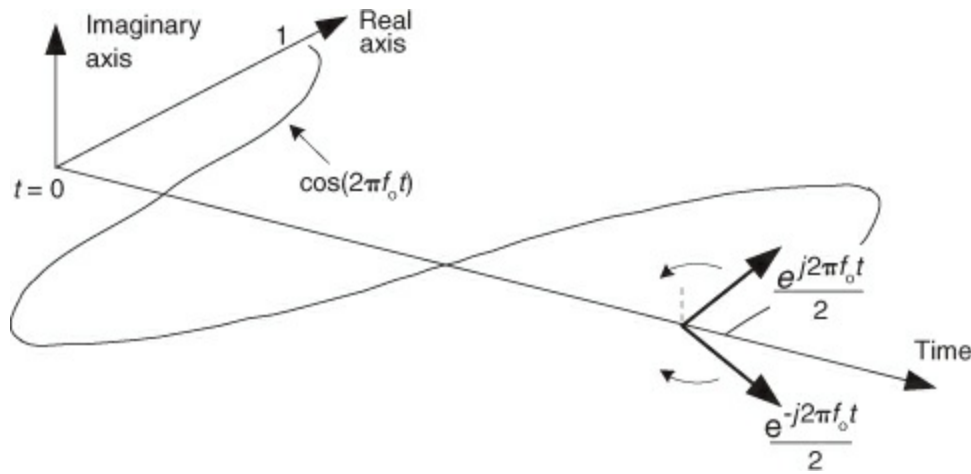
Next, what would be seen on the scope's display if the cables were mislabeled and the two signals were inadvertently swapped? We'd see another circle, but this time it would be orbiting in a clockwise direction. This would be a neat little real-world demonstration if we set the signal generators' f_0 frequencies to, say, 1 Hz.

This oscilloscope example is meaningful and helps us answer the important question “When we work with quadrature signals, how is the j -operator implemented in hardware?” The j -operator is implemented by how we treat the two signals relative to each other. We have to treat them orthogonally such that the real $\cos(2\pi f_0 t)$ signal represents an east-west value, and the real $\sin(2\pi f_0 t)$ signal represents an orthogonal north-south value. (By “orthogonal,” I mean the north-south direction is oriented exactly 90 degrees relative to the east-west direction.) So in our oscilloscope example the j -operator is implemented merely by how the connections are made to the scope. The real cosine signal controls horizontal deflection and the real sine signal controls vertical deflection. The result is a two-dimensional quadrature signal represented by the instantaneous position of the dot on the scope's display. We physically implemented the j -operator in $e^{j2\pi f_0 t} = \cos(2\pi f_0 t) + j\sin(2\pi f_0 t)$ the moment we connected the $\sin(2\pi f_0 t)$ signal to the vertical input connector of the oscilloscope. Our [Figure 8-7](#) example reminds us of an important characteristic of quadrature signals: While real signals can be transmitted over a single cable, two cables are always necessary to transmit a quadrature (complex) signal.

Returning to [Figure 8-5\(b\)](#), ask yourself: “What's the vector sum of those two phasors as they rotate in opposite directions?” Think about this for a moment. That's right, the phasors' real parts will always add constructively, and their imaginary parts will always cancel. This means the summation of these $e^{j2\pi f_0 t}$ and $e^{-j2\pi f_0 t}$ phasors will always be a purely real number. Implementations of modern-day digital communications systems are based on this property!

To emphasize the importance of the real sum of these two complex sinusoids we'll draw yet another picture. Consider the waveform in the three-dimensional [Figure 8-8](#) generated by the sum of two half-magnitude complex phasors, $e^{j2\pi f_0 t}/2$ and $e^{-j2\pi f_0 t}/2$, rotating in opposite directions about, and moving down along, the time axis.

Figure 8-8 A cosine represented by the sum of two rotating complex phasors.



Thinking about these phasors, it's clear now why the cosine wave can be equated to the sum of two complex exponentials by

(8-13)

$$\cos(2\pi f_0 t) = \frac{e^{j2\pi f_0 t} + e^{-j2\pi f_0 t}}{2} = \frac{e^{j2\pi f_0 t}}{2} + \frac{e^{-j2\pi f_0 t}}{2}.$$

Eq. (8-13), a well-known and important expression, is also one of Euler's identities. We could have derived this identity by solving Eqs. (8-7) and (8-8) for $j\sin(\theta)$, equating those two expressions, and solving that final equation for $\cos(\theta)$. Similarly, we could go through the same algebra exercise and show a real sinewave as also the sum of two complex exponentials as

(8-14)

$$\sin(2\pi f_0 t) = \frac{e^{j2\pi f_0 t} - e^{-j2\pi f_0 t}}{2j} = \frac{je^{-j2\pi f_0 t}}{2} - \frac{je^{j2\pi f_0 t}}{2}.$$

Look at Eqs. (8-13) and (8-14) carefully—they are the standard expressions for a cosine wave and a sinewave, using complex notation, and are seen throughout the literature of quadrature communications systems. Equation (8-13) tells us that the two complex exponentials are both oriented toward the positive real axis when time $t = 0$. The j operators in Eq. (8-14) tell us that the negative-frequency complex exponential is oriented along the positive imaginary axis, and the positive-frequency complex exponential is oriented along the negative imaginary axis, when time $t = 0$.

To keep the reader's mind from spinning like those complex phasors, please realize that the sole purpose of Figures 8-5 through 8-8 is to validate the complex expressions of the cosine and sinewave given in Eqs. (8-13) and (8-14). Those two equations, along with Eqs. (8-7) and (8-8), are the Rosetta Stone of quadrature signal processing.[‡] We can now easily translate, back and forth, between real sinusoids and complex exponentials.

[‡] The Rosetta Stone was a basalt slab found in Egypt in 1799. It had the same text written in three languages, two of them being Greek and Egyptian hieroglyphs. This enabled scholars to, finally, translate the ancient hieroglyphs.

Let's step back now and remind ourselves what we're doing. We are learning how real signals that can be transmitted down a coax cable, or digitized and stored in a computer's memory, can be represented in complex number notation. Yes, the constituent parts of a complex number are each real, but we're treating those parts in a special way—we're treating them in quadrature.

8.4 A Few Thoughts on Negative Frequency

It's important for us to be comfortable with the concept of negative frequency because it's essential in understanding the spectral replication effects of periodic sampling, discrete Fourier transforms, and the various quadrature signal processing techniques discussed in [Chapter 9](#). The convention of negative frequency serves as both a consistent and powerful mathematical tool in our analysis of signals. In fact, the use of negative frequency is mandatory when we represent *real signals*, such as a sine or cosine wave, in complex notation.

The difficulty in grasping the idea of negative frequency may be, for some, similar to the consternation felt in the parlors of mathematicians in the Middle Ages when they first encountered negative numbers. Until the thirteenth century, negative numbers were considered *fictitious* because numbers were normally used for counting and measuring. So up to that time, negative numbers just didn't make sense. In those days, it was valid to ask, "How can you hold in your hand something that is less than nothing?" The idea of subtracting six from four must have seemed meaningless. Math historians suggest that negative numbers were first analyzed in Italy. As the story goes, around the year 1200 the Italian mathematician Leonardo da Pisa (known as Fibonacci) was working on a financial problem whose only valid solution involved a negative number. Undaunted, Leo wrote, "This problem, I have shown to be insoluble unless it is conceded that the first man had a debt." Thus negative numbers arrived on the mathematics scene, never again to be disregarded.

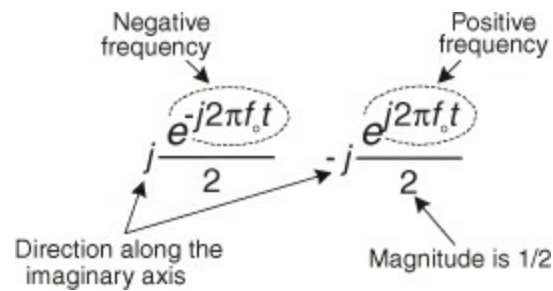
Modern men and women can now appreciate that negative numbers have a direction associated with them. The direction is backward from zero in the context that positive numbers point forward from zero. For example, negative numbers can represent temperatures measured in degrees below zero, minutes before the present if the present is considered as zero time, or money we owe the tax collector when our income is considered positive dollars. So, the notion of negative quantities is perfectly valid if we just define it properly. As comfortable as we now are with negative numbers, negative frequency remains a troublesome and controversial concept for many engineers[3,4]. This author once encountered a paper in a technical journal which stated: "since negative frequencies cannot exist—." Well, like negative numbers, negative frequency is a perfectly valid concept as long as we define it properly relative to what we're used to thinking of as positive frequency. With this thought in mind, we'll call [Figure 8-5](#)'s $e^{j2\pi f_0 t}$ signal a *positive-frequency* complex exponential because it rotates around the complex plane's origin in a circle in a positive-angle direction at a cyclic frequency of f_0 cycles per second. Likewise, we'll refer to the $e^{-j2\pi f_0 t}$ signal as a *negative-frequency* complex exponential because of its negative-angle direction of rotation.

So we've defined negative frequency in the frequency domain. If my DSP pals want to claim negative frequency doesn't exist in the time domain, I won't argue. However, our frequency-domain negative frequency definition is clean, consistent with real signals, very useful, and here to stay.

8.5 Quadrature Signals in the Frequency Domain

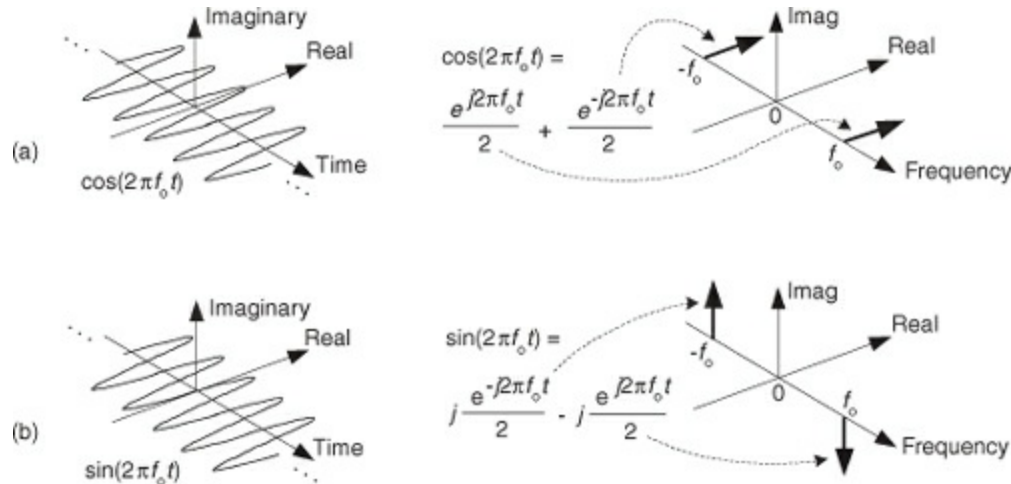
Now that we know much about the time-domain nature of quadrature signals, we're ready to look at their frequency-domain descriptions. We'll illustrate the full three-dimensional aspects of the frequency domain so none of the phase relationships of our quadrature signals will be hidden from view. [Figure 8-9](#) tells us the rules for representing complex exponentials in the frequency domain.

Figure 8-9 Frequency-domain interpretation of complex exponentials.



We'll represent a single complex exponential as a narrow impulse located at the frequency specified in the exponent. In addition, we'll show the phase relationships between those complex exponentials along the real and imaginary frequency-domain axes. To illustrate those phase relationships, a complex frequency domain representation is necessary. With all this said, take a look at [Figure 8-10](#).

Figure 8-10 Complex time- and frequency-domain representations: (a) cosine wave; (b) a sinewave.



See how a real cosine wave and a real sinewave are depicted in our complex frequency-domain representation on the right side of [Figure 8-10](#). Those bold arrows on the right of [Figure 8-10](#) are not rotating phasors but are frequency-domain impulse symbols indicating a single spectral line for a single complex exponential such as $e^{j2\pi f_0 t}$. The directions in which the spectral impulses are pointing merely indicate the relative phases of the spectral components. The amplitude of those spectral impulses is 1/2. Notice how the spectrum of $\cos(2\pi f_0 t)$ is real-only. That's because $\cos(2\pi f_0 t)$ is an *even* function in time, its value at negative time t is equal to its value at positive time t , or

(8-15)

$$\cos[2\pi f_0(-t)] = \cos(2\pi f_0 t).$$

The $\sin(2\pi f_0 t)$ function, on the other hand, has an imaginary-only spectrum because it's an *odd* function. An odd function's value at negative time t is equal to the negative of its value at positive time t , or

(8-16)

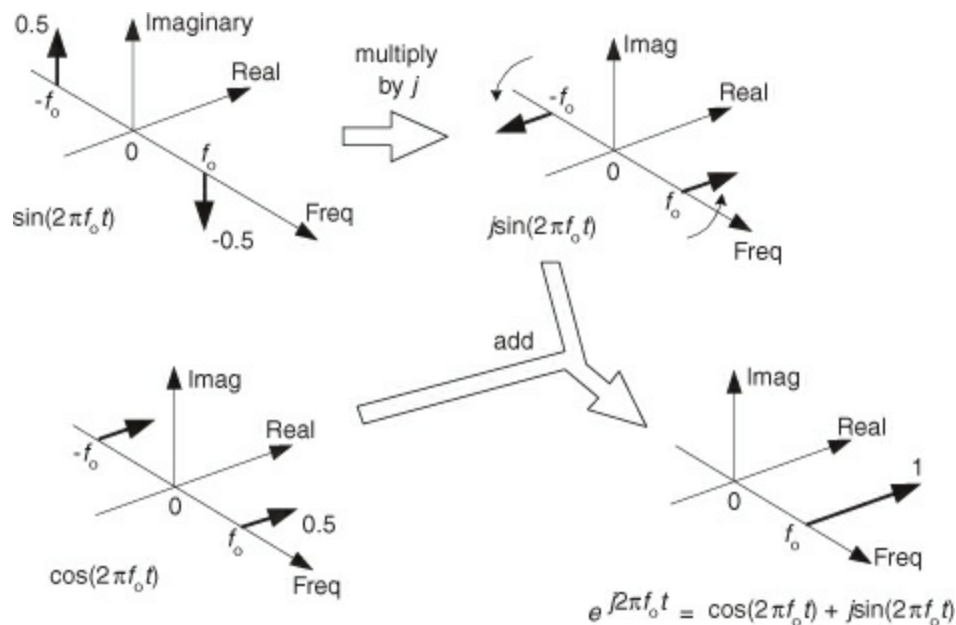
$$\sin[2\pi f_0(-t)] = -\sin(2\pi f_0 t).$$

Why are we bothering with this three-dimensional frequency-domain representation? Because it's the tool we'll use to understand the generation (modulation) and detection (demodulation) of quadrature signals in digital (and some analog) communications systems, and that's one of the goals of this chapter. Before we go there, however, let's validate this frequency-domain representation with a little example.

[Figure 8-11](#) is a straightforward example of how we use the complex frequency domain. There we

begin with a real sinewave, multiply it by j , and then add it to a real cosine wave of the same frequency. The result is the single complex exponential $e^{j2\pi f_0 t}$, illustrating graphically Euler's identity that we stated mathematically in [Eq. \(8-7\)](#).

Figure 8-11 Complex frequency-domain view of Euler's $e^{j2\pi f_0 t} = \cos(2\pi f_0 t) + j\sin(2\pi f_0 t)$.

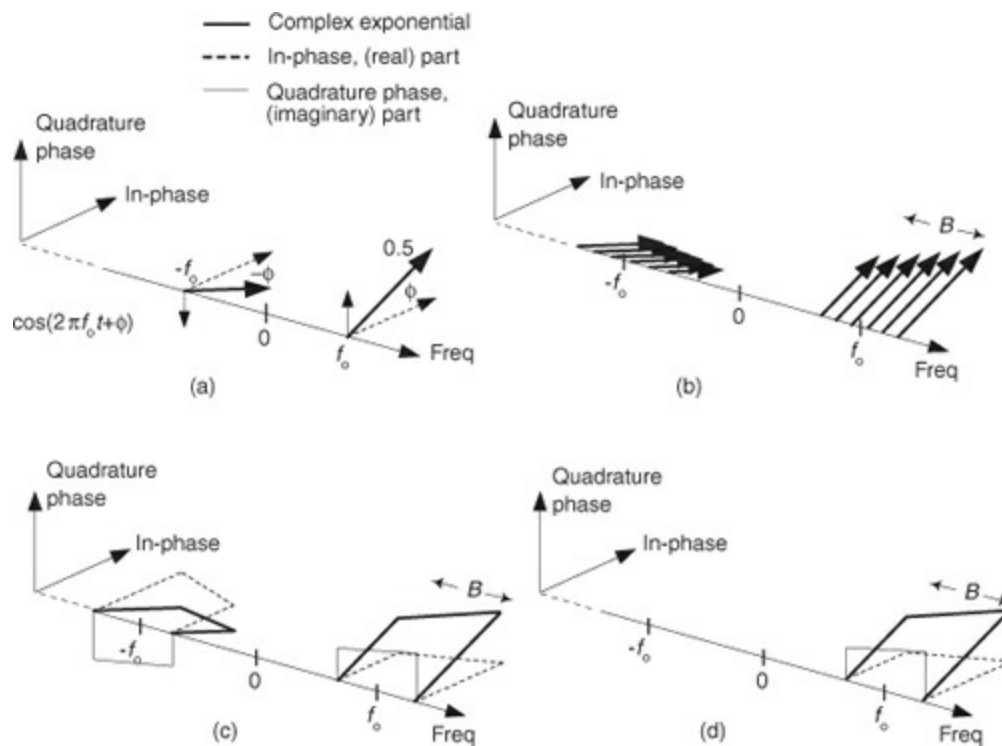


On the frequency axis, the notion of negative frequency is seen as those spectral impulses located at $-2\pi f_0$ radians/second on the frequency axis. This figure shows the big payoff: when we use complex notation, generic complex exponentials like $e^{j2\pi f t}$ and $e^{-j2\pi f t}$ are the fundamental constituents of the real sinusoids $\sin(2\pi f t)$ or $\cos(2\pi f t)$. That's because both $\sin(2\pi f t)$ and $\cos(2\pi f t)$ are made up of $e^{j2\pi f t}$ and $e^{-j2\pi f t}$ components. If you were to take the discrete Fourier transform (DFT) of discrete time-domain samples of a $\sin(2\pi f_0 t)$ sinewave, a $\cos(2\pi f_0 t)$ cosine wave, or an $e^{j2\pi f_0 t}$ complex sinusoid and plot the complex results, you'd get exactly the narrow frequency-domain impulses in [Figure 8-11](#). If you understand the notation and operations in [Figure 8-11](#), pat yourself on the back, because you now know a great deal about the nature and mathematics of quadrature signals.

8.6 Bandpass Quadrature Signals in the Frequency Domain

In quadrature processing, by convention, the real part of the spectrum is called the *in-phase component* and the imaginary part of the spectrum is called the *quadrature component*. The signals whose complex spectra are in [Figures 8-12\(a\)](#), [8-12\(b\)](#), and [8-12\(c\)](#) are real, and in the time domain they can be represented by amplitude values having nonzero real parts and zero-valued imaginary parts. We're not forced to use complex notation to represent them in the time domain—the signals are real-only.

Figure 8-12 Quadrature representation of signals: (a) real sinusoid $\cos(2\pi f_0 t + \phi)$; (b) real bandpass signal containing six sinusoids over bandwidth B ; (c) real bandpass signal containing an infinite number of sinusoids over bandwidth B Hz; (d) complex bandpass signal of bandwidth B Hz.



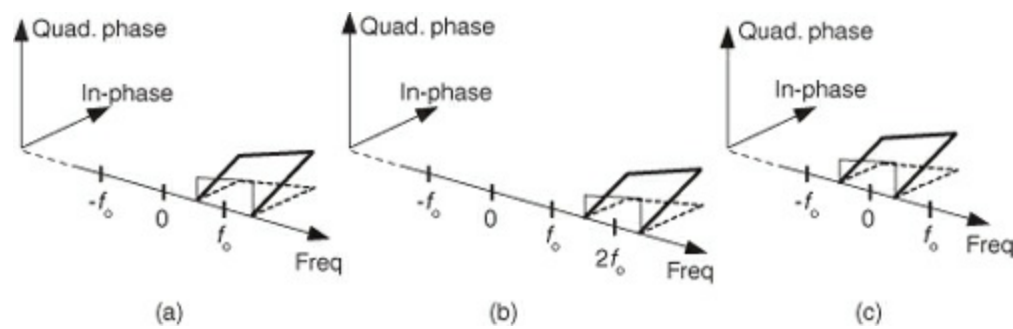
Real signals always have positive- and negative-frequency-spectral components. For any real signal, the positive- and negative-frequency components of its in-phase (real) spectrum always have even symmetry around the zero-frequency point. That is, the in-phase part's positive- and negative-frequency components are mirror images of each other. Conversely, the positive- and negative-frequency components of its quadrature (imaginary) spectrum are always negatives of each other. This means that the phase angle of any given positive quadrature frequency component is the negative of the phase angle of the corresponding negative quadrature frequency component as shown by the thin solid arrows in [Figure 8-12\(a\)](#). This *conjugate symmetry* is the invariant nature of real signals and is obvious when their spectra are represented using complex notation.

A complex-valued time signal, whose spectrum can be that in [Figure 8-12\(d\)](#), is not restricted to the above spectral conjugate symmetry conditions. We'll call that special complex signal an *analytic signal*, signifying that it has no negative-frequency spectral components.

Let's remind ourselves again: those bold arrows in [Figures 8-12\(a\)](#) and [8-12\(b\)](#) are not rotating phasors. They're frequency-domain impulses indicating a single complex exponential $e^{j2\pi f t}$. The directions in which the impulses are pointing show the relative phases of the spectral components.

There's an important principle to keep in mind before we continue. Multiplying a time signal by the complex exponential $e^{j2\pi f_0 t}$, what we call *quadrature mixing* (also called *complex mixing*), shifts a signal's spectrum upward in frequency by f_0 Hz, as shown in [Figures 8-13\(a\)](#) and [8-13\(b\)](#). Likewise, multiplying a time signal by $e^{-j2\pi f_0 t}$ (also called *complex down-conversion* or *mixing to baseband*) shifts a signal's spectrum down to a center frequency of zero Hz as shown in [Figure 8-13\(c\)](#). The process of quadrature mixing is used in many DSP applications as well as most modern-day digital communications systems.

Figure 8-13 Quadrature mixing of a bandpass signal: (a) spectrum of a complex signal $x(t)$; (b) spectrum of $x(t)e^{j2\pi f_0 t}$; (c) spectrum of $x(t)e^{-j2\pi f_0 t}$.

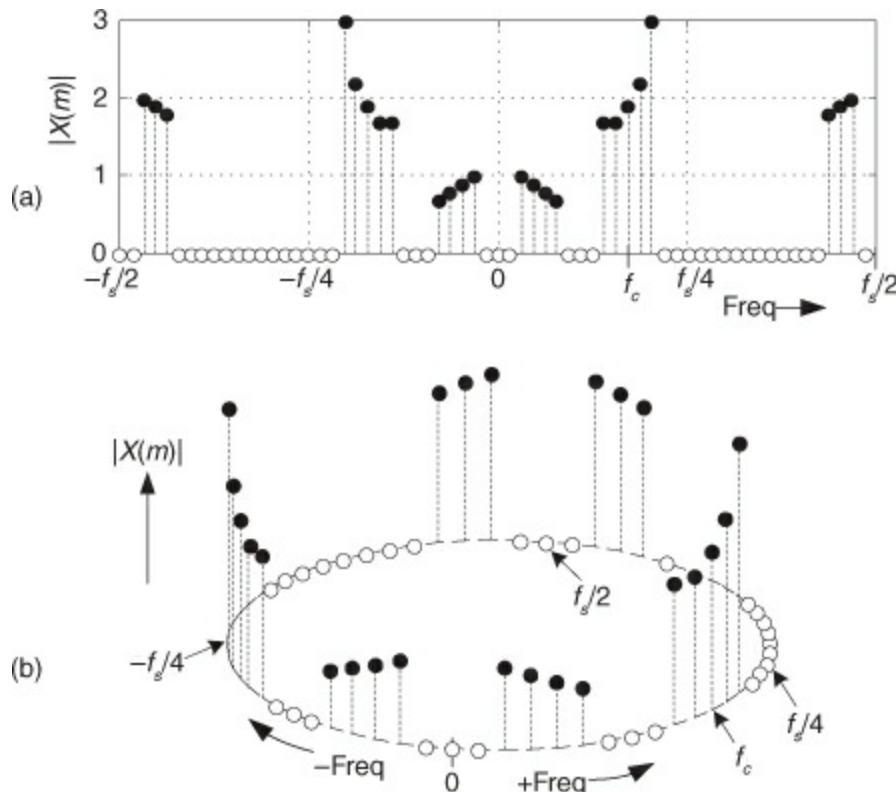


Our entire quadrature signals discussion, thus far, has been based on continuous signals, but the principles described here apply equally well to discrete-time signals. Let's look at the effect of complex down-conversion of a discrete signal's spectrum.

8.7 Complex Down-Conversion

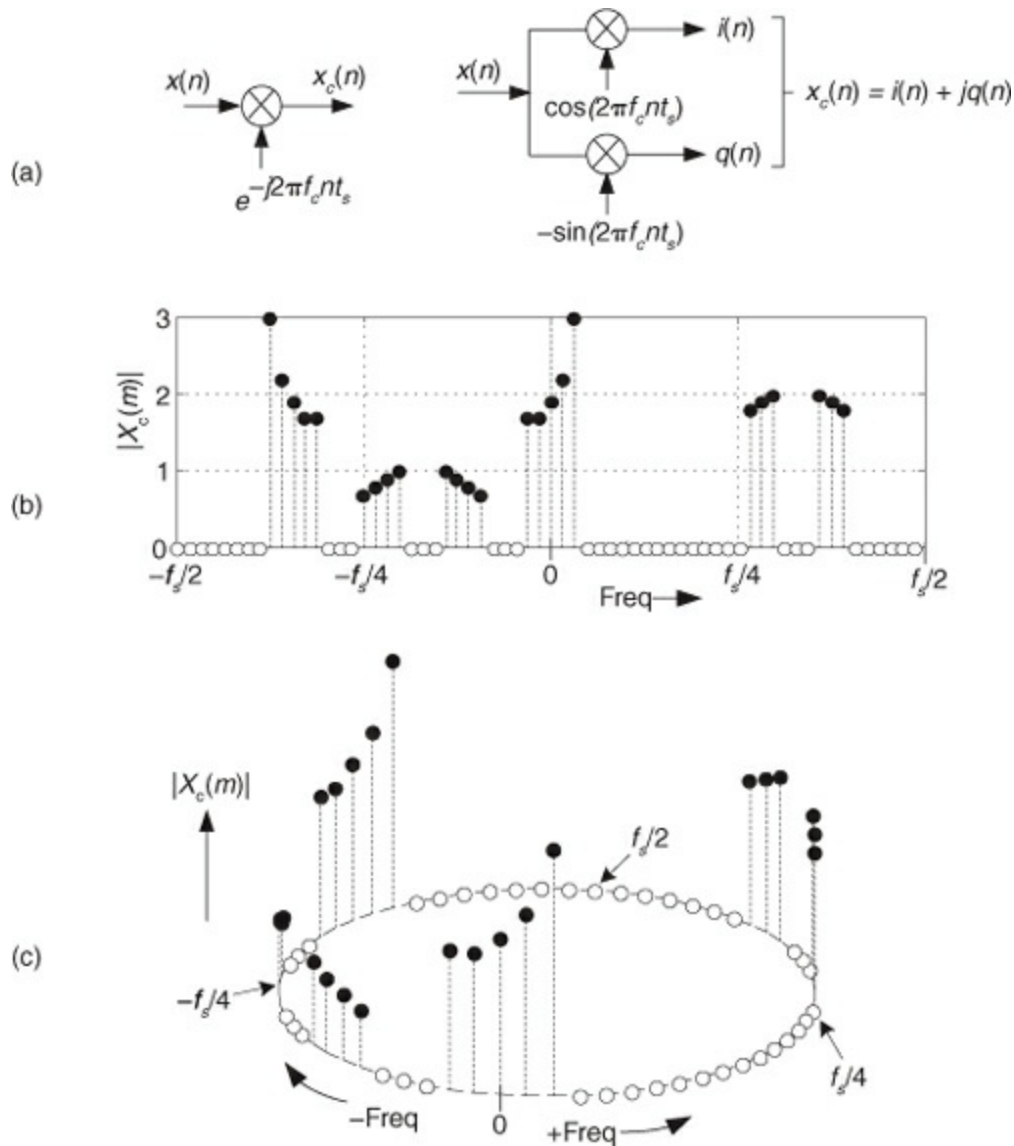
Complex down-conversion (also called *quadrature demodulation*) of discrete signals is a straightforward process and is best described by an example. Think of a real-valued discrete sequence $x(n)$ having an $|X(m)|$ spectral magnitude whose nonzero-valued samples are shown as the solid dots in [Figure 8-14\(a\)](#). Because of the periodicity of discrete spectral representations we discussed in [Sections 2.1](#) and [3.17](#) (as well as the frequency axis of the FFT lying on the unit circle in the z -plane explained in [Section 6.3](#)), we're justified in also representing the $|X(m)|$ spectrum as the three-dimensional circular plot given in [Figure 8-14\(b\)](#). There we've wrapped the linear frequency axis of [Figure 8-14\(a\)](#) around a circle whose perimeter length is equal to the sample rate f_s such that the frequencies $f_s/2$ and $-f_s/2$ are the same point on the circular axis.

Figure 8-14 Discrete $|X(m)|$ spectra of a real-valued time sequence: (a) traditional depiction; (b) circular frequency axis depiction.



With $x(n)$ being an N -point real sequence, $|X(m)|$'s spectrum is symmetrical about the zero-frequency point. If we now perform complex down-conversion (by multiplying $x(n)$ by $e^{-j2\pi f_c n t_s}$, where $t_s = 1/f_s$, using either equivalent scheme shown in [Figure 8-15\(a\)](#)), the result is the complex sequence

Figure 8-15 Discrete $|X_c(m)|$ spectra of a down-converted time sequence: (a) down-conversion symbols; (b) traditional frequency axis depiction; (c) circular frequency axis depiction.



(8-17)

$$x_c(n) = x(n)e^{-j2\pi f_c n t_s} = i(n) + jq(n)$$

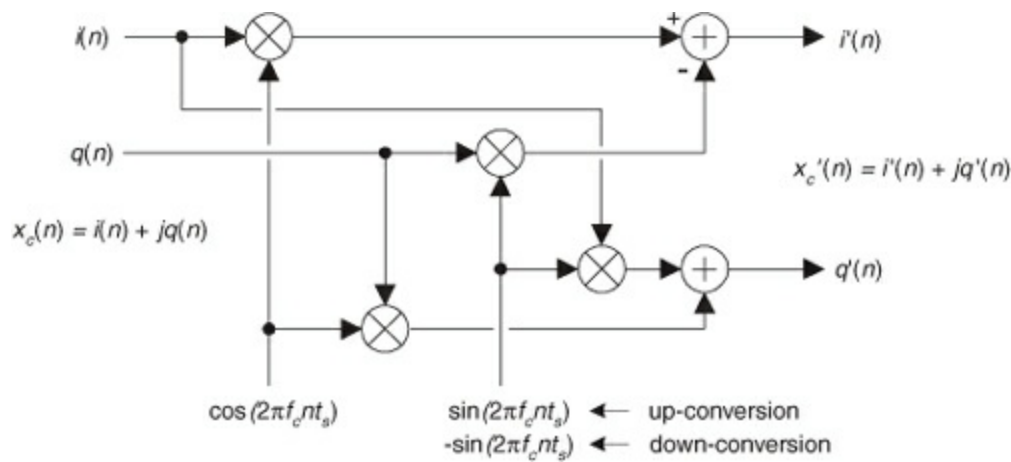
whose spectrum is given in [Figure 8-15\(b\)](#).

The minus sign in the exponent of $e^{-j2\pi f_c n t_s}$ shifts the $|X(m)|$ spectrum f_c Hz in the negative-frequency direction. Of course, because $x_c(n)$ is complex, there's no symmetry about the zero-frequency point in $|X_c(m)|$. The circular depiction of $|X_c(m)|$ is provided in [Figure 8-15\(c\)](#).

The purpose of [Figures 8-14](#) and [8-15](#) is to show how frequency translation by means of complex down-conversion causes spectral components to wrap around the $f_s/2$ point.

[Figure 8-15\(a\)](#) showed the method of down-converting a real $x(n)$ time sequence. For completeness, [Figure 8-16](#) shows how translating a complex time sequence $x_c(n) = i(n) + jq(n)$ up or down by f_c Hz requires a complex multiplier.

Figure 8-16 Complex multiplier used for up/down-conversion.



This complex multiplier computes

$$(8-18)$$

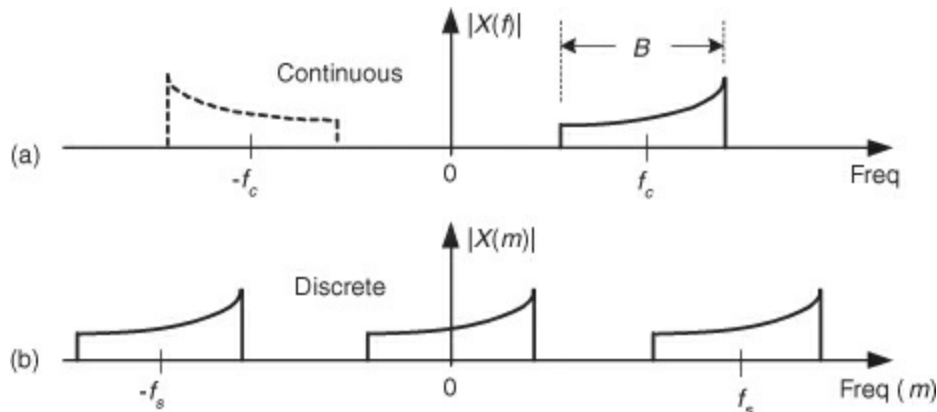
$$i'(n) + jq'(n) = x_c(n)e^{\pm j2\pi f_c n t_s} = [i(n) + jq(n)][\cos(2\pi f_c n t_s) \pm j\sin(2\pi f_c n t_s)].$$

If you use this multiplier, don't forget the minus sign at the top adder in [Figure 8-16](#). (It's an easy mistake to make. Believe me.)

8.8 A Complex Down-Conversion Example

We can use all we've learned so far about quadrature signals by exploring the process of quadrature sampling. Quadrature sampling is the process of digitizing a continuous (analog) bandpass signal and down-converting its spectrum to be centered at zero Hz. Let's see how this popular process works by thinking of a continuous bandpass signal, of bandwidth B , centered about a carrier frequency of f_c Hz as shown in [Figure 8-17\(a\)](#).

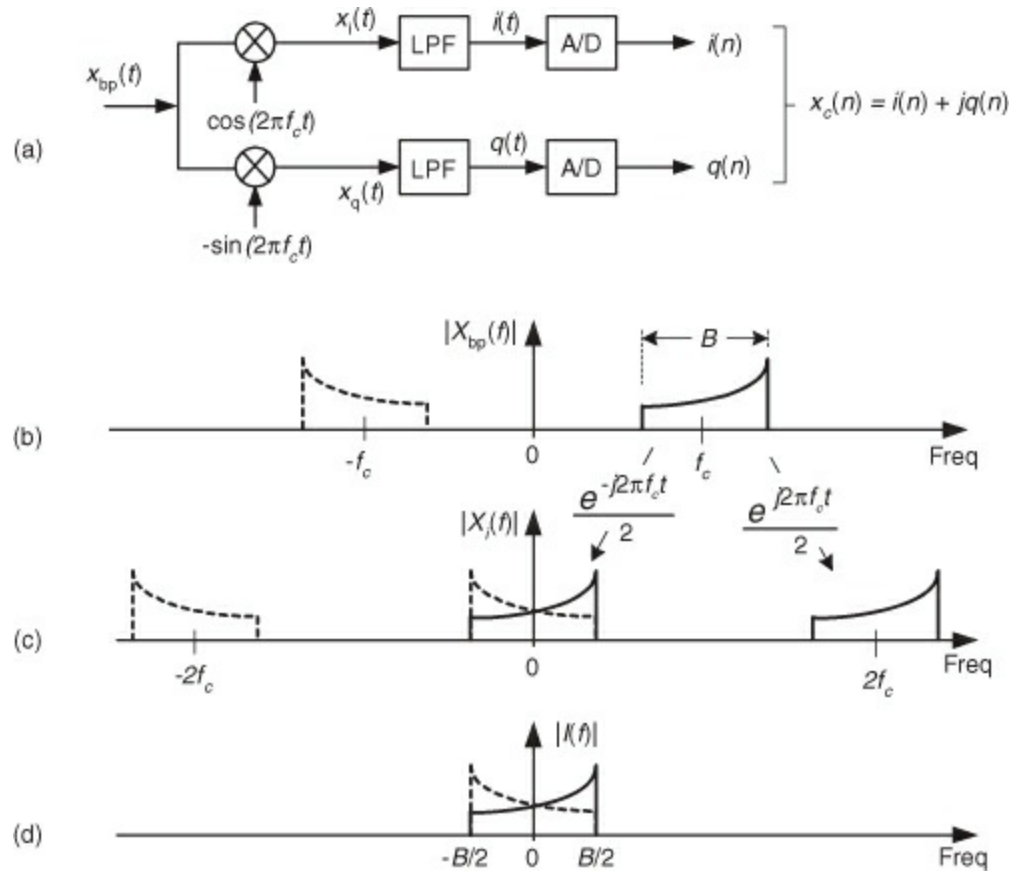
Figure 8-17 The “before and after” spectra of a quadrature-sampled signal.



Our goal in quadrature sampling is to obtain a digitized version of the analog bandpass signal, but we want the digitized signal's discrete spectrum centered about zero Hz, not f_c Hz, as in [Figure 8-17\(b\)](#). That is, we want to mix a time signal with $e^{-j2\pi f_c t}$ to perform complex down-conversion. The frequency f_s is the digitizer's sampling rate in samples/second. We show replicated spectra in [Figure 8-17\(b\)](#) to remind ourselves of this effect when A/D conversion takes place.

We can solve our sampling problem with the quadrature sampling block diagram (also known as *I/Q demodulation*) shown in [Figure 8-18\(a\)](#). That arrangement of two sinusoidal oscillators, with their relative 90-degree phase, is often called a *quadrature oscillator*. First we'll investigate the in-phase (upper) path of the quadrature sampler. With the input analog $x_{bp}(t)$'s spectrum shown in [Figure 8-18\(b\)](#), the spectral output of the top mixer is provided in [Figure 8-18\(c\)](#).

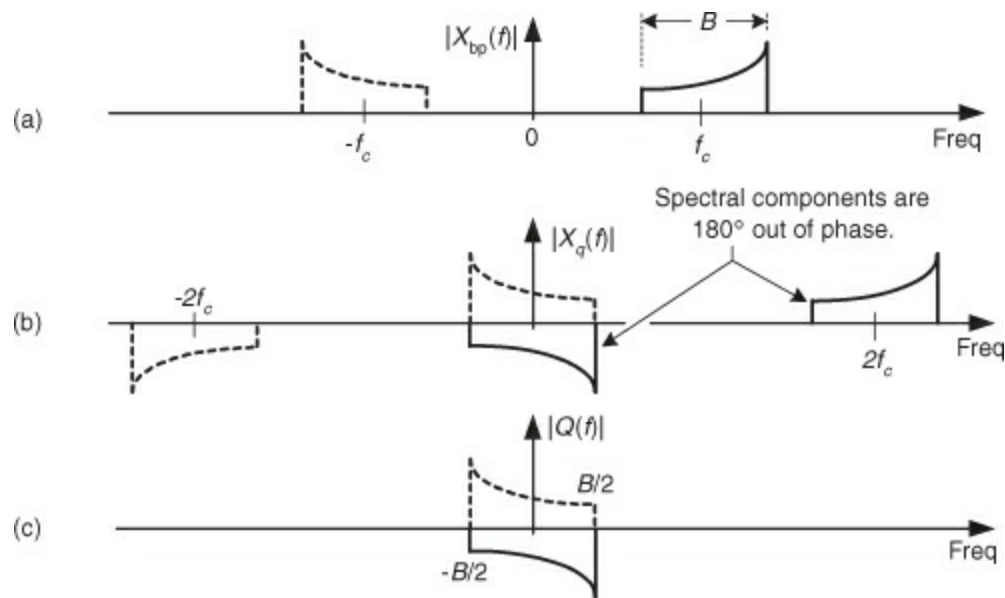
Figure 8-18 Quadrature sampling: (a) block diagram; (b) input spectrum; (c) in-phase mixer output spectrum; (d) in-phase filter output spectrum.



Those $e^{j2\pi f_c t}$ and $e^{-j2\pi f_c t}$ terms in [Figure 8-18](#) remind us, from [Eq. \(8-13\)](#), that the constituent complex exponentials comprise a real cosine duplicate and translate each part of $|X_{bp}(f)|$'s spectrum to produce the $|X_i(f)|$ spectrum. There is a magnitude loss of a factor of two in $|X_i(f)|$, but we're not concerned about that at this point. [Figure 8-18\(d\)](#) shows the output of the lowpass filter (LPF) in the in-phase path.

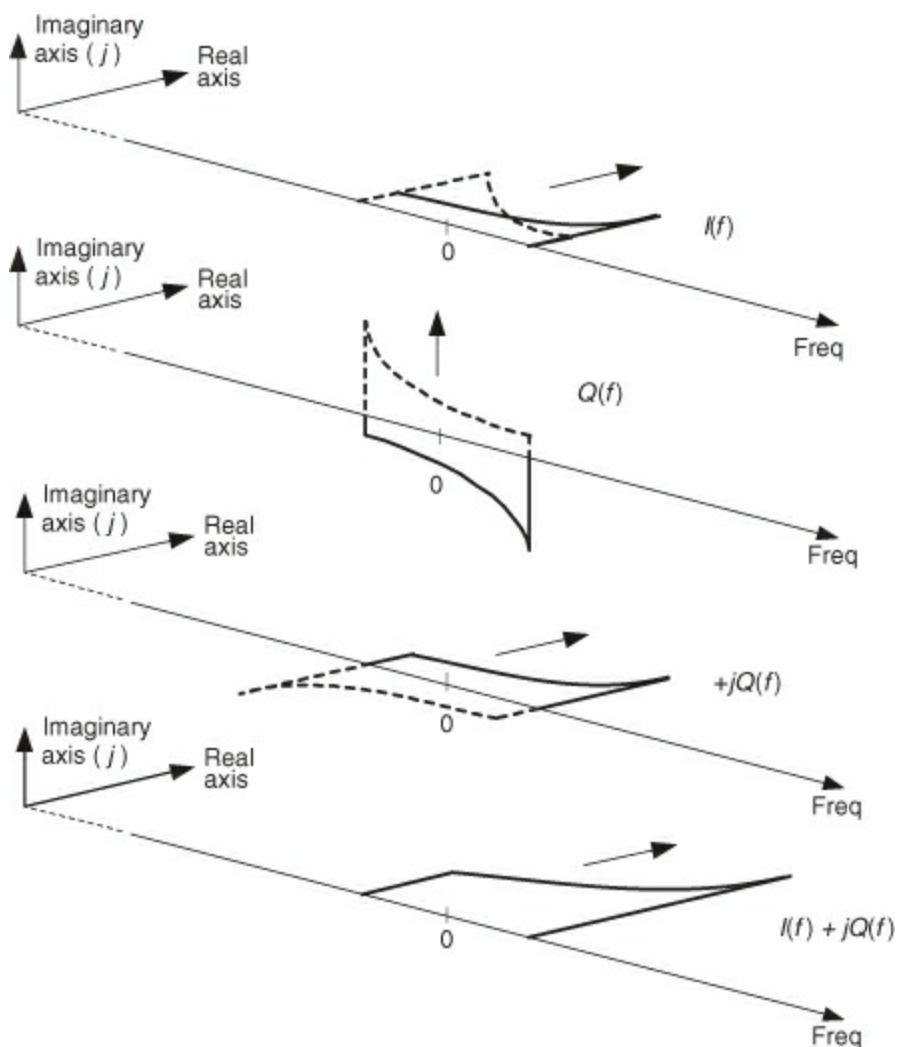
Likewise, [Figure 8-19](#) shows how we get the filtered continuous quadrature-phase portion (bottom path) of our desired complex signal by mixing $x_{bp}(t)$ with $-\sin(2\pi f_c t)$. From [Eq. \(8-14\)](#) we know that the complex exponentials comprising the real $-\sin(2\pi f_c t)$ sinewave are $e^{j2\pi f_c t}$ and $-e^{-j2\pi f_c t}$. The minus sign in the $-e^{-j2\pi f_c t}$ term accounts for the down-converted spectra in $|X_q(f)|$ being 180 degrees out of phase with the up-converted spectra.

Figure 8-19 Spectra within the quadrature phase (lower) signal path of the block diagram.



This depiction of quadrature sampling can be enhanced if we look at the situation from a three-dimensional standpoint, as in [Figure 8-20](#). There the $+j$ factor rotates the “imaginary-only” $Q(f)$ by 90 degrees, making it “real-only.” This $jQ(f)$ is then added to $I(f)$ to yield the spectrum of a complex continuous signal $x(t) = i(t) + jq(t)$. Applying this signal to two A/D converters gives our final desired discrete time samples of $x_c(n) = i(n) + jq(n)$ in [Figure 8-18\(a\)](#) having the spectrum shown in [Figure 8-17\(b\)](#).

Figure 8-20 Three-dimensional view of combining the $I(f)$ and $Q(f)$ spectra to obtain the $I(f) + jQ(f)$ spectra.

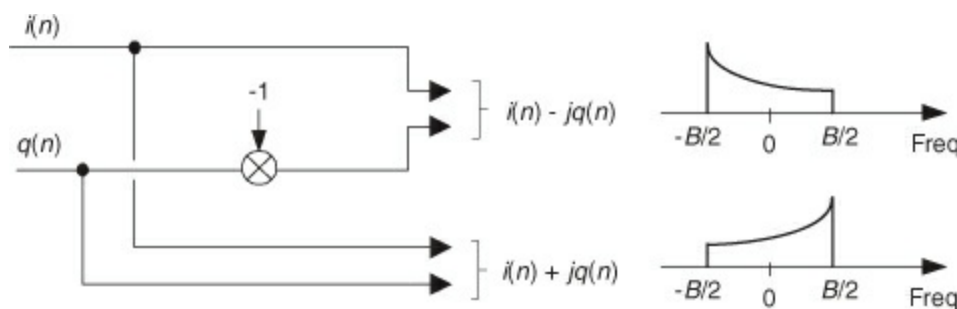


Some advantages of this quadrature sampling scheme are:

- Each A/D converter operates at half the sampling rate of standard real-signal sampling.
- In many hardware implementations, operating at lower clock rates saves power.
- For a given f_s sampling rate, we can capture wider-band analog signals.
- Quadrature sequences make FFT processing more efficient due to a wider frequency range coverage.
- Quadrature sampling also makes it easier to measure the instantaneous magnitude and phase of a signal during demodulation.
- Knowing the instantaneous phase of signals enables coherent processing.

While the quadrature sampler in [Figure 8-18\(a\)](#) performed complex down-conversion, it's easy to implement complex up-conversion by merely conjugating the $x_c(n)$ sequence, effectively inverting $x_c(n)$'s spectrum about zero Hz, as shown in [Figure 8-21](#).

Figure 8-21 Using conjugation to control spectral orientation.

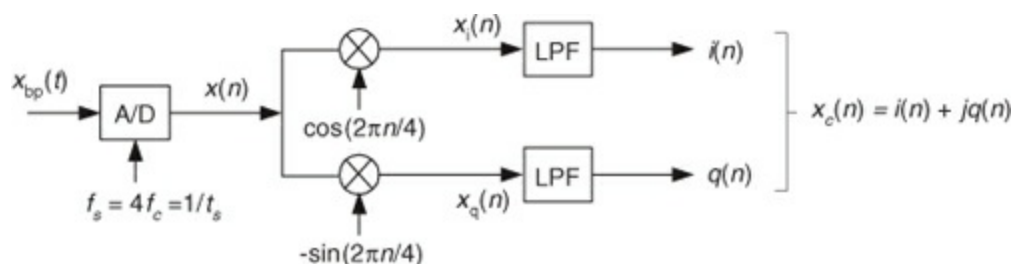


8.9 An Alternate Down-Conversion Method

The quadrature sampling method of complex down-conversion in [Figure 8-18\(a\)](#) works perfectly on paper, but it's difficult to maintain the necessary exact 90-degree phase relationships with high-frequency, or wideband, signals in practice. One- or two-degree phase errors are common in the laboratory. Ideally, we'd need perfectly phase-matched coax cables, two oscillators exactly 90 degrees out of phase, two ideal mixers with identical behavior and no DC output component, two analog lowpass filters with identical magnitude and phase characteristics, and two A/D converters with exactly identical performance. (Sadly, no such electronic components are available to us.) Fortunately, there's an easier-to-implement quadrature sampling method[\[5\]](#).

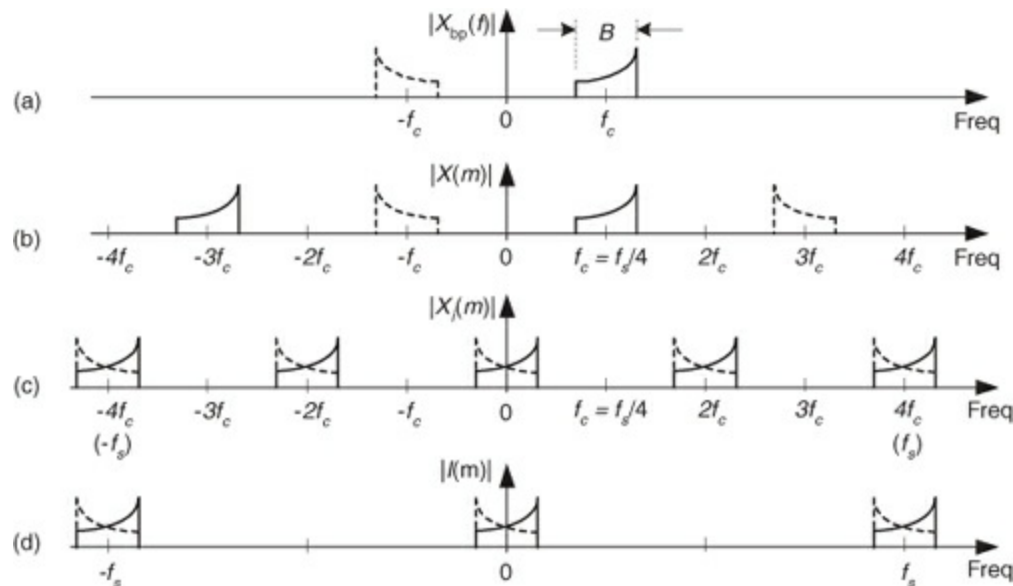
Consider the process shown in [Figure 8-22](#), where the analog $x_{bp}(t)$ signal is initially digitized with the follow-on mixing and filtering being performed digitally. This *quadrature sampling with digital mixing* method mitigates the problems with the [Figure 8-18\(a\)](#) quadrature sampling method and eliminates one of the A/D converters.

Figure 8-22 Quadrature sampling with digital mixing method.



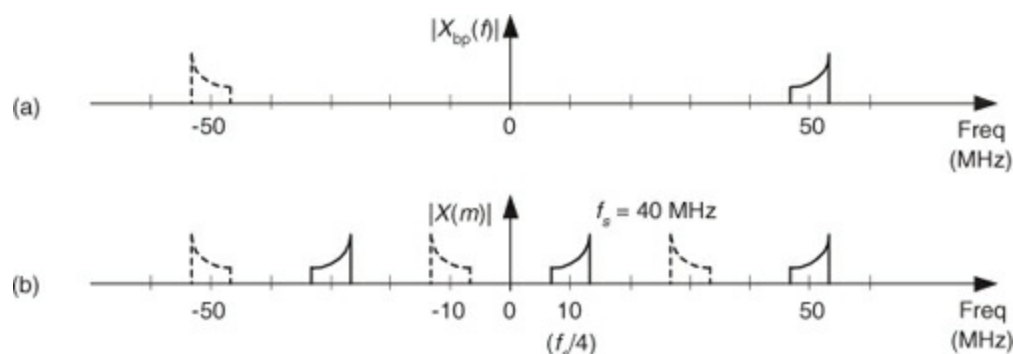
We use [Figure 8-23](#) to show the spectra of the in-phase path of this quadrature sampling with digital mixing process. Notice the similarity between the continuous $|I(f)|$ in [Figure 8-18\(d\)](#) and the discrete $|I(m)|$ in [Figure 8-23\(d\)](#). A sweet feature of this process is that with $f_c = f_s/4$, the cosine and sine oscillator outputs are the repetitive four-element $\cos(\pi n/2) = 1, 0, -1, 0$, and $-\sin(\pi n/2) = 0, -1, 0, 1$, sequences, respectively. (See [Section 13.1](#) for details of these special mixing sequences.) No actual mixers (or multiplies) are needed to down-convert our desired spectra to zero Hz! After lowpass filtering, the $i(n)$ and $q(n)$ sequences are typically decimated by a factor of two to reduce the data rate for following processing. (Decimation is a topic covered in [Section 10.1](#).)

Figure 8-23 Spectra of quadrature sampling with digital mixing within the in-phase (upper) signal path.



With all its advantages, you should have noticed one drawback of this quadrature sampling with digital mixing process: the f_s sampling rate must be four times the signal's f_c center frequency. In practice, $4f_c$ could be an unpleasantly high value. Happily, we can take advantage of the effects of bandpass sampling to reduce our sample rate. Here's an example: Consider a real analog signal whose center frequency is 50 MHz, as shown in [Figure 8-24\(a\)](#). Rather than sampling this signal at 200 MHz, we perform bandpass sampling and use Eq. (2-13) with $m_{\text{odd}} = 5$ to set the f_s sampling rate at 40 MHz. This forces one of the replicated spectra of the sampled $|X(m)|$ to be centered at $f_s/4$, as shown in [Figure 8-24\(b\)](#), which is what we wanted. The A/D converter $x(n)$ output is now ready for complex down-conversion by $f_s/4$ (10 MHz) and digital lowpass filtering.

Figure 8-24 Bandpass sampling effects used to reduce the sampling rate of quadrature sampling with digital mixing: (a) analog input signal spectrum; (b) A/D converter spectrum.



[Section 13.1](#) provides a clever trick for reducing the computational workload of the lowpass filters in

[Figure 8-22](#), when this $f_s/4$ down-conversion method is used with decimation by two.

References

[1] Struik, D. *A Concise History of Mathematics*, Dover Publications, New York, 1967.
[2] Bergamini, D. *Mathematics*, Life Science Library, Time Inc., New York, 1963.
[3] Lewis, L. J., et al. *Linear Systems Analysis*, McGraw-Hill Inc., New York, 1969, p. 193.
[4] Schwartz, M. *Information, Transmission, Modulation, and Noise*, McGraw-Hill Inc., New York, 1970, p. 35.
[5] Considine, V. “Digital Complex Sampling,” *Electronics Letters*, 19, August 4, 1983.

Chapter 8 Problems

8.1 Consider the following expressions:

- 1. $x + 6 = 0$
- 2. $2x - 7 = 0$
- 3. $x^2 - 2 = 0$
- 4. $x^2 + 2 = 0$

(a) For those expressions, fill in the table below with “Yes” or “No” describing the numerical nature of x .

Hint: A rational number is a number that can be expressed as a fraction p/q where p and q are integers, and $q \neq 0$.

	Real-only	Integer	Rational	Complex
$x + 6 = 0$				
$2x - 7 = 0$				
$x^2 - 2 = 0$				
$x^2 + 2 = 0$				

(b) Write the various values for x in both rectangular and polar (magnitude and phase) notation.

8.2 Because they are so important in understanding quadrature signals and systems, prove the following:

- (a) Multiplying a complex number C by the j -operator rotates that complex number by 90 degrees.
- (b) Multiplying a complex number C by the j -operator results in a number whose magnitude is equal to $|C|$.

8.3 In quadrature systems, we often need to compute the square of a complex time-domain sample’s magnitude in order to estimate instantaneous signal power. Prove that the product of a complex number times its conjugate, CC^* , is equal to the complex number’s magnitude squared.

- (a) Use the rectangular form of $C = M\cos(\phi) + jM\sin(\phi)$ for your proof.
- (b) Use the polar form of $C = Me^{j\phi}$ for your proof.
- (c) Which form of C , rectangular or polar, was easier to use for this proof?

8.4 Consider a complex number C having nonzero real and imaginary parts. If the sum of C plus its reciprocal ($C + 1/C$) is real-only, what can we say about the magnitude of C ? Justify your answer.

8.5 Assume that we have two complex values:

$$C_a = R_a + jI_a,$$

$$C_b = R_b + jI_b.$$

Next, suppose we want to compute a real-valued Q defined as

$$Q = \text{real part of } \left[\frac{C_a}{C_b} \right].$$

For computational efficiency reasons, to avoid unnecessary arithmetic using the imaginary parts of the complex C values, we might decide to compute Q as

$$Q_{\text{efficient}} = \frac{\text{real part of } [C_a]}{\text{real part of } [C_b]}.$$

Show the validity (the correctness) of using $Q_{\text{efficient}}$ to compute our desired Q value.

Hint: [Appendix A](#) will help you solve this problem.

8.6 To show the value of using complex numbers in our derivations of real-valued functions, use the notation

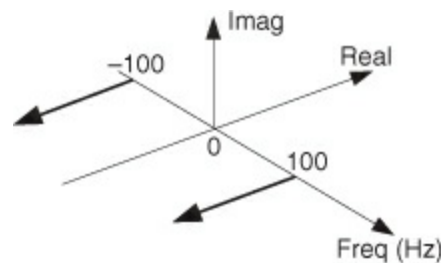
$$\cos(\phi) = \text{Re}\{e^{j\phi}\}$$

where $\text{Re}\{e^{j\phi}\}$ means “the real part of $e^{j\phi}$ ” to prove the following trigonometric identity:

$$\cos(\alpha + \beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta).$$

8.7 Given a continuous single sinusoidal $x(t)$ signal whose three-dimensional spectrum is shown in [Figure P8-7](#), write the time-domain equation for $x(t)$ in trigonometric form. (Assume the magnitudes of the spectral components are 0.5.) Justify your answer.

Figure P8-7



8.8 The sum of two general complex exponentials, of the same frequency, is

$$Ae^{j(\omega t + \alpha)} + Be^{j(\omega t + \beta)} = Me^{j(\omega t + \theta)}$$

where A , B , and M are real-valued constants. Continuous frequency ω is in radians/second, and α , β , and θ are constant phase shifts measured in radians. Find the expressions for M and θ in terms of A , B , α , and β .

Hint: Recall one of the *laws of exponents*: $x^p x^q = x^{p+q}$.

8.9 In your future mathematical work you may need to represent a real tangent function in terms of complex exponentials. Such an expression is

$$\tan(\alpha) = \frac{e^{j\alpha} - e^{-j\alpha}}{j(e^{j\alpha} - e^{-j\alpha})}.$$

Prove that the above $\tan(\alpha)$ equation is correct.

8.10 In the literature of DSP, some authors state that the *combined* frequency magnitude response of two cascaded systems is the product of their individual frequency magnitude responses. That is, $|H_{\text{combined}}(\omega)| = |H_1(\omega)| \cdot |H_2(\omega)|$. Another author might say the *combined* frequency magnitude response of two cascaded systems is equal to the magnitude of the product of their individual frequency responses. That is, $|H_{\text{combined}}(\omega)| = |H_1(\omega)H_2(\omega)|$. Can both statements be correct? We simplify, and restate, this problem as follows: Given two complex numbers, C_1 and C_2 , prove that the product of their magnitudes is equal to the magnitude of their product, i.e.,

$$|C_1| \cdot |C_2| = |C_1 C_2|.$$

8.11 To gain further experience in working with complex numbers, show that:

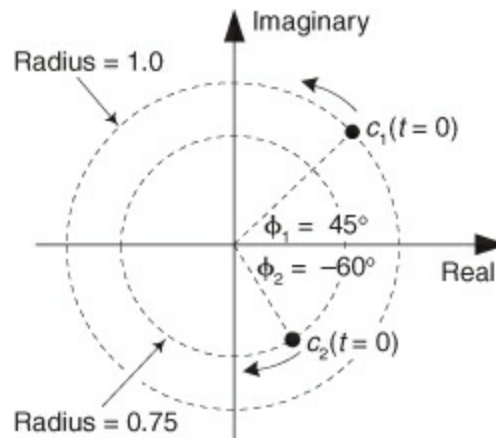
(a) The magnitude of any complex number divided by its conjugate, $|C/C^*|$, is equal to 1 (unity). Assume $C \neq 0$.

(b) The conjugate of the product of two complex numbers is equal to the product of their conjugates. That is:

$$(C_1 \cdot C_2)^* = C_1^* \cdot C_2^*.$$

8.12 Consider the continuous complex exponential signals $c_1(t)$ and $c_2(t)$, shown in [Figure P8-12](#) at the time instant $t = 0$. Signal $c_1(t)$ is orbiting the complex plane's origin counterclockwise at a frequency of 5 Hz, and signal $c_2(t)$ is orbiting the plane's origin clockwise at a frequency of 7 Hz. Write the time-domain equation for the sum of $c_1(t)$ and $c_2(t)$ where the terms in the equation are in polar form having exponents measured in radians.

Figure P8-12



8.13 In the mathematics of complex numbers, and quadrature systems, you may encounter De Moivre's Theorem. (De Moivre is pronounced duh-'mwah-vruh.) That theorem,

$$[\cos(\phi) + j\sin(\phi)]^N = \cos(N\phi) + j\sin(N\phi),$$

is used to compute the result of raising a complex number to some integer (N) power. Prove De Moivre's Theorem.

8.14 Consider the complex time-domain sequence $q(n)$ that is in polar form having a magnitude factor and a phase factor, defined by

$$q(n) = 0.9^n e^{j2\pi n/8}$$

where n is our time-domain index. Starting at time $n = 0$:

- (a) State, in words, how the magnitude of $q(n)$ changes as time index n increases.
- (b) State, in words, how the phase of $q(n)$ changes as n increases.
- (c) State, in words, how the real part of $q(n)$ changes as n increases.
- (d) State, in words, how the imaginary part of $q(n)$ changes as n increases.
- (e) On a complex plane, draw the $q(n)$ sequence for time index $0 \leq n \leq 7$.

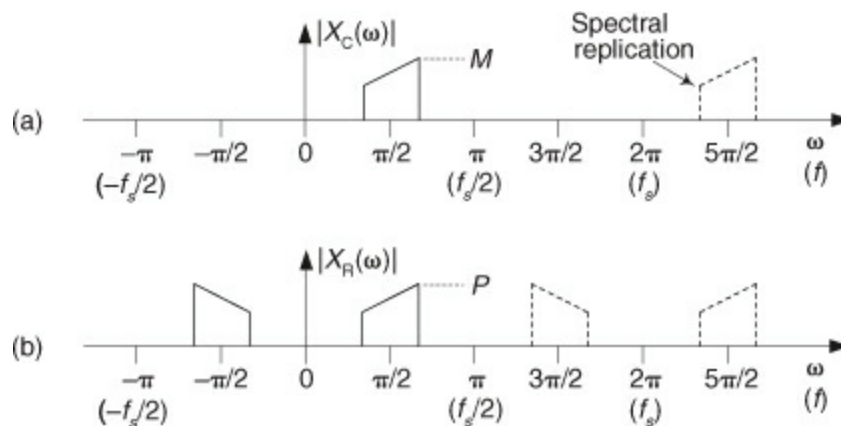
8.15 Consider the real-valued sequence defined by

$$x(n) = \cos(2\pi f_0 n t_s + \pi/4)$$

where $f_0 = -1$ kHz. What is the phase, measured in degrees, of the negative-frequency 1 kHz spectral component of $x(n)$?

8.16 Consider a complex $x_C(n)$ sequence whose $X_C(\omega)$ spectrum is depicted in [Figure P8-16\(a\)](#). The frequency axis in that figure is labeled in both our ω digital frequency (radians/sample) and an f cyclic frequency (Hz). If we take just the real part of $x_C(n)$, we have a real-valued $x_R(n)$ sequence whose $X_R(\omega)$ spectrum is depicted in [Figure P8-16\(b\)](#). If the magnitude of the maximum spectral component of $|X_C(\omega)|$ is M , what is the magnitude of the maximum spectral component of $|X_R(\omega)|$? That is, what is P in terms of M ? Justify your solution.

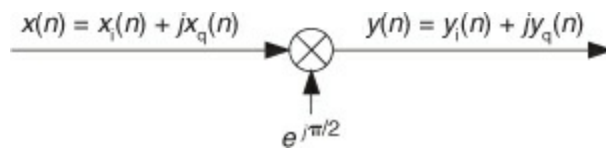
Figure P8-16



Note: This problem is *not* “busy work.” Extracting the real part of a complex time sequence is a common operation in quadrature processing.

8.17 Assume you are required to multiply a complex $x(n)$ sequence by $e^{j\pi/2}$ to obtain the complex $y(n)$ sequence as shown in [Figure P8-17](#). Draw the block diagram, showing real-valued sequences only, that implements the process in [Figure P8-17](#).

Figure P8-17



Hint: Ask yourself, “Factor $e^{j\pi/2}$ is equal to what?”

- 8.18** Many digital communications systems (your cell phone, for example) are designed using the following principle regarding sinusoids whose frequencies are f Hz: The product of

$$\sin(2\pi ft)\sin(2\pi ft + \theta)$$

results in a sinusoid of frequency $2f$ Hz superimposed on a constant DC level whose value is proportional to $\cos(\theta)$. Prove this principle to be true.

Note: The value of this exercise is that we now know how to determine the phase θ of $\sin(2\pi ft + \theta)$ by computing the average of $\sin(2\pi ft)\sin(2\pi ft + \theta)$. Thus if the phase θ of $\sin(2\pi ft + \theta)$ represents some sort of information, we can measure (extract) that information.

- 8.19** Think about programming a DSP chip to generate the complex time-domain sequence, the sum of two complex exponentials that both oscillate through one cycle every 20 samples, defined by

$$x(n) = 5e^{j(2\pi n/20 + \pi/4)} + 3e^{j(2\pi n/20 + \pi/6)}.$$

To optimize your code (reduce its number of computations), you can generate the $x(n)$ sequence using

$$x(n) = Ae^{j(Bn + C)}.$$

To exercise our skills using both the mathematical notation of complex signals and Euler’s equations, determine the values for the above real-only variables A , B , and C .

Hint: Recall one of the *laws of exponents*: $\alpha^p \alpha^q = \alpha^{p+q}$.

- 8.20** Think about the continuous (analog) signal comprising a constant value of one minus a complex sinusoid whose frequency is one Hz defined by

$$x(t) = 1 - e^{j2\pi[1]t}.$$

Draw rough sketches, over the time interval $0 \leq t \leq 1$ second, of

- (a) The real part of $x(t)$,
- (b) The imaginary part of $x(t)$,
- (c) The magnitude of $x(t)$.

- 8.21** In many quadrature processing applications we seek to compute the angle θ of a single complex time-domain sample by using the arctangent operation in the text’s [Eq. \(8-6\)](#). Given the complex number $C=I+jQ=Me^{j\theta}$, three simple algorithms for estimating angle θ are the following polynomials, where $X=Q/I$:

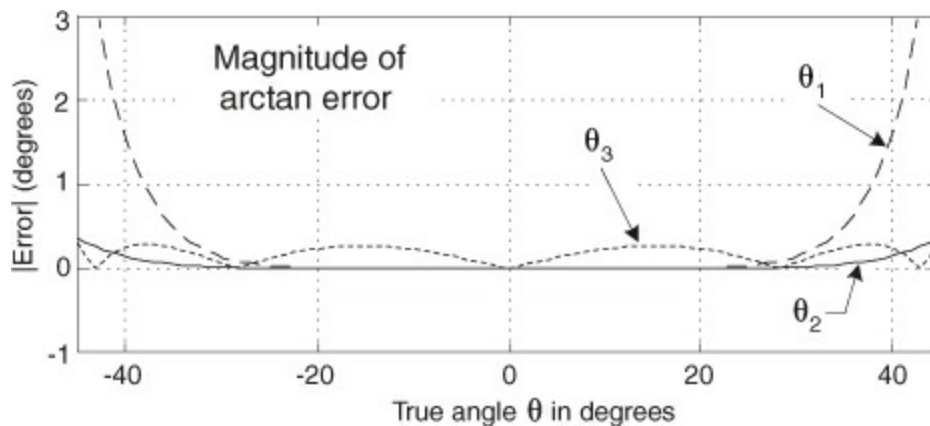
$$\theta \approx \theta_1 = X - \frac{1}{3}(X)^3 + \frac{1}{5}(X)^5,$$

$$\theta \approx \theta_2 = \frac{15X + 4X^3}{15 + 9X^2}, \text{ and}$$

$$\theta \approx \theta_3 = -0.19329X^3 + 0.97327X.$$

The magnitudes of the angular error associated with these arctangent algorithms, measured in degrees, over the true angle θ range of $-45^\circ \leq \theta \leq 45^\circ$ are shown in [Figure P8-21](#).

Figure P8-21



- (a) Which arctangent algorithm, θ_1 , θ_2 , or θ_3 , has the lowest average error magnitude over the range $-45^\circ \leq \theta \leq 45^\circ$?
- (b) To compare the computational workload associated with these algorithms, create a table (like the following) and fill in the number of multiplies, additions, and divisions necessary to compute an estimated arctangent for each algorithm.

Hint: Do not count redundant multiplies. That is, if X^2 has been computed, use that X^2 value for computing higher orders of X .

Algorithm	Multiplies	Additions	Divisions
θ_1			
θ_2			
θ_3			

- (c) Assume an arctangent computation needs to be implemented on a programmable DSP chip, and that chip requires 1, 1, and 30 processor clock cycles to implement a multiply, an addition, and a division, respectively. Which arctangent algorithm, θ_1 , θ_2 , or θ_3 , would you use to minimize the computational workload (processor clock cycles) and increase real-time data throughput?
- (d) If you know that the true angle θ is always in the range of $-22.5^\circ \leq \theta \leq 22.5^\circ$, which arctangent algorithm, θ_1 , θ_2 , or θ_3 , would be your first choice? Explain why. (The assumptions in Part (c) regarding processor clock cycles still apply.)

8.22 An interesting contender in the *arctangent race* (to calculate reasonably accurate arctangents with as few computations as possible) for estimating the angle of a complex time-domain sample is the following algorithm:

$$\tan^{-1}\left(\frac{Q}{I}\right) \approx \theta' = \frac{Q/I}{1+0.28125(Q/I)^2} \text{ radians}$$

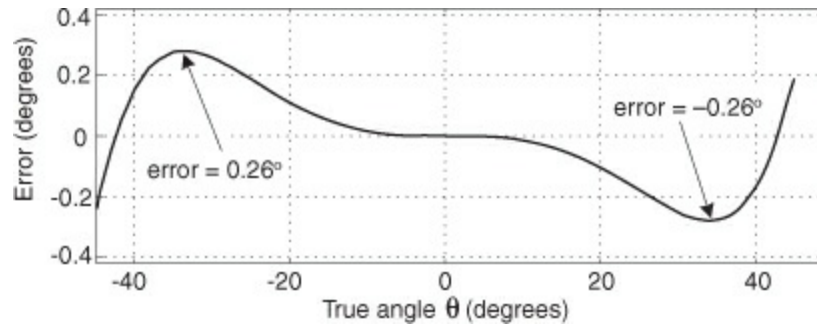
or

$$\tan^{-1}\left(\frac{Q}{I}\right) \approx \theta' = \frac{Q/I}{1+0.28125(Q/I)^2} \cdot \frac{180}{\pi} \text{ degrees,}$$

where θ' is the approximated angle of the complex sample $C=I+jQ$. This algorithm is useful when the true angle of C is in the range of $-\pi/2$ to $\pi/2$ radians (-45 to 45 degrees).

The error in degrees in using the above θ' approximation is shown in [Figure P8-22](#). To further investigate this arctangent algorithm, and exercise your algebraic function analysis skills, at what angles of the true θ is the algorithm's error at its peak values of ± 0.26 degrees?

Figure P8-22

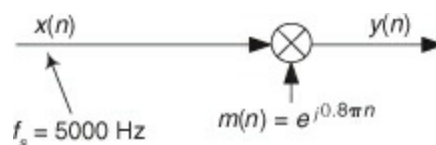


Hint: To make the algebra a little simpler, replace Q/I with X and replace 0.28125 with A . With those changes, the error $E(X)$ in this algorithm (in radians) is the true arctangent of X minus the approximation of the arctangent of X , or

$$E(X) = \tan^{-1}(X) - \frac{X}{1+AX^2} \text{ radians.}$$

8.23 Consider the process shown in [Figure P8-23](#). The $x(n)$ input sequence, whose f_s sample rate is 5000 Hz, is multiplied by a complex exponential $m(n)$ sequence to produce the complex $y(n)$ output sequence. What is the frequency, measured in Hz, of the complex exponential $m(n)$ sequence?

Figure P8-23



8.24 To gain further understanding of the quadrature sampling (complex down-conversion) in the text's [Figure 8-18\(a\)](#), we investigate the quadrature sampler's behavior when the $x_{bp}(t)$ input is a continuous (analog) cosine wave whose frequency is $f_c + 10$ Hz. That is:

$$x_{bp}(t) = \cos[2\pi(f_c + 10)t].$$

Given the above $x_{bp}(t)$ input signal:

- What is the spectral content of $x_i(t)$ in [Figure 8-18\(a\)](#)?
- What is the spectral content of $x_q(t)$?
- What is the phase shift between the low- and high-frequency components in $x_q(t)$?

(d) What is the spectral content of $i(t)$ in [Figure 8-18\(a\)](#)?

(e) What is the spectral content of $q(t)$?

8.25 In the text's [Section 8.6](#) we said that multiplying a continuous (analog) $x(t)$ time-domain signal, centered at f_o Hz, by a complex exponential whose frequency is negative f_c Hz, $e^{-j2\pi f_c t}$, results in a signal whose spectrum is shifted down in frequency by f_c Hz. This property is an exceedingly important concept to understand and remember!

(a) Prove that the above statement is true for discrete $x(n)$ time-domain signals originally centered at f_o Hz. Assume that $x(n) = \cos(2\pi f_o n t_s)$, a real-valued sinusoid where $f_o > f_c$.

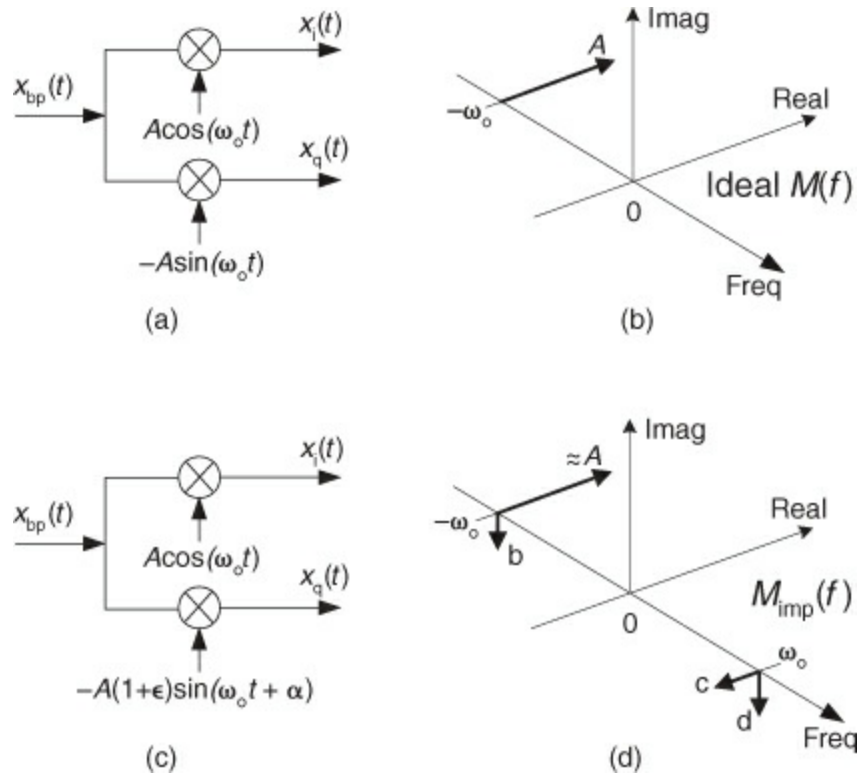
(b) Draw the spectrum of the original $x(n)$ and the spectrum of the down-shifted sequence.

(c) What is the spectral amplitude loss when multiplying a discrete $x(n)$ time-domain signal by $e^{-j2\pi f_c t}$?

8.26 Here is a problem of great practical importance. In the text we discussed analog signal down-conversion using the process shown in [Figure P8-26\(a\)](#) where $x_{bp}(t)$ was an analog bandpass signal centered at a frequency of ω_o radians/second. The ideal mixing (frequency translation) signal is

$$m(t) = A\cos(\omega_o t) - jA\sin(\omega_o t) = Ae^{-j\omega_o t},$$

Figure P8-26



whose spectrum is the single spectral component shown in [Figure P8-26\(b\)](#).

A more realistic situation is shown in [Figure P8-26\(c\)](#) where, due to real-world analog signal generator imperfections, the quadrature part of $m(t)$ contains a small amplitude error ϵ and a small phase error α (in radians). That is, the imperfect $m(t)$ becomes

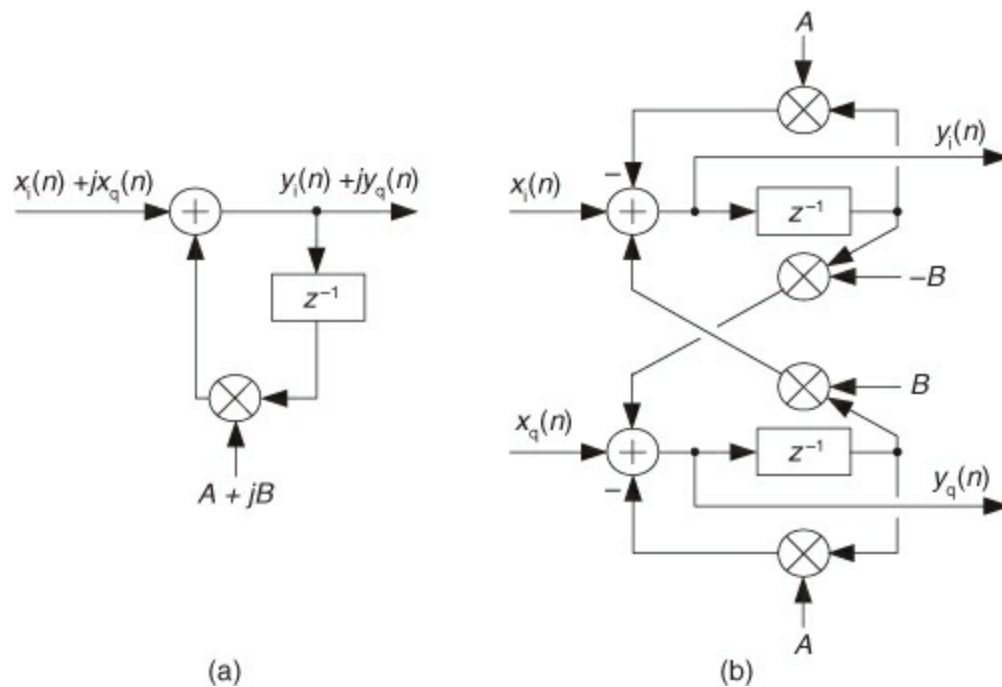
$$m_{\text{imp}}(t) = A\cos(\omega_o t) - jA(1+\epsilon)\sin(\omega_o t + \alpha).$$

When those quadrature errors exist, the spectrum of the imperfect $m_{\text{imp}}(t)$ is that shown in [Figure P8-26\(d\)](#). In practice, we want to keep the b, c, and d spectral components of $M_{\text{imp}}(f)$ as small as possible. Your problem is: What are the values for ε and α such that the magnitude of the unwanted ω_0 frequency component of $M_{\text{imp}}(f)$ is equal to 0.1 percent of (60 dB below) the magnitude of the desired $-\omega_0$ frequency component of $M_{\text{imp}}(f)$?

Hint: Start by converting the $m_{\text{imp}}(t)$ expression to polar (exponential) form and determine the complex-amplitude terms of its positive- and negative-frequency components. At the end of that exercise, assume that ε and α are much less than one part in a hundred to simplify your equations.

8.27 I once encountered an Internet website that presented a narrowband quadrature (complex) bandpass filter similar to that shown in [Figure P8-27\(a\)](#). The input sequence is assumed to be complex, and the filter's feedback coefficient is complex, $A + jB$. (Such filters are useful in both spectrum analysis and quadrature digital filtering applications.) The website stated that the real-valued coefficient implementation of this complex filter is that shown in [Figure P8-27\(b\)](#).

Figure P8-27



(a) As it turns out, [Figure P8-27\(b\)](#) is not a correct implementation of the quadrature filter. (Don't believe everything you read on the Internet!) Draw the *correct* real-coefficient implementation of the filter in [Figure P8-27\(a\)](#).

(b) Putting on our filter analysis hat, represent the complex input to the narrowband quadrature filter in [Figure P8-27\(a\)](#) as $x(n)$, the filter's complex output as $y(n)$, and the feedback coefficient as $e^{j2\pi f_r/f_s}$. (Variable f_r is the narrowband filter's resonant frequency in Hz, and f_s is the system's sample rate in Hz.) What is the time-domain difference equation, in complex notation, describing the quadrature bandpass filter?

(c) What is the z -domain equation for the filter's $H(z) = Y(z)/X(z)$ transfer function? (Transfer functions help us determine both the frequency response and stability of the filter.)

(d) Is this complex filter stable? Justify your answer.

(e) What is the frequency-domain equation for the filter's $H(f)$ frequency response? (The frequency response expression allows us to plot the filter's frequency-domain magnitude and phase behavior using signal processing software.)

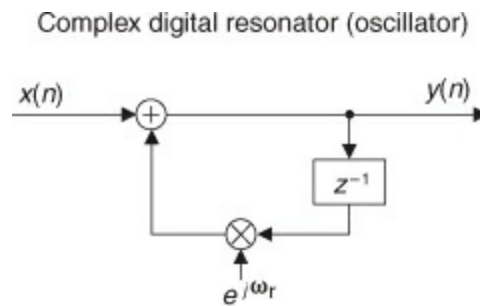
8.28 There are many signal processing applications that require a signal to be translated in frequency (particularly in digital communications systems). In the text we referred to frequency translation as *complex up-conversion*, *complex down-conversion*, and *complex mixing*. With this frequency translation notion in mind:

(a) Assuming we want to frequency translate a real-only time sequence, draw the block diagram of a discrete complex frequency translation network whose input is a real-only-valued discrete sequence, and whose output is a complex-valued discrete sequence.

(b) Assuming we only need the real part of a frequency-translated complex time sequence, draw the block diagram of a discrete complex frequency translation network whose input is a complex-valued discrete sequence and whose output sequence is real-valued.

8.29 In the literature of quadrature processing, we often encounter a network called a *complex digital resonator* whose block diagram is shown in [Figure P8-29](#). The feedback coefficient is a complex number, where the ω_r frequency value is a normalized angle measured in radians in the range of 0 to 2π radians, corresponding to a resonator cyclic frequency range of 0 to f_s , where f_s is the sample rate in Hz. Should we want to build a resonator (oscillator) whose cyclic frequency is $f_s/4$, then we'd merely set ω_r equal to $\pi/2$ and apply a single unity-valued sample, $x(n)$, to the input of the resonator to initiate its oscillations.

Figure P8-29



Here's the homework problem: If you had to build this complex resonator in hardware (because you needed a complex $e^{j\omega_r}$ quadrature sequence for some follow-on processing), what would be your resonator's block diagram where, of course, all of the discrete sequence values and feedback coefficients are real-only values?

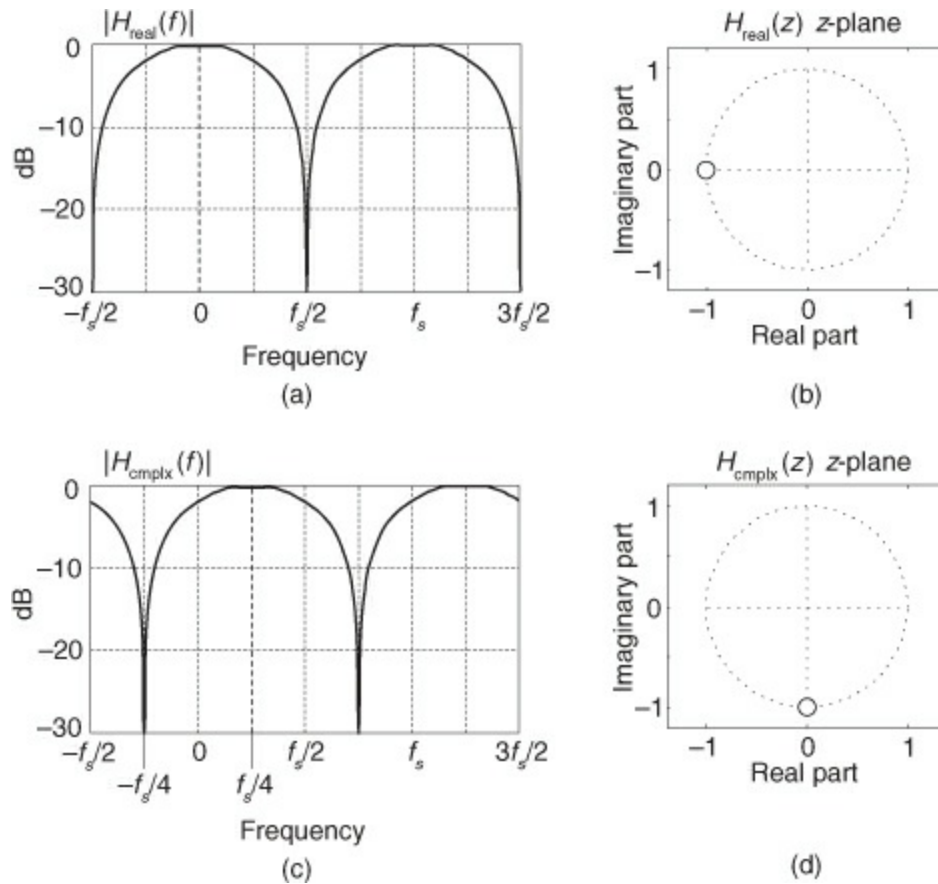
8.30 Assume that, on the job, you encounter the real-coefficient recursive lowpass filter whose z -domain transfer function is

$$H_{\text{real}}(z) = 1 + z^{-1}.$$

The filter's frequency magnitude response and z -plane pole/zero plot are given in [Figures P8-30\(a\)](#) and [P8-30\(b\)](#). Your problem is to design a complex-coefficient $H_{\text{cmplx}}(z)$ bandpass filter whose frequency magnitude response is shifted up in frequency by $f_s/4$ Hz relative to $|H_{\text{real}}(f)|$. That is, design a complex filter whose magnitude response is that given in [Figure P8-30\(c\)](#), having the pole/zero plot shown in [Figure P8-30\(d\)](#).

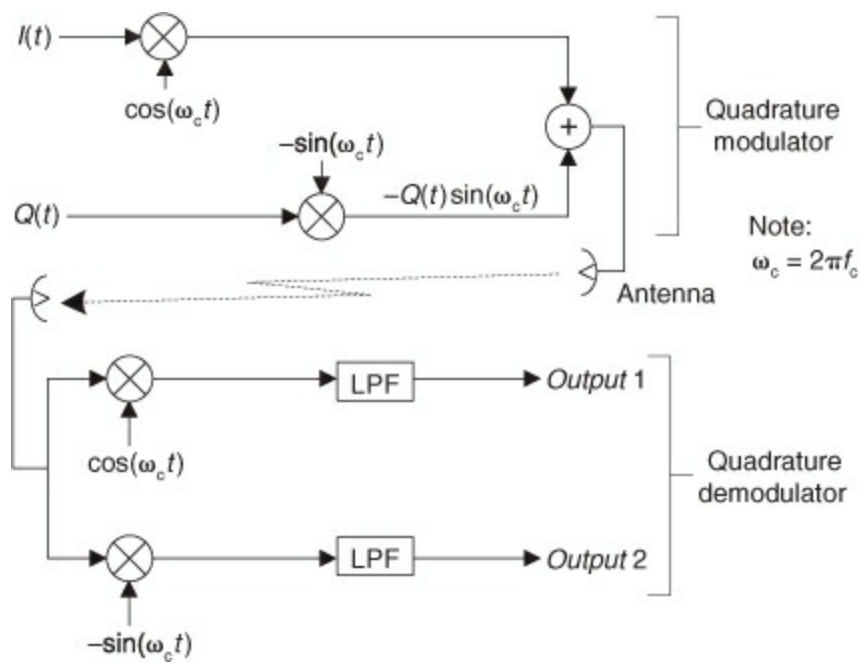
- (a) What is the transfer function equation of the complex-coefficient $H_{\text{cmplx}}(z)$ filter?
- (b) Draw the block diagram of the $H_{\text{cmplx}}(z)$ filter.

Figure P8-30

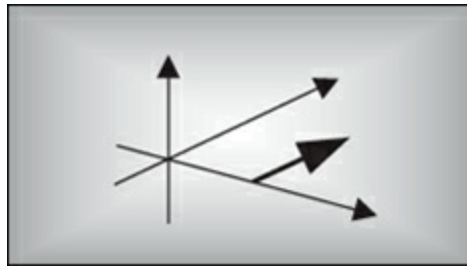


8.31 The process of quadrature modulation and demodulation has become very important and popular in modern communications systems. This means we can transmit two totally independent signals, $I(t)$ and $Q(t)$, at the same RF carrier frequency (f_c) and still receive and demodulate those two signals separately as shown in [Figure P8-31](#). In addition, we only have to use one transmitting antenna! To convince ourselves this communications method works, show algebraically what will be the outputs (in terms of $I(t)$ and $Q(t)$) of the lowpass filters of the quadrature demodulator in [Figure P8-31](#). (The $I(t)$ and $Q(t)$ signals are binary and have the values of either $+1$ or -1 . The “LPF” stages are identical lowpass filters whose cutoff frequencies are $\omega_c/2$ radians/second.) Justify your solution.

Figure P8-31



Chapter Nine. The Discrete Hilbert Transform



The discrete Hilbert transform is a process used to generate complex-valued signals from real-valued signals. Using complex signals in lieu of the real signals simplifies and improves the performance of many signal processing operations. If you've read about the discrete Hilbert transform in the DSP literature, you've probably plowed through the mathematical descriptions of analytic functions, with the constraints on their z -transforms in their regions of convergence, and perhaps you've encountered the Cauchy integral theorem used in the definition of the Hilbert transform.[‡] Well, the discrete Hilbert transform is not as complicated as it first appears; this chapter attempts to support that claim.

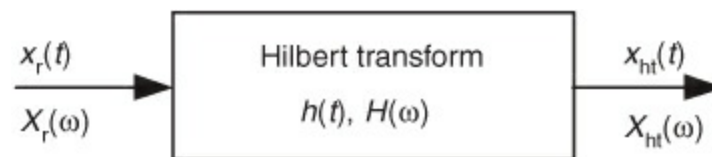
[‡] The Hilbert transform is named in honor of the great German mathematician David Hilbert (1862–1943). On his tomb in Göttingen, Germany, is inscribed, “Wir müssen wissen, wir werden wissen.” (We need to know, we shall know.)

Here we gently introduce the Hilbert transform from a practical standpoint, explain the mathematics behind its description, and show how it's used in DSP systems. In addition to providing some of the algebraic steps missing from some textbooks, we'll illustrate the time- and frequency-domain characteristics of the transform, with an emphasis on the physical meaning of the quadrature (complex) signals associated with Hilbert transform applications. Finally, nonrecursive Hilbert transformer design examples and techniques for generating complex, so-called analytic signals are presented. (If you're not well versed in the notation and behavior of complex signals at this point, a review of [Chapter 8](#) would be useful.)

9.1 Hilbert Transform Definition

The Hilbert transform (HT) is a mathematical process performed on a real signal $x_r(t)$, yielding a new real signal $x_{ht}(t)$, as shown in [Figure 9-1](#).

Figure 9-1 The notation used to define the continuous Hilbert transform.



Our goal here is to ensure that $x_{ht}(t)$ is a 90-degree phase-shifted version of $x_r(t)$. So, before we carry on, let's make sure we understand the notation used in [Figure 9-1](#). The variables are defined as follows:

$x_r(t)$ = a real continuous time-domain input signal
 $h(t)$ = the time impulse response of a Hilbert transformer
 $x_{ht}(t)$ = the HT of $x_r(t)$, ($x_{ht}(t)$ is also a real time-domain signal)
 $X_r(\omega)$ = the Fourier transform of real input $x_r(t)$
 $H(\omega)$ = the frequency response (complex) of a Hilbert transformer
 $X_{ht}(\omega)$ = the Fourier transform of output $x_{ht}(t)$
 ω = continuous frequency measured in radians/second
 t = continuous time measured in seconds

We'll clarify that $x_{ht}(t) = h(t) * x_r(t)$, where the $*$ symbol means convolution. In addition, we can define the spectrum of $x_{ht}(t)$ as $X_{ht}(\omega) = H(\omega) \cdot X_r(\omega)$. (These relationships sure make the HT look like a filter, don't they? We'll cogitate on this notion again later in this chapter.)

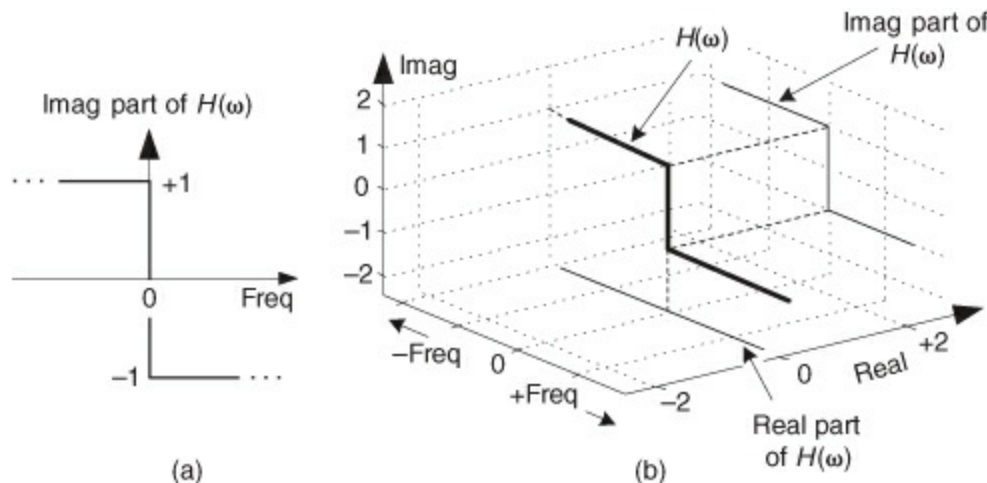
Describing how the new $x_{ht}(t)$ signal, the HT of $x_r(t)$, differs from the original $x_r(t)$ is most succinctly done by relating their Fourier transforms, $X_r(\omega)$ and $X_{ht}(\omega)$. In words, we can say that all of $x_{ht}(t)$'s positive-frequency components are equal to $x_r(t)$'s positive-frequency components shifted in phase by -90 degrees. Also, all of $x_{ht}(t)$'s negative-frequency components are equal to $x_r(t)$'s negative-frequency components shifted in phase by $+90$ degrees. Mathematically, we recall

(9-1)

$$X_{ht}(\omega) = H(\omega)X_r(\omega)$$

where $H(\omega) = -j$ over the positive-frequency range, and $H(\omega) = +j$ over the negative-frequency range. We show the nonzero imaginary part of $H(\omega)$ in [Figure 9-2\(a\)](#).

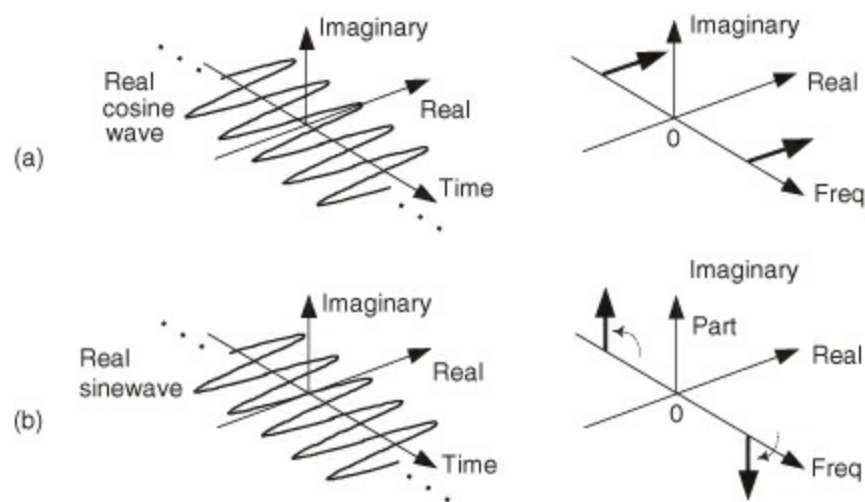
Figure 9-2 The complex frequency response of $H(\omega)$.



To fully depict the complex $H(\omega)$, we show it as floating in a three-dimensional space in [Figure 9-2\(b\)](#). The bold curve is our complex $H(\omega)$. On the right side is an upright plane on which we can project the imaginary part of $H(\omega)$. At the bottom of [Figure 9-2\(b\)](#) is a flat plane on which we can project the real part of $H(\omega)$. In rectangular notation, we say that $H(\omega) = 0 + j1$ for negative frequencies and $H(\omega) = 0 - j1$ for positive frequencies. (We introduce the three-dimensional axes of [Figure 9-2\(b\)](#) now because we'll be using them to look at other complex frequency-domain functions later in this discussion.)

To show a simple example of an HT, and to reinforce our graphical viewpoint, [Figure 9-3\(a\)](#) shows the three-dimensional time and frequency representations of a real cosine wave $\cos(\omega t)$. [Figure 9-3\(b\)](#) shows the HT of $\cos(\omega t)$ to be the sinewave $\sin(\omega t)$.

Figure 9-3 The Hilbert transform: (a) $\cos(\omega t)$; (b) its transform is $\sin(\omega t)$.



The complex spectrum on the right side of [Figure 9-3\(b\)](#) shows how the HT rotates the cosine wave's positive-frequency spectral component by $-j$, and the cosine wave's negative-frequency spectral component by $+j$. You can see on the right side of [Figure 9-3](#) that our definition of the $+j$ multiplication operation is a $+90$ -degree rotation of a spectral component counterclockwise about the frequency axis. (The length of those spectral components is half the peak amplitude of the original cosine wave.) We're assuming those sinusoids on the left in [Figure 9-3](#) exist for all time, and this allows us to show their spectra as infinitely narrow impulses in the frequency domain.

Now that we have this frequency response of the HT defined, it's reasonable for the beginner to ask: "Why would anyone want a process whose frequency-domain response is that weird $H(\omega)$ in [Figure 9-2\(b\)](#)?"

9.2 Why Care about the Hilbert Transform?

The answer is: We need to understand the HT because it's useful in so many complex-signal (quadrature) processing applications. A brief search on the Internet reveals HT-related signal processing techniques being used in the following applications:

- Quadrature modulation and demodulation (communications)
- Automatic gain control (AGC)
- Analysis of two- and three-dimensional complex signals
- Medical imaging, seismic data and ocean wave analysis
- Instantaneous frequency estimation
- Radar/sonar signal processing, and time-domain signal analysis using wavelets
- Time difference of arrival (TDOA) measurements
- High-definition television (HDTV) receivers
- Loudspeaker, room acoustics, and mechanical vibration analysis
- Audio and color image compression
- Nonlinear and nonstationary system analysis

All of these applications employ the HT either to generate or to measure complex time-domain signals, and that's where the HT's power lies. The HT delivers to us, literally, another dimension of signal processing capabilities as we move from two-dimensional real signals to three-dimensional

complex signals. Here's how.

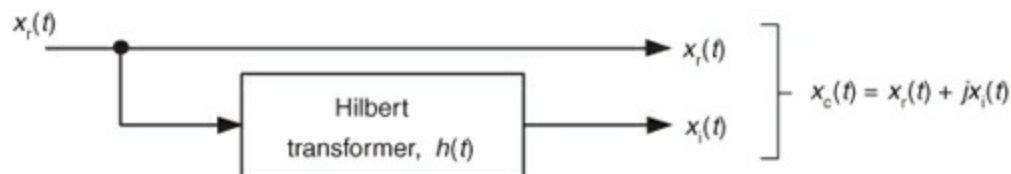
Let's consider a few mathematical definitions. If we start with a real time-domain signal $x_r(t)$, we can associate with it a complex signal $x_c(t)$, defined as

(9-2)

$$x_c(t) = x_r(t) + jx_i(t).$$

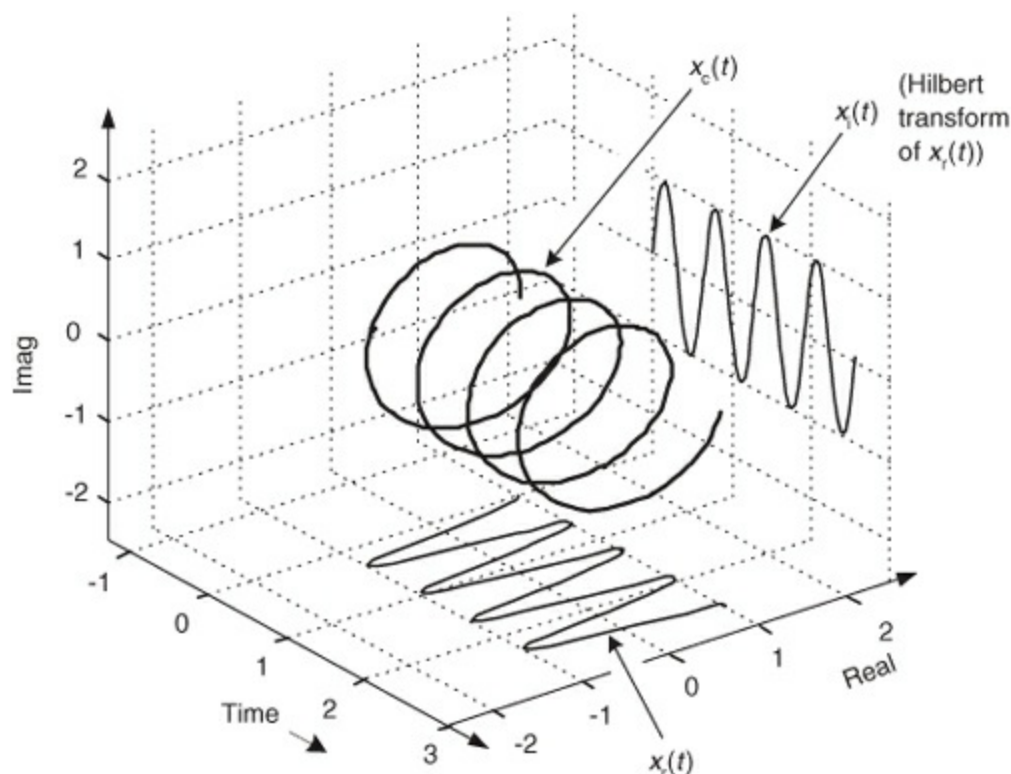
The complex $x_c(t)$ signal is known as an *analytic signal* (because it has no negative-frequency spectral components), and its real part is equal to the original real input signal $x_r(t)$. The key here is that $x_c(t)$'s imaginary part, $x_i(t)$, is the HT of the original $x_r(t)$ as shown in [Figure 9-4](#).

Figure 9-4 Functional relationship between the $x_c(t)$ and $x_r(t)$ signals.



As we'll see shortly, in many real-world signal processing situations $x_c(t)$ is easier, or more meaningful, to work with than the original $x_r(t)$. Before we see why that is true, we'll explore $x_c(t)$ further to attempt to give it some physical meaning. Consider a real $x_r(t) = \cos(\omega_0 t)$ signal that's simply four cycles of a cosine wave and its HT $x_i(t)$ sinewave as shown in [Figure 9-5](#). The $x_c(t)$ analytic signal is the bold *corkscrew* function.

Figure 9-5 The Hilbert transform and the analytic signal of $\cos(\omega_0 t)$.



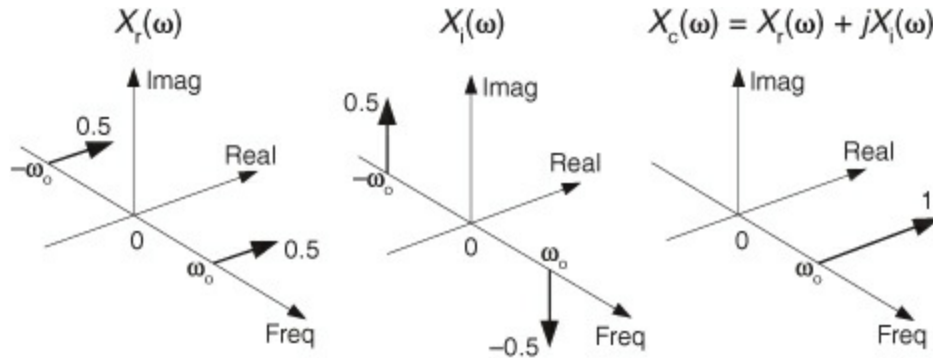
We can describe $x_c(t)$ as a complex exponential using one of Euler's equations. That is:

(9-3)

$$x_c(t) = x_r(t) + jx_i(t) = \cos(\omega_0 t) + j\sin(\omega_0 t) = e^{j\omega_0 t}.$$

The spectra of those signals in [Eq. \(9-3\)](#) are shown in [Figure 9-6](#). Notice three things in [Figure 9-6](#). First, following the relationships in [Eq. \(9-3\)](#), if we rotate $X_i(\omega)$ by $+90$ degrees counterclockwise ($+j$) and add it to $X_r(\omega)$, we get $X_c(\omega) = X_r(\omega) + jX_i(\omega)$. Second, note how the magnitude of the component in $X_c(\omega)$ is double the magnitudes in $X_r(\omega)$. Third, notice how $X_c(\omega)$ is zero over the negative-frequency range. This property of zero spectral content for negative frequencies is why $X_c(\omega)$ is called an *analytic signal*. Some people call $X_c(\omega)$ a *one-sided spectrum*.

Figure 9-6 HT spectra: (a) spectrum of $\cos(\omega_0 t)$; (b) spectrum of the Hilbert transform of $\cos(\omega_0 t)$, $\sin(\omega_0 t)$; (c) spectrum of the analytic signal of $\cos(\omega_0 t)$, $e^{j\omega_0 t}$.



To appreciate the physical meaning of our discussion here, let's remember that the $x_c(t)$ signal is not just a mathematical abstraction. We can generate $x_c(t)$ in our laboratory and route it via cables to the laboratory down the hall. (This idea is described in [Section 8.3](#).)

To illustrate the utility of this analytic signal $x_c(t)$ notion, let's see how analytic signals are very useful in measuring instantaneous characteristics of a time-domain signal: measuring the magnitude, phase, or frequency of a signal at some given instant in time. This idea of instantaneous measurements doesn't seem so profound when we think of characterizing, say, a pure sinewave. But if we think of a more complicated signal, like a modulated sinewave, instantaneous measurements can be very meaningful. If a real sinewave $x_r(t)$ is amplitude modulated so its *envelope* contains information, from an analytic version of the signal we can measure the instantaneous envelope $E(t)$ value using

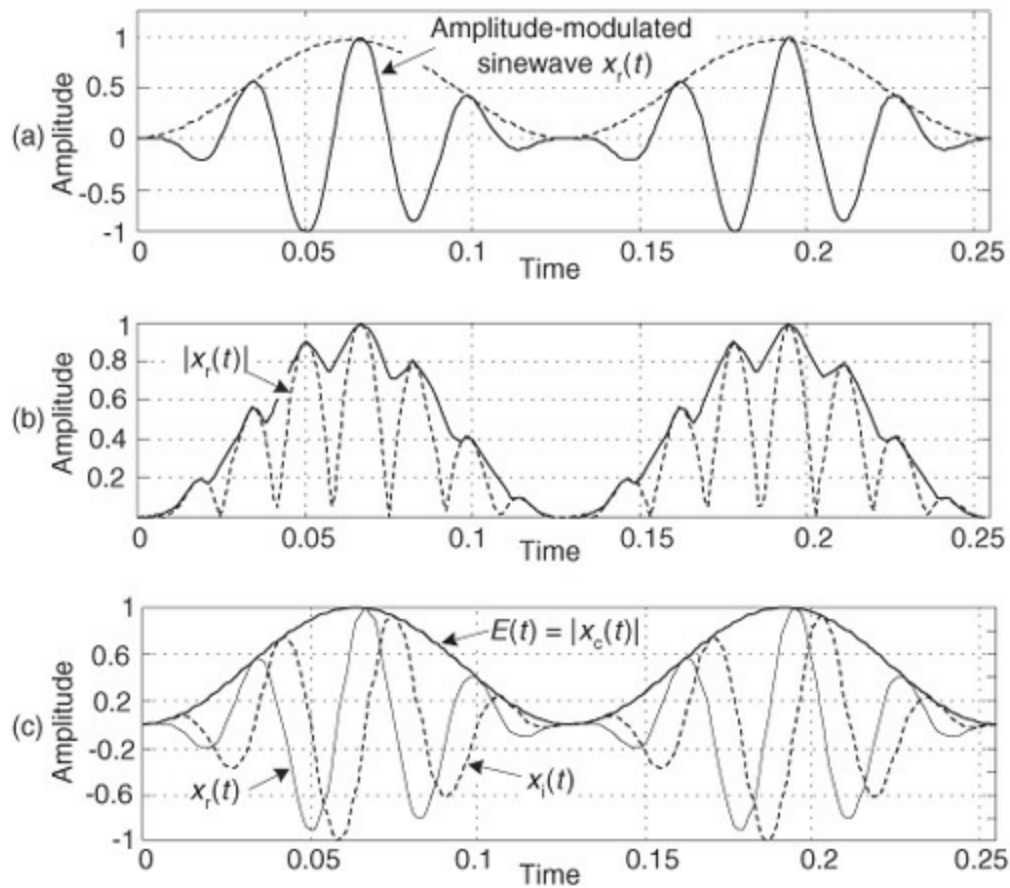
(9-4)

$$E(t) = |x_c(t)| = \sqrt{x_r(t)^2 + x_i(t)^2}.$$

That is, the envelope of the signal is equal to the magnitude of $x_c(t)$. We show a simple example of this AM demodulation idea in [Figure 9-7\(a\)](#), where a sinusoidal signal is amplitude modulated by a low-frequency sinusoid (dashed curve). To recover the modulating waveform, traditional AM demodulation would rectify the amplitude-modulated sinewave, $x_r(t)$, and pass the result through a lowpass filter. The filter's output is represented by the solid curve in [Figure 9-7\(b\)](#) representing the modulating waveform. Instead, we can compute the HT of $x_r(t)$, yielding $x_i(t)$, and use $x_i(t)$ to generate the $x_c(t) = x_r(t) + jx_i(t)$ analytic version of $x_r(t)$. Finally, we compute the magnitude of $x_c(t)$ using [Eq. \(9-4\)](#) to extract the modulating waveform shown as the bold solid curve in [Figure 9-7\(c\)](#). The $|x_c(t)|$ function is a *much* more accurate representation of the modulating waveform than the solid curve in [Figure 9-7\(b\)](#).

Figure 9-7 Envelope detection: (a) input $x_r(t)$ signal; (b) traditional lowpass filtering of $|x_r(t)|$; (c)

complex envelope detection result $|x_c(t)|$.



Suppose, on the other hand, some real $x_r(t)$ sinewave is phase modulated. We can estimate $x_c(t)$'s instantaneous phase $\phi(t)$, using

(9-5)

$$\phi(t) = \tan^{-1} \left(\frac{x_i(t)}{x_r(t)} \right).$$

Computing $\phi(t)$ is equivalent to phase demodulation of $x_r(t)$. Likewise (and more often implemented), should a real sinewave carrier be frequency modulated, we can measure its instantaneous frequency $F(t)$ by calculating the instantaneous time rate of change of $x_c(t)$'s instantaneous phase using

(9-6)

$$F(t) = \frac{d}{dt} \phi(t) = \frac{d}{dt} \tan^{-1} \left(\frac{x_i(t)}{x_r(t)} \right).$$

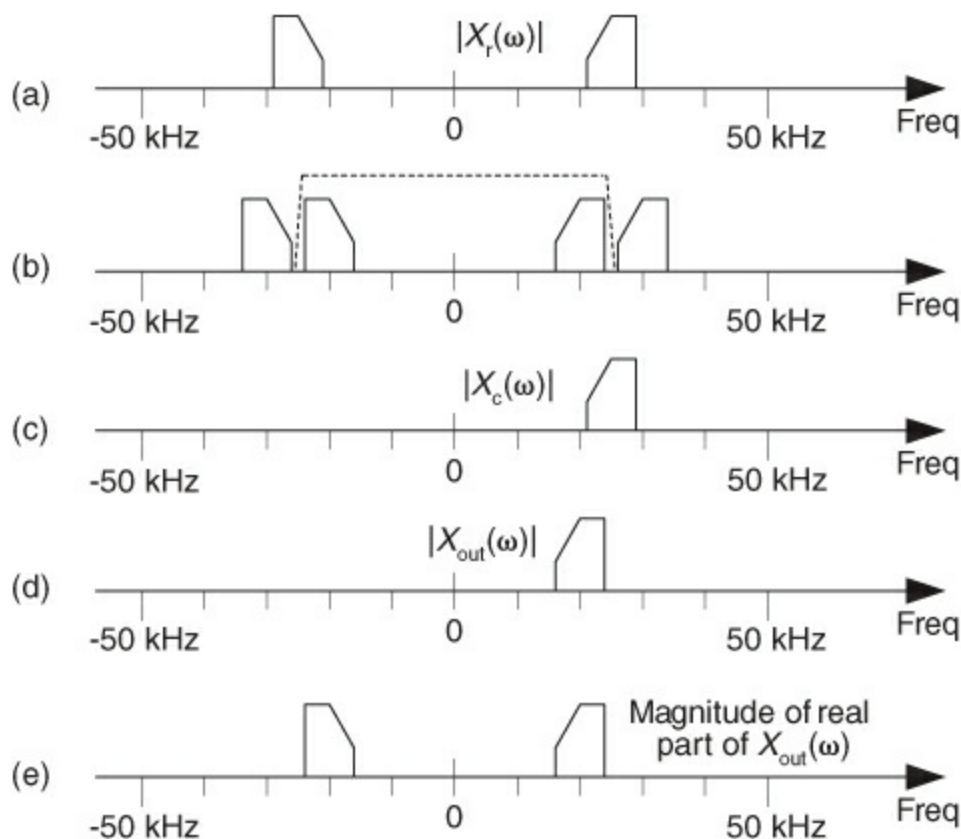
The process in [Eq. \(9-6\)](#) is a form of digital differentiation, a topic we discuss in some detail in [Chapter 7](#).

Calculating $F(t)$ is equivalent to frequency demodulation of $x_r(t)$. By the way, if $\phi(t)$ is measured in radians, then $F(t)$ in [Eq. \(9-6\)](#) is measured in radians/second. Dividing $F(t)$ by 2π will give it dimensions of Hz. (Another frequency demodulation method is discussed in [Section 13.22](#).)

For another HT application, consider a real $x_r(t)$ signal whose $|X_r(\omega)|$ spectral magnitude is centered at 25 kHz as shown in [Figure 9-8\(a\)](#). Suppose we wanted to translate that spectrum to be centered at 20 kHz. We could multiply $x_r(t)$ by the real sinusoid $\cos(2\pi 5000t)$ to obtain a real signal whose spectrum is shown in [Figure 9-8\(b\)](#). The problem with this approach is we'd need an impractically

high-performance filter (the dashed curve) to eliminate those unwanted high-frequency spectral images.

Figure 9-8 Spectra associated with frequency translation of a real signal $x_r(t)$.



On the other hand, if we compute the HT of $x_r(t)$ to obtain $x_i(t)$, and combine the two signals to form the analytic signal $x_c(t) = x_r(t) + jx_i(t)$, we'll have the complex $x_c(t)$ whose one-sided spectrum is given in [Figure 9-8\(c\)](#). Next we multiply the complex $x_c(t)$ by the complex $e^{-j2\pi 5000t}$, yielding a frequency-translated $x_{out}(t)$ complex signal whose spectrum is shown in [Figure 9-8\(d\)](#). Our final step is to take the real part of $x_{out}(t)$ to obtain a real signal with the desired spectrum centered about 20 kHz, as shown in [Figure 9-8\(e\)](#).

Now that we're convinced of the utility of the HT, let's determine the HT's time-domain impulse response and use it to build Hilbert transformers.

9.3 Impulse Response of a Hilbert Transformer

Instead of following tradition and just writing down the impulse response equation for a device that performs the HT, we'll show how to arrive at that expression. To determine the HT's impulse response expression, we take the inverse Fourier transform of the HT's frequency response $H(\omega)$. The garden-variety continuous inverse Fourier transform of an arbitrary frequency function $X(f)$ is defined as

$$(9-7) \quad x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi ft} df,$$

where f is frequency measured in cycles/second (hertz). We'll make three changes to [Eq. \(9-7\)](#). First, in terms of our original frequency variable $\omega = 2\pi f$ radians/second, and because $df = d\omega/2\pi$, we

substitute $d\omega/2\pi$ for the df term. Second, because we know our discrete frequency response will be periodic with a repetition interval of the sampling frequency ω_s , we'll evaluate [Eq. \(9-7\)](#) over the frequency limits of $-\omega_s/2$ to $+\omega_s/2$. Third, we partition the original integral into two separate integrals. This algebraic massaging gives us the following:

(9-8)

$$\begin{aligned} h(t) &= \frac{1}{2\pi} \int_{-\omega_s/2}^{\omega_s/2} H(\omega) e^{j\omega t} d\omega = \frac{1}{2\pi} \int_{-\omega_s/2}^0 j e^{j\omega t} d\omega + \frac{1}{2\pi} \int_0^{\omega_s/2} -j e^{j\omega t} d\omega \\ &= \frac{1}{2\pi t} \left(\left[e^{j\omega t} \right]_{-\omega_s/2}^0 - \left[e^{j\omega t} \right]_0^{\omega_s/2} \right) = \frac{1}{2\pi t} \left(e^{j0} - e^{-j\omega_s t/2} - e^{j\omega_s t/2} + e^{j0} \right) \\ &= \frac{1}{2\pi t} [2 - 2\cos(\omega_s t/2)] = \frac{1}{\pi t} [1 - \cos(\omega_s t/2)]. \end{aligned}$$

Whew! OK, we're going to plot this impulse response shortly, but first we have one hurdle to overcome. Heartache occurs when we plug $t = 0$ into [Eq. \(9-8\)](#) because we end up with the indeterminate ratio $0/0$. Hardcore mathematics to the rescue here. We merely pull out the Marquis de L'Hopital's Rule, take the time derivatives of the numerator and denominator in [Eq. \(9-8\)](#), and then set $t = 0$ to determine $h(0)$. Following through on this:

(9-9)

$$h(0) = \frac{\frac{d}{dt}(1 - \cos(\omega_s t/2))}{\frac{d}{dt}\pi t} \Big|_{t \rightarrow 0} = \frac{\omega_s \sin(\omega_s t/2)}{2\pi} \Big|_{t \rightarrow 0} = 0.$$

So now we know that $h(0) = 0$. Let's find the discrete version of [Eq. \(9-8\)](#) because that's the form we can model in software and actually use in our DSP work. We can *go digital* by substituting the discrete-time variable nt_s for the continuous-time variable t in [Eq. \(9-8\)](#). Using the following definitions

n = discrete time-domain integer index ($\dots, -3, -2, -1, 0, 1, 2, 3, \dots$)

f_s = sample rate measured in samples/second

t_s = time between samples, measured in seconds ($t_s = 1/f_s$)

$\omega_s = 2\pi f_s$

we can rewrite [Eq. \(9-8\)](#) in discrete form as

(9-10)

$$h(n) = \frac{1}{\pi n t_s} [1 - \cos(\omega_s n t_s/2)].$$

Substituting $2\pi f_s$ for ω_s , and $1/f_s$ for t_s , we have

(9-11)

$$h(n) = \frac{1}{\pi n t_s} [1 - \cos(2\pi f_s n / 2f_s)] = \frac{f_s}{\pi n} [1 - \cos(\pi n)],$$

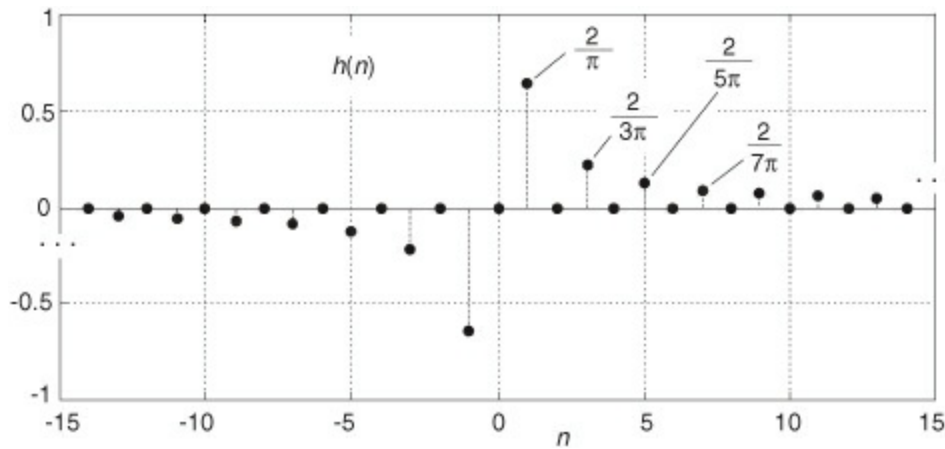
for $n \neq 0$, and $[h(n) = 0, \text{ for } n = 0]$.

Finally, we plot HT's $h(n)$ impulse response in [Figure 9-9](#). The f_s term in [Eq. \(9-11\)](#) is simply a scale factor; its value does not affect the shape of $h(n)$. An informed reader might, at this time, say, “Wait a minute. [Equation \(9-11\)](#) doesn't look at all like the equation for the HT's impulse response that's in my other DSP textbook. What gives?” The reader would be correct because one popular expression in the literature for $h(n)$ is

(9-12)

$$\text{alternate form: } h(n) = \frac{2 \sin^2(\pi n / 2)}{\pi n}.$$

Figure 9-9 The Hilbert transform's discrete impulse response when $f_s = 1$.



Here's the answer: The derivation of [Eq. \(9-12\)](#) is based on the assumption that the f_s sampling rate is normalized to unity. If we set $f_s = 1$ in [Eq. \(9-11\)](#), then that new expression will be equal to [Eq. \(9-12\)](#). We leave the proof of that equality as a homework problem.

Looking again at [Figure 9-9](#), we can reinforce the validity of our $h(n)$ derivation. Notice that for $n > 0$ the values of $h(n)$ are nonzero only when n is odd. In addition, the amplitudes of those nonzero values decrease by factors of $1/1$, $1/3$, $1/5$, $1/7$, etc. Of what does that remind you? That's right, the Fourier series of a periodic squarewave! This makes sense because our $h(n)$ is the inverse Fourier transform of the squarewave-like $H(\omega)$ in [Figure 9-2](#). Furthermore, our $h(n)$ is antisymmetric, and this is consistent with a purely imaginary $H(\omega)$. (If we were to make $h(n)$ symmetrical by inverting its values for all $n < 0$, the new sequence would be proportional to the Fourier series of a periodic real squarewave.)

Now that we have the expression for the HT's impulse response $h(n)$, let's use it to build a discrete Hilbert transformer.

9.4 Designing a Discrete Hilbert Transformer

Discrete Hilbert transformations can be implemented in either the time or frequency domains. Let's look at time-domain Hilbert transformers first.

9.4.1 Time-Domain Hilbert Transformation: FIR Filter Implementation

Looking back at [Figure 9-4](#), and having $h(n)$ available, we want to know how to generate the discrete

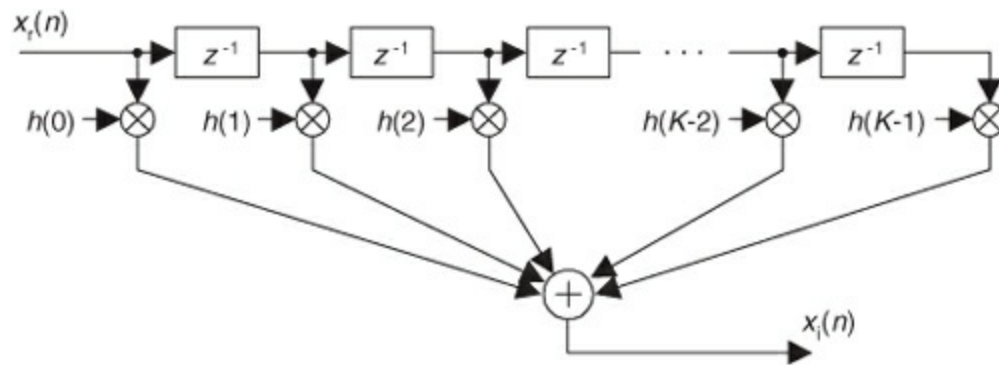
$x_i(n)$. Recalling the frequency-domain product in [Eq. \(9-1\)](#), we can say $x_i(n)$ is the convolution of $x_r(n)$ and $h(k)$. Mathematically, this is

$$(9-13)$$

$$x_i(n) = \sum_{k=-\infty}^{\infty} h(k) x_r(n-k).$$

So this means we can implement a Hilbert transformer as a discrete nonrecursive finite impulse response (FIR) filter structure as shown in [Figure 9-10](#).

Figure 9-10 FIR implementation of a K -tap Hilbert transformer.



Designing a traditional time-domain FIR Hilbert transformer amounts to determining those $h(k)$ values so the functional block diagram in [Figure 9-4](#) can be implemented. Our first thought is merely to take the $h(n)$ coefficient values from [Eq. \(9-11\)](#), or [Figure 9-9](#), and use them for the $h(k)$ s in [Figure 9-10](#). That's almost the right answer. Unfortunately, the [Figure 9-9](#) $h(n)$ sequence is infinite in length, so we have to truncate the sequence. Figuring out what the truncated $h(n)$ should be is where the true design activity takes place.

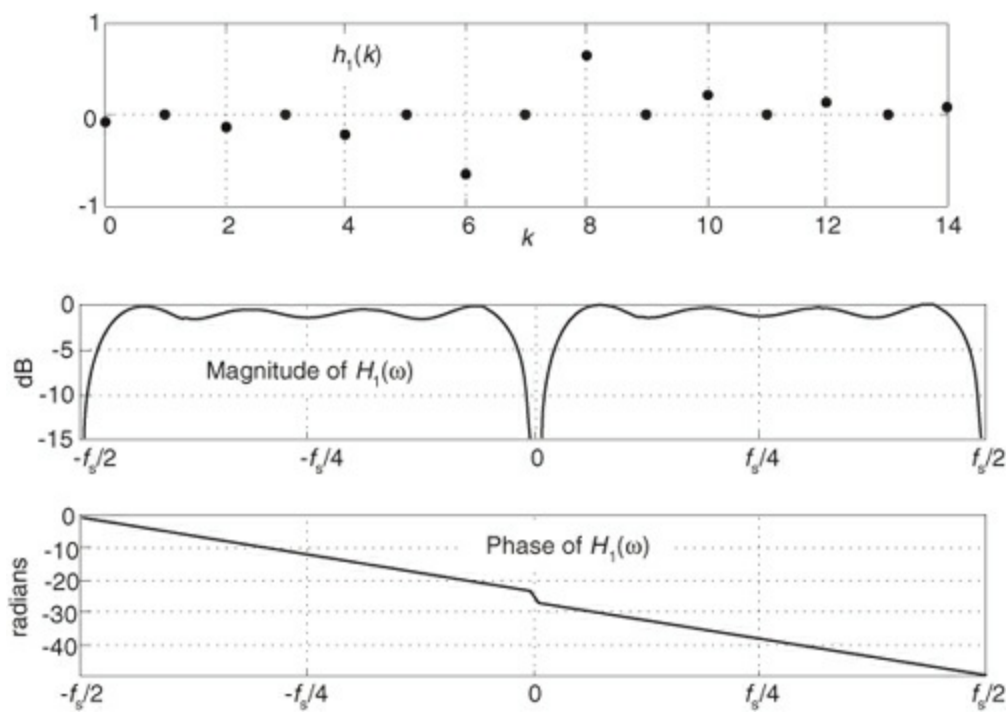
To start with, we have to decide if our truncated $h(n)$ sequence will have an odd or even length. We make this decision by recalling that FIR implementations having antisymmetric coefficients and an odd, or even, number of taps are called a Type-III, or a Type-IV, system respectively[1–3]. These two antisymmetric filter types have the following unavoidable restrictions with respect to their frequency magnitude responses $|H(\omega)|$:

$h(n)$ length: \rightarrow	Odd (Type-III)	Even (Type-IV)
	$ H(0) = 0$	$ H(0) = 0$
	$ H(\omega_s/2) = 0$	$ H(\omega_s/2) $ no restriction

What this little table tells us is odd-tap Hilbert transformers always have a zero magnitude response at both zero Hz and at half the sample rate. Even-tap Hilbert transformers always have a zero magnitude response at zero Hz. Let's look at some examples.

[Figure 9-11](#) shows the frequency response of a 15-tap (Type-III, odd-tap) FIR Hilbert transformer whose coefficients are designated as $h_1(k)$. These plots have much to teach us.

Figure 9-11 $H_1(\omega)$ frequency response of $h_1(k)$, a 15-tap Hilbert transformer.

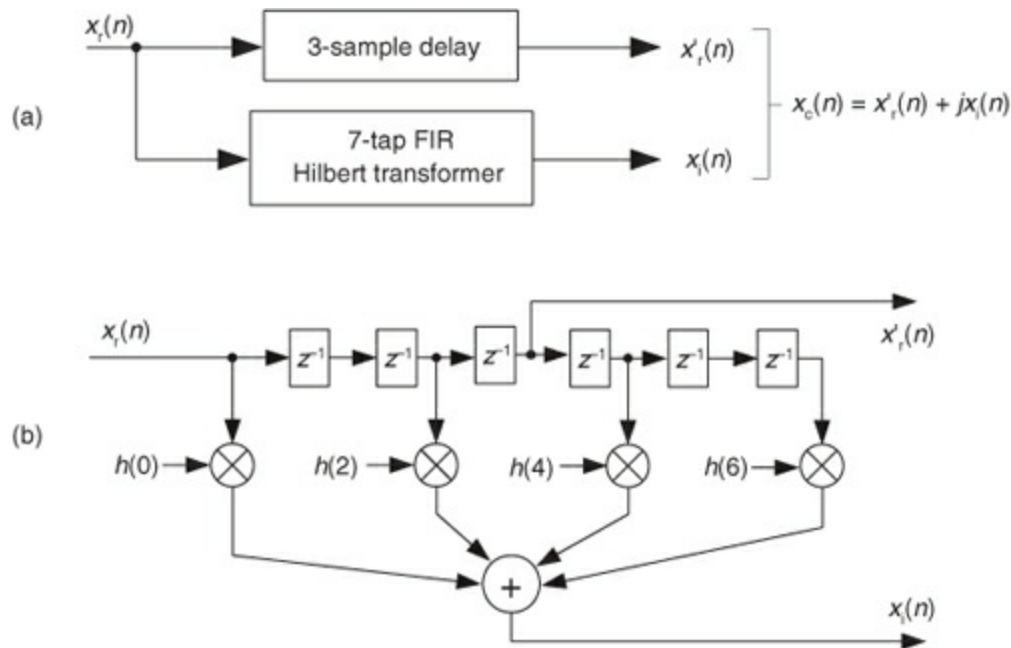


1. For example, an odd-tap FIR implementation does indeed have a zero magnitude response at 0 Hz and $\pm f_s/2$ Hz. This means odd-tap (Type-III) FIR implementations turn out to be bandpass in performance.
2. There's ripple in the $H_1(\omega)$ passband. We should have expected this because we were unable to use an infinite number of $h_1(k)$ coefficients. Here, just as it does when we're designing standard lowpass FIR filters, truncating the length of the time-domain coefficients causes ripples in the frequency domain. (When we abruptly truncate a function in one domain, Mother Nature pays us back by invoking the Gibbs phenomenon, resulting in ripples in the other domain.) You guessed it. We can reduce the ripple in $|H_1(\omega)|$ by windowing the truncated $h_1(k)$ sequence. However, windowing the coefficients will narrow the bandwidth of $H_1(\omega)$ somewhat, so using more coefficients may be necessary after windowing is applied. You'll find windowing the truncated $h_1(k)$ sequence to be to your advantage.
3. It's exceedingly difficult to compute the HT of low-frequency signals. We can widen the passband and reduce the transition region width of $H_1(\omega)$'s magnitude response, but that requires many filter taps.
4. The phase response of $H_1(\omega)$ is linear, as it should be when the coefficients' absolute values are symmetrical. The slope of the phase curve (that is constant in our case) is proportional to the time delay a signal sequence experiences traversing the FIR filter. More on this in a moment. That discontinuity in the phase response at 0 Hz corresponds to π radians, as [Figure 9-2](#) tells us it should. Whew, good thing. That's what we were after in the first place!

In our relentless pursuit of correct results, we're forced to compensate for the linear phase shift of $H_1(\omega)$ —that constant time value equal to the group delay of the filter—when we generate our analytic $x_c(n)$. We do this by delaying, in time, the original $x_r(n)$ by an amount equal to the group delay of the $h_1(k)$ FIR Hilbert transformer. Recall that the group delay, G , of a tapped-delay line FIR filter, having antisymmetrical coefficients, is $G = D/2$ samples where D is the number of unit-delay elements in the delay line. So our block diagram for generating a complex $x_c(n)$ signal, using an FIR structure, is

given in [Figure 9-12\(a\)](#). In this example, the 7-tap Hilbert filter has $D = 6$ delay elements as shown in [Figure 9-12\(b\)](#). There we delay $x_r(n)$ by $G = 6/2 = 3$ samples, generating the delayed sequence $x'_r(n)$. This delayed sequence now aligns properly in time with $x_i(n)$.

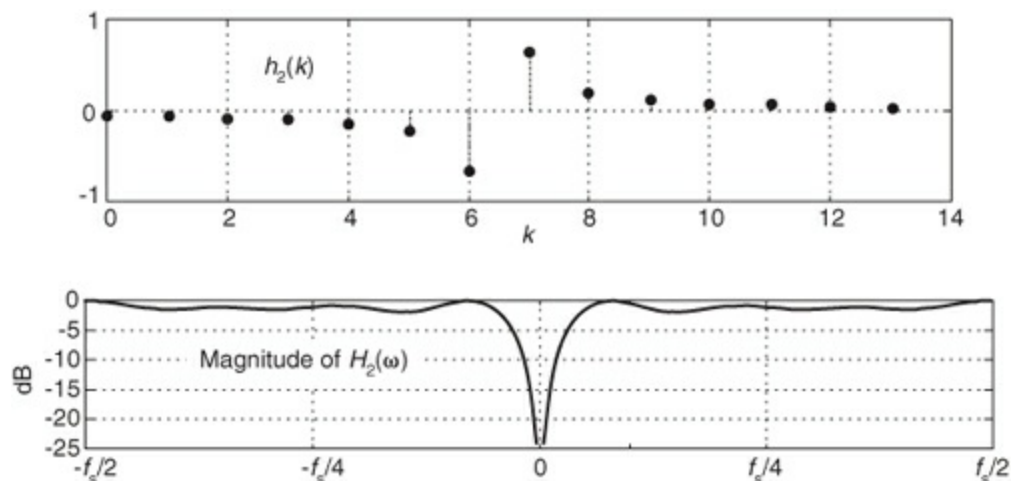
Figure 9-12 Generating an $x_c(n)$ sequence when $h(k)$ is a 7-tap FIR Hilbert filter: (a) processing steps; (b) filter structure.



If you're building your odd-tap FIR Hilbert transform in hardware, an easy way to obtain $x'_r(n)$ is to *tap off* the original $x_r(n)$ sequence at the center tap of the FIR Hilbert transformer structure as in [Figure 9-12\(b\)](#). If you're modeling [Figure 9-12\(a\)](#) in software, the $x'_r(n)$ sequence can be had by inserting $G = 3$ zeros at the beginning of the original $x_r(n)$ sequence.

We can, for example, implement an FIR Hilbert transformer using a Type-IV FIR structure, with its even number of taps. [Figure 9-13](#) shows this notion where the coefficients are, say, $h_2(k)$. See how the frequency magnitude response is nonzero at $\pm f_s/2$ Hz. Thus this even-tap filter approximates an ideal Hilbert transformer somewhat better than an odd-tap implementation.

Figure 9-13 $H_2(\omega)$ frequency response of $h_2(k)$, a 14-tap Hilbert transformer.



One of the problems with this traditional Hilbert transformer is that the passband gain in $|H_2(\omega)|$ is not unity for all frequencies, as is the $x'_r(n)$ path in [Figure 9-12](#). So to minimize errors, we must use many

$h_2(k)$ coefficients (or window the coefficients) to make $|H_2(\omega)|$'s passband as flat as possible.

Although not shown here, the negative slope of the phase response of $H_2(\omega)$ corresponds to a filter group delay of $G = (14-1)/2 = 6.5$ samples. This requires us to delay the original $x_r(n)$ sequence by a noninteger (fractional) number of samples in order to achieve time alignment with $x_i(n)$. Fractional time-delay filters are beyond the scope of this material, but reference [4] is a source of further information on the topic.

Let's recall that alternate coefficients of a Type-III (odd-tap) FIR are zeros. Thus the odd-tap Hilbert transformer is more attractive than an even-tap version from a computational workload standpoint. Almost half of the multiplications in Figure 9-10 can be eliminated for a Type-III FIR Hilbert transformer. Designers might even be able to further reduce the number of multiplications by a factor of two by using the *folded* FIR structure (discussed in Section 13.7) that's possible with symmetric coefficients (keeping in mind that half the coefficients are negative).

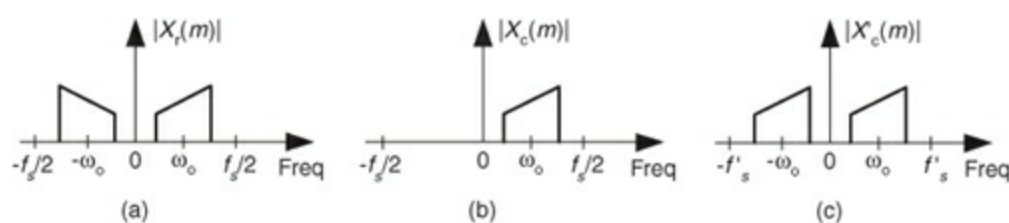
A brief warning: Here's a mistake sometimes even the *professionals* make. When we design standard linear-phase FIR filters, we calculate the coefficients and then use them in our hardware or software designs. Sometimes we forget to *flip* the coefficients before we use them in an FIR filter. This forgetfulness usually doesn't hurt us because typical FIR coefficients are symmetrical. Not so with FIR Hilbert filters, so please don't forget to reverse the order of your coefficients before you use them for convolutional filtering. Failing to flip the coefficients will distort the desired HT phase response.

As an aside, Hilbert transformers can be built with IIR filter structures, and in some cases they're more computationally efficient than FIR Hilbert transformers at the expense of a slight degradation in the 90-degree phase difference between $x_r(n)$ and $x_i(n)$ [5,6].

9.4.2 Frequency-Domain Hilbert Transformation

Here's a frequency-domain Hilbert processing scheme deserving mention because the HT of $x_r(n)$ and the analytic $x_c(n)$ sequence can be generated simultaneously. We merely take an N -point DFT of a real even-length- N $x_r(n)$ signal sequence, obtaining the discrete $X_r(m)$ spectrum given in Figure 9-14(a). Next, create a new spectrum $X_c(m) = 2X_r(m)$. Set the negative-frequency $X_c(m)$ samples, that's $(N/2)+1 \leq m \leq N-1$, to zero, leaving us with a new one-sided $X_c(m)$ spectrum as in Figure 9-14(b). Next, divide the $X_c(0)$ (the DC term) and the $X_c(N/2)$ spectral samples by 2. Finally, we perform an N -point inverse DFT of the new $X_c(m)$, the result being the desired analytic $x_c(n)$ time-domain sequence. The real part of $x_c(n)$ is the original $x_r(n)$, and the imaginary part of $x_c(n)$ is the HT of $x_r(n)$. Done!

Figure 9-14 Spectrum of the original $x_r(n)$ sequence, and the one-sided spectrum of the analytic $x_c(n)$ sequence.



There are several issues to keep in mind concerning this straightforward frequency-domain analytic

signal generation scheme:

1. If possible, restrict the $x_r(n)$ input sequence length to an integer power of two so the radix-2 FFT algorithm can be used to efficiently compute the DFT.
2. Make sure the $X_c(m)$ sequence has the same length as the original $X_r(m)$ sequence. Remember, you zero out the negative-frequency $X_c(m)$ samples; you don't discard them.
3. The factor of two in the above $X_c(m) = 2X_r(m)$ assignment compensates for the amplitude loss by a factor of two in losing the negative-frequency spectral energy.
4. If your HT application is block-oriented in the sense that you only have to generate the analytic sequence from a fixed-length real-time sequence, this technique is sure worth thinking about because there's no time delay headache associated with time-domain FIR implementations to worry about. With the advent of fast hardware DSP chips and pipelined FFT techniques, the above analytic signal generation scheme may be viable for a number of applications. One scenario to consider is using the efficient $2N$ -Point Real FFT technique, described in [Section 13.5.2](#), to compute the forward DFT of the real-valued $x_r(n)$. Of course, the thoughtful engineer would conduct a literature search to see what algorithms are available for efficiently performing inverse FFTs when many of the frequency-domain samples are zeros.

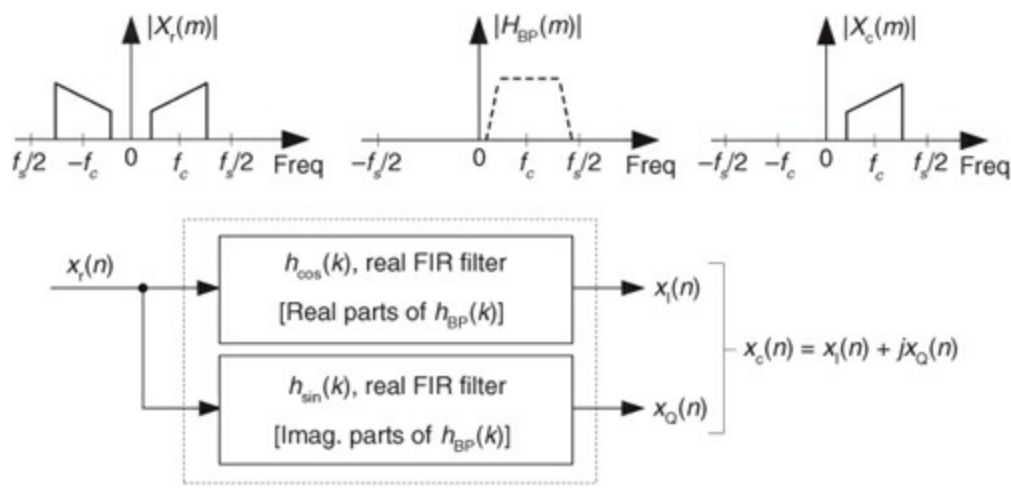
Should you desire a decimated-by-two analytic $x'_c(n)$ sequence based on $x_r(n)$, it's easy to do, thanks to reference [\[7\]](#). First, compute the N -point $X_r(m)$. Next, create a new spectral sequence $X'_c(k) = 2X_r(k)$ for $1 \leq k \leq (N/2)-1$. Set $X'_c(0)$ equal to $X_r(0) + X_r(N/2)$. Finally, compute the $(N/2)$ -point inverse DFT of $X'_c(m)$, yielding the decimated-by-two analytic $x'_c(n)$. The $x'_c(n)$ sequence has a sample rate of $f'_s = f_s/2$ and the spectrum shown in [Figure 9-14\(c\)](#).

In [Section 13.28.2](#) we discuss a scheme to generate interpolated analytic signals from $x_r(n)$.

9.5 Time-Domain Analytic Signal Generation

In digital communications applications, the FIR implementation of the HT (such as that in [Figure 9-12\(b\)](#)) is used to generate a complex analytic signal $x_c(n)$. Some practitioners now use a time-domain complex filtering technique to achieve analytic signal generation when dealing with real bandpass signals[\[8\]](#). This scheme, which does not specifically perform the HT of an input sequence $x_r(n)$, uses a complex filter implemented with two real FIR filters with essentially equal magnitude responses, but whose phase responses differ by exactly 90 degrees, as shown in [Figure 9-15](#).

Figure 9-15 Generating an $x_c(n)$ sequence with a complex filter (two real FIR filters).



Here's how it's done. A standard K -tap FIR lowpass filter is designed, using your favorite FIR design software, to have a two-sided bandwidth slightly wider than the original real bandpass signal of interest. The real coefficients of the lowpass filter, $h_{LP}(k)$, are then multiplied by the complex exponential $m_{ix}(k)$ sequence specified by

$$(9-14)$$

$$m_{ix}(k) = e^{j2\pi(k-D)f_c/f_s} = \cos[2\pi(k-D)f_c/f_s] + j\sin[2\pi(k-D)f_c/f_s]$$

using the following definitions:

k = time index term of the exponential mixing sequence ($k = 0, 1, 2, \dots, K-1$)

D = time delay of the K -tap lowpass filter, $D = (K-1)/2$

f_c = bandpass signal's center frequency, in Hz

f_s = sample rate of the original $x_I(n)$ bandpass signal sequence in Hz

The results of the $h_{LP}(k)$ times $m_{ix}(k)$ multiplication are complex coefficients whose rectangular-form representation is

$$(9-14')$$

$$h_{BP}(k) = h_{\cos}(k) + jh_{\sin}(k),$$

creating a complex bandpass filter centered at f_c Hz. The delay value D in [Eq. \(9-14\)](#) ensures that the $h_{\cos}(k)$ and $h_{\sin}(k)$ coefficients are symmetrical (or antisymmetrical) to obtain a linear-phase complex filter[\[9\]](#).

Next we use the real and imaginary parts of the filter's $h_{BP}(k)$ coefficients in two separate real-valued-coefficient FIR filters as shown in [Figure 9-15](#). In DSP parlance, the filter producing the $x_I(n)$ sequence is called the I-channel for *in-phase*, and the filter generating $x_Q(n)$ is called the Q-channel for *quadrature phase*. There are several interesting aspects of this mixing analytic signal generation scheme in [Figure 9-15](#):

1. The mixing of the $h_{LP}(k)$ coefficients by $m_{ix}(k)$ induces a loss, by a factor of two, in the frequency magnitude responses of the two real FIR filters. Doubling the $h_{LP}(k)$ values before mixing will eliminate the loss.
2. We can window the $h_{LP}(k)$ coefficients before mixing to reduce the passband ripple in, and to

minimize differences between, the magnitude responses of the two real filters. (We'd like to keep the passband gains as similar as possible.) Of course, windowing will degrade the lowpass filter's roll-off somewhat, so using more coefficients may be necessary before windowing is applied.

3. Odd- or even-tap lowpass filters can be used, with equal ease, in this technique.
4. Because the $h_{\cos}(k)$ or $h_{\sin}(k)$ coefficients are exactly symmetrical, or exactly antisymmetrical, the *folded* FIR structure (see [Section 13.7](#)) can be used to reduce the number of multiplies per filter output sample by roughly a factor of two.
5. If the original bandpass signal and the complex filter are centered at one-fourth the sample rate ($f_0 = f_s/4$), count your blessings. In this case we set $D = 0$ in [Eq. \(9-14\)](#), making nearly half the coefficients of each real filter zero-valued, maintaining linear phase, and reduce our FIR computational workload further by a factor of two.
6. A particularly efficient complex bandpass filter is a lowpass half-band FIR filter whose center frequency has been translated to $f_c = f_s/4$. The result is that half the $h_{\sin}(k)$ coefficients are zeros, and all but one of the $h_{\cos}(k)$ coefficients are zeros! (Two specialized analytic signal generation schemes using half-band filters are described in [Sections 13.1](#) and [13.37](#).)
7. For hardware applications, both of the two real FIR filters in [Figure 9-15](#) must be implemented. If your analytic signal generation is strictly a high-level software language exercise, such as using MATLAB, the language being used may allow $h_{BP}(k)$ to be implemented as a single complex filter.

Keep in mind, now, that the $x_Q(n)$ sequence in [Figure 9-15](#) is not the Hilbert transform of the $x_I(n)$ input. That wasn't our goal here. Our intent was to generate an analytic $x_c(n)$ sequence whose $x_Q(n)$ quadrature component is the Hilbert transform of the $x_I(n)$ in-phase sequence.

9.6 Comparing Analytic Signal Generation Methods

Time-domain FIR Hilbert transformer design is essentially an exercise in lowpass filter design. As such, an ideal discrete FIR Hilbert transformer, like an ideal lowpass FIR filter, cannot be achieved in practice. Fortunately, we can usually ensure that the bandpass of our [Figure 9-12](#) Hilbert transformer covers the bandwidth of the $x_I(n)$ input signal. The [Figure 9-12](#) Hilbert transformer has magnitude roll-off in the $x_I(n)$ channel but not in the $x'_I(n)$ channel. Using more filter taps improves the transformer's performance by

- minimizing passband ripple in the $x_I(n)$ channel,
- minimizing passband gain differences between the two channels, and
- broadening the transformer's passband width.

The choice of an odd or even number of taps depends on whether the gain at $f_s/2$ should be zero or not, and whether an integer-sample delay is needed.

For roughly the same passband ripple performance, the complex filter in [Figure 9-15](#) requires a longer impulse response (more filter taps) than the [Figure 9-12](#) Hilbert transformer. However, the complex filter has the nice feature that it actually implements bandpass filtering where the [Figure 9-](#)

[12](#) Hilbert transformer performs no filtering. A possible drawback of the complex filter method is that the real part of the generated complex signal is not equal to the original real input signal. (That is, $x_i(n)$ does not equal $x_r(n)$.)

For very high-performance analytic signal generation (where the equivalent Hilbert transformer bandwidth must be very close to the full $f_s/2$ bandwidth, and passband ripple must be very small), the DFT method should be considered. With the DFT method the real part of $x_c(n)$ is equal to the real input $x_r(n)$. Choosing which of the above analytic signal generation methods to use (or perhaps a complex down-conversion scheme from [Chapter 8](#)) in any given application requires modeling with your target computing hardware, using your expected number format (fixed point versus floating point), against the typical input signals you intend to process.

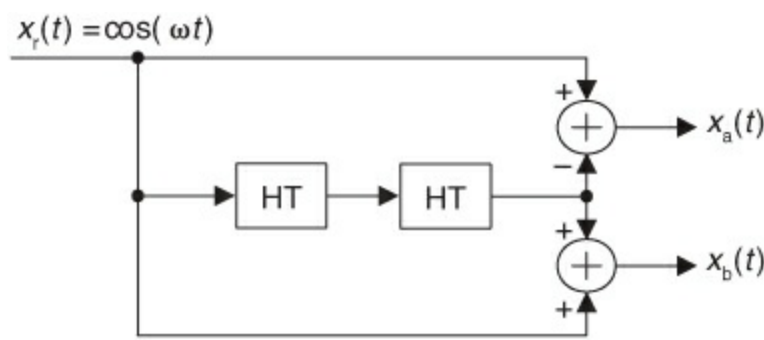
References

- [1] Proakis, J., and Manolakis, D. *Digital Signal Processing: Principles, Algorithms and Applications*, Prentice Hall, Upper Saddle River, New Jersey, 1996, pp. 618, 657.
- [2] Rabiner, L., and Gold, B. *The Theory and Application of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1975, pp. 67, 168.
- [3] Oppenheim, A., and Schaffer, R. *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1st ed. 1989, 2nd ed. 1999.
- [4] Laasko, T., et al. “Splitting the Unit Delay,” *IEEE Signal Processing Magazine*, January 1996.
- [5] Gomes, J., and Petraglia, A. “An Analog Sampled-Data DSB to SSB Converter Using Recursive Hilbert Transformer for Accurate I and Q Channel Matching,” *IEEE Trans. on Circuits and Sys.-II, Analog and Digital Signal Processing*, Vol. 39, No. 3, March 2002, pp. 177–187.
- [6] Ansari, R. “IIR Hilbert Transformers,” *IEEE Trans. Acoust., Speech Signal Processing*, Vol. 35, August 1987, pp. 1116–1119.
- [7] Marple, S., Jr. “Computing the Discrete-Time ‘Analytic’ Signal via FFT,” *IEEE Trans. on Signal Proc.*, Vol. 47, No. 9, September 1999, pp. 2600–2603.
- [8] Reilly, A., et al. “Analytic Signal Generation—Tips and Traps,” *IEEE Trans. on Signal Proc.*, Vol. 42, No. 11, November 1994.
- [9] Bell, D. Subject: “Hilbert Transform Using FFT Approach,” Internet Newsgroup: *comp.dsp*, September 28, 2005.

Chapter 9 Problems

9.1 Consider the real-valued continuous $x_r(t) = \cos(\omega t)$ sinusoidal signal applied to the network in [Figure P9-1](#). The “HT” notation means Hilbert transform. Write the algebraic expressions for the signals $x_a(t)$ and $x_b(t)$. Justify your solutions.

Figure P9-1



9.2 Consider the real-valued continuous $x_r(t)$ sinusoidal signal defined by

$$x_r(t) = A \sin(2\pi f_0 t)$$

where A is a scalar constant, and f_0 is a fixed frequency measured in Hz.

- Draw the three-dimensional spectrum of the Hilbert transform of $x_r(t)$.
- What is the equation for the analytic signal whose real part is $x_r(t)$, in terms of $\sin()$ and $\cos()$ functions?
- What is the equation for the analytic signal whose real part is $x_r(t)$, in complex exponential notation? Show your work.
- Is the analytic signal whose real part is $x_r(t)$ a positive- or negative-frequency complex exponential? Justify your answer.
- On a complex plane, draw the location of the analytic signal whose real part is $x_r(t)$, at time $t = 0$.

9.3 Consider the effect of repeated Hilbert transformations of a real-valued continuous $x(t) = \cos(\omega_0 t)$ sinusoidal signal, expressed as

$$u(t) = \text{HT}[x(t)]$$

$$v(t) = \text{HT}[u(t)]$$

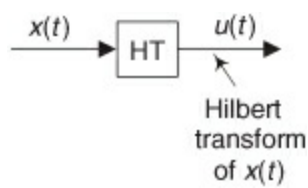
$$w(t) = \text{HT}[v(t)]$$

$$y(t) = \text{HT}[w(t)]$$

where “ $\text{HT}[x(t)]$ ” means the Hilbert transform of $x(t)$.

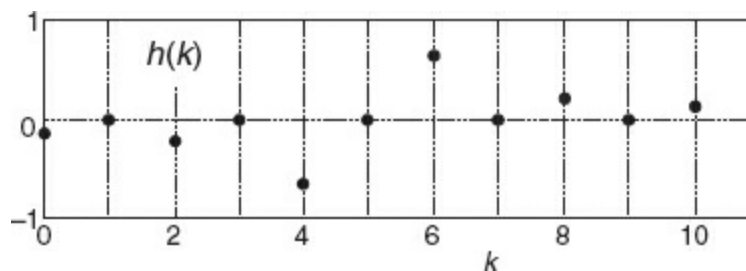
- Which of $u(t)$, $v(t)$, $w(t)$, or $y(t)$ equals $x(t)$? Justify your answer.
- Which of $u(t)$, $v(t)$, $w(t)$, or $y(t)$ equals $-x(t)$? Justify your answer.
- Which of $u(t)$, $v(t)$, $w(t)$, or $y(t)$ equals the inverse Hilbert transform of $x(t)$?
- Draw a block diagram of the system that implements the inverse Hilbert transform using just one forward Hilbert transform operation. Use a simple block, as shown in [Figure P9-3](#), to represent a forward Hilbert transform.

Figure P9-3



- 9.4 Looking at the odd- and even-length Hilbert transformers' time-domain impulse responses in the text's [Figures 9-11](#) and [9-13](#):
- Why should we expect that their frequency magnitude responses are zero (a magnitude null) at zero Hz?
 - Write the z -domain transfer function for a 6-tap tapped-delay line Hilbert transform filter having coefficients $h(0)$, $h(1)$, $h(2)$, ..., $h(5)$. Evaluate that function to determine the filter's frequency response at zero Hz.
- 9.5 Draw the block diagram (the structure) of a real-coefficient 11-tap FIR Hilbert transformer that will compute an $x_a(n) = x_r(n) + jx_i(n)$ analytic signal associated with a real $x_r(n)$ input signal.
- 9.6 Think about the tapped-delay line FIR Hilbert transformer whose coefficients are shown in [Figure P9-6](#). The coefficients are $h(0)$, $h(1)$, $h(2)$, ..., $h(10)$.

Figure P9-6



- Write the z -domain transfer function polynomial, $H(z)$, for this 11-coefficient Hilbert transformer in terms of the $h(k)$ coefficients and z .
 - We often describe a Hilbert transformer by the [order](#) of the transformer's z -domain transfer function polynomial. What is the order of the transformer whose coefficients are shown in [Figure P9-6](#)?
 - How many unit-delay elements are needed in the tapped-delay line to implement the Hilbert transformer whose coefficients are shown in [Figure P9-6](#)?
 - How many multipliers are needed in the standard tapped-delay line implementation of this Hilbert transformer?
- 9.7 The text's [Eq. \(9-11\)](#) provided the expression for the impulse response of a discrete Hilbert transformer. That expression is repeated here as

$$h(n) = \frac{f_s [1 - \cos(\pi n)]}{\pi n}.$$

The text also presented an alternate expression for the impulse response of a discrete Hilbert transformer, [Eq. \(9-12\)](#), under the assumption that f_s is normalized to unity. That alternate expression is repeated here as

$$h'(n) = \frac{2 \sin^2(\pi n / 2)}{\pi n}.$$

Show that the expressions for $h(n)$ and $h'(n)$ are equal.

9.8 Assume that we've performed the discrete Fourier transform (DFT) of an N -sample $x(n)$ time-domain sequence to obtain a frequency-domain $X(m)$ sequence. $X(m)$'s integer frequency index is $0 \leq m \leq N-1$. If N is even and $X(0) = X(N/2) = 0$, describe what operations must be performed on $X(m)$ to create a new frequency-domain $X_{\text{new}}(m)$ sequence whose inverse DFT is the Hilbert transform of $x(n)$. (Stated in different words: How do we perform the Hilbert transform in the frequency domain?)

9.9 In the literature of DSP you may learn that we can design an N -tap Hilbert transformer by first designing an N -tap half-band nonrecursive FIR filter (where $N+1$ is an integer multiple of four), yielding coefficients $h(n)$. Next, we multiply the half-band filter's $h(n)$ coefficients by $2\sin(\pi n/2)$ to obtain the desired $h_{\text{hilb}}(n)$ Hilbert transformer coefficients. That is,

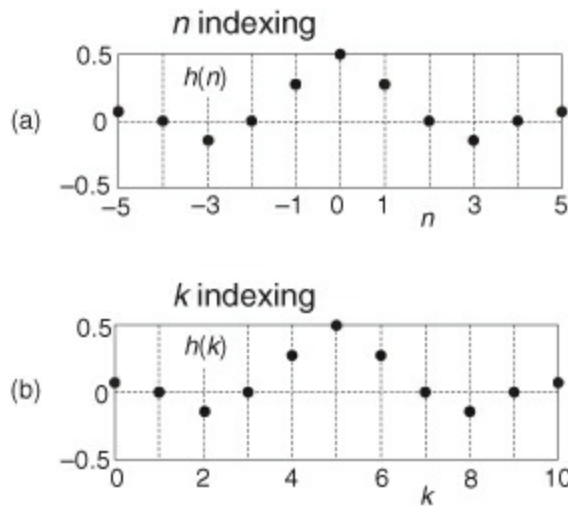
$$h_{\text{hilb}}(n) = 2\sin(\pi n/2)h(n).$$

Not emphasized in the literature, however, is the fact that the above $h_{\text{hilb}}(n)$ expression is only valid when the $h(n)$ coefficients' n index is in the range

$$-\frac{N-1}{2} \leq n \leq \frac{N-1}{2},$$

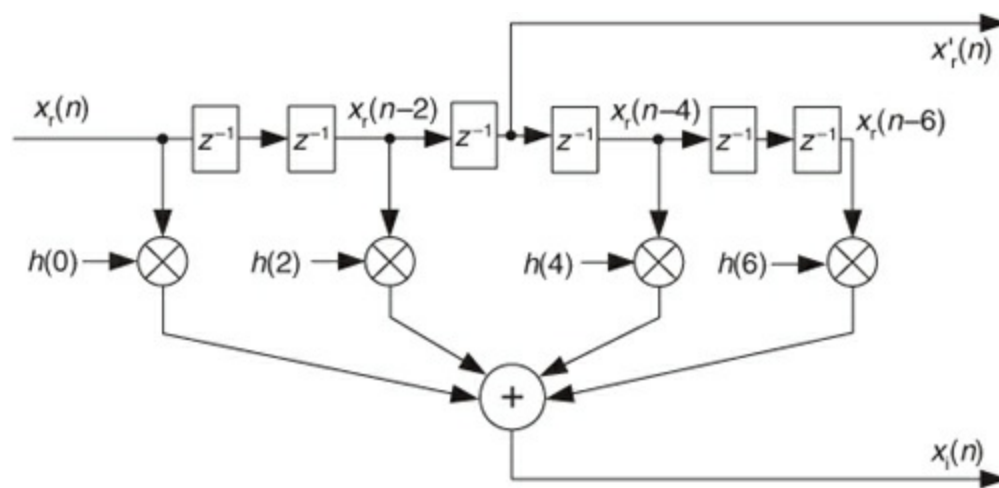
where $h(0)$ is the middle coefficient of $h(n)$ as shown for an 11-tap half-band FIR filter in [Figure P9-9\(a\)](#). In this textbook we typically use the more traditional FIR filter coefficient indexing notation of $k = 0, 1, 2, \dots, N-1$ as shown in [Figure P9-9\(b\)](#). What is the correct equation for the $h_{\text{hilb}}(k)$ coefficients when the half-band filter's coefficients are $h(k)$ with index k defined using our standard notation of $k = 0, 1, 2, \dots, N-1$ as shown in [Figure P9-9\(b\)](#)?

Figure P9-9



9.10 In [Section 9.4](#) we presented a 7-tap (6th-order) FIR Hilbert transformer filter, used to compute the complex sequence $x_c(n) = x_r(n) + jx_i(n)$ based on the real input sequence $x_r(n)$, as shown in [Figure P9-10](#).

Figure P9-10



Clever DSP engineers always seek to reduce the number of multipliers in their systems. Redesign the Hilbert transformer in [Figure P9-10](#) to a form that reduces the number of necessary multiplications per output sample. Draw the block diagram of your new design.

Hint: Write the difference equation for the $x_i(n)$ output sequence, and recall the symmetry relationships between the Hilbert transformer filter's coefficients.

9.11 Similar to the structure in the text's [Figure 9-12](#):

- Draw the block diagram of a tapped-delay line FIR Hilbert transformer filter, having 6 taps, that will generate both $x'_r(n)$ and $x_i(n)$.
- Comment on the practicality of this even-tap implementation relative to building a Hilbert transformer having an odd number of taps. (That is, is an even-tap or an odd-tap Hilbert transformer easier to implement?) Someday, if you're designing a Hilbert transformer for a real-world project, the answer to this question may be *very* important!

9.12 In this chapter we discussed the notion of building a tapped-delay line FIR filter whose coefficients are complex-valued. Explain why, or why not, such a filter's frequency magnitude response is periodic.

9.13 Because an FIR Hilbert transformer is a linear time-invariant (LTI) system, there are two ways to determine its frequency response. The first way is to derive an equation for the transformer's frequency response as a function of a continuous frequency variable, just as we did for IIR filters in [Section 6.4](#). Given that, we could plot that equation's magnitude and phase, using software, on our computer screen. The second method to determine a Hilbert transformer's frequency response is to perform the discrete Fourier transform (DFT) of the Hilbert transformer's unit impulse response, and plot the magnitude and phase of the DFT results.

- Derive the expression for the frequency response of a $(K-1)$ th-order (K taps) Hilbert transformer as a function of a continuous frequency variable ω in the range of $-\pi$ to $+\pi$ radians/sample.
- If you were to model your equation from Part (a) using software, over what range would you define your continuous frequency variable?
- The above Parts (a) and (b) were related to the algebraic method for finding a Hilbert transformer's frequency response in the form of an equation that is a function of a continuous frequency variable. We can also use the discrete Fourier transform (DFT) to estimate a Hilbert

transformer's frequency response. Provide the equation for the K -point DFT of a $(K-1)$ th-order Hilbert transformer's unit impulse response.

- (d) For a 43rd-order Hilbert transformer, how many frequency-domain samples would result from your Part (c) DFT equation? How can you increase the number of DFT frequency-domain samples to see finer detail (improved granularity) of the Hilbert transformer's frequency response?

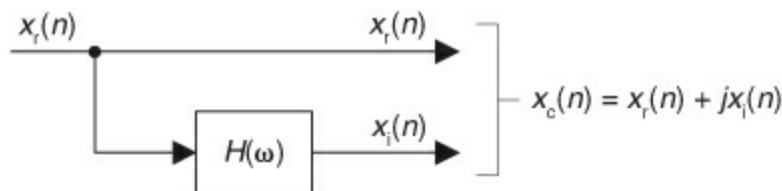
- 9.14 Determine the closed-form equation, as a function of the number of coefficients K , of the number of multipliers needed by a tapped-delay line FIR Hilbert transformer when K is an odd number. That is, write the expression for $N_{\text{mults}} = \text{some function of } K$. Do not consider the case of a *folded* delay line structure alluded to in the text.

Hint: Due to the unusual situation when $K - 1$ is an integer multiple of four, you'll have to use the mathematical notation of $\lfloor X \rfloor$ which means the "integer part of number X ."

- 9.15 Suppose we wish to implement the quadrature processing system in [Figure P9-15](#) where $x_r(n)$ is a real-valued sequence whose discrete-time Fourier transform is $X_r(\omega)$, and ω is our digital frequency ranging from $-\pi$ to π radians/sample. The sequences $x_r(n)$ and $x_i(n)$ are interpreted as the real and imaginary parts of a complex sequence $x_c(n) = x_r(n) + jx_i(n)$. Under the restriction that we want the discrete-time Fourier transform of $x_c(n)$ to be defined by

$$X_c(\omega) = \begin{cases} X_r(\omega), & \text{for } -\pi < \omega \leq 0 \\ 0, & \text{for } 0 < \omega < \pi, \end{cases}$$

Figure P9-15



fill in the following blank lines defining the $H(\omega)$ network's frequency response:

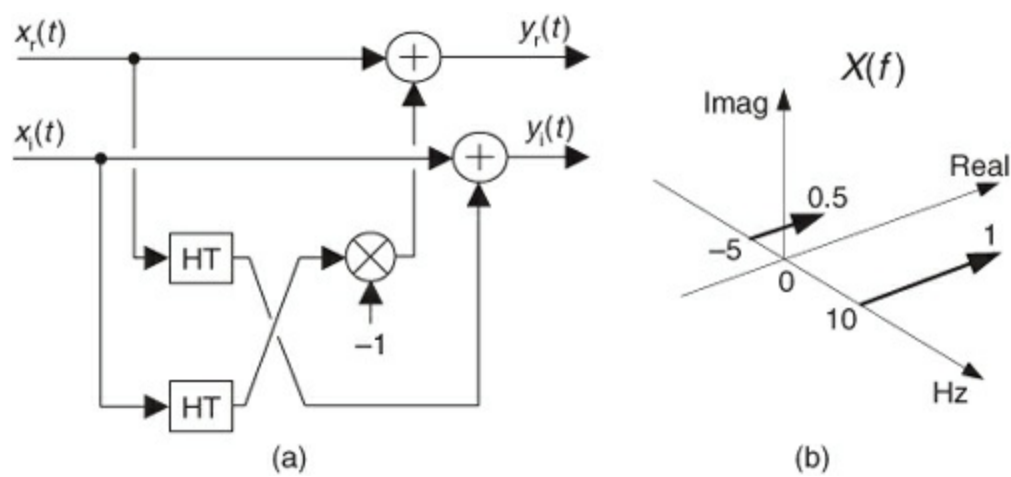
$$H(\omega) = \begin{cases} \underline{\hspace{1cm}}, & \text{for } -\pi < \omega \leq 0 \\ \underline{\hspace{1cm}}, & \text{for } 0 < \omega < \pi. \end{cases}$$

That is, what is $H(\omega)$ so that $X_c(\omega)$ is equal to $X_r(\omega)$ for negative frequencies, and equal to zero for positive frequencies. Show your work.

Hint: Begin by writing an expression for $X_c(\omega)$ in terms of $X_r(\omega)$ and $H(\omega)$.

- 9.16 Consider the network given in [Figure P9-16\(a\)](#) whose input is the complex $x(t) = x_r(t) + jx_i(t)$ signal, and whose output is the complex $y(t) = y_r(t) + jy_i(t)$ signal. The block labeled "H" represents a Hilbert transform. The $x(t)$ input has the spectrum shown by the bold arrows in [Figure P9-16\(b\)](#). Using a format similar to [Figure P9-16\(b\)](#), draw the three-dimensional spectrum of the $y(t)$ output.

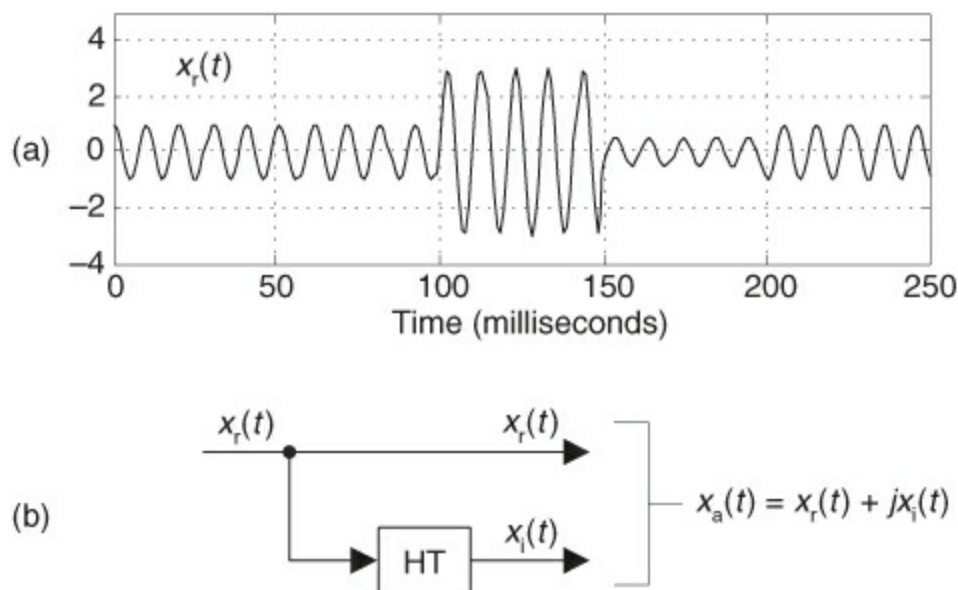
Figure P9-16



9.17 Consider the real-valued continuous $x_r(t)$ cosine signal shown in [Figure P9-17\(a\)](#). That cosine wave's peak amplitude fluctuates as follows:

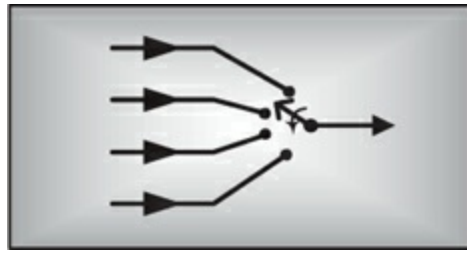
Peak amp.:	Time interval:
1.0	$0 \leq t < 100$ milliseconds
3.0	$100 \leq t < 150$ milliseconds
0.5	$150 \leq t < 200$ milliseconds
1.0	$200 \leq t \leq 250$ milliseconds

Figure P9-17



If we generate the analytic $x_a(t)$ signal, using a Hilbert transformer as shown in [Figure P9-17\(b\)](#), draw the time-domain magnitude of $x_a(t)$.

Chapter Ten. Sample Rate Conversion



The useful, and fascinating, process of sample rate conversion is a scheme for changing the effective sampling rate of a discrete time-domain signal sequence. We'll fully explain that puzzling notion in a moment. But first, know that sample rate conversion has many applications; it's primarily used to minimize computations by reducing signal data rates when a signal of interest's bandwidth has been narrowed by lowpass filtering. Sample rate conversion is mandatory in real-time processing when two separate hardware processors operating at two different sample rates must exchange digital signal data. In satellite and medical image processing, sample rate conversion is necessary for image enhancement, scale change, and image rotation. Sample rate conversion is also used to reduce the computational complexity of certain narrowband digital filters.

In this chapter we'll explore sample rate conversion by first looking at the process by way of a few examples. Then we introduce what are known as polyphase filters. With some knowledge under our belts, next we'll review the standard mathematical notation used to describe sample rate conversion. Finally, we'll examine the behavior of specialized digital filters that have found wide use in sample rate conversion applications.

We can define sample rate conversion as follows: Consider the process where a continuous signal $x(t)$ has been sampled at a rate of $f_{s,\text{old}} = 1/T_{\text{old}}$, and the discrete samples are $x_{\text{old}}(n) = x(nT_{\text{old}})$. Sample rate conversion is necessary when we need $x_{\text{new}}(n) = x(nT_{\text{new}})$, and direct sampling of the continuous $x(t)$ at the rate of $f_{s,\text{new}} = 1/T_{\text{new}}$ is not possible. For example, imagine we have an analog-to-digital (A/D) conversion system supplying a sample value every T_{old} seconds. But our processor can only accept data at a rate of one sample every T_{new} seconds. How do we obtain $x_{\text{new}}(n)$ directly from $x_{\text{old}}(n)$? One possibility is to digital-to-analog (D/A) convert the $x_{\text{old}}(n)$ sequence to regenerate the continuous $x(t)$ and then A/D convert $x(t)$ at a sampling rate of $f_{s,\text{new}}$ to obtain $x_{\text{new}}(n)$. Due to the spectral distortions induced by D/A followed by A/D conversion, this technique limits our effective dynamic range and is typically avoided in practice. Fortunately, accurate all-digital sample rate conversion schemes have been developed, as we shall see.

Sampling rate changes come in two flavors: rate decreases and rate increases. Decreasing the sampling rate is typically called *decimation*. When the sampling rate is being increased, the process is known as *interpolation*, i.e., computing intermediate sample values. Because decimation is the simpler of the two sample rate change operations, let's examine it first.

10.1 Decimation

Decimation is the two-step process of lowpass filtering followed by an operation known as *downsampling*. Let's first consider the notion of downsampling. We can downsample a sequence of sampled signal values by a factor of M by retaining every M th sample and discarding all the

remaining samples. Relative to the original sample rate, $f_{s,old}$, the sample rate of the downsampled sequence is

$$(10-1)$$

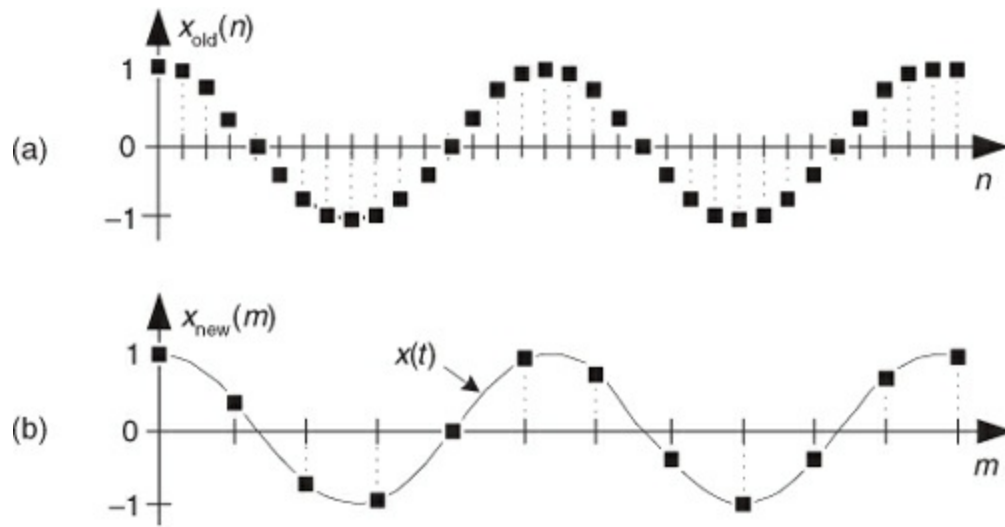
$$f_{s,new} = \frac{f_{s,old}}{M}.$$

For example, assume that an analog sinewave $x(t)$ has been sampled to produce the $x_{old}(n)$ sequence shown in [Figure 10-1\(a\)](#). To downsample $x_{old}(n)$ by a factor of $M = 3$, we retain $x_{old}(0)$ and discard $x_{old}(1)$ and $x_{old}(2)$, retain $x_{old}(3)$ and discard $x_{old}(4)$ and $x_{old}(5)$, retain $x_{old}(6)$, and so on as shown in [Figure 10-1\(b\)](#). Mathematically, we describe the downsampled sequence as

$$(10-1')$$

$$x_{new}(m) = x_{old}(Mm)$$

Figure 10-1 Sample rate conversion: (a) original sequence; (b) downsampled by $M = 3$ sequence.



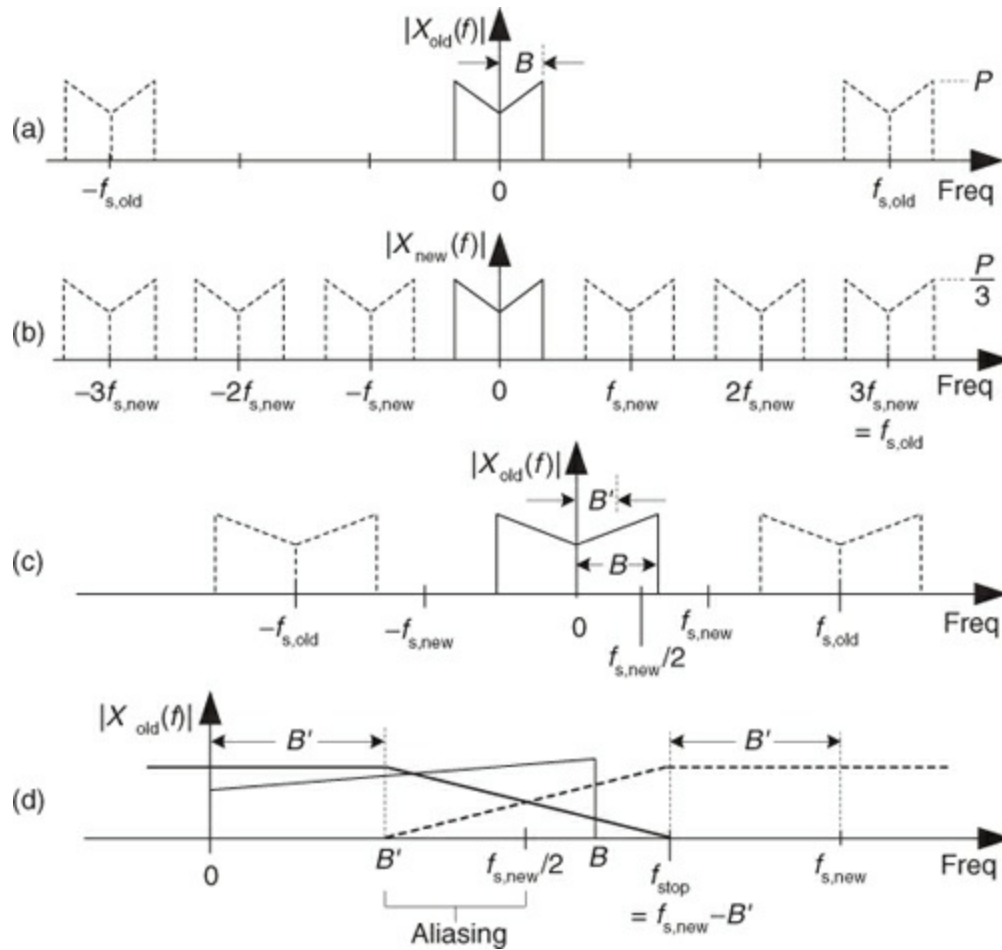
where $M = 3$, and $m = 0, 1, 2, 3$, etc.

Notice in [Eq. \(10-1'\)](#) that we're using an alternate time index variable m , rather than n , in $x_{new}(m)$ to remind us that the time period between the $x_{new}(m)$ samples is different from the time period between the $x_{old}(n)$ samples. That is, the absolute time instant corresponding to $m = 3$ is not equal to the absolute time instant corresponding to $n = 3$.

The spectral implications of downsampling are what we should expect as shown in [Figure 10-2](#), where the spectrum of an original band-limited sampled $x_{old}(n)$ signal is indicated by the solid lines, and the spectral replications are indicated by the dashed lines. With $x_{new}(m) = x_{old}(3n)$, $x_{new}(m)$'s spectrum, $X_{new}(f)$, is shown in [Figure 10-2\(b\)](#). Two important features are illustrated in [Figure 10-2](#). First, $X_{new}(f)$ could have been obtained directly by sampling the original continuous $x(t)$ signal at a rate of $f_{s,new}$, as opposed to downsampling $x_{old}(n)$ by a factor of three. Second, there is a limit to the amount of downsampling that can be performed relative to the bandwidth B of the original signal. We must ensure that $f_{s,new} > 2B$ to prevent overlapped spectral replications (aliasing errors) after downsampling.

Figure 10-2 Decimation by a factor of three: (a) spectrum of original $x_{old}(n)$ signal; (b) spectrum after

downsampling by three; (c) bandwidth B' is to be retained; (d) lowpass filter's frequency response relative to bandwidth B' .



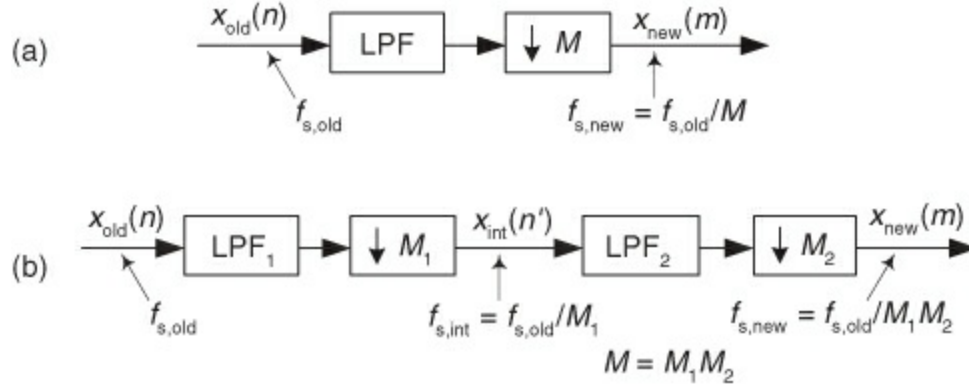
If a decimation application requires $f_{s,new}$ to be less than $2B$, then $x_{old}(n)$ must be lowpass filtered before the downsampling process is performed, as shown in [Figure 10-2\(c\)](#). (To clarify our terminology, we refer to decimation as the two-step process of lowpass filtering followed by downsampling.) If the original signal has a bandwidth B , and we're interested in retaining only the band B' , the signal spectrum above B' must be lowpass filtered, with full attenuation in the stopband beginning at f_{stop} , before the downsample-by- M process is performed. [Figure 10-2\(d\)](#) shows this in more detail where the frequency response of the lowpass filter, the bold lines, must attenuate the signal spectral components whose frequencies are greater than B' . Review the busy [Figure 10-2\(d\)](#) carefully and notice how the lowpass filter's f_{stop} frequency can be as high as $f_{stop} = f_{s,new} - B'$ and no spectral aliasing will occur in the B' band of interest.

In practice, the nonrecursive tapped-delay line FIR filter structure in [Figure 5-13](#) is the prevailing choice for [decimation filters](#) due to its linear phase response[1]. However, we need not apply $x_{old}(n)$ samples, one at a time, to an FIR lowpass filter and discard $M-1$ out of every M filter output samples. Instead, we could apply one $x_{old}(n)$ sample to the filter and compute an output sample, apply the next M consecutive $x_{old}(n)$ samples to the filter's delay line and compute the next output, and continue applying M consecutive $x_{old}(n)$ samples for each new filter output sample. That way we do not compute filter output samples that are discarded. In [Section 10.7](#) we'll learn how to minimize the number of filter multiplications needed.

10.2 Two-Stage Decimation

When the desired decimation factor M is large, say $M > 20$, there is an important feature of the filter/decimation process to keep in mind. Significant lowpass filter (LPF) computational savings may be had by implementing the single-stage decimation, shown in [Figure 10-3\(a\)](#), in two stages as shown in [Figure 10-3\(b\)](#). There we decimate sequence $x_{\text{old}}(n)$ by integer factor M_1 to produce the intermediate $x_{\text{int}}(n')$ sequence, which is then decimated by integer factor M_2 . The downsampling, sample rate decrease operation “ $\downarrow M_1$ ” in [Figure 10-3\(b\)](#) means discard all but every M_1 th sample. The product of M_1 and M_2 is our desired decimation factor; that is, $M = M_1 M_2$.

Figure 10-3 Decimation: (a) single-stage; (b) two-stage.



As a brief aside, the systems in [Figure 10-3](#) are called *multirate systems* because there are two or more different data sample rates within a single system.

10.2.1 Two-Stage Decimation Concepts

Considering [Figure 10-3\(b\)](#), an important question is “Given a desired total downsampling factor M , what should be the values of M_1 and M_2 to minimize the number of taps in lowpass filters LPF_1 and LPF_2 ?” If, for example, $M = 100$, should $M_1 M_2$ be $5 \cdot 20$, $20 \cdot 5$, $25 \cdot 4$, or maybe $10 \cdot 10$? Thankfully, thoughtful DSP pioneers answered this question for us[\[1\]](#). For two-stage decimation, the optimum value for M_1 is

(10-2)

$$M_{1,\text{opt}} \approx 2M \cdot \frac{1 - \sqrt{MF/(2-F)}}{2 - F(M+1)}$$

where F is the ratio of [Figure 10-3\(a\)](#)’s single-stage lowpass filter’s transition region width to that filter’s stopband frequency. That is,

(10-2’)

$$F = \frac{f_{\text{stop}} - B'}{f_{\text{stop}}}$$

After using [Eq. \(10-2\)](#) to determine the optimum $M_{1,\text{opt}}$ factor, and setting M_1 equal to the integer submultiple of M that is closest to $M_{1,\text{opt}}$, the second downsampling factor is

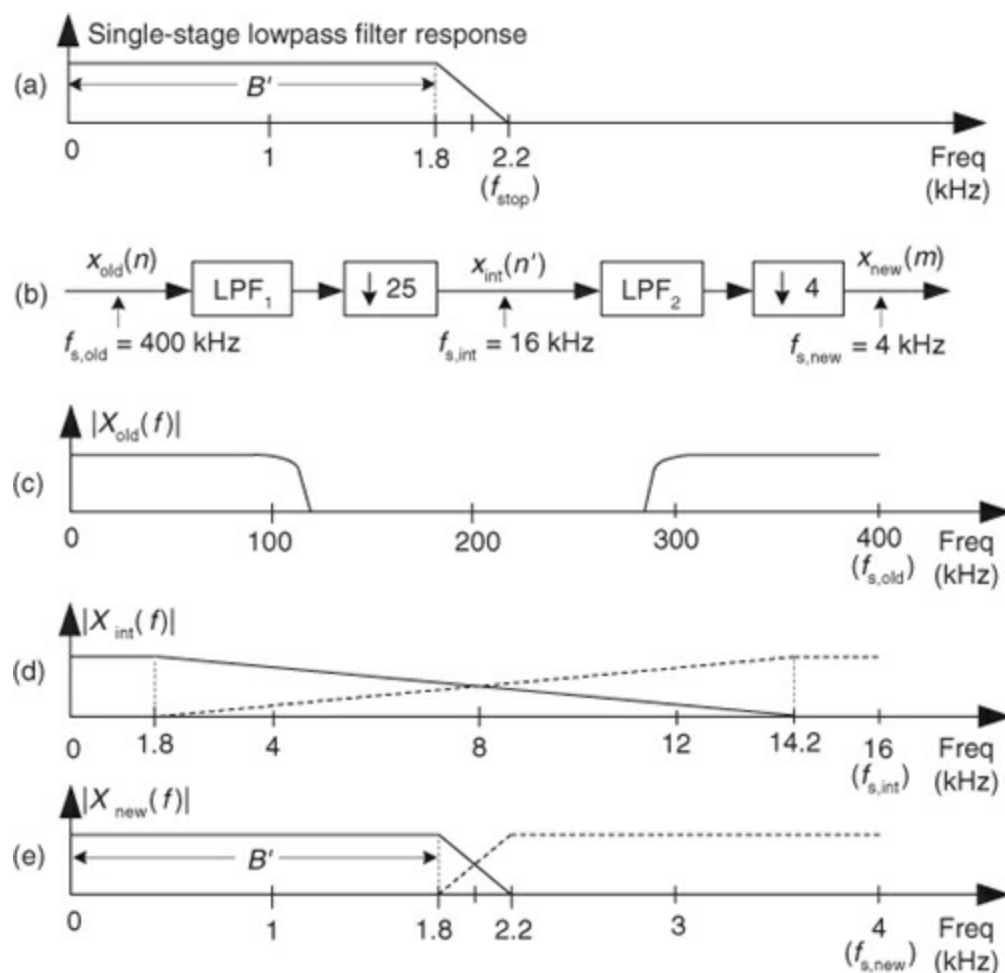
(10-2’')

$$M_2 = \frac{M}{M_1}$$

10.2.2 Two-Stage Decimation Example

By way of example, let's assume we have an $x_{\text{old}}(n)$ input signal arriving at a sample rate of 400 kHz, and we must decimate that signal by a factor of $M = 100$ to obtain a final sample rate of 4 kHz. Also, let's assume the baseband frequency range of interest is from 0 to $B' = 1.8$ kHz, and we want 60 dB of filter stopband attenuation. As such, a single-stage-decimation lowpass filter's frequency magnitude response is shown in [Figure 10-4\(a\)](#).

Figure 10-4 Two-stage decimation: (a) single-stage filter response; (b) decimation by 100; (c) spectrum of original signal; (d) output spectrum of the $M = 25$ downsampler; (e) output spectrum of the $M = 4$ downsampler.



So, with $f_{s,\text{new}} = 4$ kHz, we must filter out all $x_{\text{old}}(n)$'s signal energy above f_{stop} by having our filter transition region extend from 1.8 kHz to $f_{\text{stop}} = 4 - 1.8 = 2.2$ kHz. Now let's estimate the number of taps, N , required of a single-stage decimation-by-100 process. Using [Chapter 5's Eq. \(5-49\)](#), and the notation $f_{\text{pass}} = B' = 1.8$ kHz, we estimate the filter tap length to be

(10-3)

$$N \approx \frac{\text{Atten}}{22(f_{\text{stop}} - f_{\text{pass}})} = \frac{60}{22(2.2/400 - 1.8/400)} \approx 2727$$

taps. That's a painfully large number! (Resist all temptation to propose using a 2727-tap FIR filter in any system design review meeting at your company, or else you may be forced to update your résumé.)

Happily, to reduce the number of necessary filter taps we can partition our decimation problem into

two stages. With $M = 100$ and $F = (2200-1800)/2200 = 4/22$, [Eq. \(10-2\)](#) yields an optimum $M_{1,\text{opt}}$ downsample factor of 26.4. The integer submultiple of 100 closest to 26.4 is 25, so we set $M_1 = 25$. Next, from [Eq. \(10-2'\)](#), $M_2 = 4$ as shown in [Figure 10-4\(b\)](#).

In this two-stage decimation example we'll assume the original $X_{\text{old}}(f)$ input signal spectrum extends from zero Hz to something greater than 100 kHz as shown in [Figure 10-4\(c\)](#). If the first lowpass filter LPF_1 has a passband cutoff frequency of 1.8 kHz and its f_{stop} is defined as $f_{s,\text{int}} - B' = 16 - 1.8 = 14.2$ kHz, the output of the $M_1 = 25$ decimator will have the spectrum shown in [Figure 10-4\(d\)](#). When filter LPF_2 has a passband cutoff frequency of 1.8 kHz and its f_{stop} is set equal to $4 - 1.8 = 2.2$ kHz, the output of the $M_2 = 4$ decimator will have our desired spectrum shown in [Figure 10-4\(e\)](#). The point is, the total number of taps in the two lowpass filters, N_{total} , is greatly reduced from the 2727 taps needed by a single filter stage. From the expression in [Eq. \(10-3\)](#) for the combined LPF_1 and LPF_2 filters, the total number of two-stage filter taps is roughly

(10-3')

$$N_{\text{total}} = N_{\text{LPF1}} + N_{\text{LPF2}}$$

$$\approx \frac{60}{22(14.2/400 - 1.8/400)} + \frac{60}{22(2.2/16 - 1.8/16)} \approx 197.$$

This is an impressive computational savings, and it shows the kind of processing efficiency afforded by two-stage decimation[1,2]. Had we used $M_1 = 50$ and $M_2 = 2$ (or $M_1 = 10$ and $M_2 = 10$) in our decimation-by-100 example, the total number of two-stage filter taps would have been greater than 250. Thus $M_1 = 25$ and $M_2 = 4$ is the better choice.

10.2.3 Two-Stage Decimation Considerations

The multistage decimation design curves in reference [1] tell us that, for computational efficiency reasons, it's always to our benefit to decimate in order from the largest to the smallest factor. That is, we make sure that M_1 is greater than M_2 .

In two-stage decimation applications it is advantageous to consider setting the M_1 and M_2 decimation factors equal to integer powers of two because we can use computationally efficient half-band filters for the lowpass filters in [Figure 10-4\(b\)](#). We discuss the use of multirate half-band filtering later in [Section 10.11](#).

There are two practical issues to consider for two-stage decimation. First, as we discussed regarding cascaded filters in [Section 6.8.1](#), if the dual-filter system in [Figure 10-4\(b\)](#) is required to have a passband peak-peak ripple of R dB (R decibels), then both filters must be designed to have a passband peak-peak ripple of no greater than $R/2$ dB. Second, the number of multiplications needed to compute each $x_{\text{new}}(m)$ output sample in [Figure 10-4\(b\)](#) is much larger than N_{total} because we must compute so many LPF_1 and LPF_2 output samples destined to be discarded. Later we'll introduce an efficient decimation filter implementation scheme called *polyphase decomposition* that only requires N_{total} multiplications per $x_{\text{new}}(m)$ output sample.

The advantages of two-stage decimation, over single-stage decimation, are

- an overall reduction in computational workload,
- reduced signal and filter coefficient data storage,
- simpler filter designs, and
- a decrease in the ill effects of finite binary-word-length filter coefficients.

These advantages become more pronounced as the overall desired decimation factor M becomes larger. To conclude our two-stage decimation discussion, be aware that reference [\[3\]](#) discusses aspects of multistage decimation where the number of stages is greater than two.

10.3 Properties of Downsampling

Let us now quickly review several interesting aspects of downsampling a discrete sequence (retaining every M th sample and discarding all the remaining samples).

10.3.1 Time and Frequency Properties of Downsampling

First, we realize that downsampling is one of those rare processes that is not time invariant. From the very nature of its operation, we know if we delay the input sequence by one sample, a downsampler will generate an entirely different output sequence. For example, if we apply an input sequence $x(n) = x(0), x(1), x(2), x(3), x(4)$, etc., to a downsampler and $M = 3$, the output $y(m)$ will be the sequence $x(0), x(3), x(6)$, etc. Should we delay the input sequence by one sample, our delayed $x_d(n)$ input would be $x(1), x(2), x(3), x(4), x(5)$, etc. In this case the downsampled output sequence $y_d(m)$ would be $x(1), x(4), x(7)$, etc., which is *not* a delayed version of $y(m)$. Thus a downsampler is not time invariant. What this means is that if a downsampling operation is in cascade with other operations, we are not permitted to swap the order of any of those operations and the downsampling process without modifying those operations in some way. We first discussed this notion of time invariance in [Section 1.7](#), and we'll see an example of it in [Section 10.13](#).

Second, downsampling does not cause time-domain signal amplitude loss. A sinusoid with a peak-peak amplitude of 10 retains this peak-peak amplitude after downsampling. However, downsampling by M does induce a magnitude loss by a factor of M in the frequency domain. That's because, as we learned in [Chapter 3](#), DFT magnitudes are proportional to the number of time-domain samples used in the transformation.

10.3.2 Drawing Downsampled Spectra

To illustrate the frequency properties of downsampling, let's review an algorithm (a recipe) that tells us how to draw the spectrum of a downsampled signal. Drawing the spectrum of a downsampled lowpass signal is easy; we saw that in [Figures 10-2\(a\)](#) and [10-2\(b\)](#). However, drawing the spectra of bandpass and highpass signals that have been downsampled can be a bit tricky. Here's the process I use to draw the spectra of any type of downsampled signal.

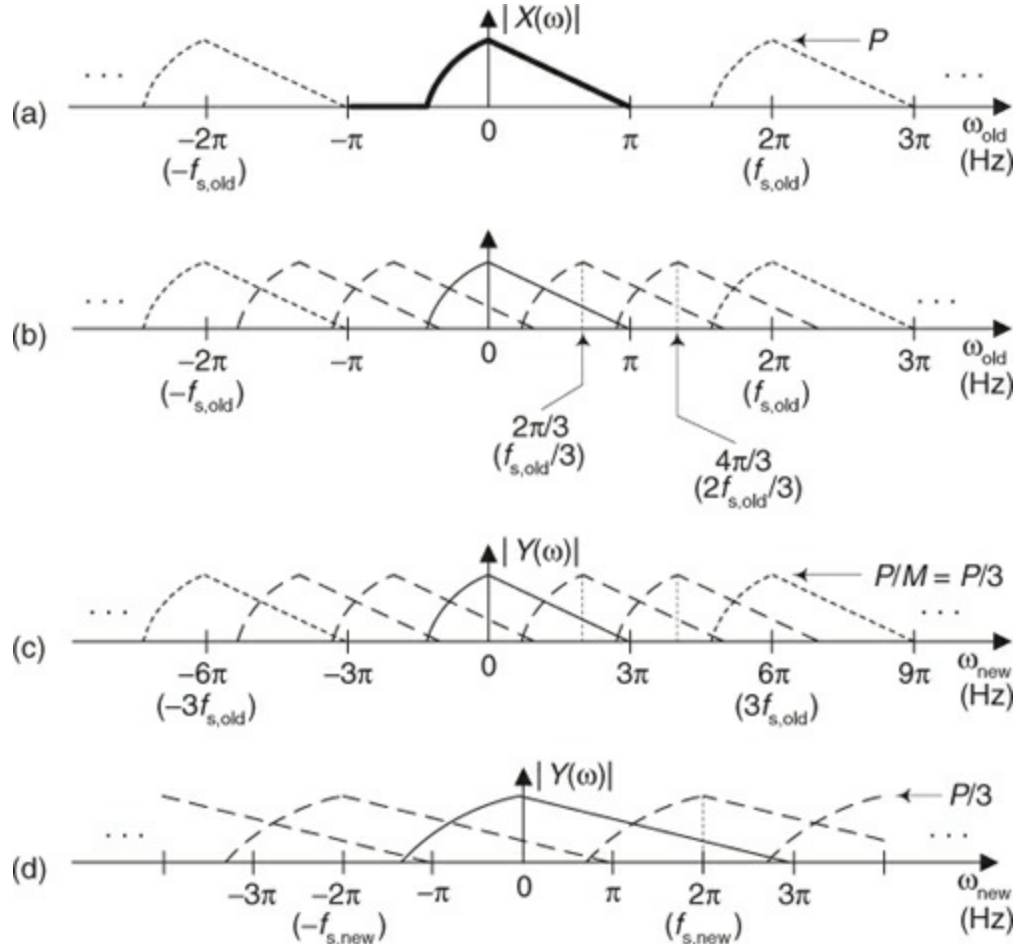
We begin by looking at the spectral magnitude, $|X(\omega)|$ in [Figure 10-5\(a\)](#), of an $x(n)$ time signal containing spectral energy at both low and high frequencies. To help clarify our discussion by making the associated spectra (we hope) easier to interpret, we use a complex-valued lowpass $x(n)$ for this example. Regarding [Figure 10-5\(a\)](#), notice the following:

- The baseband spectral envelope of $|X(\omega)|$ is centered at zero Hz covering the frequency range of $-\pi \leq \omega_{\text{old}} \leq \pi$ radians/sample ($-f_{s,\text{old}}/2$ to $f_{s,\text{old}}/2$ Hz), shown by the bold solid curve. Frequency $f_{s,\text{old}}$

is the original sample rate of $x(n)$, measured in Hz.

- For clarity, and reference, we label the frequency axis in both radians/sample and Hz.
- The spectral replications in $|X(\omega)|$ are shown by the short-dashed curves, spaced at integer multiples of 2π radians/sample ($f_{s,\text{old}}$ Hz).
- $|X(\omega)|$ has a peak magnitude of P .

Figure 10-5 Spectra associated with downsampling by $M = 3$.



Assuming we want to downsample $x(n)$ by a factor of $M = 3$ to create a $y(m)$ sequence, the following steps show how to determine the $|Y(\omega)|$ spectrum based on the known $|X(\omega)|$:

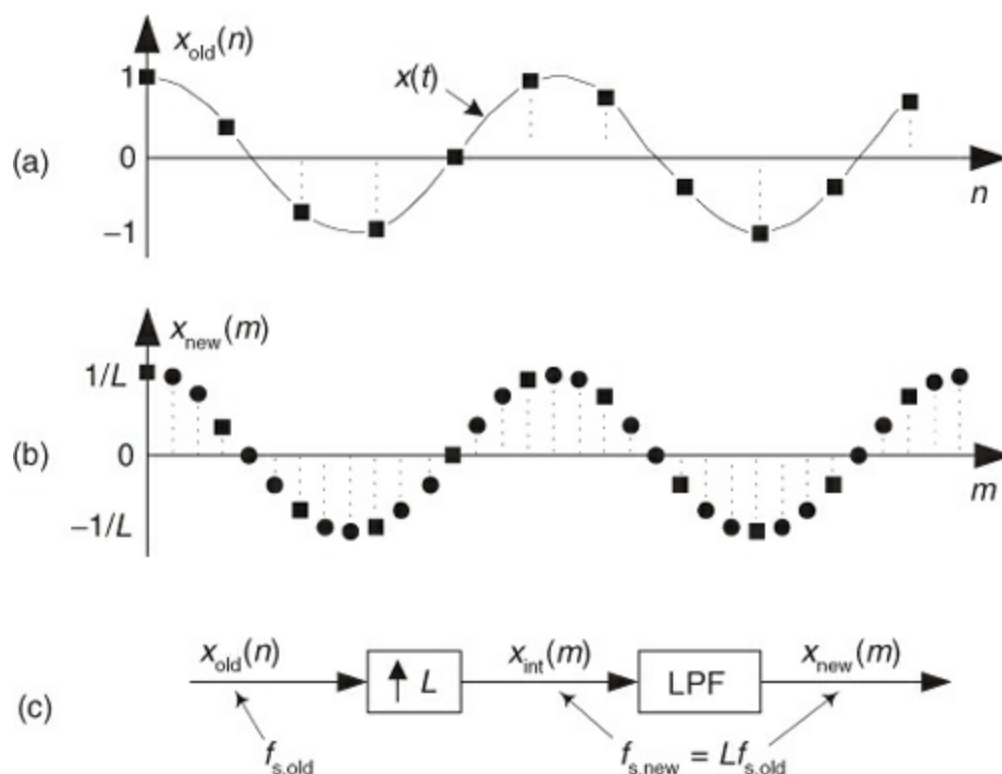
1. Draw the $|X(\omega)|$ spectrum of sequence $x(n)$ showing at least one spectral replication in both the positive- and negative-frequency directions. We did that in [Figure 10-5\(a\)](#).
2. Insert $M-1$ equally spaced copies of the primary spectral envelope between the primary spectral envelope and the spectral replications centered at $\omega_{\text{old}} = \pm 2\pi$. The spectral spacing of the $M-1$ inserted copies should be multiples of $2\pi/M$ radians/sample as shown by the long-dashed curves in [Figure 10-5\(b\)](#).
3. Scale upward the frequency axis values of $|Y(\omega)|$ by a factor of M , yielding the new ω_{new} frequency axis variable as shown in [Figure 10-5\(c\)](#).
4. Finally, scale downward the vertical axis of $|Y(\omega)|$ by a factor of $1/M$. This produces a peak magnitude for $|Y(\omega)|$ of P/M as shown in [Figure 10-5\(c\)](#).

We zoom in on the $|Y(\omega)|$ spectrum in [Figure 10-5\(d\)](#) to show enhanced detail.

10.4 Interpolation

As we said before, downsampling is only part of the sample rate conversion story—let’s now consider interpolation. Sample rate increase by interpolation is a bit more involved than decimation because with interpolation new sample values need to be calculated. Conceptually, interpolation comprises the generation of a continuous $x(t)$ curve passing through our $x_{\text{old}}(n)$ sampled values, as shown in [Figure 10-6\(a\)](#), followed by sampling that curve at the new sample rate $f_{\text{s,new}}$ to obtain the interpolated sequence $x_{\text{new}}(m)$ in [Figure 10-6\(b\)](#). Of course, continuous curves cannot exist inside a digital machine, so we’re forced to obtain $x_{\text{new}}(m)$ directly from $x_{\text{old}}(n)$. To increase a given $f_{\text{s,old}}$ sample rate by an integer factor of L we must insert $L-1$ zero-valued samples between each sample in $x_{\text{old}}(n)$, creating a longer-length sequence. To the end of that longer sequence we append $L-1$ zero-valued samples. Those two steps are what we call *upsampling*, indicated by the “ $\uparrow L$ ” operation in [Figure 10-6\(c\)](#). Next, we apply the upsampled sequence to a lowpass filter whose output is the interpolated sequence in [Figure 10-6\(b\)](#).

Figure 10-6 Interpolation: (a) original time sequence; (b) interpolated by $L = 3$ sequence; (c) interpolation functional notation.

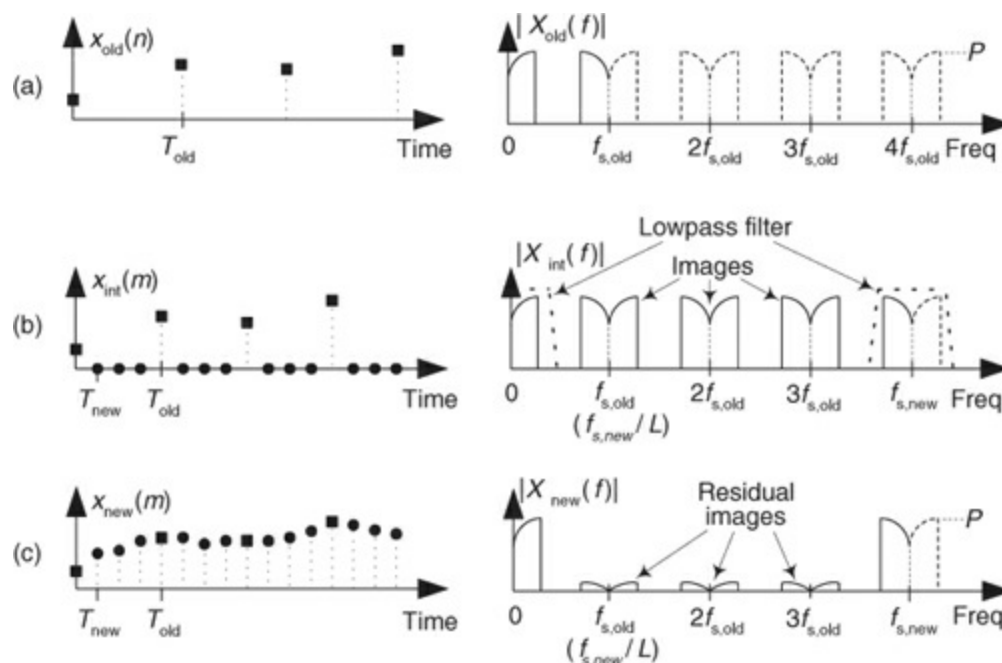


We formally refer to interpolation as the two-step process of upsampling followed by lowpass filtering. The process of interpolation is beautifully straightforward and best understood by way of an example.

Let’s assume we have the sequence $x_{\text{old}}(n)$, part of which is shown in [Figure 10-7\(a\)](#), and we want to increase its sample rate by a factor of $L = 4$. The $x_{\text{old}}(n)$ sequence’s spectrum is provided in [Figure 10-7\(a\)](#) where the signal spectrum between zero Hz and $4f_{\text{s,old}}$ is shown. Please notice that the dashed curves in $X_{\text{old}}(f)$ are spectral replications. To upsample $x_{\text{old}}(n)$ by a factor of four, we insert three zeros between each sample of $x_{\text{old}}(n)$ and append the last three zeros, as shown in [Figure 10-7\(b\)](#), to create the new intermediate sequence $x_{\text{int}}(m)$. Notice that the old sequence is embedded in the new

sequence. The insertion of the zeros (a process often called *zero stuffing*) establishes the sample index for the intermediate sequence $x_{\text{int}}(m)$ where the interpolated values will be assigned.

Figure 10-7 Interpolation by four: (a) original sampled sequence and its spectrum; (b) zeros inserted in original sequence and resulting spectrum; (c) output sequence of interpolation filter and final spectrum.



The spectrum of $x_{\text{int}}(m)$, $X_{\text{int}}(f)$, is shown in [Figure 10-7\(b\)](#) where $f_{s,\text{new}} = 4f_{s,\text{old}}$. The solid curves in $X_{\text{int}}(f)$, centered at multiples of $f_{s,\text{old}}$, are called *images*. What we've done by adding the zeros is merely increase the effective sample frequency to $f_s = f_{s,\text{new}}$ in [Figure 10-7\(b\)](#). The final step in interpolation is to filter the $x_{\text{int}}(m)$ sequence with the lowpass filter shown in [Figure 10-6\(c\)](#). That filter's frequency magnitude response is crudely shown as the dashed lines centered at zero Hz, and $f_{s,\text{new}}$ Hz, in [Figure 10-7\(b\)](#). The lowpass filter's job is to attenuate the spectral images shown in [Figure 10-7\(b\)](#). This lowpass filter is called an *interpolation filter*, and its output sequence is the desired $x_{\text{new}}(m)$ sequence in [Figure 10-7\(c\)](#) having the spectrum $X_{\text{new}}(f)$ containing residual spectral images. We'll discuss those residual images in a moment.

10.5 Properties of Interpolation

Here we discuss several important aspects of the interpolation (upsampling followed by lowpass filtering) process depicted in [Figure 10-7](#).

10.5.1 Time and Frequency Properties of Interpolation

Because we cannot implement an ideal lowpass interpolation filter, $x_{\text{new}}(m)$ will not be an exact interpolation of $x_{\text{old}}(n)$. The error manifests itself as the residual spectral images in $X_{\text{new}}(f)$ as indicated in [Figure 10-7\(c\)](#). With an ideal filter, these images would not exist, but we can only approximate an ideal lowpass interpolation filter. The issue to remember is that the accuracy of our entire interpolation process depends on the stopband attenuation of our lowpass filter. The greater the stopband attenuation, the more accurate the interpolation. As with decimation, interpolation can be thought of as an exercise in lowpass filter design.

Note that our interpolation process, because of the zero-valued samples, has an inherent amplitude

loss factor of L when a unity-gain lowpass filter is used. That is, the peak sample value of $x_{\text{new}}(m)$ is equal to the peak sample value of $x_{\text{old}}(n)$ divided by L . Thus, to achieve unity gain between sequences $x_{\text{old}}(n)$ and $x_{\text{new}}(m)$, the lowpass interpolation filter must have a gain of L at zero Hz.

Although there is a time-domain gain (amplitude) loss of L by upsampling and filtering, that loss is canceled in the discrete frequency domain by the L -fold gain in the magnitudes of the discrete Fourier transform (DFT) of an $x_{\text{new}}(m)$ sequence that is L times longer in duration than the original $x_{\text{old}}(n)$ time sequence. (We're repeating a fact we learned in [Chapter 3](#)—DFT magnitudes are proportional to the length of the time sequence being transformed.)

Rather than perform the upsampling in [Figure 10-7\(b\)](#), we might be inclined to merely repeat each $x_{\text{old}}(n)$ sample three times to generate the new upsampled $x_{\text{int}}(m)$ sequence. Such a maneuver would indeed help attenuate the unwanted spectral images, but sadly the resulting low-frequency $X_{\text{int}}(m)$ spectral magnitude shape will be the original desired $X_{\text{old}}(m)$ spectrum multiplied by a $\sin(x)/x$ function. If this happens, then the follow-on lowpass filter must compensate for that spectral magnitude roll-off distortion. Such non-flat passband $\sin(x)/x$ -compensation filters require so many additional taps that the “repeat each $x_{\text{old}}(n)$ sample” scheme is unwise. In fact, later we'll discuss an efficient interpolation filtering scheme called *polyphase filtering* wherein we don't bother to create the upsampled $x_{\text{int}}(m)$ sequence at all.

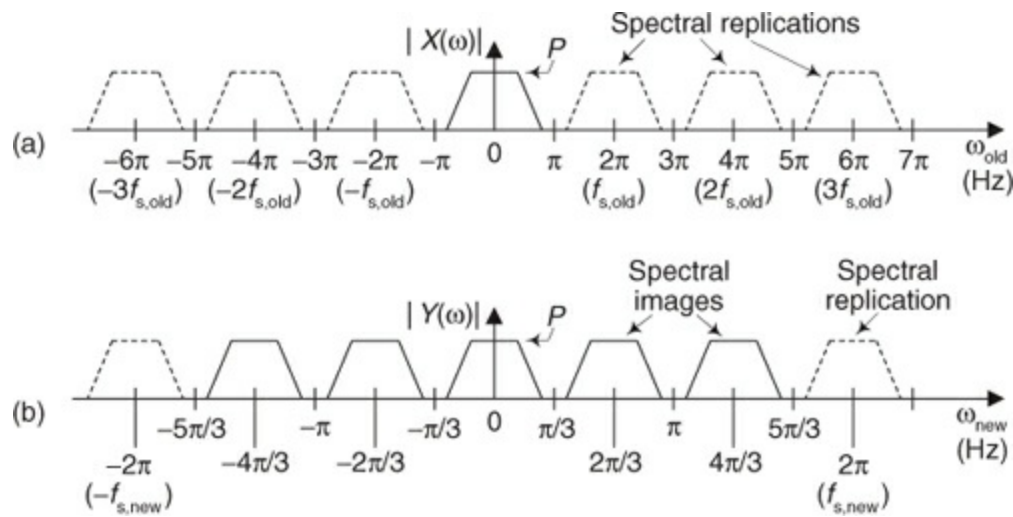
There's one further issue regarding interpolation. You might tend to think that interpolation was born of our modern-day signal processing applications such as cell phones and compact disc players. Please don't. Ancient astronomical cuneiform tablets, originating from Uruk and Babylon (200 years before the birth of Jesus), indicate that linear interpolation was used to fill in the missing tabulated positions of celestial bodies for those times when atmospheric conditions prevented direct observation[\[4\]](#). Interpolation has been used ever since, for *filling in* missing data.

10.5.2 Drawing Upsampled Spectra

To illustrate the frequency properties of upsampling (insertion of zero-valued samples), and to demonstrate the method for drawing the spectra of upsampled signals, consider the spectral magnitude, $|X(\omega)|$ in [Figure 10-8\(a\)](#), of a lowpass $x(n)$ time signal. Regarding [Figure 10-8\(a\)](#), notice the following:

- The baseband spectral envelope of $|X(\omega)|$ is centered at zero Hz covering the frequency range of roughly $-\pi \leq \omega_{\text{old}} \leq \pi$ radians/sample ($-f_{s,\text{old}}/2$ to $f_{s,\text{old}}/2$ Hz), shown by the solid lines. Frequency $f_{s,\text{old}}$ is the original sample rate of $x(n)$, measured in Hz.
- For clarity, and reference, we label the frequency axis in both radians/sample and Hz.
- The spectral replications in $|X(\omega)|$ are shown by the dashed-line spectral envelopes spaced at integer multiples of 2π radians/sample ($f_{s,\text{old}}$ Hz).
- $|X(\omega)|$ has a peak magnitude of P .

Figure 10-8 Spectra associated with upsampling by $L = 3$.



Assuming we want to upsample $x(n)$ by a factor of $L = 3$, for example, to create a $y(m)$ sequence, the following steps show how to determine the $|Y(\omega)|$ spectrum of $y(m)$:

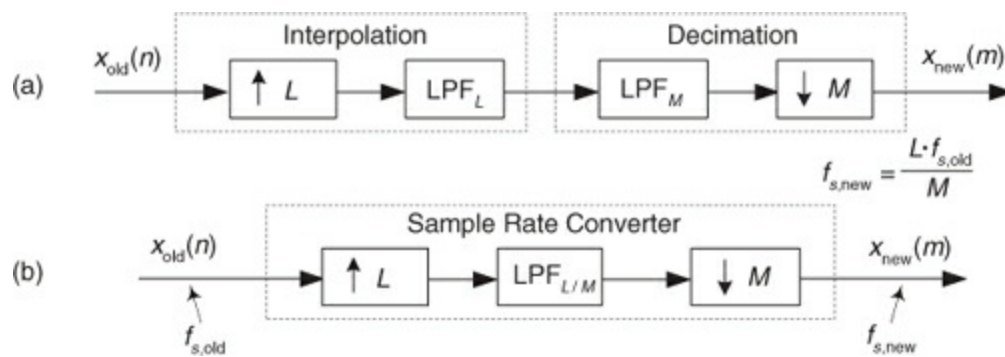
1. Draw the $|X(\omega)|$ spectrum of sequence $x(n)$ showing at least $L = 3$ spectral replications in both the positive- and negative-frequency directions. We did that in [Figure 10-8\(a\)](#).
2. Scale downward the frequency axis values of $X(\omega)$ by a factor of L , yielding the new ω_{new} frequency variable as shown in [Figure 10-8\(b\)](#).
3. Finally, indicate the spectral images (destined to be attenuated by subsequent lowpass filtering) by using solid lines to represent their spectral envelopes, as we did in [Figure 10-8\(b\)](#).

10.6 Combining Decimation and Interpolation

Although changing sampling rates, through decimation or interpolation, by integer factors is quite common in practice, what can we do if we need a sample rate change that is not an integer? The good news is that we can implement sample rate conversion by any rational fraction L/M with interpolation by an integer factor L followed by decimation by an integer factor M . Because the ratio L/M can be obtained as accurately as we want, with the correct choice of integers L and M , we can change sample rates by almost any factor in practice. For example, a sample rate increase by a factor of 7.125 can be performed by an interpolation by $L = 57$ followed by a decimation by $M = 8$, because $7.125 = 57/8$.

This L/M sample rate change is illustrated as the processes shown in [Figure 10-9\(a\)](#). The neat part here is that the computational burden of changing the sample rate by the ratio of L/M is less than the sum of an individual interpolation followed by an individual decimation. That's because we can combine the interpolation filter LPF_L and the decimation filter LPF_M into a single filter shown as $\text{LPF}_{L/M}$ in [Figure 10-9\(b\)](#). The process in [Figure 10-9\(b\)](#) is normally called a *sample rate converter* because if $L > M$ we have interpolation, and when $M > L$ we have decimation. (The filter $\text{LPF}_{L/M}$ is often called a *multirate filter*.)

Figure 10-9 Sample rate conversion by a rational factor: (a) combination interpolation/decimation; (b) single lowpass filter method.



Filter $LPF_{L/M}$ must sufficiently attenuate the interpolation spectral images so they don't contaminate our desired signal beyond acceptable limits after decimation. To accomplish this task, lowpass filter $LPF_{L/M}$ must attenuate all spectral components whose frequencies are above $f_{s,old}/2$ or $(f_{s,old}/2) \cdot (L/M)$, whichever is smaller, where $f_{s,old}$ is $x_{old}(n)$'s sample rate in Hz. The stopband attenuation of $LPF_{L/M}$ must be great enough that the attenuated upsampled images do not induce intolerable levels of noise when they're aliased by downsampling by M into the final band of 0 to $f_{s,new}/2$ Hz, where $f_{s,new}$ is the filter's data rate, in Hz.

Again, our interpolator/decimator designs are exercises in lowpass filter design, and all the knowledge and tools we have to design lowpass filters can be applied to this task. In software interpolator/decimator design, we want our lowpass filter algorithm to prevent aliasing images and be fast in execution time. For hardware interpolator/decimators, we strive to implement designs optimizing the conflicting goals of high performance (minimum spectral aliasing), simple architecture, high data throughput speed, and low power consumption.

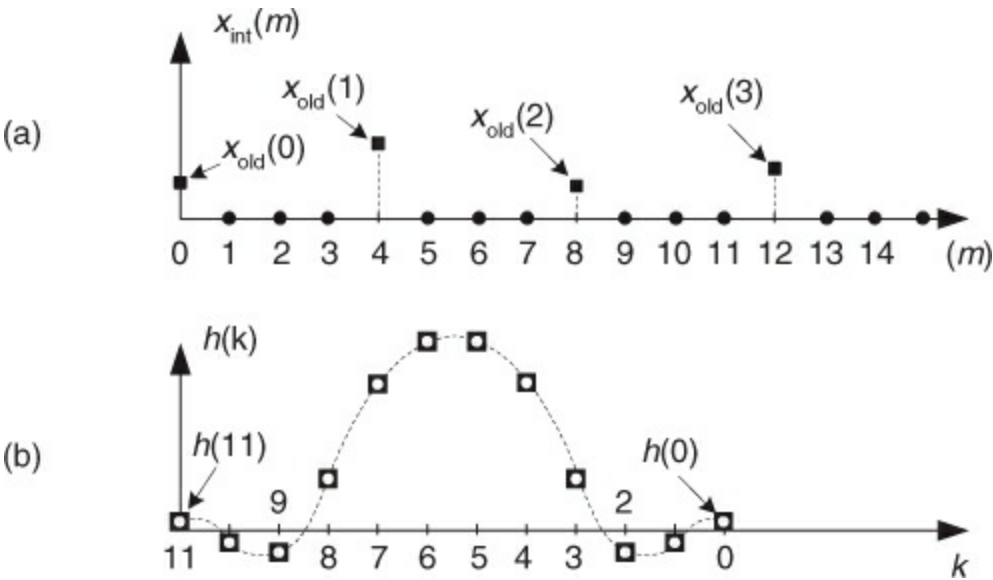
The filtering computational workload in rational-factor sample rate conversion, as we've presented it here, is sadly inefficient. Think about interpolating a signal sequence by a factor of 4/3; we'd insert the zero-valued samples into the original time sequence and apply it to a lowpass filter. Three-fourths of the filter multiplication products would necessarily be zero. Next, we'd discard two-thirds of our computed filter output values. Very inefficient! Fortunately, we are now prepared to introduce special sample rate conversion filters, called *digital polyphase* filters, that avoid these computational inefficiencies.

10.7 Polyphase Filters

In this section we introduce the fascinating, and exceedingly useful, subject of digital polyphase FIR filters. These filters have the ability to eliminate all multiply by zero operations in interpolation, as well as avoid the wasteful computation of filter output samples that are subsequently discarded in decimation applications.

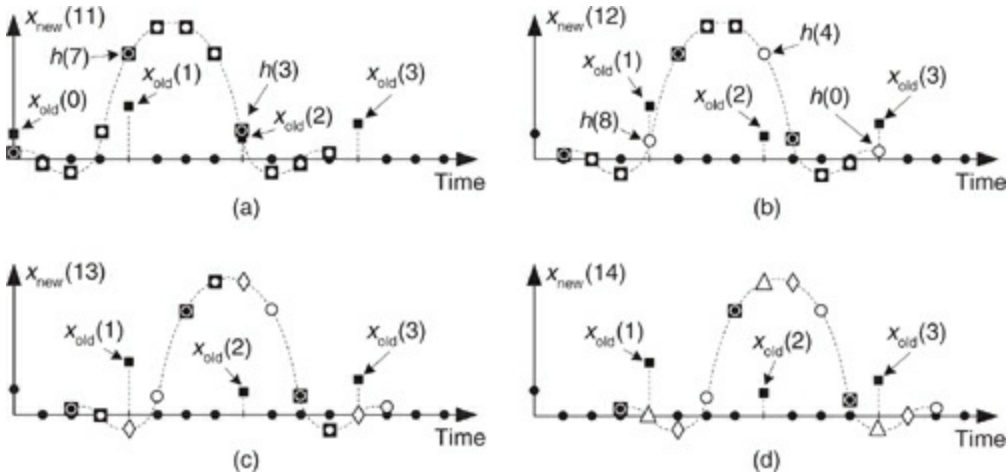
Let's assume that a linear-phase FIR interpolation filter design requires an $N = 12$ -tap filter; our initial plan is to pass the upsampled by $L = 4$ $x_{int}(m)$ sequence in [Figure 10-10\(a\)](#) through the 12-tap FIR filter coefficients shown in [Figure 10-10\(b\)](#) to obtain the desired $x_{new}(m)$ sequence. (This filter, whose coefficients are the $h(k)$ sequence, is often called the *prototype* FIR filter. That's because later we're going to modify it.) Notice that with time advancing to the right in [Figure 10-10\(a\)](#), the filter coefficients are in reversed order as shown in [Figure 10-10\(b\)](#). This filtering requires 12 multiplications for each $x_{new}(m)$ output sample, with 9 of the products always being zero. As it turns out, we need not perform all 12 multiplications.

Figure 10-10 Interpolation by four with a 12-tap lowpass FIR filter: (a) filter input samples; (b) filter coefficients, \blacksquare s, used to compute $x_{\text{new}}(m)$.



To show this by way of an example, [Figure 10-11\(a\)](#) shows the $x_{\text{int}}(m)$ samples just filling the filter’s delay line so that we can compute the $x_{\text{new}}(m=11)$ output sample. The 12 filter coefficients are indicated by the \blacksquare symbols.

Figure 10-11 Filter coefficients used to calculate various $x_{\text{new}}(m)$ samples.



With the dots in [Figure 10-11\(a\)](#) representing the $x_{\text{int}}(m)$ sequence, we see that although there are nine \blacksquare s and three \blacksquare s, only the three \blacksquare s generate nonzero products contributing to the convolution sum $x_{\text{new}}(11)$. Those three \blacksquare s represent FIR filter coefficients $h(3)$, $h(7)$, and $h(11)$. The issue here is that we need not perform the multiplications associated with the zero-valued samples in $x_{\text{int}}(m)$. We only need to perform three multiplications to obtain $x_{\text{new}}(11)$. To see the polyphase concept, remember that we use the prototype filter coefficients indicated by the \blacksquare s to compute $x_{\text{new}}(12)$. When we slide the filter’s impulse response to the right one sample, we use the coefficients indicated by the circles, in [Figure 10-11\(b\)](#), to calculate $x_{\text{new}}(12)$ because the nonzero values of $x_{\text{int}}(m)$ will line up under the circled coefficients. Those circles represent filter coefficients $h(0)$, $h(4)$, and $h(8)$.

Likewise, when we slide the impulse response to the right one more sample to compute $x_{\text{new}}(13)$, we use the coefficients indicated by the diamonds in [Figure 10-11\(c\)](#). Finally, we slide the impulse response to the right once more and use the coefficients indicated by the triangles in [Figure 10-11\(d\)](#).

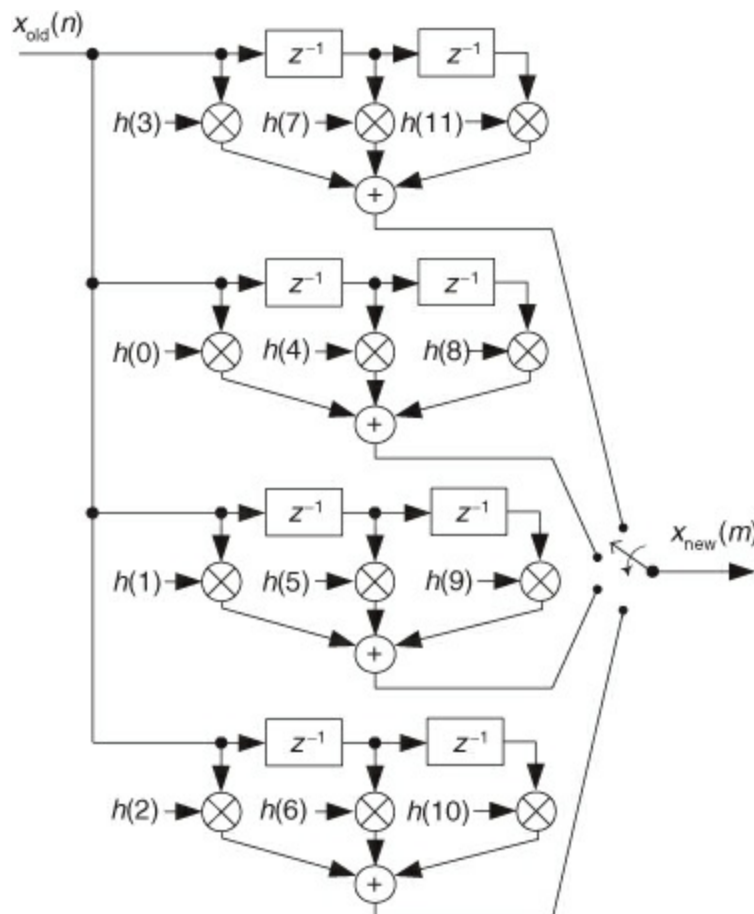
to compute $x_{\text{new}}(14)$. Sliding the filter's impulse response once more to the right, we would return to using the coefficients indicated by the \blacksquare s to calculate $x_{\text{new}}(15)$. You can see the pattern here—there are $L = 4$ different sets of coefficients used to compute $x_{\text{new}}(m)$ from the $x_{\text{old}}(n)$ samples. Each time a new $x_{\text{new}}(m)$ sample value is to be computed, we rotate one step through the four sets of coefficients and calculate as

$x_{\text{new}}(11) = h(3)x_{\text{old}}(2) + h(7)x_{\text{old}}(1) + h(11)x_{\text{old}}(0)$	← uses the \blacksquare coefficients
$x_{\text{new}}(12) = h(0)x_{\text{old}}(3) + h(4)x_{\text{old}}(2) + h(8)x_{\text{old}}(1)$	← uses the \circ coefficients
$x_{\text{new}}(13) = h(1)x_{\text{old}}(3) + h(5)x_{\text{old}}(2) + h(9)x_{\text{old}}(1)$	← uses the \diamond coefficients
$x_{\text{new}}(14) = h(2)x_{\text{old}}(3) + h(6)x_{\text{old}}(2) + h(10)x_{\text{old}}(1)$	← uses the Δ coefficients
$x_{\text{new}}(15) = h(3)x_{\text{old}}(3) + h(7)x_{\text{old}}(2) + h(11)x_{\text{old}}(1)$	← uses the \blacksquare coefficients
$x_{\text{new}}(16) = h(0)x_{\text{old}}(4) + h(4)x_{\text{old}}(3) + h(8)x_{\text{old}}(2)$	← uses the \circ coefficients
$x_{\text{new}}(17) = h(1)x_{\text{old}}(4) + h(5)x_{\text{old}}(3) + h(9)x_{\text{old}}(2)$	← uses the \diamond coefficients
$x_{\text{new}}(18) = h(2)x_{\text{old}}(4) + h(6)x_{\text{old}}(3) + h(10)x_{\text{old}}(2)$	← uses the Δ coefficients

and so on. The beautiful parts here are that we don't actually have to create the $x_{\text{int}}(m)$ sequence at all, and we perform no multiply by zero computations. *That* is polyphase filtering.

The above list of calculations not only shows us what filtering to do, it shows us how to do it. We can implement our polyphase interpolation filtering technique with a bank of four subfilters as shown in [Figure 10-12](#). This depiction is called the *commutator model* for polyphase interpolation filters. We have a commutator switch rotating one complete cycle after the arrival of each new $x_{\text{old}}(n)$ sample. This way, four $x_{\text{new}}(m)$ samples are computed for each $x_{\text{old}}(n)$ input sample.

Figure 10-12 Polyphase interpolation by $L = 4$ filter structure as a bank of FIR subfilters.

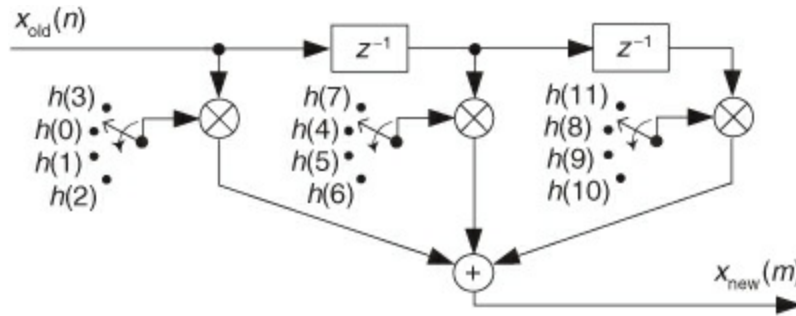


In the typical case, if our polyphase filter is interpolating by a factor of L , then we'll have L subfilters. As such, for convenience the number of taps in (the impulse response length of) the original prototype lowpass FIR filter, N , is chosen to be an integer multiple of L . Again, the passband width of

the prototype lowpass FIR filter must not be greater than $f_{s,\text{old}}/2$ where $f_{s,\text{old}}$ is $x_{\text{old}}(n)$'s sample rate in Hz.

A minimum data storage structure for the polyphase interpolation filter is shown in [Figure 10-13](#), where three commutators rotate (in unison) counterclockwise through four sets of filter coefficients upon the arrival of each new $x_{\text{old}}(n)$ sample. Again, four $x_{\text{new}}(m)$ samples are computed for each $x_{\text{old}}(n)$ sample.

Figure 10-13 Minimum-storage polyphase interpolation filter structure using commutated coefficients.



This commutated-coefficients scheme has the advantage of reducing the number of storage registers for the $x_{\text{old}}(n)$ input samples. If our polyphase filter is interpolating by a factor of L , then we have L sets of coefficients.

We can validate our polyphase FIR filter block diagrams with z -transform equations. We start by describing our [Figure 10-12](#) polyphase FIR filter with

(10-4)

$$H(z) = h(0) + h(4)z_{\text{in}}^{-1} + h(8)z_{\text{in}}^{-2} \\ + [h(1) + h(5)z_{\text{in}}^{-1} + h(9)z_{\text{in}}^{-2}]z_{\text{out}}^{-1} \\ + [h(2) + h(6)z_{\text{in}}^{-1} + h(10)z_{\text{in}}^{-2}]z_{\text{out}}^{-2} \\ + [h(3) + h(7)z_{\text{in}}^{-1} + h(11)z_{\text{in}}^{-2}]z_{\text{out}}^{-3},$$

where z_{in}^{-1} is a unit delay at the input sample rate, and z_{out}^{-1} is a unit delay at the output sample rate implemented with the commutator. Because $z_{\text{in}}^{-1} = z_{\text{out}}^{-4}$, and $z_{\text{in}}^{-2} = z_{\text{out}}^{-8}$, we can write

(10-4')

$$H(z) = h(0) + h(4)z_{\text{out}}^{-4} + h(8)z_{\text{out}}^{-8} \\ + [h(1) + h(5)z_{\text{out}}^{-4} + h(9)z_{\text{out}}^{-8}]z_{\text{out}}^{-1} \\ + [h(2) + h(6)z_{\text{out}}^{-4} + h(10)z_{\text{out}}^{-8}]z_{\text{out}}^{-2} \\ + [h(3) + h(7)z_{\text{out}}^{-4} + h(11)z_{\text{out}}^{-8}]z_{\text{out}}^{-3} \\ = h(0) + h(4)z_{\text{out}}^{-4} + h(8)z_{\text{out}}^{-8} \\ + h(1)z_{\text{out}}^{-1} + h(5)z_{\text{out}}^{-5} + h(9)z_{\text{out}}^{-9} \\ + h(2)z_{\text{out}}^{-2} + h(6)z_{\text{out}}^{-6} + h(10)z_{\text{out}}^{-10} \\ + h(3)z_{\text{out}}^{-3} + h(7)z_{\text{out}}^{-7} + h(11)z_{\text{out}}^{-11} \\ = \sum_{k=0}^{11} h(k)z_{\text{out}}^{-k},$$

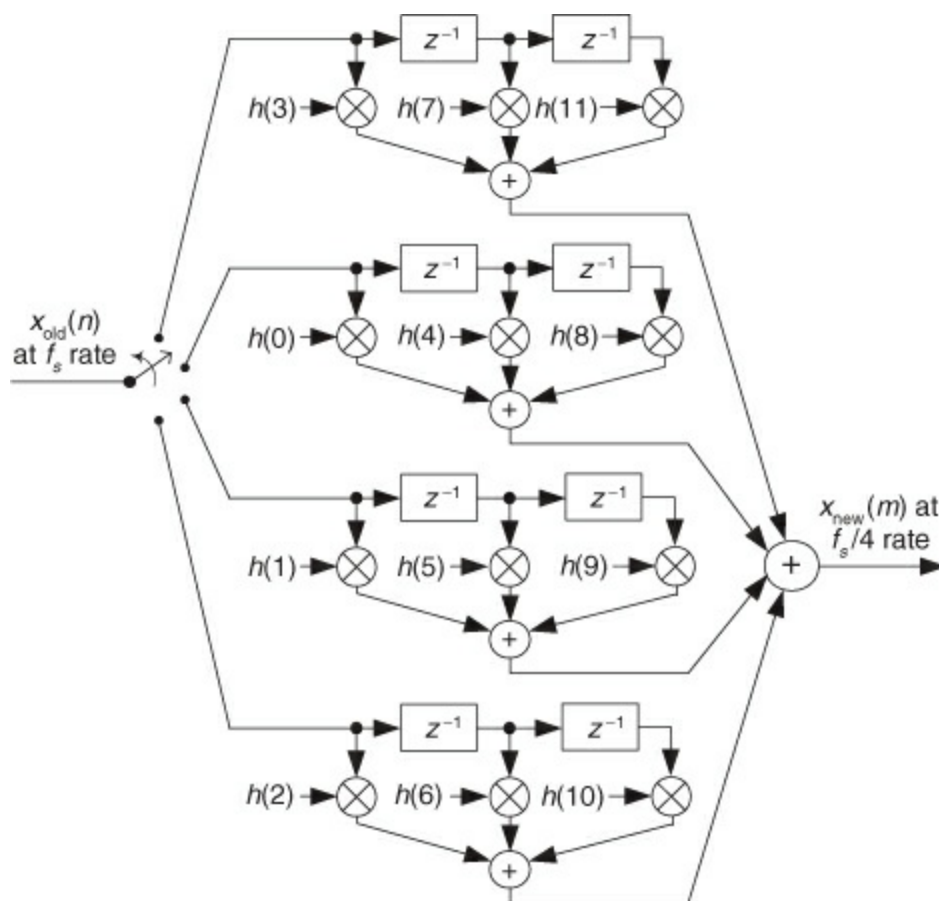
which is the classic z -domain transfer function for a 12-tap FIR filter. [Equation \(10-4\)](#) is called a *polyphase decomposition* of [Eq. \(10-4'\)](#).

Concerning our [Figure 10-11](#) example, there are several issues to keep in mind:

- For an interpolation factor of L , most people make sure the prototype FIR has an integer multiple of L number of stages for ease of implementation.
- As with the zeros-insertion and filtering method of interpolation, the polyphase method has a gain loss equal to the interpolation factor L . To compensate for this amplitude loss we can increase the filter's coefficients by a factor of L , or perhaps multiply the $x_{\text{new}}(m)$ output sequence by L .
- Our [Figure 10-11](#) example used a prototype filter with an even number of taps, but an odd-tap prototype FIR interpolation filter can also be used[\[5\]](#). For example, you could have a 15-tap prototype FIR and interpolate by 5.
- Because the subfilter coefficient sets in [Figure 10-13](#) are not necessarily symmetrical, we can't reduce the number of multiplications by means of the *folded FIR* structure discussed in [Section 13.7](#).

With the commutating switch structure of [Figure 10-12](#) in mind, we can build a decimation-by-four polyphase filter using a commutating switch as shown in [Figure 10-14](#). The switch rotates through its four positions ($M = 4$), applying four $x_{\text{old}}(n)$ input samples to the subfilters, then the four subfilters' outputs are accumulated to provide a single $x_{\text{new}}(m)$ output sample. In this filter the commutating switch rotates in the counterclockwise direction.

Figure 10-14 Polyphase decimation by $M = 4$ filter structure as a bank of FIR subfilters.



Notice that the subfilters in [Figure 10-14](#) are unchanged from the interpolation filter in [Figure 10-12](#). Again, the benefit of polyphase decimation filtering means no unnecessary computations are performed. We're decimating before filtering, so no filter computational results are discarded.

In the typical case, if our polyphase filter is decimating by a factor of M , then we'll have M

subfilters. As such, for convenience the number of taps in (the impulse response length of) the original prototype lowpass FIR filter, N , is chosen to be an integer multiple of M . The passband width of the prototype lowpass filter must not be greater than $(f_{s,\text{old}}/2) \cdot (L/M)$ where $f_{s,\text{old}}$ is $x_{\text{old}}(n)$'s sample rate in Hz.

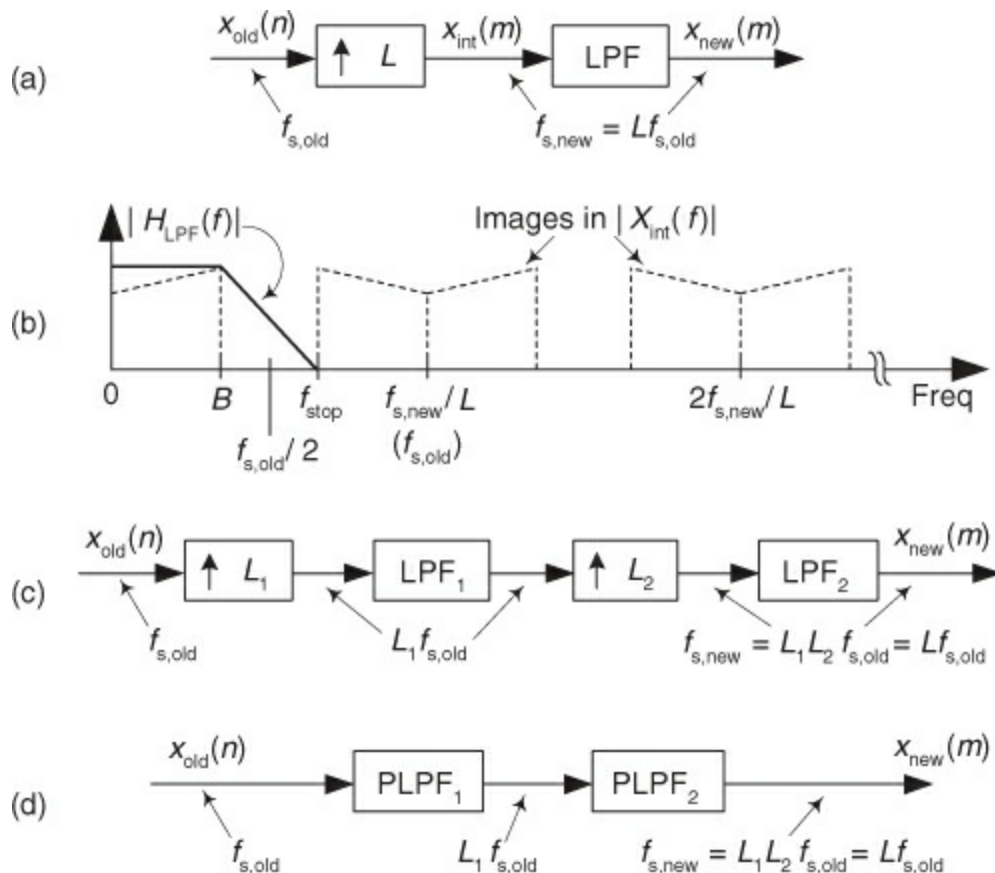
Again, in practice, large changes in sampling rate are accomplished with multiple stages (where [Figure 10-14](#), for example, is a single decimation stage) of cascaded smaller rate change operations of decimation and interpolation as discussed in [Sections 10.2](#) and [10.8](#). With that thought in mind, now is the appropriate time to discuss two-stage interpolation.

This concludes our brief introduction to the important topic of discrete polyphase filters. (For my money, the development of polyphase filters arguably resides in the stratosphere of brilliant DSP innovations, along with the radix-2 FFT algorithm and the Parks-McClellan FIR filter design algorithm.) More detailed information on polyphase filters can be found in references [\[6–8\]](#) and my favorite, reference [\[9\]](#).

10.8 Two-Stage Interpolation

Because we're now familiar with the notion of polyphase filtering, we're ready to consider the process of two-stage interpolation. When a desired interpolation factor L is large, say $L > 20$, significant interpolation filter computational savings may be had by implementing the interpolation in [Figure 10-15\(a\)](#) in two stages as shown in [Figure 10-15\(c\)](#). In the later figure we interpolate input sequence $x_{\text{old}}(n)$ by integer factor L_1 followed by interpolation by integer factor L_2 , where $L = L_1 L_2$.

Figure 10-15 Interpolation: (a) single-stage; (b) lowpass filter (LPF) magnitude response and upsampled $x_{\text{int}}(m)$ spectrum; (c) two-stage interpolation; (d) two-stage polyphase interpolation.



10.8.1 Two-Stage Interpolation Concepts

Let's assume we want to interpolate [Figure 10-15\(a\)](#)'s input $x_{\text{old}}(n)$ sequence by L , so we insert the $L-1$ zero-valued samples appropriately in $x_{\text{old}}(n)$ to create the $x_{\text{int}}(m)$ sequence whose spectral magnitude is shown as the dashed lines in [Figure 10-15\(b\)](#). The lowpass filter (LPF) in [Figure 10-15\(a\)](#) must have a frequency magnitude response, shown as the solid lines in [Figure 10-15\(b\)](#), that eliminates the spectral images in the $x_{\text{int}}(m)$ sequence's $X_{\text{int}}(f)$ spectrum. As such, the filter LPF's transition region extends from B Hz to $f_{\text{stop}} = f_{\text{s,old}} - B = f_{\text{s,new}}/L - B$ Hz. (Frequency f_{stop} is the beginning of the lowpass filter's stopband.) Given that frequency response requirement, we could now begin to design the lowpass filter LPF.

However, using [Figure 10-15\(c\)](#)'s two-stage interpolation, we can accomplish our overall interpolation by L where the combined number of computations in filters LPF_1 and LPF_2 is much smaller than the computations needed in the single [Figure 10-15\(a\)](#) LPF filter. This computational workload reduction can be achieved by determining the optimum L_1 and L_2 factors for our two-stage interpolation in [Figure 10-15\(c\)](#), just as we did in finding the optimum downsampling factors in two-stage decimation.

Given the desired upsampling factor L in [Figure 10-15\(a\)](#), we can determine the L_1 and L_2 upsampling factors that minimize the number of overall two-stage filtering multiplications per input sample using

$$(10-5) \quad L_{2,\text{opt}} \approx 2L \cdot \frac{1 - \sqrt{LF / (2 - F)}}{2 - F(L + 1)},$$

where F is the ratio of the LPF filter's transition region width over the filter's stopband frequency, as shown in [Figure 10-15\(b\)](#). That is,

$$(10-5') \quad F = \frac{f_{\text{stop}} - B}{f_{\text{stop}}} = \frac{f_{\text{s,old}} - 2B}{f_{\text{stop}}}.$$

Upon using [Eq. \(10-5\)](#) to compute $L_{2,\text{opt}}$, and setting L_2 equal to the integer submultiple of L that is closest to $L_{2,\text{opt}}$, the first interpolation factor L_1 is found using

$$(10-5'') \quad L_1 = \frac{L}{L_2}.$$

So, once we know the optimum values for L_1 and L_2 , we proceed by designing the LPF_1 and LPF_2 lowpass filters in [Figure 10-15\(c\)](#). Finally, we implement the two-stage interpolation using two polyphase interpolation filters, PLPF_1 and PLPF_2 , as shown in [Figure 10-15\(d\)](#). Let's illustrate this two-stage interpolation concept with an example.

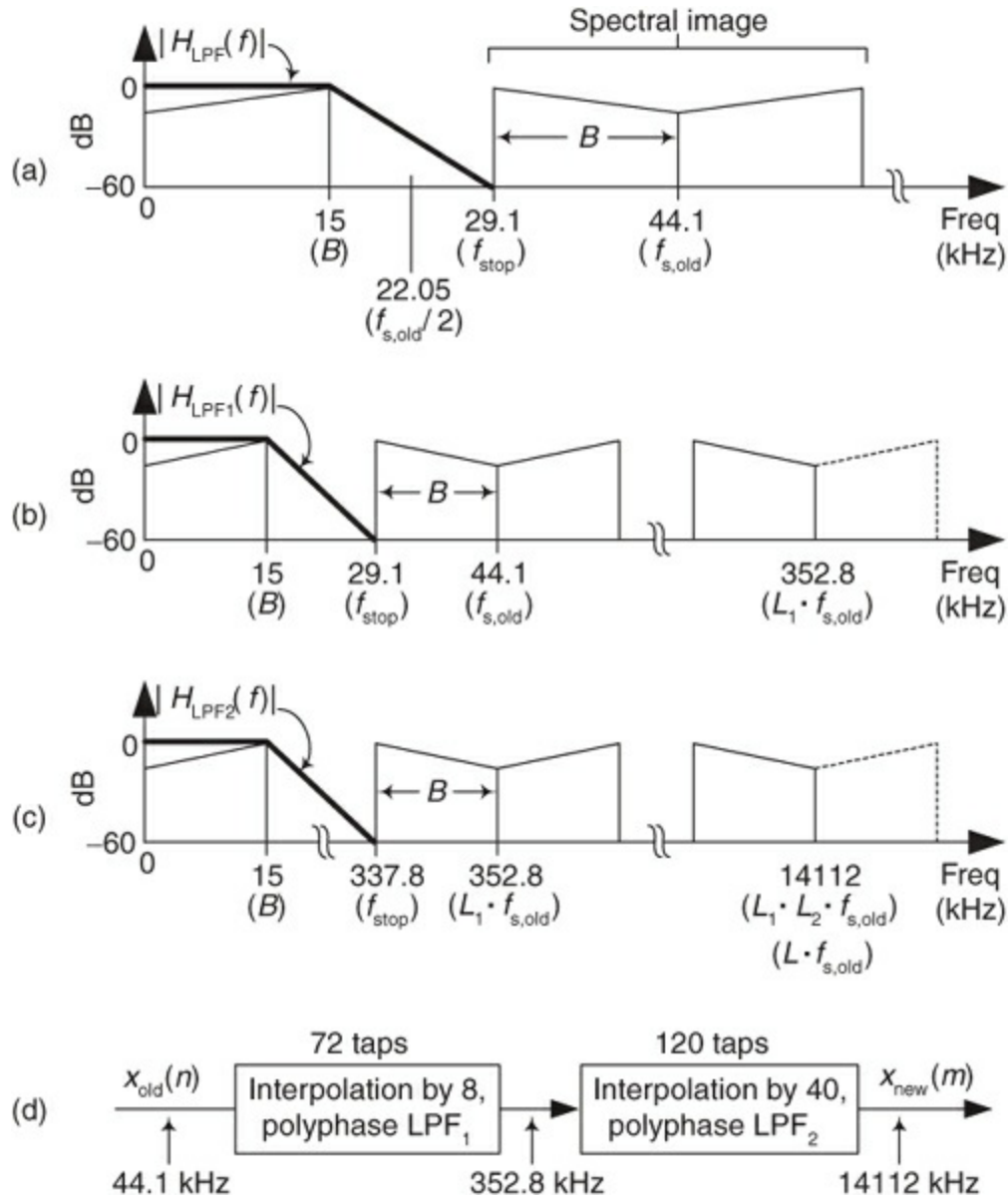
10.8.2 Two-Stage Interpolation Example

Assume we must convert a compact disc (CD) audio signal, having a signal bandwidth of 15 kHz and a sample rate of 44.1 kHz, to the sample rate of 96 kHz used by a high-performance audio system. In addition, let's assume that our interpolation filtering requires a stopband attenuation of 60 dB. We can

accomplish this sample rate conversion by interpolating the CD signal by a factor of $L = 320$, and then decimate the interpolated signal by $M = 147$. So this two-stage interpolation example will show how to efficiently interpolate an input signal sequence by $L = 320$, yielding an interpolated sequence having a sample rate of $f_{s,\text{new}} = L \cdot f_{s,\text{old}} = 320 \cdot 44.1 = 14112$ kHz.

The bold lines in [Figure 10-16\(a\)](#) show the frequency requirements of the lowpass filter that we need for a single-stage $L = 320$ interpolation process. It is that magnitude response that we will implement using two cascaded polyphase interpolation filter stages. The sample rate in [Figure 10-16\(a\)](#) is $f_{s,\text{new}} = 14112$ kHz.

Figure 10-16 Two-stage interpolation: (a) single-stage filter frequency parameters; (b) LPF₁ filter parameters; (c) LPF₂ filter parameters; (d) polyphase implementation.



First we determine the optimum L_1 and L_2 interpolation factors. With $f_{\text{stop}} = 29.1$ kHz and $B = 15$ kHz, we use [Eq. \(10-5'\)](#) to compute ratio F as

$$(10-6)$$

$$F = \frac{f_{s,\text{old}} - 2B}{f_{\text{stop}}} = \frac{44.1 - 30}{29.1} = 0.4845.$$

Next, we compute $L_{2,\text{opt}}$ using [Eq. \(10-5\)](#) as

(10-6')

$$L_{2,\text{opt}} \approx 2L \frac{1 - \sqrt{LF / (2 - F)}}{2 - F(L + 1)}$$

$$= 2 \cdot 320 \frac{1 - \sqrt{320 \cdot 0.485 / (2 - 0.485)}}{2 - 0.485(320 + 1)} = 37.98.$$

The integer submultiple of $L = 320$ that's closest to $L_{2,\text{opt}} = 37.98$ is 40. So we set $L_2 = 40$, and using [Eq. \(10-5'\)](#), we compute $L_1 = 320/40 = 8$.

So the first polyphase lowpass filter, LPF_1 , must have the frequency magnitude response shown in [Figure 10-16\(b\)](#) when its operating sample rate is $L_1 \cdot f_{s,\text{old}} = 8 \cdot 44.1 = 352.8$ kHz. (That 352.8 kHz sample rate would have been the LPF_1 filter's input rate had we inserted the $L_1 - 1$ zero-valued samples between each of the original CD samples. Recall that with polyphase filtering we don't actually insert any zero-valued samples, but we must design a polyphase filter assuming the upsampled 352.8 kHz sample rate.)

Using [Eq. \(10-3\)](#) to estimate the number of taps in LPF_1 , N_{LPF1} , with $\text{Atten} = 60$, we compute

(10-7)

$$N_{\text{LPF1}} \approx \frac{\text{Atten}}{22(f_{\text{stop}} - f_{\text{pass}})}$$

$$= \frac{60}{22(29.1 / 352.8 - 15 / 352.8)} \approx 68 \text{ taps.}$$

Because we must partition the LPF_1 coefficients into a polyphase bank of $L_1 = 8$ subfilters, N_{LPF1} must be an integer multiple of 8. So we'll set $N_{\text{LPF1}} = 72$ taps, and the polyphase LPF_1 filter will have 8 subfilters.

The second polyphase lowpass filter, LPF_2 , must have the frequency magnitude response shown in [Figure 10-16\(c\)](#) when its operating sample rate is $L_1 \cdot L_2 \cdot f_{s,\text{old}} = 14112$ kHz. Using [Eq. \(10-3\)](#) to estimate the number of taps in LPF_2 , N_{LPF2} , with $\text{Atten} = 60$, we compute

(10-8)

$$N_{\text{LPF2}} \approx \frac{60}{22(337.8 / 14112 - 15 / 14112)} \approx 119 \text{ taps.}$$

Because we must partition the LPF_2 coefficients into a polyphase bank of $L_2 = 40$ subfilters, N_{LPF2} must be an integer multiple of 40. So we'll set $N_{\text{LPF2}} = 120$ taps, and the polyphase LPF_2 filter will have 40 subfilters. We implement our two-stage interpolation as shown in [Figure 10-16\(d\)](#), and that completes our two-stage interpolation example.

The number of multiplies in our two-stage polyphase interpolation process is $N_{\text{LPF1}} + L_1 \cdot N_{\text{LPF2}} = 1032$ multiplies per $x_{\text{old}}(n)$ input sample. If we had implemented our interpolation by $L = 320$ using a single polyphase filter having 320 subfilters, we would have had to perform 2880 multiplies per $x_{\text{old}}(n)$ input sample. So, happily, our two-stage interpolation process reduced the number of

necessary filter multiplies by almost a factor of three relative to a single-stage interpolation.

10.8.3 Two-Stage Interpolation Considerations

Due to the duality between decimation and interpolation, for computational efficiency reasons as presented in reference [3], it's beneficial to interpolate in order from the smallest to the largest factor. That is, we make sure that L_1 is smaller than L_2 .

Also, it is advantageous to consider setting the L_1 and L_2 interpolation factors equal to integer powers of two because we can use computationally efficient half-band filters for the lowpass filters in Figure 10-15(c). We discuss the use of multirate half-band filtering later in Section 10.11.

As with dual-stage decimation, if the single-filter system in Figure 10-15(a) is required to have a passband peak-peak ripple of R dB (R decibels), then each filter in Figure 10-15(c) must be designed to have passband peak-peak ripple of no greater than $R/2$ dB. We have previously mentioned that interpolation has an inherent amplitude loss. Thus, to achieve unity gain between sequences $x_{\text{old}}(n)$ and $x_{\text{new}}(m)$ in Figure 10-15(c), the product of the DC (zero Hz) gains of the LPF_1 and LPF_2 filters must be equal to L .

The advantages of two-stage interpolation, over single-stage interpolation, are identical to the advantages of two-stage decimation listed at the end of Section 10.2. Be aware that references [1] and [3] discuss aspects of multistage interpolation where the number of stages is greater than two.

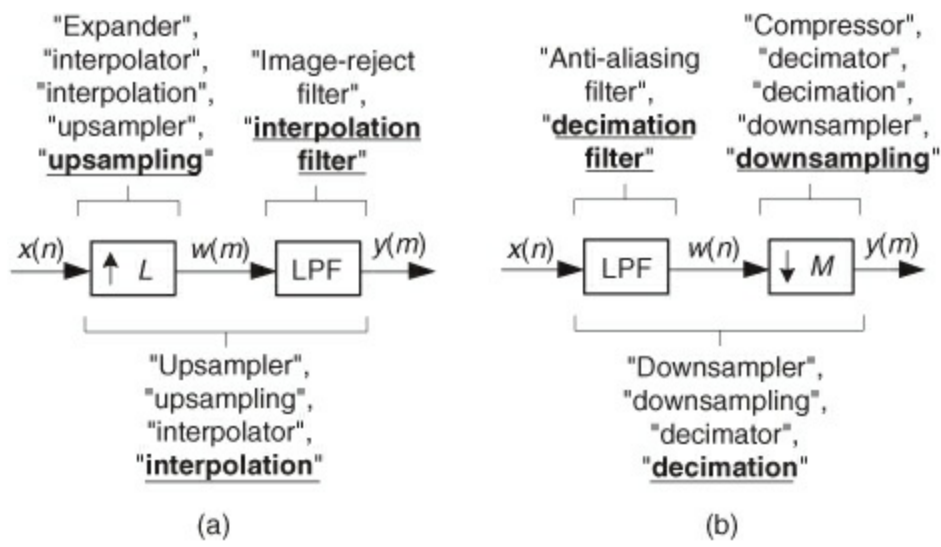
In concluding this section, we mention that Chapter 13 contains three DSP tricks regarding interpolation of time-domain signals. Now that we have some familiarity with sample rate conversion, for completeness let's review the standard mathematical notation used to describe these operations using polyphase filters. Learning that notation will aid readers as they encounter other descriptions of sample rate conversion in the literature of DSP.

10.9 z -Transform Analysis of Multirate Systems

In preparation for the multirate filter material in the following sections, here we formalize both our terminology and notation of sample rate conversion operations.

First, there is a fair amount of variety (some would call it “ambiguity”) in the literature of DSP regarding the language of sample rate conversion. If you've been reading the literature, you may have noticed that the terminology used has been, unfortunately, very inconsistent—sometimes downright confusing. A wide variety of terms are used in the literature as shown in Figure 10-17 where “LPF” means lowpass filter. In the spirit of consistency, from here on we'll use the terminology indicated by the bold underlined font in Figure 10-17.

Figure 10-17 Sample rate conversion terminology: (a) sample rate increase; (b) sample rate reduction.



10.9.1 Signal Mathematical Notation

Compared to the written language of sample rate conversion, the mathematical notation of sample rate conversion is quite consistent if we use z-transform representations. For example, if a time-domain sequence $x(n)$, having a z-transform of

$$(10-9)$$

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

is upsampled by two ($L = 2$, a single zero-valued sample is inserted between each $x(n)$ sample), producing a $w(m)$ sequence as shown in [Figure 10-17\(a\)](#), then we can describe $w(m)$ as

$$(10-10)$$

$$w(m) = \begin{cases} x(m/2), & m = 0, \pm 2, \pm 4, \dots \\ 0, & \text{when } m \text{ is } \pm \text{odd.} \end{cases}$$

[Equation \(10-10\)](#) indicates that every other $w(m)$ sample is zero. Considering only the nonzero values of $w(m)$, the z-transform of $w(m)$ is expressed as

$$(10-11)$$

$$\begin{aligned} W(z) &= \sum_{m=-\infty}^{\infty} w(m)z^{-m} = \sum_{k=-\infty}^{\infty} w(2k)z^{-2k} \\ &= \sum_{k=-\infty}^{\infty} x(2k/2)z^{-2k} = \sum_{k=-\infty}^{\infty} x(k)z^{-2k} = X(z^2) \end{aligned}$$

where m represents even-valued integers and k represents all integers. If the $w(m)$ sequence is an upsampled-by-integer- L version of $x(n)$ (inserting $L-1$ zero-valued samples between each $x(n)$ sample), then $w(m)$'s z-transform is expressed as

$$(10-12)$$

$$W(z) = X(z^L).$$

In a similar manner, some authors express the z-transform of sequence $x(n)$ as

$$(10-13)$$

$$X(z) = W(z^{1/L}).$$

So here is the point: When we see expressions like [Eqs. \(10-11\)](#), [\(10-12\)](#), or [\(10-13\)](#), they merely mean that sequence $w(m)$ is an upsampled-by- L version of sequence $x(n)$, and sequence $x(n)$ is a decimated-by- L version of sequence $w(m)$.

10.9.2 Filter Mathematical Notation

With the above multirate notation fresh in our minds, let's consider how we can use that notation to describe digital polyphase filters. If we have a tapped-delay line FIR filter, having N taps, whose impulse response is $h(k)$, then we can represent the filter's z -domain transfer function as

(10-14)

$$H(z) = \sum_{k=0}^{N-1} h(k)z^{-k}.$$

For an $N = 9$ -tap FIR filter, for example, from [Eq. \(10-14\)](#) its z -domain transfer function is

(10-15)

$$H(z) = h(0) + h(1)z^{-1} + h(2)z^{-2} + h(3)z^{-3} + \dots + h(8)z^{-8}.$$

In both up- and downsample-by-integer-factor- Q filtering applications, for computational efficiency reasons, we partition $H(z)$ into Q separate subfilters using the polyphase decomposition process. For example, if $Q = 3$, we can write $H(z)$ as

(10-16)

$$\begin{aligned} H(z) = & h(0) + h(3)z^{-3} + h(6)z^{-6} + [h(1) + h(4)z^{-3} + h(7)z^{-6}]z^{-1} \\ & + [h(2) + h(5)z^{-3} + h(8)z^{-6}]z^{-2}. \end{aligned}$$

(Read no further until you convince yourself that [Eqs. \(10-15\)](#) and [\(10-16\)](#) are equivalent.) Due to the exponents of z in [Eq. \(10-16\)](#) we can write

(10-17)

$$\begin{aligned} H(z) = & h(0)(z^{-0})^3 + h(3)(z^{-1})^3 + h(6)(z^{-2})^3 \\ & + [h(1)(z^{-0})^3 + h(4)(z^{-1})^3 + h(7)(z^{-2})^3]z^{-1} \\ & + [h(2)(z^{-0})^3 + h(5)(z^{-1})^3 + h(8)(z^{-2})^3]z^{-2} \\ = & H_0(z^3) + H_1(z^3)z^{-1} + H_2(z^3)z^{-2}, \end{aligned}$$

where

(10-17')

$$\begin{aligned} H_0(z^3) = & h(0) + 0z^{-1} + 0z^{-2} + h(3)z^{-3} + 0z^{-4} + 0z^{-5} + h(6)z^{-6} \\ H_1(z^3) = & h(1) + 0z^{-1} + 0z^{-2} + h(4)z^{-3} + 0z^{-4} + 0z^{-5} + h(7)z^{-6} \\ H_2(z^3) = & h(2) + 0z^{-1} + 0z^{-2} + h(5)z^{-3} + 0z^{-4} + 0z^{-5} + h(8)z^{-6}. \end{aligned}$$

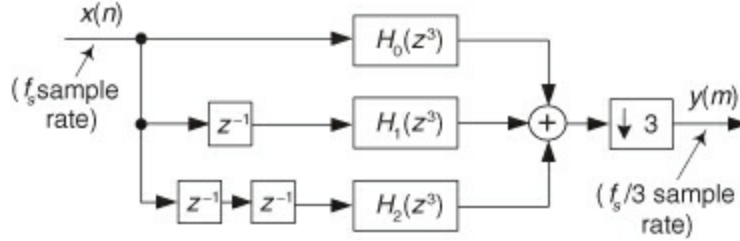
The notation in the last line of [Eq. \(10-17\)](#) seems, at first, like a needless complication in describing the 9-tap $h(k)$ filter, but shortly we will see why such notation is *very* useful.

10.10 Polyphase Filter Implementations

Let's now use the above z -domain transfer functions to help us understand the most popular forms of

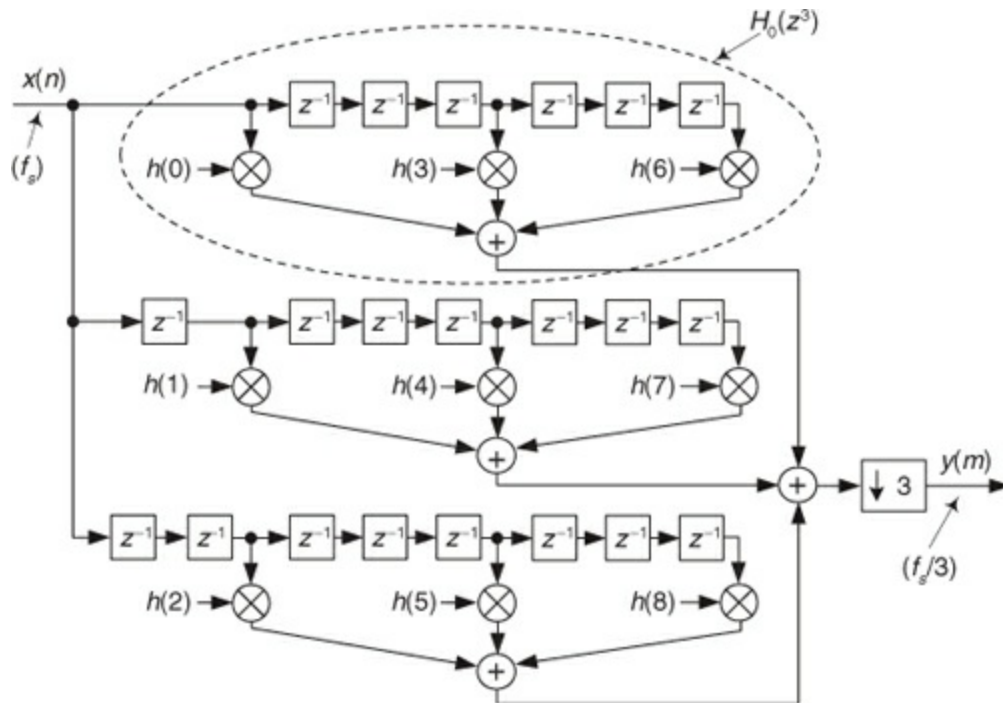
polyphase filtering in multirate systems. [Equation \(10-17\)](#), when followed by downsampling by $M = 3$, is depicted graphically in [Figure 10-18](#), showing the three subfilters. We interpret the notation of the top subfilter, $H_0(z^3)$ in [Figure 10-18](#), as a tapped-delay line wherein there are $M = 3$ delay elements between each tap. To pause for a moment, what we're doing here is showing the algebraic and graphical notation used to describe the polyphase decomposition of a 9-tap prototype FIR filter used in a decimation-by-three application.

Figure 10-18 Polyphase decomposition of $H(z)$ prior to downsampling by $M = 3$.



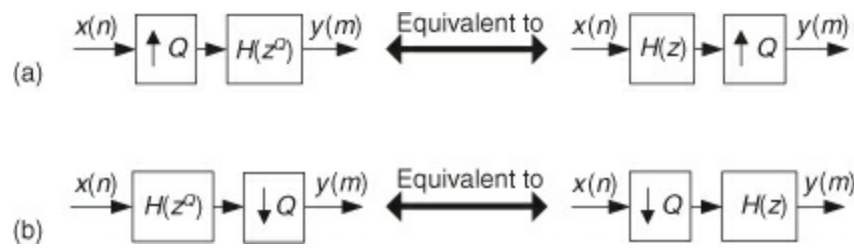
The detailed structure of the FIR filter in [Figure 10-18](#) is shown in [Figure 10-19](#), where we see the polyphase decomposition of $h(k)$ into three subfilters, creating a polyphase filter.

Figure 10-19 Details of the polyphase decomposition of $H(z)$ for decimation $M = 3$.



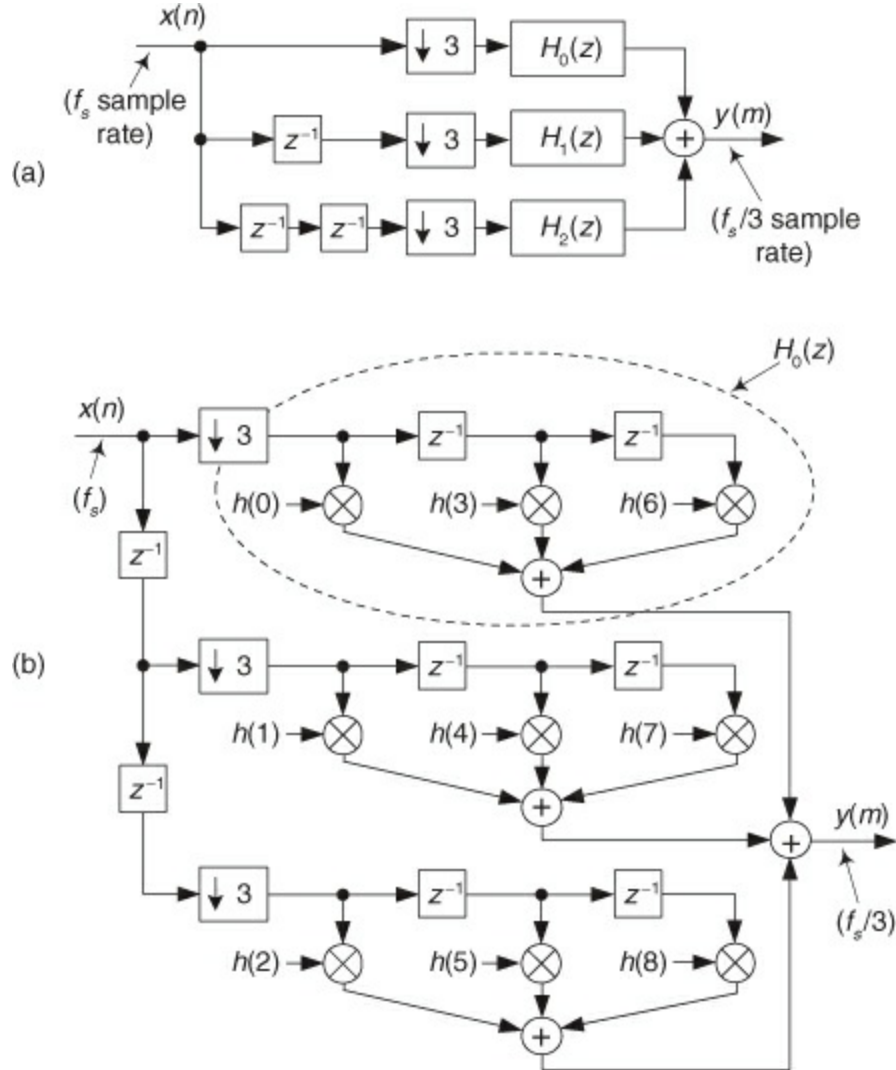
When the multirate concepts described above were first applied to the impulse responses of digital filters, DSP pioneers quickly arrived at the impressive-sounding “noble identities” graphically depicted in [Figure 10-20](#). Those complementary identities, showing the equivalency of swapping the order of filters and up/downsamplers, are exceedingly useful in the analysis and implementation of multirate systems as we shall see in the next section. In [Figure 10-20](#) the $H(z)$ term is the z -transform of a filter's $h(n)$ impulse response, and the $H(z^Q)$ term is the z -transform of $h(n)$ upsampled by integer Q , similar in form to [Eqs. \(10-17\)](#) and [\(10-17'\)](#).

Figure 10-20 Noble identities of multirate systems: (a) sample rate increase; (b) sample rate reduction.



Using the noble identities, we can move the downsampling by $M = 3$ operation in front of the subfilters in [Figure 10-18](#) as shown in [Figure 10-21\(a\)](#). A detailed depiction of the polyphase filter is provided in [Figure 10-21\(b\)](#), where we also rearranged the initial delay elements at the input of the filter.

Figure 10-21 Polyphase decomposition of $h(k)$, for decimation by $M = 3$: (a) simplified depiction; (b) detailed depiction.



In that figure we see that the delay lines between the filter coefficients now contain only a single delay element and the subfilters can be described by

$$(10-18)$$

$$H_0(z) = h(0) + h(3)z^{-1} + h(6)z^{-2}$$

$$H_1(z) = h(1) + h(4)z^{-1} + h(7)z^{-2}$$

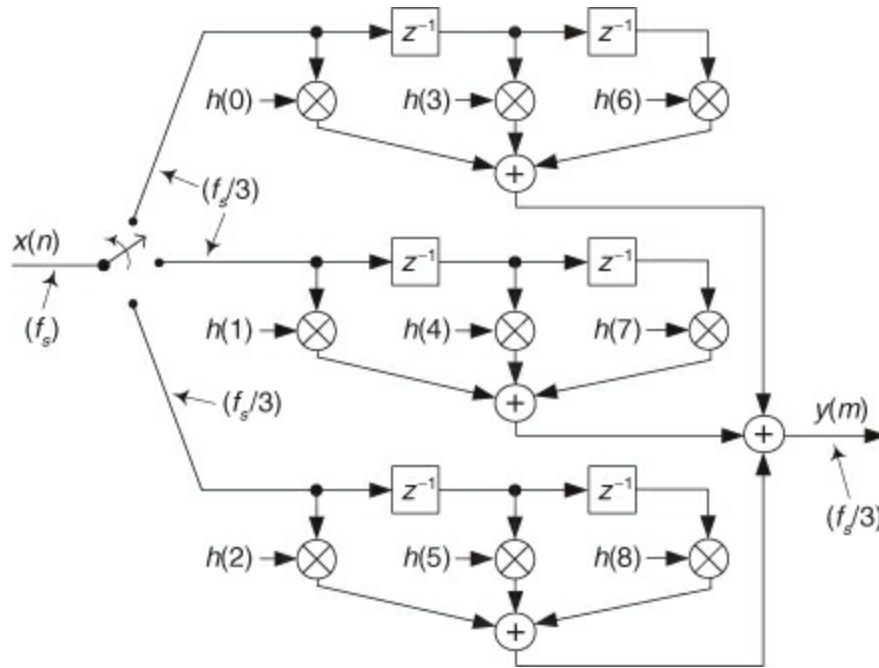
$$H_2(z) = h(2) + h(5)z^{-1} + h(8)z^{-2}.$$

The upper subfilters in [Figures 10-19](#) and [10-21\(b\)](#) make obvious the meaning of our notation regarding $H_0(z^3)$ and $H_0(z)$, for example. That is, $H_0(z^3)$ is merely an upsampled-by-three version of

$H_0(z)$.

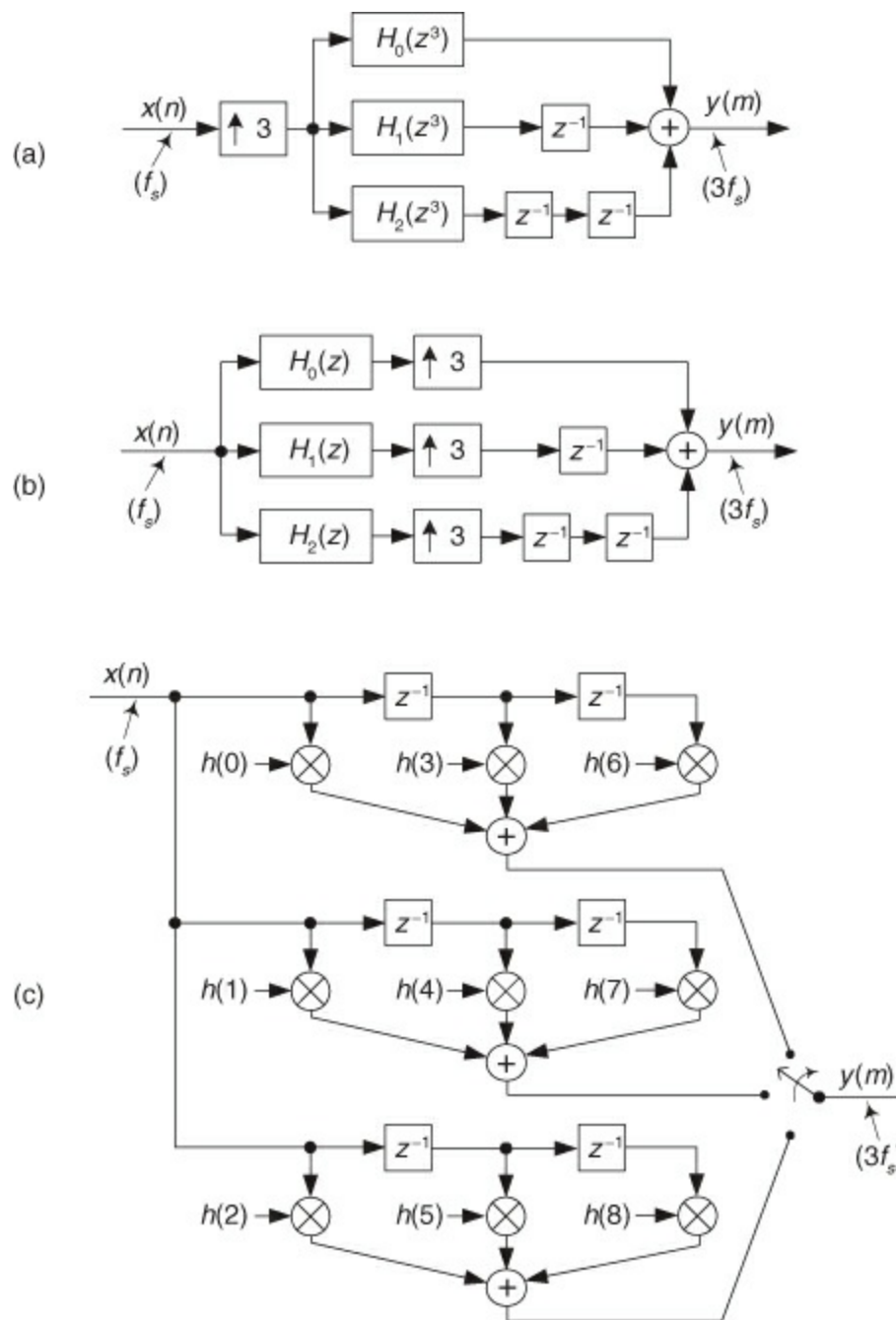
One final simplification available for polyphase decimation filters is shown in [Figure 10-22](#), where the two initial delay elements and the downsampling by $M = 3$ operations in [Figure 10-21\(b\)](#) are replaced by a three-position commutating (multiplexing) switch. One $y(m)$ output sample is produced each time the switch completes a single full (three-step) rotation.

Figure 10-22 Simplified polyphase decomposition of $h(k)$, for decimation by $M = 3$.



In an identical manner, interpolation by $L = 3$ (upsampling by three followed by lowpass filtering) by way of polyphase decomposition is depicted in [Figure 10-23\(a\)](#). The subfilters in that figure are identical to the subfilters from [Eq. \(10-17\)](#) and [Figure 10-19](#). Looking at [Figure 10-23\(b\)](#), we see that the upsamplers insert two zero-valued samples between each output sample of the three subfilters. The delay elements delay those upsampled sequences by various delay times such that at each output time instant only one of the inputs to the final summation is nonzero. So instead of performing a summation of mostly zero-valued samples, we can select only the path to the summer that contains a nonzero sample.

Figure 10-23 Polyphase decomposition of $h(k)$, for interpolation by $L = 3$: (a) simple depiction; (b) reduced-length subfilters; (c) final structure.



Thinking about this path selection process (multiplexing), happily we can use the three-path commutating switch in [Figure 10-23\(c\)](#) for multipath selection and eliminate the delay elements, the upsamplers, and the final summation. As each new $x(n)$ input sample is available, the switch completes a single full (three-step) rotation, producing three $y(m)$ output samples.

Again, the purpose of the material in this section is to show the algebraic and graphical notation typically used to describe FIR polyphase filters used in sample rate conversion applications.

The major benefits of using polyphase filters for sample rate conversion are:

- Signal data storage requirements are minimized.
- No multiply by zero computations are performed (for interpolation).
- No computational results are discarded (for decimation).
- A key benefit is that the computations are performed at the lower sample rate. For an N -tap FIR filter, polyphase decimation implementations reduce the number of multiplications per unit time to $1/M$ times the number of multiplications per unit time with no polyphase decomposition. This advantage may be critical in high-data-rate applications and leads to lower power consumption in

battery-powered devices.

In the following sections we introduce several specialized digital filters developed specifically to minimize the computational workload encountered in sample rate conversion applications. As such, let's have a look at rational-factor sample rate change filters first.

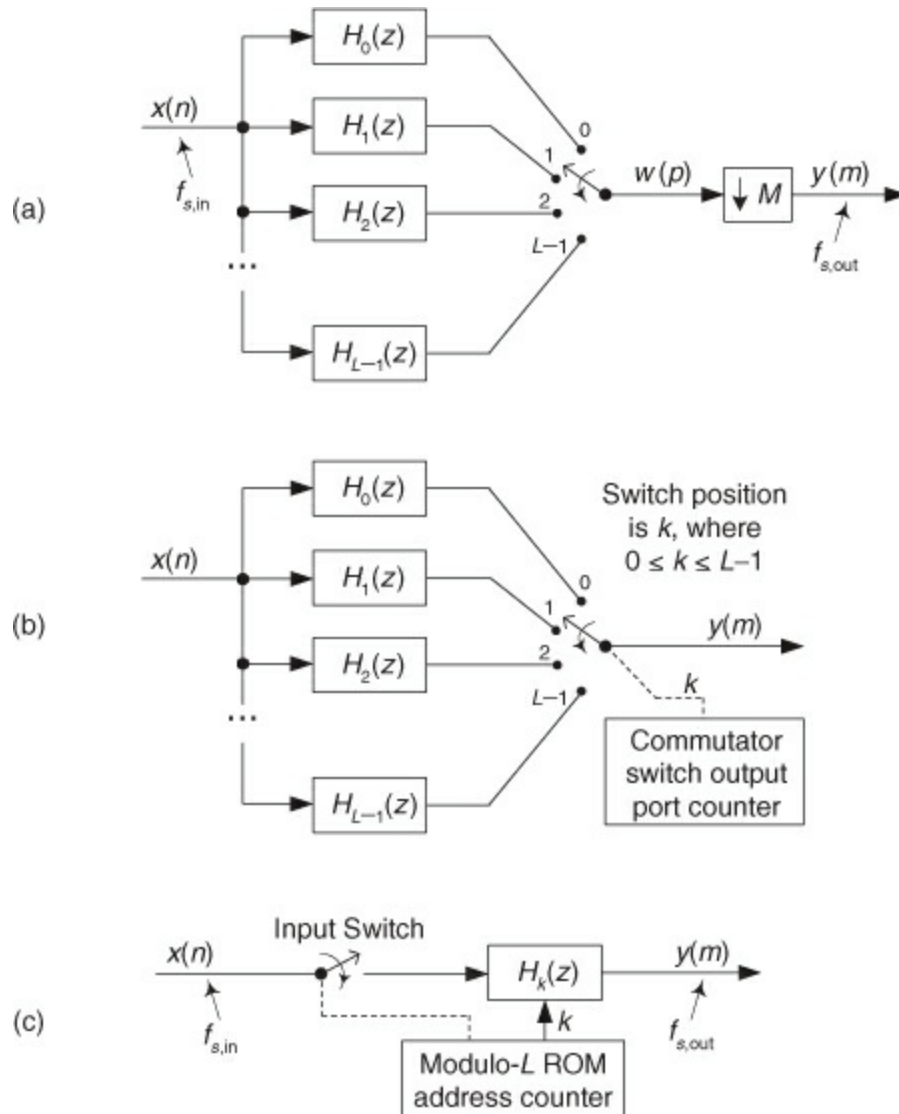
10.11 Sample Rate Conversion by Rational Factors

In the event that we wish to resample a signal by a rational factor L/M (as in [Figure 10-9\(b\)](#)), we can interpolate by integer factor L followed by downsampling by integer factor M . Our single lowpass filter comprises the L polyphase subfilters shown in [Figure 10-24\(a\)](#), where the input and output sample rates are related by

(10-19)

$$f_{s,\text{out}} = \frac{L \cdot f_{s,\text{in}}}{M}.$$

Figure 10-24 Resampling by rational-factor L/M : (a) fundamental process; (b) no downsampling; (c) addressed-ROM scheme.



However, this naive approach would not be sensible because we'd be computing some $w(p)$ samples that are destined to be discarded by the downsample-by- M process.

Attempting to avoid that computational inefficiency, we can omit the downsampling process

altogether and merely control the position of the interpolator's output commutating switch position as depicted in [Figure 10-24\(b\)](#). For example, if we rotate the switch but skip alternate switch output ports, we achieve resampling by a factor of $L/2$. If we advance the switch to every third output position, for each $y(m)$ output sample, we'll have resampling by a factor of $L/3$, and so on. This commutating switch control mechanism idea means that we need only compute the output of a single subfilter for each $y(m)$ output sample. In resampling by a rational factor L/M , the switch output port (index of a single subfilter) used to compute a $y(m)$ output sample is found using

$$(10-20)$$

$$k = \langle mM \rangle_L$$

where $\langle mM \rangle_L$ means compute the product mM modulo- L . So the switch output port counter in [Figure 10-24\(b\)](#) is a binary modulo- L counter. As the resampler operates, the index n , of the most recent input $x(n)$ applied to the subfilters, is given by

$$(10-20')$$

$$n = \lfloor mM/L \rfloor$$

where $\lfloor mM/L \rfloor$ means the integer part of mM/L . The actual resampler difference equation is

$$(10-20'')$$

$$y(m) = \sum_{p=0}^{(N/L)-1} h(pL+k) \cdot x(n-p)$$

where N is the number of taps in the prototype FIR filter from which the polyphase $H_k(z)$ subfilters in [Figure 10-24\(b\)](#) were obtained.

For a numerical example, the left side of [Table 10-1](#) shows the commutating switch output port index k (index of a single subfilter), and the input $x(n)$ index n , as a function of a resampler's m th $y(m)$ output sample index for resampling by a factor of $L/M = 4/3$. In this case, the switch counter counts as $k = 0, 3, 2, 1, 0, 3, 2, 1$, and so on. The right side of [Table 10-1](#) shows the switch indexing for resampling by $3/4$. In that scenario, the switch counter counts as $k = 0, 1, 2, 0, 1, 2$, and so on.

Table 10-1 Indexing for Resampling by $4/3$ and $3/4$

Interpolation by $4/3$ $L = 4, M = 3$				Decimation by $3/4$ $L = 3, M = 4$		
m	k	n	Apply new $x(n)$	k	n	Apply new $x(n)$
0	0	0	←	0	0	←
1	3	0		1	1	←
2	2	1	←	2	2	←
3	1	2	←	0	3,4	←
4	0	3	←	1	5	←
5	3	3		2	6	←
6	2	4	←	0	7,8	←
7	1	5	←	1	9	←
8	0	6	←	2	10	←
9	3	6		0	11,12	←
10	2	7	←	1	13	←

In our daily lives we hear the phrase “Timing is everything.” Well, that’s certainly true in our resampling schemes. In [Figure 10-24\(b\)](#) we must remember that when the commutating switch resides at position $k = 0$, and when during its cycling it crosses the $k = 0$ position, we must input a new $x(n)$ sample *before* we compute a $y(m)$ output sample. The times when a new $x(n)$ input sample is applied to the subfilters, before a $y(m)$ sample is computed, are indicated by the left-pointing arrows in [Table 10-1](#).

Be aware that it’s possible that more than one $x(n)$ input sample must be applied to the resampler prior to an output $y(m)$ computation for decimation applications. For example, on the right side of [Table 10-1](#), when $m = 3$, we are forced to apply both the $x(3)$ and $x(4)$ input samples to the resampler before computing $y(3)$.

OK, let’s stop and catch our breath here. If we were to substitute the expressions for k and n , from [Eqs. \(10-20\)](#) and [\(10-20'\)](#), into [Eq. \(10-20''\)](#), we would produce a rather complicated algebraic expression for $y(m)$. However, we will not let such an equation for $y(m)$ intimidate us because the $h(pL + k)$ term in [Eq. \(10-20''\)](#) merely specifies the coefficients of the k th subfilter, and the $x(n - p)$ term simply defines the $x(n)$ input samples residing in that k th subfilter. As such, we see that [Eq. \(10-20''\)](#) is no more than a convolution equation where the summation index p accounts for each of the N/L coefficients in a subfilter. (N/L is an integer.)

Notice that the tapped-delay lines of each subfilter in [Figure 10-24\(b\)](#) contain the same $x(n)$ time samples. To reduce input signal data storage requirements, we can use a single tapped-delay line as we described for [Figure 10-13](#). So in our rational-factor resampling implementation, shown in [Figure 10-24\(c\)](#), the modulo- L counter output index k now becomes a pointer pointing to a bank of read-only memory (ROM) locations that contain the N/L coefficients of the k th subfilter. For each updated value of k in [Table 10-1](#) we use the k th set of subfilter coefficients to compute $y(m)$. The control of applying a new $x(n)$ input sample, or samples, to the resampler before computing a $y(m)$ output sample is indicated by the dashed line to the Input Switch in [Figure 10-24\(c\)](#). As such, each time the modulo- L ROM address counter overflows, we apply new $x(n)$ input samples to the resampler.

To conclude this rational-factor resampling discussion, there are three practical issues we must keep in mind. First, if we want the DC (zero Hz) gain of our resampling process to be unity, then the original prototype lowpass FIR filter must have a DC gain of L to compensate for the amplitude loss by a factor of L caused by interpolation. (The downsampling by M causes no amplitude change.) To achieve a DC gain of L , the sum of the prototype filter’s $h(k)$ coefficients must equal L .

Second, to avoid aliasing errors after downsampling, in designing the original prototype lowpass FIR filter, the filter’s passband width must not be greater than $f_{s,\text{in}}/2$ or $(f_{s,\text{in}}/2) \cdot (L/M)$, whichever is smaller, where $f_{s,\text{in}}$ is $x(n)$ ’s sample rate, and $f_{s,\text{out}}$ is the filter’s data rate, in Hz. The stopband attenuation of the prototype filter must be such that the attenuated upsampled images do not induce intolerable levels of noise when they’re aliased by downsampling by M into the final band of 0 to $f_{s,\text{out}}/2$ Hz.

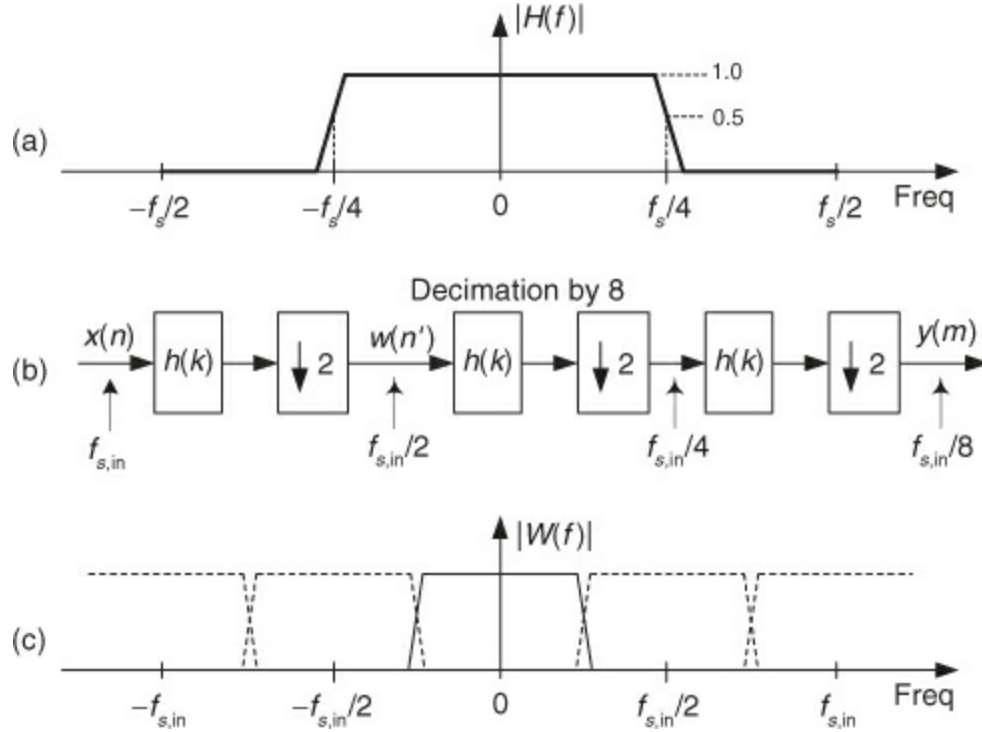
Third, from a computational efficiency standpoint, the rational-factor resampling scheme described in this section has the power of George Foreman’s right hand.

10.12 Sample Rate Conversion with Half-band Filters

Recall that the half-band filters we introduced in [Section 5.7](#) have a frequency magnitude response

with transition regions centered at $\pm f_s/4$ as shown in [Figure 10-25\(a\)](#). Those filters are linear-phase lowpass tapped-delay line FIR filters in which every other filter coefficient is zero, except the center coefficient. We discuss half-band filters here because their sparse nonzero coefficient sets make them ideal for use in sample rate conversion applications where the resampling factor is an integer power of two (2, 4, 8, etc.).

Figure 10-25 Half-band filters: (a) filter frequency magnitude response; (b) decimation by eight; (c) spectral overlap after decimation by two.



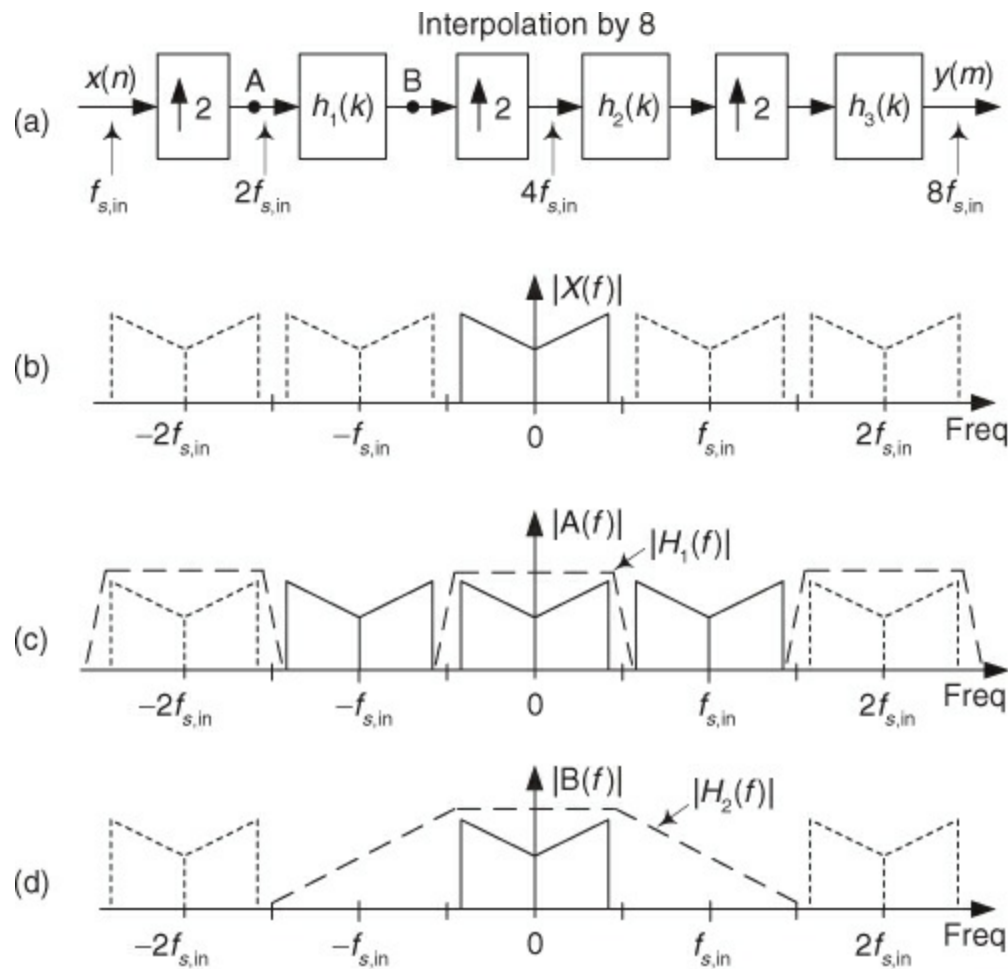
10.12.1 Half-band Filtering Fundamentals

An example of sample rate change by an integer power of two is shown in [Figure 10-25\(b\)](#) where the same $h(k)$ half-band filter is used three times to achieve decimation by eight. If the sample rate at the input of the three-stage decimation is $f_{s,\text{in}}$, the sample rate at the output is $f_{s,\text{in}}/8$.

We remind the reader that due to the nature of half-band filters there will be some amount of spectral overlap, and thus some aliasing, after each downsample-by-two operation. This is shown in [Figure 10-25\(c\)](#) for the first decimation-by-two stage, where the spectral replications are shown as dotted lines centered at integer multiples of the sample rate $f_{s,\text{in}}/2$. The amount of spectral overlap is proportional to the transition region width of the filters (inversely proportional to the number of $h(k)$ half-band filter taps).

It's normal to use the same half-band filter in multistage decimation by two as was done in [Figure 10-25\(b\)](#). However, in multistage interpolation by factors of two it would be computationally inefficient to use the same half-band filter in each stage. [Figure 10-26](#) helps explain why this is true. Consider the $x(n)$ signal in [Figure 10-26\(a\)](#) that we wish to interpolate by a factor of eight. The $x(n)$ signal's spectrum is that shown in [Figure 10-26\(b\)](#) where the spectral replications are shown as dotted lines centered at integer multiples of the input sample rate $f_{s,\text{in}}$. The signal at node A, after $x(n)$ has been upsampled by two via zeros insertion, has the $|A(f)|$ spectrum shown in [Figure 10-26\(c\)](#) where the new sample rate is $2f_{s,\text{in}}$.

Figure 10-26 Multistage interpolation using half-band filters.



The job of the $h_1(k)$ filter in [Figure 10-26\(a\)](#) is to eliminate the spectral images in $|A(f)|$ centered at $\pm f_{s,in}$ (half the sample rate at node A). We show $h_1(k)$'s magnitude response as the dashed $|H_1(f)|$ lines in [Figure 10-26\(c\)](#). The output of the $h_1(k)$ half-band filter, node B, has the $|B(f)|$ spectrum shown in [Figure 10-26\(d\)](#). After the signal at node B is upsampled by two, the $h_2(k)$ half-band filter must have the frequency magnitude response shown as $|H_2(f)|$ in [Figure 10-26\(d\)](#). Because the transition region width of $|H_2(f)|$ is so much wider than the transition region width of $|H_1(f)|$, the $h_2(k)$ filter will require fewer coefficients than did the $h_1(k)$ filter. For similar reasons the $h_3(k)$ filter will require fewer coefficients than the $h_2(k)$ filter.

What we're saying is this: Unlike multistage decimation by powers of two, in our relentless pursuit of computational efficiency, multistage interpolation by powers of two should not use the same half-band filter in each stage. In multistage interpolation each follow-on half-band filter requires fewer taps than the preceding filter. Because we like to minimize the number of necessary multiplications per second in real-time applications, we take comfort in the fact that the half-band interpolation filter requiring the most computations per output sample, $h_1(k)$, operates at the lowest sample rate.

From a practical standpoint, we remind the reader that if we use an FIR filter design software package to design half-band filters, unavoidable numerical computation errors will yield alternating filter coefficients that are indeed very small but not exactly zero-valued. So in our filter modeling efforts, we must force those very small coefficient values to zero before we proceed to analyze half-band filter frequency responses.

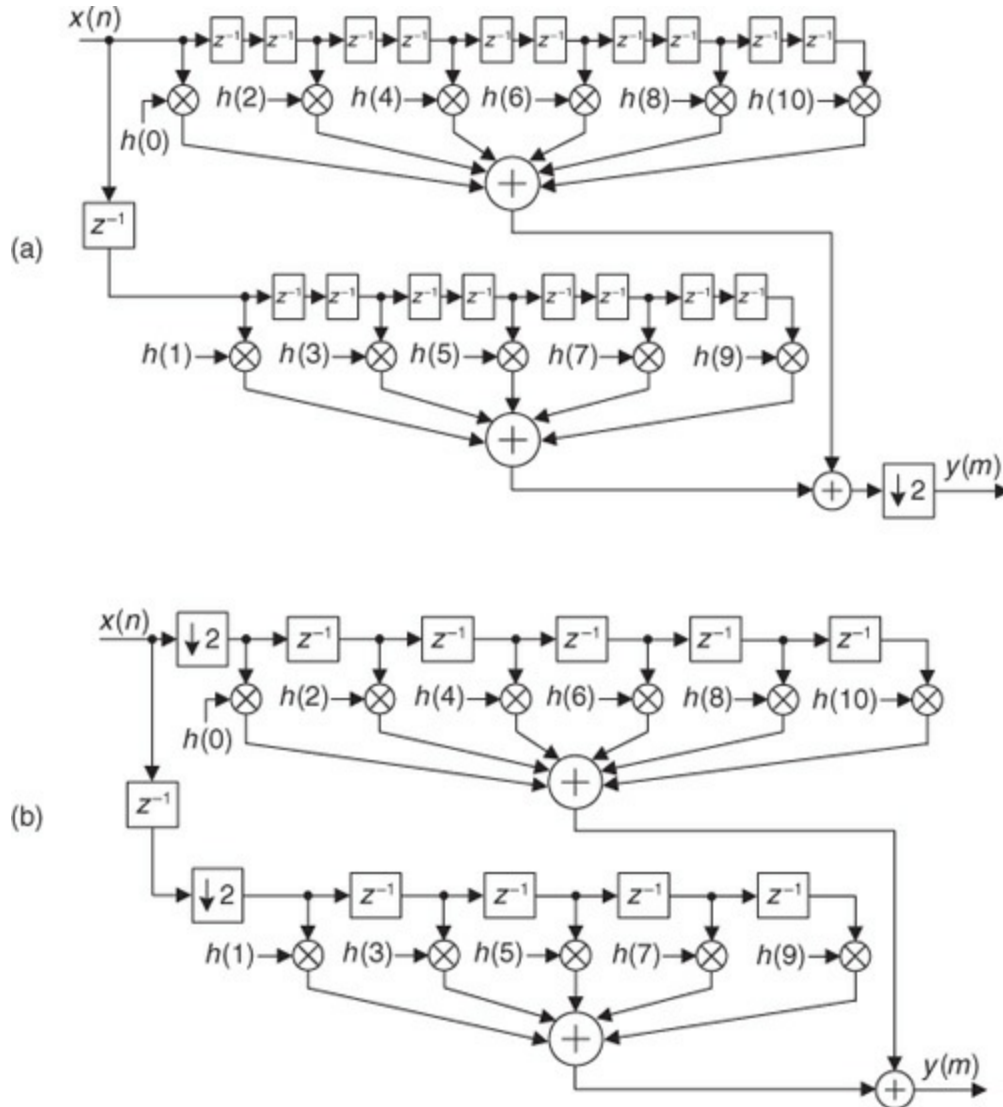
10.12.2 Half-band Filter Implementations

Here we discuss several important aspects of implementing half-band FIR filters for sample rate

conversion and show why these filters are computationally efficient. We illustrate half-band filter implementations in sample rate conversion applications with a decimation-by-two example showing the details of a polyphase decomposition process.

Suppose we need an $N = 11$ -tap half-band FIR filter in a decimation-by-two application. We could use a standard 11-tap tapped-delay line half-band filter, as discussed in [Chapter 5](#), followed by a downsample-by-two operation. Instead we choose to use polyphase decomposition as shown in [Figure 10-27\(a\)](#).

Figure 10-27 An 11-tap polyphase half-band decimation filter: (a) polyphase form; (b) polyphase with downsampling prior to filtering.



Recall that a prototype FIR filter, which we want to decompose into Q polyphase subfilters for a resample by Q application, must have an integer multiple of Q taps. So we can think of our 11-tap FIR filter as being a 12-tap filter with the $h(11)$ twelfth coefficient being zero-valued.

Read no further until you convince yourself that the two subfilters in [Figure 10-27\(a\)](#), whose outputs are summed, is equivalent to a standard 11-tap tapped-delay line half-band filter, where both implementations have a z -domain transfer function of

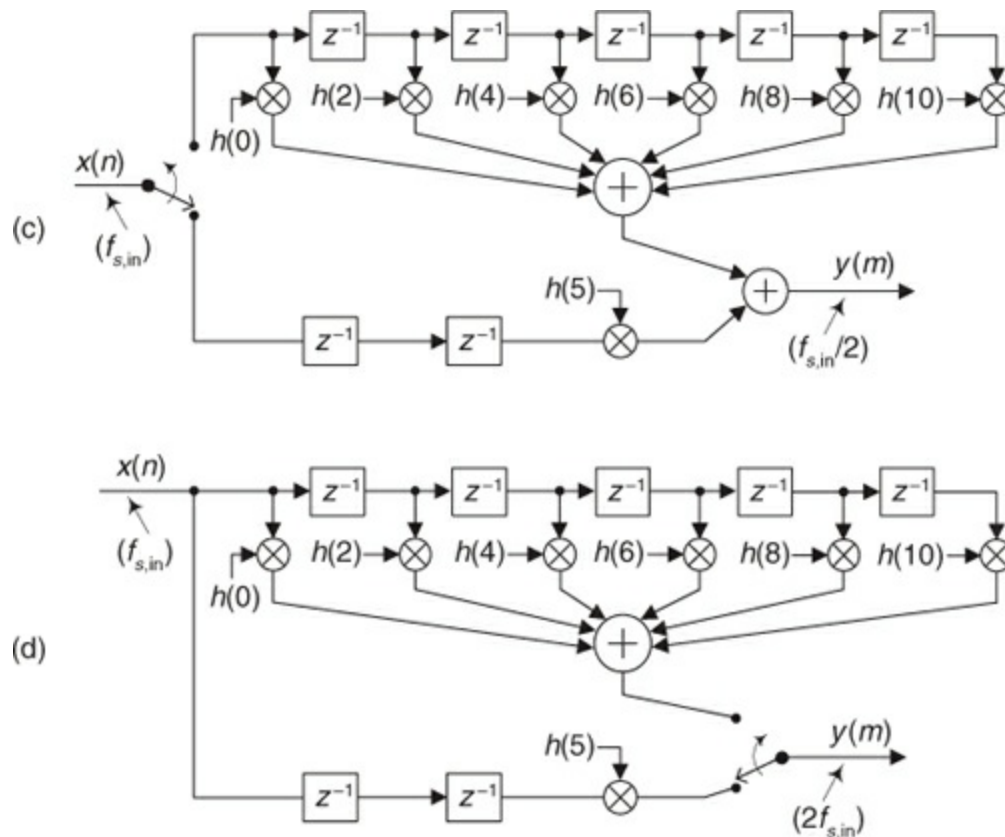
$$(10-21)$$

$$H(z) = h(0) + h(1)z^{-1} + h(2)z^{-2} + h(3)z^{-3} + \dots + h(10)z^{-10}.$$

Next, we place the downsample-by-two operation in [Figure 10-27\(a\)](#) ahead of the tapped-delay lines

as shown in [Figure 10-27\(b\)](#). That modification, because of our noble identities, reduces each dual delay element in [Figure 10-27\(a\)](#) to a single delay element as shown in [Figure 10-27\(b\)](#).

Figure 10-27 Half-band filter implementations: (c) decimation by two; (d) interpolation by two.



Applying the input commutating switch implementation introduced in [Figure 10-24](#), our [Figure 10-27\(b\)](#) decimation-by-two polyphase half-band filter becomes what is shown in [Figure 10-27\(c\)](#). Because only one of the odd-indexed filter coefficients is nonzero, namely $h(5) \neq 0$, we have only one multiply operation in the bottom path of our final polyphase half-band filter. Again, by using this polyphase implementation, we compute no filter output samples destined to be discarded by the downsample-by-two operation, and happily all filter computations take place at the decimated (lower) sample rate.

[Figure 10-27\(d\)](#) presents the structure of a polyphase version of a half-band filter that eliminates any multiply by zero computations in an interpolation-by-two application.

If the number of taps in a half-band filter is N , where $N+1$ is an integer multiple of four, then the number of unit-delay elements in the filters' bottom paths in [Figures 10-27\(c\)](#) and [10-27\(d\)](#) is $(N-3)/4$.

Because the half-band filter coefficients in the top path are symmetrical, thankfully, we can use the *folded* FIR filter scheme described in [Section 13.7](#) to reduce the number of multipliers in the top path by a factor of two. This means we can achieve the filtering performance of an N -tap half-band FIR filter while performing only, roughly, $N/4$ multiplies per filter output sample. Neat!

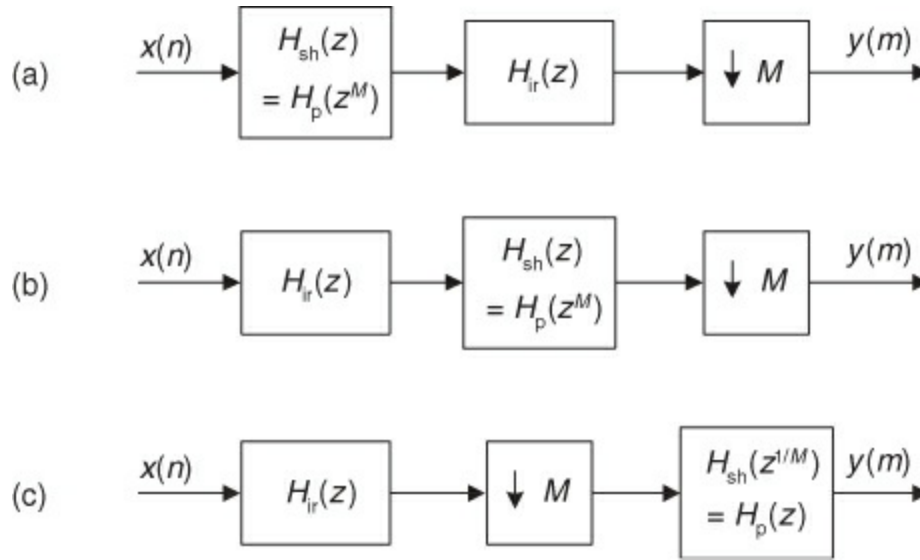
If [Figures 10-27\(c\)](#) and [10-27\(d\)](#)'s half-band filters' coefficients are designed such that $h(5) = 0.5$, which is often the case with commercial filter design software, the bottom path's multiplication by $h(5)$ can be replaced with a binary right-shift-by-one-bit operation. On the other hand, to compensate for the amplitude loss by a factor of two inherent in interpolation by two, the coefficients in [Figure 10-27\(d\)](#) are multiplied by two to make the filter's gain equal to two. In that case coefficient $h(5)$ becomes one, eliminating the bottom path multiplication altogether.

10.13 Sample Rate Conversion with IFIR Filters

The interpolated FIR (IFIR) filters that we introduced in [Chapter 7](#) are particularly useful in sample rate conversion applications because they're computationally efficient, and their signal data storage requirements can be reduced in such applications.

To see why this is so, we refer to [Figure 10-28\(a\)](#) showing a standard IFIR filter with its cascaded shaping and image-reject subfilters followed by downsampling by integer M (discard all but every M th sample). The high-order $H_{\text{sh}}(z)$ shaping filter is an upsampled (zero-stuffed) by M version of an $H_{\text{p}}(z)$ prototype lowpass filter as discussed in [Chapter 7](#). Because the $H_{\text{sh}}(z)$ shaping subfilter and the $H_{\text{ir}}(z)$ image-reject subfilter are linear and time invariant, we can swap their order as depicted in [Figure 10-28\(b\)](#). Now comes the good part.

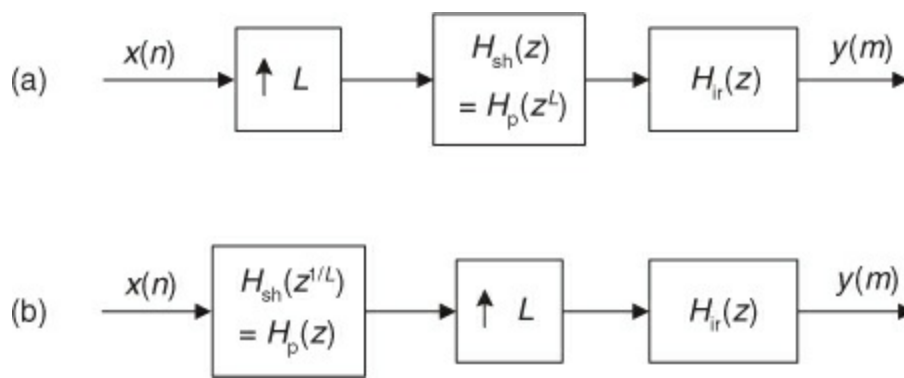
Figure 10-28 IFIR filter structures used for decimation.



Due to the noble identities we can swap the order of the $H_{\text{sh}}(z)$ subfilter with the downsampler and arrive at the structure shown in [Figure 10-28\(c\)](#). Every M -unit delay in the $H_{\text{sh}}(z)$ filter's tapped-delay line is now replaced by a single unit delay, which takes us back to using our original low-order $H_{\text{p}}(z)$ prototype filter. This fortunate scenario reduces the signal data storage requirements of our traditional IFIR filter. In addition, the $H_{\text{ir}}(z)$ and M downsampler combination can be implemented using polyphase filtering to further reduce their computational complexity.

In a similar manner, IFIR filters can be used for interpolation as shown in [Figure 10-29\(a\)](#). There we show an upsampling process followed by a standard IFIR filter structure. Again, we can swap the order of subfilter $H_{\text{sh}}(z)$ with the upsampler and arrive at the structure shown in [Figure 10-29\(b\)](#). Every L -unit delay in $H_{\text{sh}}(z)$ is now replaced by a single unit delay, which, again, takes us back to using our original low-order prototype filter $H_{\text{p}}(z)$ with its reduced data storage requirements. The L upsampler and $H_{\text{ir}}(z)$ combination can be implemented using polyphase filtering to reduce their computational workload.

Figure 10-29 IFIR filter structures used for interpolation.



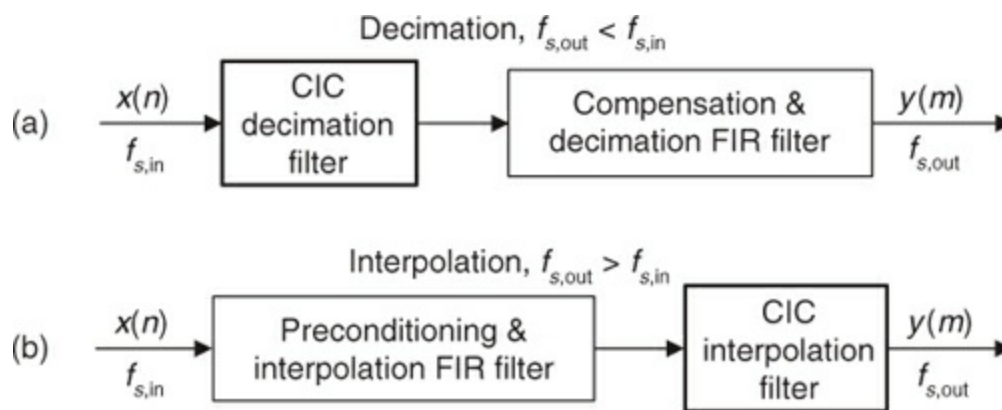
Before concluding this chapter on sample rate conversion, we introduce one final topic, cascaded integrator-comb filters. These important filters have become popular for sample rate conversion in the hardware design of modern digital communications systems.

10.14 Cascaded Integrator-Comb Filters

Cascaded integrator-comb (CIC) filters are computationally efficient implementations of narrowband lowpass filters and, as such, are used in hardware implementations of decimation and interpolation.

CIC filters are well suited to improve the efficiency of anti-aliasing filtering prior to decimation, as shown in [Figure 10-30\(a\)](#), and for anti-imaging filtering for interpolating signals as in [Figure 10-30\(b\)](#). Both applications are associated with very high-data-rate filtering such as hardware quadrature modulation and demodulation in modern wireless systems, and delta-sigma A/D and D/A converters.

Figure 10-30 CIC filter applications: (a) decimation; (b) interpolation.



Because their frequency magnitude response envelopes are $\sin(x)/x$ -like, CIC filters are typically followed, or preceded, by higher-performance linear-phase lowpass tapped-delay line FIR filters whose task is to compensate for the CIC filter's non-flat passband as shown in [Figure 10-30](#). That cascaded-filter architecture has valuable benefits. For example, with decimation, narrowband lowpass filtering can be attained at a greatly reduced computational complexity from that of a single lowpass FIR filter due to the initial CIC filtering. In addition, the follow-on FIR filter operates at reduced clock rates, minimizing power consumption in high-speed hardware applications. A crucial bonus in using CIC filters, the property that makes them popular in hardware devices, is that they require no multiplications. Their arithmetic is additions and subtractions only.

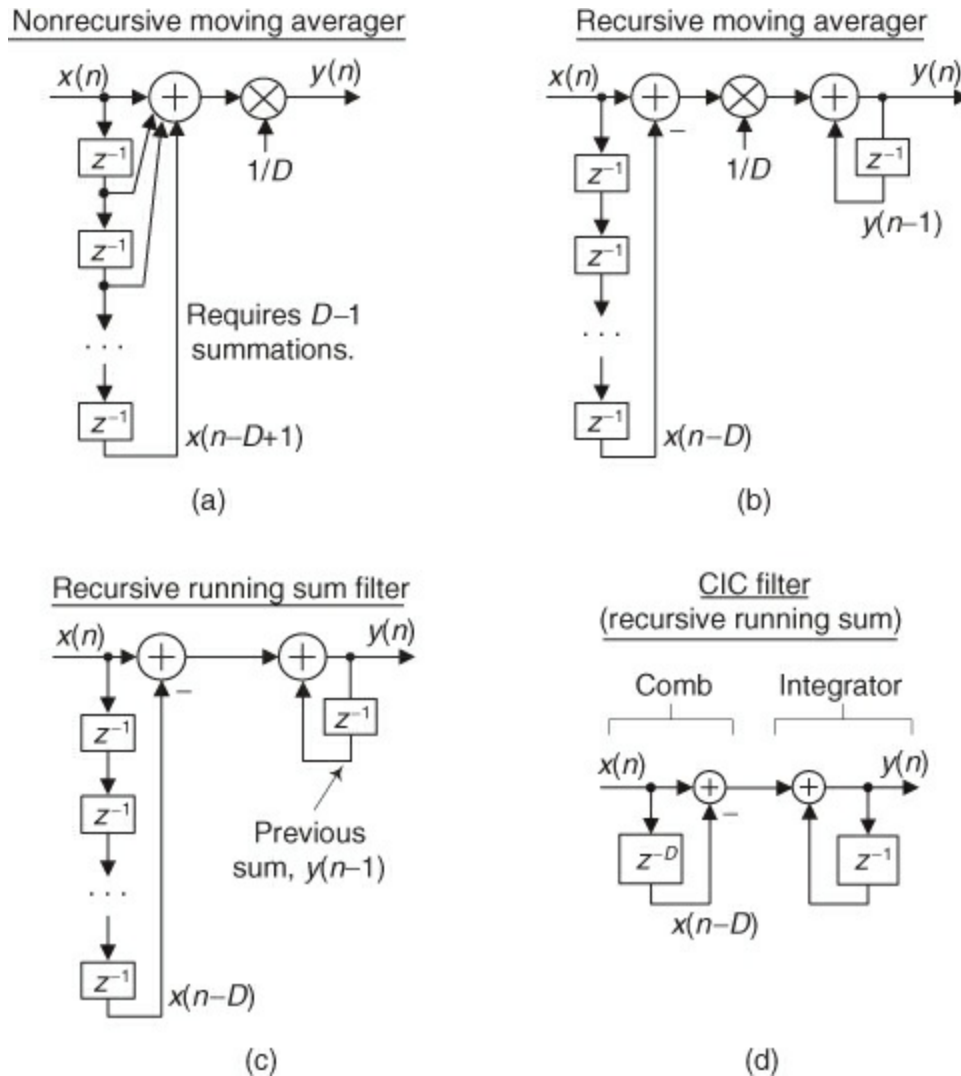
While CIC filters were introduced to the signal processing community over two decades ago, their application possibilities have grown in recent years[\[10\]](#). That's because improvements in VLSI integrated circuit technology, increased use of polyphase filtering techniques, advances in delta-sigma converter implementations, and the significant growth in wireless communications systems have

spurred much interest in, and improvements upon, traditional CIC filters. Here we'll introduce the structure and behavior of traditional CIC filters, present their frequency-domain performance, and discuss several important implementation issues.

10.14.1 Recursive Running Sum Filter

CIC filters originate from the notion of a *recursive running sum* filter, which is itself an efficient version of the standard nonrecursive *moving averager*. Reviewing a D -point nonrecursive moving average process in [Figure 10-31\(a\)](#), we see that $D-1$ summations (plus one multiply by $1/D$) are necessary to compute each $y(n)$ output sample.

Figure 10-31 D -point averaging filters: (a) nonrecursive moving averager; (b) recursive moving averager; (c) recursive running sum filter; (d) CIC version of a recursive running sum filter.



The D -point nonrecursive moving average filter's $y(n)$ time-domain output is expressed as

$$(10-22)$$

$$y(n) = \frac{1}{D} [x(n) + x(n-1) + x(n-2) + x(n-3) + \dots + x(n-D+1)].$$

The z -domain expression for this nonrecursive moving averager is

$$(10-23)$$

$$Y(z) = \frac{1}{D} [X(z) + X(z)z^{-1} + X(z)z^{-2} + \dots + X(z)z^{-(D-1)}]$$

while its z -domain $H_{\text{ma}}(z)$ transfer function is

(10-24)

$$H_{\text{ma}}(z) = \frac{Y(z)}{X(z)} = \frac{1}{D} [1 + z^{-1} + z^{-2} + \dots + z^{-D+1}] = \frac{1}{D} \sum_{n=0}^{D-1} z^{-n}$$

where the subscript “ma” means “moving average.”

An equivalent, but more computationally efficient, form of a moving averager is the recursive moving averager depicted in [Figure 10-31\(b\)](#). The recursive moving averager has the sweet advantage that only two additions are required per output sample, regardless of the delay length D !

Notice that the delay line of the recursive moving averager has D delay elements, while the nonrecursive moving averager has $D-1$ delay elements. The recursive moving averager’s difference equation is

(10-25)

$$y(n) = \frac{1}{D} [x(n) - x(n-D)] + y(n-1),$$

having a z -domain $H_{\text{rma}}(z)$ transfer function of

(10-26)

$$H_{\text{rma}}(z) = \frac{1}{D} \cdot \frac{1 - z^{-D}}{1 - z^{-1}}$$

where the subscript “rma” means “recursive moving average.” What is interesting is that the nonrecursive moving averager and the recursive moving averager have identical behavior and, as such, $H_{\text{ma}}(z) = H_{\text{rma}}(z)$. The transfer functions of the two averagers are equal to each other! (Actually, we saw the equivalence of nonrecursive FIR filters and special recursive structures once before—it was in regard to frequency sampling filters in [Section 7.1](#).)

If we ignore the $1/D$ gain factor, we have a structure known as a *recursive running sum filter* shown in [Figure 10-31\(c\)](#). Next we’ll see how a CIC filter is itself a recursive running sum filter.

10.14.2 CIC Filter Structures

If we condense the delay line representation in [Figure 10-31\(c\)](#), we obtain the classic representation of a single-stage (1st-order) CIC filter, whose cascade structure (block diagram) is shown in [Figure 10-31\(d\)](#). The feedforward portion of the CIC filter is called the *comb* section, whose *differential delay* is D , and the feedback section is called an *integrator*. The comb stage subtracts a delayed input sample from the current input sample, and the integrator is simply an accumulator (performing summations). The CIC filter’s difference equation is

(10-27)

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

and its z -domain transfer function is

(10-28)

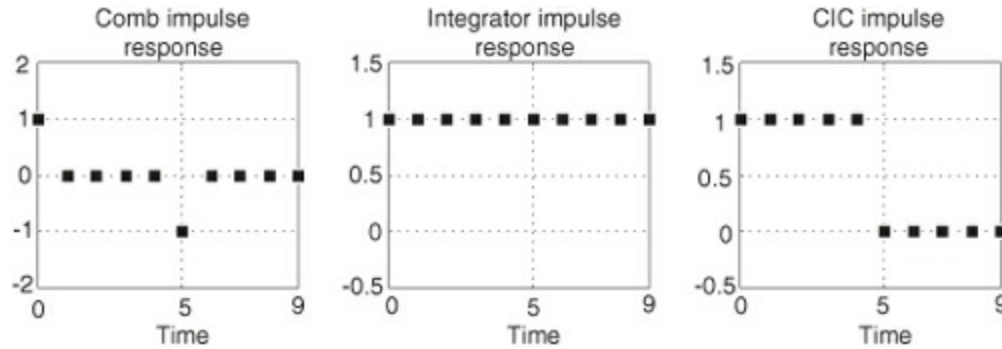
$$H_{\text{cic}}(z) = \frac{1 - z^{-D}}{1 - z^{-1}}.$$

Looking at [Eq. \(10-28\)](#), we see that the numerator is the transfer function of the comb filter and the

denominator is the transfer function of the integrator.

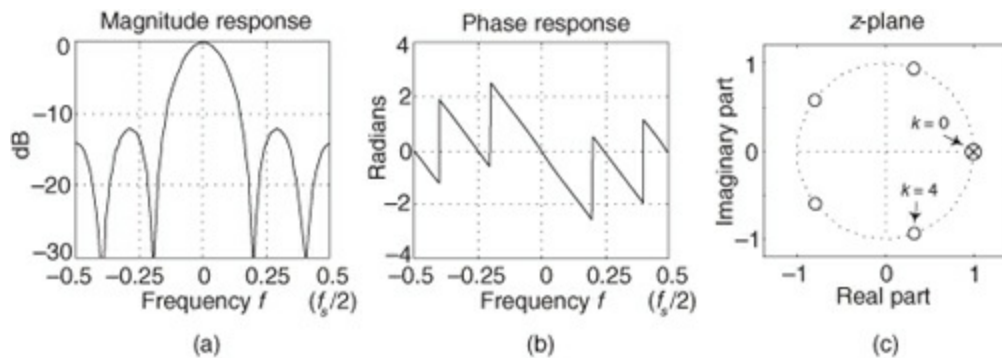
To see why the CIC filter is of interest, first we examine its time-domain behavior, for $D = 5$, shown in [Figure 10-32](#). If a unit impulse sequence, a unity-valued sample followed by many zero-valued samples, was applied to the comb stage, that stage's output is as shown in [Figure 10-32\(a\)](#). Think, now, what would be the output of the integrator if its input was the comb stage's impulse response? The initial positive impulse from the comb filter starts the integrator's all-ones output. Then, D samples later, the negative impulse from the comb stage arrives at the integrator to make all further CIC filter output samples equal to zero.

Figure 10-32 Single-stage CIC filter time-domain responses when $D = 5$.



The key issue is the combined unit impulse response of the CIC filter being a rectangular sequence, identical to the unit impulse response of the recursive running sum filter. (Moving averagers, recursive running sum filters, and CIC filters are close kin. They have the same z -domain pole/zero locations, their frequency magnitude responses have identical shapes, their phase responses are identical, and their transfer functions differ only by a constant scale factor.) The frequency magnitude (in dB) and linear-phase response of a $D = 5$ CIC filter are shown in [Figure 10-33\(a\)](#).

Figure 10-33 Characteristics of a single-stage CIC filter when $D = 5$: (a) magnitude response; (b) phase response; (c) pole/zero locations.



We can obtain an expression for the CIC filter's frequency response by evaluating [Eq. \(10-28\)](#)'s $H_{\text{cic}}(z)$ transfer function on the z -plane's unit circle, by setting $z = e^{j\omega} = e^{j2\pi f}$, yielding

$$(10-29)$$

$$H_{\text{cic}}(f) = \frac{1 - e^{-j2\pi f D}}{1 - e^{-j2\pi f}} = \frac{e^{-j2\pi f D/2} (e^{j2\pi f D/2} - e^{-j2\pi f D/2})}{e^{-j2\pi f/2} (e^{j2\pi f/2} - e^{-j2\pi f/2})}.$$

In [Eq. 10-29](#) the frequency variable f is in the range of -0.5 to 0.5 , corresponding to a continuous-time frequency range of $-f_s/2$ to $f_s/2$ Hz. Using Euler's identity $2j\sin(\alpha) = e^{j\alpha} - e^{-j\alpha}$, we can write

$$(10-30)$$

$$H_{\text{cic}}(f) = \frac{e^{-j2\pi f D/2}}{e^{-j2\pi f/2}} \frac{2j \sin(2\pi f D/2)}{2j \sin(2\pi f/2)} = e^{-j\pi f (D-1)} \frac{\sin(\pi f D)}{\sin(\pi f)}.$$

The first positive-frequency magnitude null in [Figure 10-33\(a\)](#), when $D = 5$ for example, is located at a frequency of $f_s/D = f_s/5 = 0.2f_s$ Hz ($f = 0.2$). [Equation \(10-30\)](#) is in the form of [Eq. \(3-46\)](#). This means, ignoring the linear-phase factor, a 1st-order CIC filter's frequency magnitude response is roughly equal to a $\sin(x)/x$ function centered at zero Hz as we see in [Figure 10-33\(a\)](#). (This is why CIC filters are sometimes called *sinc* filters.)

Let's stop here for a moment and mention a subtle characteristic of the phase of $H_{\text{cic}}(f)$. The phase angle, the $-\pi f(D-1)$ in [Eq. \(10-30\)](#), is a linear function of frequency. Plotting that phase, over the frequency range of $-0.5 \leq f \leq 0.5$, would yield a straight line (with negative slope). However, the $\sin(\pi f D)/\sin(\pi f)$ amplitude portion of [Eq. \(10-30\)](#) changes sign (polarity) between its amplitude nulls (zero amplitude). So those sign changes show up as phase discontinuities of π radians (180 degrees) in phase plots. For example, notice the phase discontinuity in [Figure 10-33\(b\)](#) at frequency $f = 0.2$. That discontinuity is π radians, because the $\sin(\pi f D)/\sin(\pi f)$ amplitude term changed sign from positive to negative at $f = 0.2$.

The z -plane pole/zero characteristics of a $D = 5$ CIC filter are provided in [Figure 10-33\(c\)](#), where the comb filter produces D zeros, equally spaced around the unit circle, and the integrator produces a single pole canceling the zero at $z = 1$. Each of the comb's zeros, being a D th root of 1, are located at $z(k) = e^{j2\pi k/D}$, where $k = 0, 1, 2, \dots, D-1$.

The normally risky situation of having a filter pole directly on the unit circle need not trouble us here because there is no coefficient quantization error in our $H_{\text{cic}}(z)$ transfer function. CIC filter coefficients are ones and can be implemented with perfect precision using binary numbers. Although recursive, CIC filters are guaranteed stable, linear phase as shown in [Figure 10-33\(b\)](#) and have finite-length impulse responses.

If we examine just the magnitude of $H_{\text{cic}}(f)$ from [Eq. \(10-30\)](#), we can determine the DC (zero Hz) gain of our single-stage [Figure 10-31\(d\)](#) CIC filter. However, setting $f = 0$ in [Eq. \(10-30\)](#), we have

$$(10-31)$$

$$\text{CIC filter gain} = |H_{\text{cic}}(f)|_{f=0} = \left| \frac{\sin(0)}{\sin(0)} \right| = \frac{0}{0}$$

which is indeterminate. But don't worry, we can apply the Marquis de L'Hopital's rule to the magnitude-only portion of [Eq. \(10-30\)](#), then set $f = 0$, to yield

$$(10-32)$$

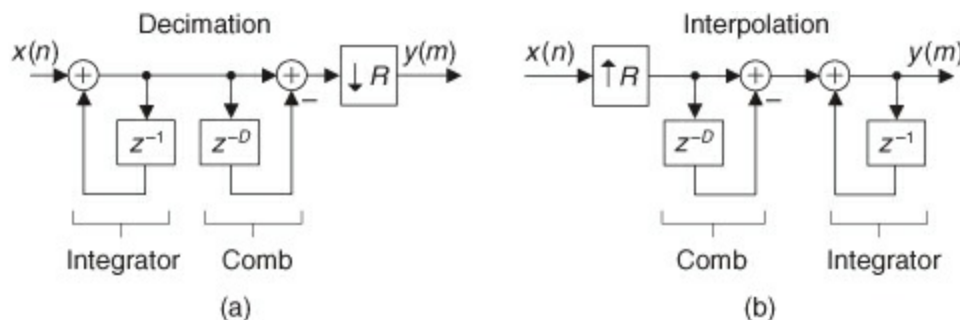
$$\begin{aligned} \text{CIC filter gain} &= \frac{d[\sin(\pi f D)] / d[f]}{d[\sin(\pi f)] / d[f]} \\ &= \frac{\cos(\pi f D)(\pi D)}{\cos(\pi f)(\pi)} \Big|_{f=0} = \frac{\cos(0)(\pi D)}{\cos(0)(\pi)} = D. \end{aligned}$$

So, the DC gain of a 1st-order CIC filter is equal to the comb filter delay D . This fact will be very important to us when we actually implement a CIC filter in hardware.

CIC filters are primarily used for anti-aliasing filtering prior to decimation and for anti-imaging

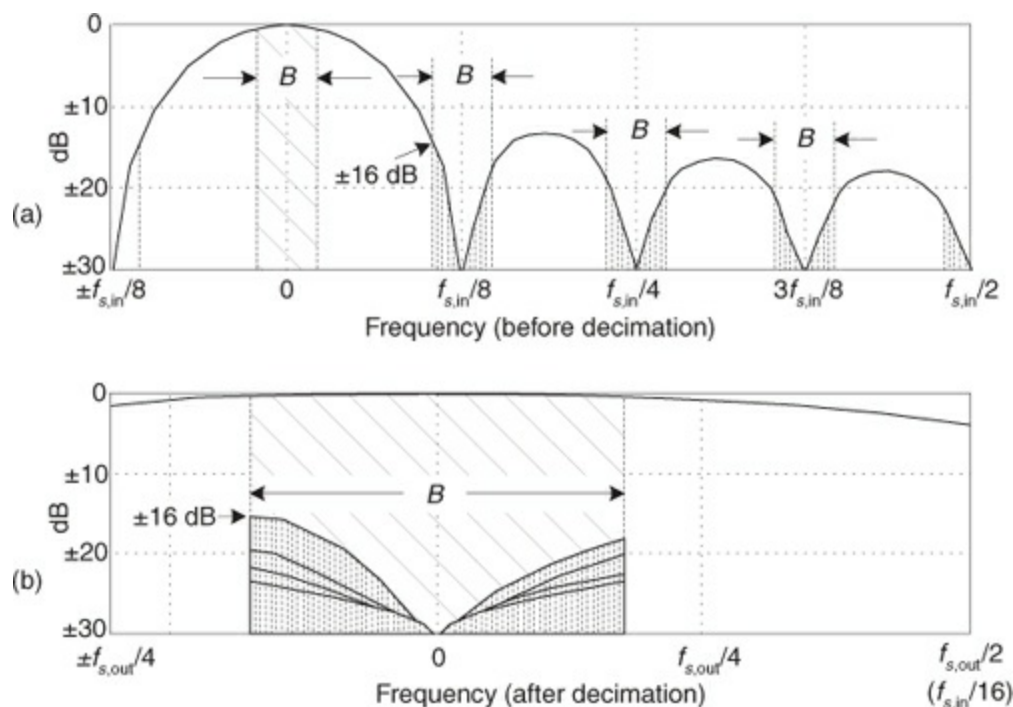
filtering for interpolated signals. With those notions in mind, we swap the order of [Figure 10-31\(c\)](#)'s comb and integrator—we're permitted to do so because those are linear time-invariant operations—and include downsampling by a sample rate conversion factor R in [Figure 10-34\(a\)](#). (Readers should prove to themselves that the unit impulse response of the integrator/comb combination, prior to the sample rate conversion, in [Figure 10-34\(a\)](#) is equal to that in [Figure 10-32\(c\)](#).) In most CIC filter applications the sample rate change factor R is equal to the comb's differential delay D , but we'll keep them as separate design parameters for now.

Figure 10-34 Single-stage CIC filters, used in: (a) decimation; (b) interpolation.



The downsampling operation in [Figure 10-34\(a\)](#) results in an output sample rate of $f_{s,\text{out}} = f_{s,\text{in}}/R$. To investigate a CIC filter's frequency-domain behavior in more detail, [Figure 10-35\(a\)](#) shows the frequency magnitude response of a $D = 8$ CIC filter prior to downsampling. The spectral band, of width B , centered at zero Hz, is the desired passband of the filter. A key aspect of CIC filters is the spectral aliasing that takes place due to downsampling.

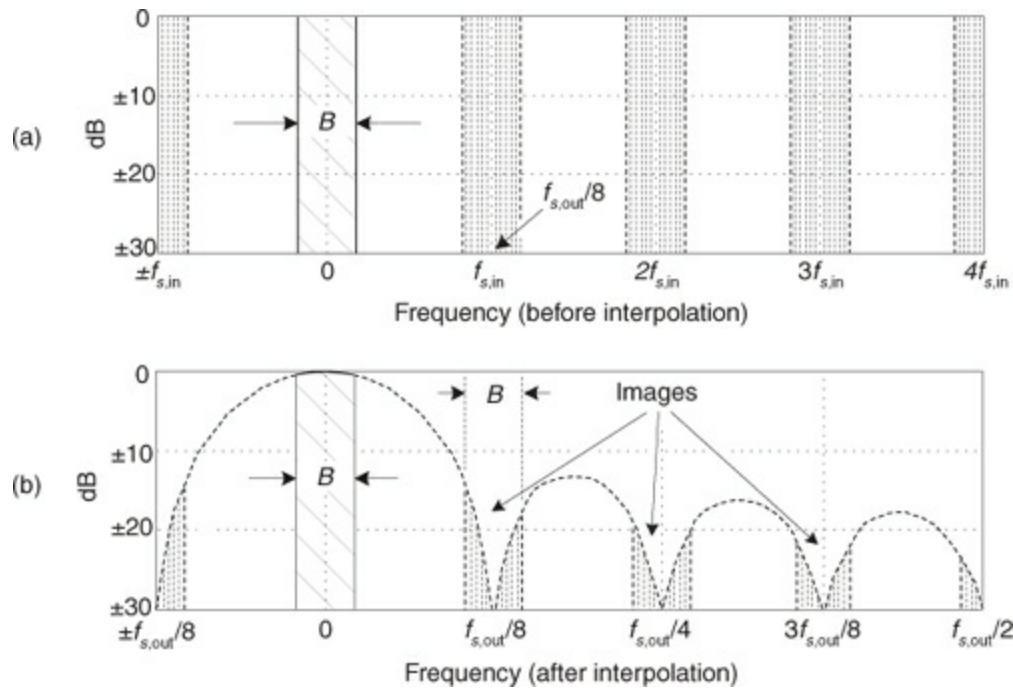
Figure 10-35 Frequency magnitude response of a 1st-order, $D = 8$, decimating CIC filter: (a) response before decimation; (b) response and aliasing after $R = 8$ downsampling.



Those B -width shaded spectral bands centered at multiples of $f_{s,\text{in}}/R$ in [Figure 10-35\(a\)](#) will alias directly into our desired passband after downsampling by $R = 8$ as shown in [Figure 10-35\(b\)](#). Notice how the largest *aliased* spectral component, in this example, is approximately 16 dB below the peak of the band of interest. Of course, the aliased power levels depend on the bandwidth B —the smaller B is, the lower the aliased energy after downsampling.

[Figure 10-34\(b\)](#) shows a CIC filter used for interpolation where upsampling by R yields a $y(m)$ output sample rate of $f_{s,\text{out}} = Rf_{s,\text{in}}$. (In this CIC filter discussion, interpolation is defined as zeros-insertion upsampling followed by filtering.) [Figure 10-36\(a\)](#) shows an arbitrary baseband spectrum, with its spectral replications, of a signal applied to the $D = R = 8$ interpolating CIC filter of [Figure 10-34\(b\)](#). The filter's output spectrum in [Figure 10-36\(b\)](#) shows how imperfect filtering gives rise to the undesired spectral images.

Figure 10-36 Spectra of a 1st-order, $D = R = 8$, interpolating CIC filter: (a) input spectrum before interpolation; (b) output spectral images.

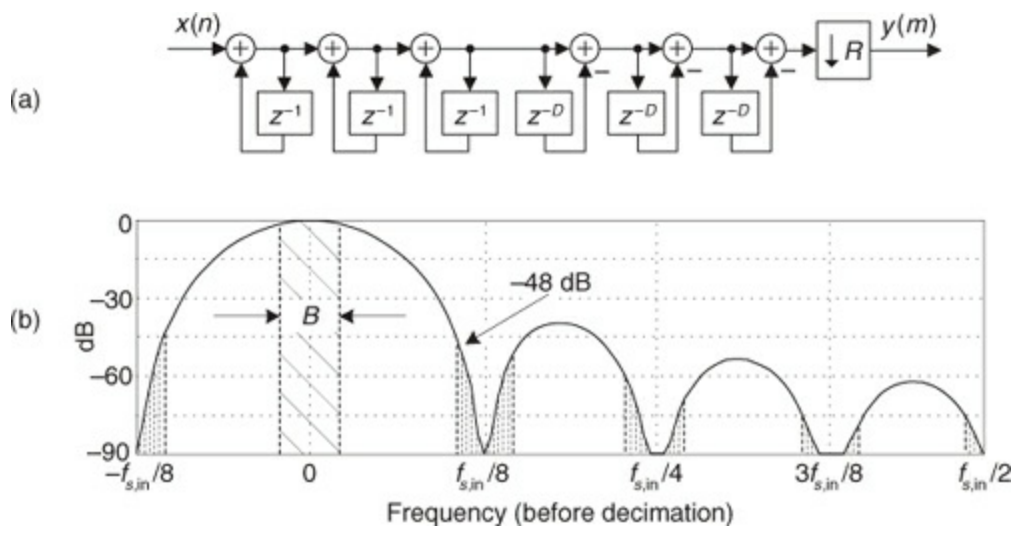


After interpolation, unwanted images of the B -width baseband spectrum reside at the null centers, located at integer multiples of $f_{s,\text{out}}/D$. If we follow the CIC filter with a traditional lowpass tapped-delay line FIR filter, whose stopband includes the first image band, fairly high image rejection can be achieved.

10.14.3 Improving CIC Attenuation

The most common method to improve CIC filter anti-aliasing and image attenuation is by increasing the order Q of the CIC filter using multiple stages. [Figure 10-37](#) shows the structure and frequency magnitude response of a 3rd-order ($Q = 3$) CIC decimation filter.

Figure 10-37 A 3rd-order ($Q = 3$), $D = R = 8$ CIC decimation filter: (a) structure; (b) frequency magnitude response before decimation.



Notice the increased attenuation at multiples of $f_{s,\text{in}}/D$ in [Figure 10-37\(b\)](#) compared to the 1st-order CIC filter in [Figure 10-35\(a\)](#). Because the $Q = 3$ CIC stages are in cascade, the overall before-decimation transfer function will be the product of their individual single-stage transfer functions, or

(10-33)

$$H_{\text{cic},Q\text{th-order}}(z) = \left[\frac{1 - z^{-D}}{1 - z^{-1}} \right]^Q.$$

The overall frequency magnitude response of the $Q = 3$ cascaded stages, before decimation, will be

(10-34)

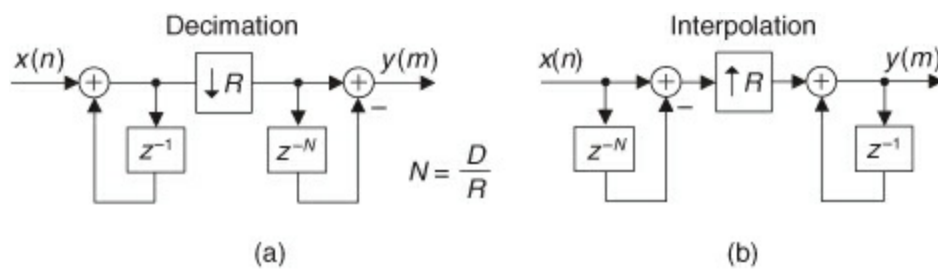
$$|H_{\text{cic},Q\text{th-order}}(f)| = \left| \frac{\sin(\pi f D)}{\sin(\pi f)} \right|^Q.$$

where, again, the frequency variable f is in the range of -0.5 to 0.5 corresponding to a continuous-time frequency range of $-f_s/2$ to $f_s/2$ Hz. The price we pay for improved anti-alias attenuation is additional hardware adders and increased CIC filter passband droop. An additional penalty of increased orders comes from the DC (zero Hz) gain of the decimation filter, which is D^Q . That potentially large gain causes significant binary data word-width growth for higher-order filters. Even so, this multistage decimation implementation is common in commercial integrated circuits, where a Q th-order CIC filter is called a *sinc* ^{Q} filter.

10.14.4 CIC Filter Implementation Issues

With CIC filters, the comb section can precede, or follow, the integrator section. However, it's sensible to put the comb section on the side of the filter operating at the lower sample rate to reduce the length of the delay line. Using the noble identities discussed earlier in this chapter, swapping the [Figure 10-34](#) comb filters with the rate conversion operations results in the most common implementation of CIC filters as shown in [Figure 10-38](#). Notice that the decimation filter's comb section now has a delay length (differential delay) of $N = D/R$. That's because an N -sample delay after downsampling by R is equivalent to a D -sample delay before downsampling by R . Likewise for the interpolation filter; an N -sample delay before upsampling by R is equivalent to a D -sample delay after upsampling by R .

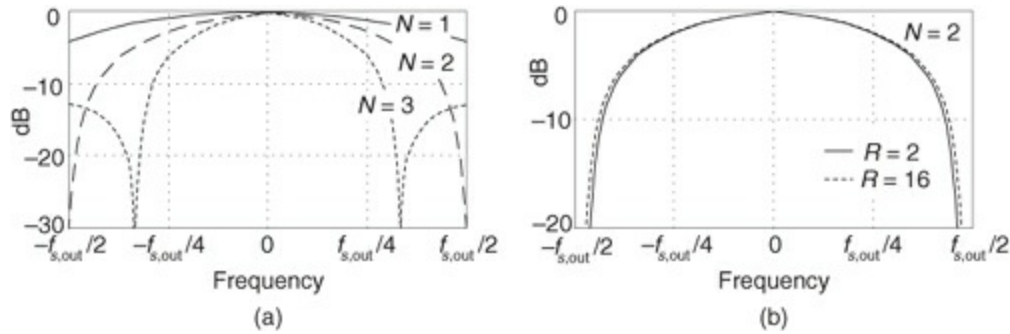
Figure 10-38 Single-stage CIC filter implementations: (a) for decimation; (b) for interpolation.



Those [Figure 10-38](#) configurations yield two major benefits: First, the comb section's new differential delay is decreased to $N = D/R$, reducing data storage requirements; second, the comb section now operates at a reduced clock rate. Both of these effects reduce hardware power consumption.

The comb section's differential delay design parameter $N = D/R$ is typically 1 or 2 for high-sample-rate conversion ratios as is often done in commercial up/down-converter chips. Value N effectively sets the number of nulls in the frequency response of a decimation filter, as shown in [Figure 10-39\(a\)](#).

Figure 10-39 CIC decimation filter frequency responses: (a) for various values of differential delay N , when $R = 8$; (b) for two R downsampling factors when $N = 2$.

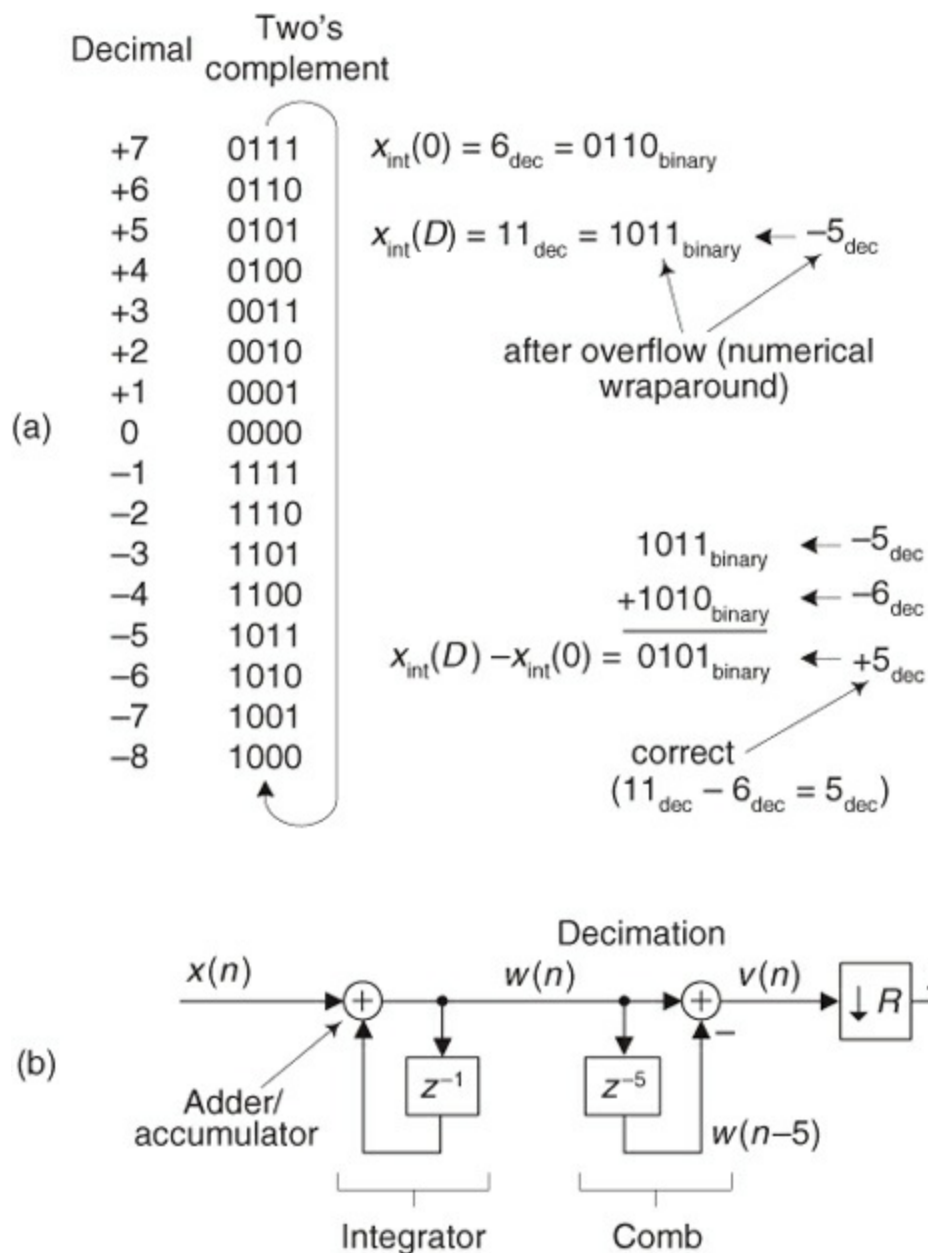


An important characteristic of a CIC decimator is that the shape of the filter response, relative to its $f_{s,out}$ output sample rate, changes very little as a function of the downsampling factor R , as shown in [Figure 10-39\(b\)](#). For R larger than roughly 16, the change in the filter shape is negligible. Fortunately, this allows the same compensation FIR filter to be used for variable-decimation ratio systems.

The gain of a Q th-order CIC decimation filter is D^Q , and individual integrators within the filter can experience overflow. (An integrator's gain is infinite at DC!) As such, the use of two's complement (non-saturating) arithmetic resolves this overflow situation just so long as the integrator word width accommodates the maximum value expected at the CIC filter output. Happily, using the two's complement binary number format, with its modular wraparound property, the follow-on comb filter will properly compute the correct difference between two successive integrator output samples.

To show this behavior, assume we're using a four-bit two's complement number format, and a CIC decimation filter's integrator must sum the values $7 + 4$ and the comb filter must subtract 6 from that sum. [Figure 10-40\(a\)](#) shows how a previous integrator output $x_{int}(0)$ sample of decimal 6 can be subtracted by the comb filter from a later $x_{int}(D)$ integrator output sample of decimal 11 ($11 = 7 + 4$, a temporary overflow condition), resulting in a correct difference of decimal plus 5 ($+5_{dec}$).

Figure 10-40 Two's complement overflow (numerical wraparound): (a) difference example; (b) $D = 5$ decimation example.



This two's complement wraparound issue is so important that it deserves a second example. Think of the $D = 5$ decimation filter in [Figure 10-40\(b\)](#). If we applied a unit step input (an all-ones $x(n)$ sequence) at time $n = 1$, we expect the $v(n)$ sequence to ramp up to a decimal value of 5 and remain at that value. Now if the integrator's adder/accumulator register was only three bits wide, it will *not* accommodate the $v(n)$ output of 5 because the most positive value of a three-bit word in two's complement format is +3. That scenario is shown on the left side of [Table 10-2](#), where all the values are shown in decimal format. There we see that the $v(n)$ sequence goes to an incorrect value of -3.

Table 10-2 Accumulator Example for $D = 5$ Decimation

n	Three-bit integrator accumulator				Four-bit integrator accumulator		
	$w(n)$	$w(n-5)$	$v(n)$		$w(n)$	$w(n-5)$	$v(n)$
0	0	0	0		0	0	0
1	1	0	1		1	0	1
2	2	0	2		2	0	2
3	3	0	3		3	0	3
4	-4	0	-4		4	0	4
5	-3	0	-3		5	0	5
6	-2	1	-3		6	1	5
7	-1	2	-3		7	2	5
8	0	3	-3		-8	3	-11=5
9	1	-4	-3		-7	4	-11=5
10	2	-3	-3		-6	5	-11=5
11	3	-2	-3		-5	6	-11=5
12	-4	-1	-3		-4	7	-11=5
13	-3	0	-3		-3	-8	5
14	-2	1	-3		-2	-7	5
15	-1	2	-3		-1	-6	5

If we increase the integrator's accumulator width to four bits, the integrator accumulator experiences overflow but the comb filter compensates for that situation and provides the correct $v(n)$ sequence as shown on the right side of [Table 10-2](#).

So here's the bottom line: When two's complement fixed-point arithmetic is used, the number of bits in a Q th-order CIC decimation filter's integrator and comb registers must accommodate the filter's input signal times the filter's total gain of D^Q . To be specific, overflow errors are avoided if the number of integrator and comb register bit widths is at least

$$(10-35)$$

$$\text{register bit widths} = \text{number of bits in } x(n) + \lceil Q \log_2(D) \rceil,$$

where $x(n)$ is the input to the CIC filter, and $\lceil k \rceil$ means that if k is not an integer, round it up to the next larger integer. For example, if a $Q = 3$ -stage CIC decimation filter accepts one-bit binary input words from a sigma-delta A/D converter and the decimation factor is $R = D = 64$, binary overflow errors are avoided if the three integrator and three comb registers' bit widths are no less than

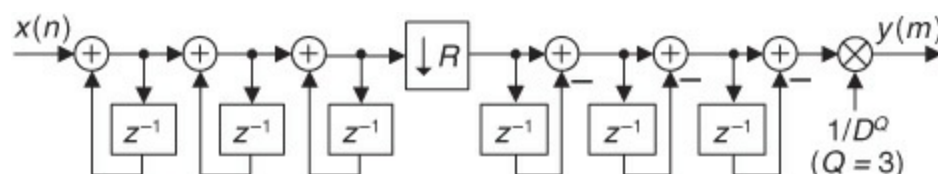
$$(10-36)$$

$$\text{register bit widths} = 1 + \lceil 3 \cdot \log_2(D) \rceil = 1 + 3 \cdot 6 = 19 \text{ bits.}$$

Regarding a CIC decimation filter's gain of D^Q , we often see a multistage CIC decimation filter implemented as shown in [Figure 10-41](#) where $R = D$, and a gain reduction (by $1/D^Q$) stage is included as a final operation. If D is an integer power of two, the multiply operation can be performed with a binary right shift. That's one of the computational benefits of decimating by an integer power of two. In the [Figure 10-41](#) scenario, the data words out of the final comb filter are shifted to the right by

$Q \log_2(D)$ bits to achieve an overall decimation filter gain of unity.

Figure 10-41 Unity gain, $Q = 3$, $D = R$, CIC decimation filter.

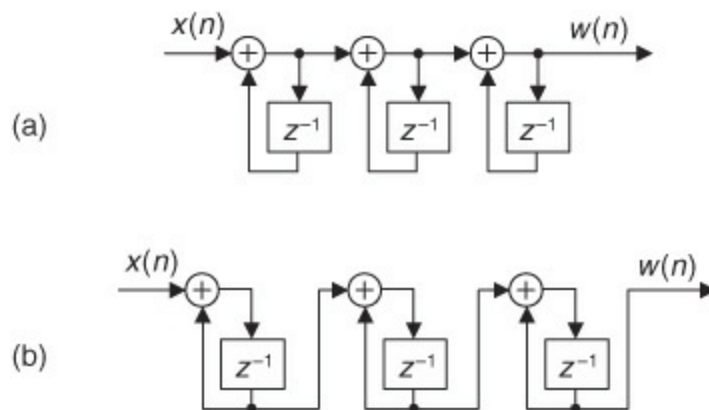


Interpolating CIC filters have zero-valued samples inserted after each original input sample reducing its gain by a factor of $1/R$, so the net gain of a CIC interpolation filter is D^Q/R . For multistage interpolation CIC filters, the integrators' register bit widths grow in size in successive integrator stages. This means that not all integrator accumulator registers need to have the same bit width, so there is some flexibility in discarding some of the least significant bits (lsbs) within the stages of a multistage CIC interpolation filter. The specific effects of this lsb removal are, however, a complicated issue, so we refer the reader to references [9,10] for more details.

While the preceding discussion focused on hardwired CIC filters, these filters can also be implemented with programmable fixed-point DSP chips. Although those chips have inflexible data paths and fixed word widths, their use of CIC filtering can be advantageous for high-sample-rate conversion. Large word widths can be accommodated with multiword additions at the expense of extra instructions. Even so, for large R the computational workload per output sample may be small compared to computations required using a more conventional tapped-delay line FIR filter approach in fixed-point DSP chips.

One further CIC filter implementation issue deserves mention. When we need to implement cascaded integrators, we showed those integrators as in [Figure 10-42\(a\)](#). As it turns out, depending on the architecture of your hardware implementation, it may be advantageous to implement those cascaded integrators as shown in [Figure 10-42\(b\)](#), where placing the unit-delay elements in the forward path reduces the pipelined critical-path delay from three adder delays to a single adder delay[11]. While the [Figure 10-42\(b\)](#) cascaded network adds additional time delay, the frequency magnitude responses are identical for the two networks in [Figure 10-42](#).

Figure 10-42 Cascaded integrator implementations: (a) traditional method; (b) reduced pipelined critical-path delay method.



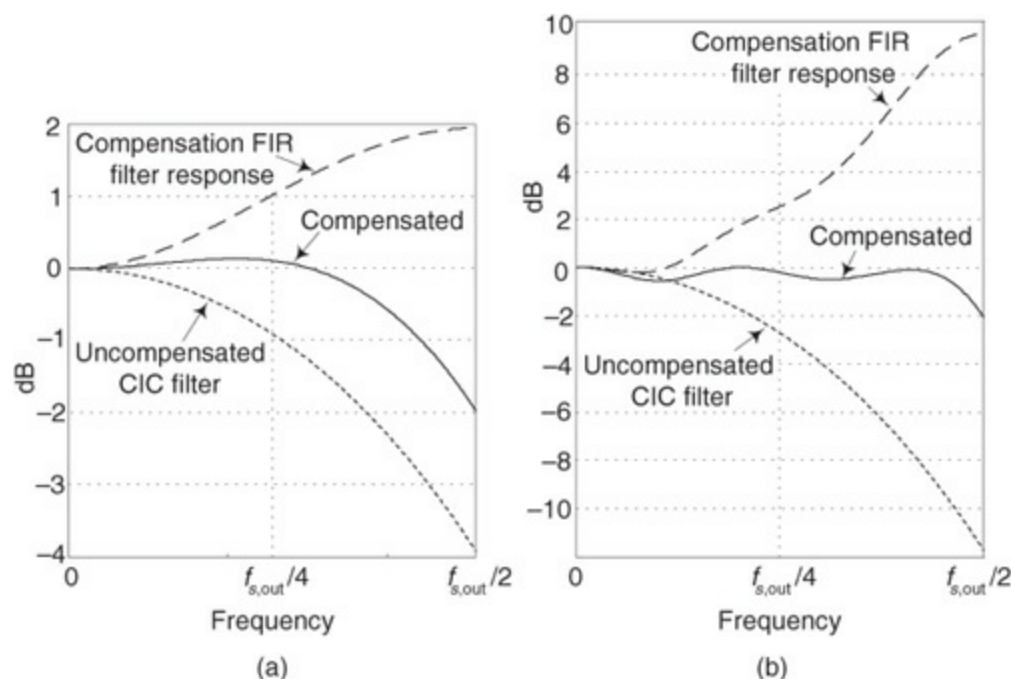
10.14.5 Compensation/Preconditioning FIR Filters

In typical decimation/interpolation filtering applications we desire a reasonably flat passband and narrow transition region filter response. These desirable properties are not provided by CIC filters alone, with their drooping passband gains and wide transition regions. We alleviate this problem, in

decimation for example, by following the CIC filter with a compensation nonrecursive FIR filter (often called an *inverse sinc filter*), as in [Figure 10-30\(a\)](#), to narrow the output bandwidth and flatten the passband gain.

The compensation FIR filter's frequency magnitude response is ideally an inverted version of the CIC filter passband response similar to that shown by the dashed curve in [Figure 10-43\(a\)](#) for a simple 3-tap FIR filter whose coefficients are $[-1/16, 9/8, -1/16]$. With the dotted curve representing the uncompensated passband droop of a 1st-order $R = 8$ CIC filter, the solid curve represents the compensated response of the cascaded filters. If either the CIC filter's order or passband width increases, the correction becomes more demanding, requiring more compensation FIR filter taps. An example of this situation is shown in [Figure 10-43\(b\)](#) where the dotted curve represents the passband droop of a 3rd-order $R = 8$ CIC filter and the dashed curve, taking the form of $[x/\sin(x)]^3$, is the response of a 15-tap compensation FIR filter having the coefficients $[-1, 4, -16, 32, -64, 136, -352, 1312, -352, 136, -64, 32, -16, 4, -1]$.

Figure 10-43 Compensation FIR filter magnitude responses, dashed curves: (a) with a 1st-order decimation CIC filter; (b) with a 3rd-order decimation CIC filter.

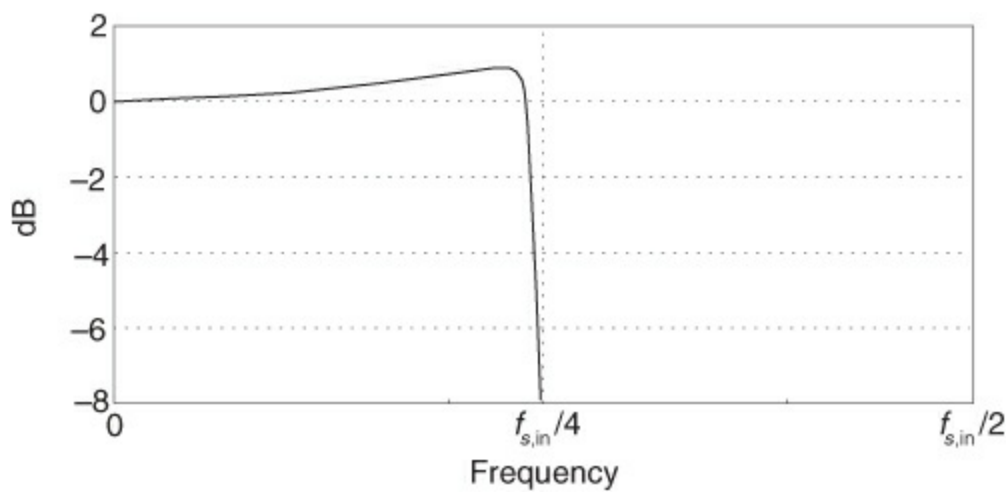


Wideband compensation also means that signals near $f_{s,out}/2$ are attenuated with the CIC filter and then must be amplified in the correction filter, which adds noise. As such, practitioners often limit the passband width of the compensation FIR filter to roughly one-fourth the frequency of the first null in the CIC filter response.[‡]

[‡] I thank my DSP pal Ray Andraka, of Andraka Consulting Group Inc., for his guidance on these implementation issues.

Those dashed curves in [Figure 10-43](#) represent the frequency magnitude responses of compensating FIR filters within which no sample rate change takes place. (The FIR filters' input and output sample rates are equal to the $f_{s,out}$ output rate of the decimating CIC filter.) If a compensating FIR filter were designed to provide an additional decimation by two, its frequency magnitude response would look similar to that in [Figure 10-44](#), where $f_{s,in}$ is the compensation filter's input sample rate.

Figure 10-44 Frequency magnitude response of a decimate-by-two compensation FIR filter.



After all of this discussion, just keep in mind that a decimating CIC filter is merely a very efficient recursive implementation of a moving average filter, having $D = NR$ taps, whose output is decimated by R . Likewise, the interpolating CIC filter is insertion of $R-1$ zero-valued samples after each original input sample followed by a $D = NR$ -tap moving average filter running at the output sample rate $f_{s,out}$. The cascade implementations in [Figure 10-30](#) result in total computational workloads far less than those when using a single tapped-delay line FIR filter alone for high-sample-rate conversion by decimation or interpolation. CIC filter structures are designed to maximize the amount of low-sample-rate processing to minimize power consumption in high-speed hardware applications. Again, CIC filters require no multiplications; their arithmetic is strictly additions and subtractions. Their performance allows us to state that, technically speaking, CIC filters are lean, mean, fat-free filtering machines.

[Section 13.24](#) provides a few advanced tricks allowing us to implement nonrecursive CIC filters, and this eases the word-width growth problem of the above traditional recursive CIC filters.

This chapter's discussion of sample rate conversion has, by necessity, only touched the surface of this important signal processing technique. Fortunately for us, the excellent work of early signal processing engineers and mathematicians is well documented in the literature of DSP. Several standard DSP textbooks briefly discuss multirate filter design concepts[[12–14](#)], and other texts are devoted exclusively to polyphase filters and multirate processing[[6–9](#)]. The inquisitive reader can probe further to learn how to choose the number of stages in a multistage process[[1,3](#)], the interrelated considerations of designing optimum FIR filters[[1,15](#)], the benefits of half-band FIR filters[[5,16](#)], when IIR filter structures may be advantageous[[15](#)], what special considerations are applicable to sample rate conversion in image processing[[17–19](#)], guidance in developing the control logic necessary for hardware implementations of rate conversion algorithms[[15](#)], how rate conversion improves the usefulness of commercial test equipment[[20,21](#)], and software development tools for designing multirate filters[[22](#)].

References

- [1] Crochiere, R., and Rabiner, L. "Optimum FIR Digital Implementations for Decimation, Interpolation, and Narrow-band Filtering," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-23, No. 5, October 1975.
- [2] Ballanger, M. "Computation Rate and Storage Estimation in Multirate Digital Filtering with Half-Band Filters," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-25, No. 4, August 1977.

- [3] Crochiere, R., and Rabiner, L. "Interpolation and Decimation of Digital Signals—A Tutorial Review," *Proceedings of the IEEE*, Vol. 69, No. 3, March 1981.
- [4] Neugebauer, O. *Astronomical Cuneiform Texts: Babylonian Ephemerides of the Seleucid Period for the Motion of the Sun, the Moon and the Planets*, Lund Humphries, London, 1955.
- [5] Schafer, R., and Rabiner, L. "A Digital Signal Processing Approach to Interpolation," *Proceedings of the IEEE*, Vol. 61, No. 6, June 1973.
- [6] Fliege, N. *Multirate Digital Signal Processing: Multirate Systems, Filter Banks, Wavelets*, John Wiley and Sons, New York, 1995.
- [7] Crochiere, R., and Rabiner, L. *Multirate Digital Signal Processing*, Prentice Hall, Upper Saddle River, New Jersey, 1983.
- [8] Vaidyanathan, P. *Multirate Systems and Filter Banks*, Prentice Hall, Upper Saddle River, New Jersey, 1992.
- [9] Harris, F. *Multirate Signal Processing for Communication Systems*, Prentice Hall, Upper Saddle River, New Jersey, 2004, [Chapter 11](#).
- [10] Hogenauer, E. "An Economical Class of Digital Filters for Decimation and Interpolation," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-29, April 1981, pp. 155–162.
- [11] Brandt, F. "Oversampled Analog-to-Digital Conversion, *Stanford Electronics Laboratories*, Technical Report No. ICL91-009, April 1991, p. 108.
- [12] Proakis, J., and Manolakis, D. *Digital Signal Processing: Principles, Algorithms and Applications*, Prentice Hall, Upper Saddle River, New Jersey, 1996.
- [13] Oppenheim, A., and Schafer, R. *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1st ed. 1989, 2nd ed. 1999.
- [14] Rorabaugh, C. *DSP Primer*, McGraw-Hill, New York, 1999.
- [15] Crochiere, R., and Rabiner, L. "Further Considerations in the Design of Decimators and Interpolators," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-24, No. 4, August 1976.
- [16] Ballanger, M., et al. "Interpolation, Extrapolation, and Reduction of Computational Speed in Digital Filters," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-22, No. 4, August 1974.
- [17] Hou, H., and Andrews, H. "Cubic Splines for Image Interpolation and Digital Filtering," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-26, No. 6, August 1978.
- [18] Keys, R. "Cubic Convolution Interpolation for Digital Image Processing," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-29, No. 6, August 1981.
- [19] Parker, J., et al. "Comparison of Interpolating Methods for Image Resampling," *IEEE Trans. on Medical Imaging*, Vol. MI-2, No. 1, August 1983.
- [20] Blue, K., et al. "Vector Signal Analyzers for Difficult Measurements on Time-Varying and Complex Modulated Signals," *Hewlett-Packard Journal*, December 1993.
- [21] Bartz, M., et al. "Baseband Vector Signal Analyzer Hardware Design," *Hewlett-Packard Journal*, December 1993.

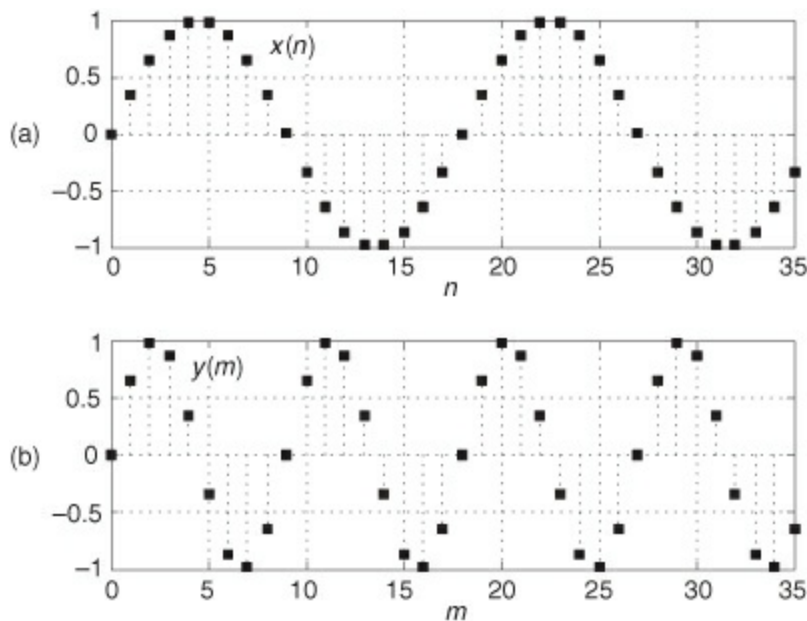
Chapter 10 Problems

10.1 Assume we want to decimate an $x(n)$ time-domain sequence by four.

- (a) Should the $x(n)$ sequence be lowpass filtered before or after we discard every fourth sample?
- (b) Draw the frequency magnitude response of an ideal lowpass filter used in this decimation-by-four process. Label the frequency axis of your drawing in both Hz (in terms of the filter’s input data sampling rate f_s Hz) and our “discrete-system” frequency notation of radians/sample.
- (c) What should be the lowpass filter’s zero-Hz (DC) magnitude so that there is no time-domain amplitude gain or loss in our decimation process?

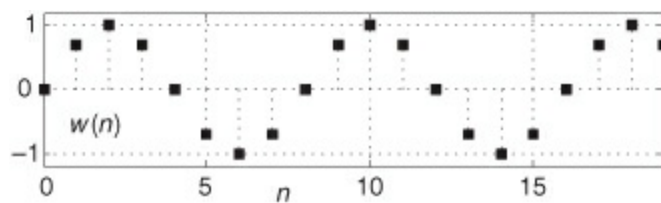
10.2 Assume we have a 72-sample sinusoidal $x(n)$ time-domain sequence, the first 36 samples of which are shown in [Figure P10-2\(a\)](#). Next we decimate $x(n)$ by two to generate 36 samples of the $y(m)$ sequence shown in [Figure P10-2\(b\)](#). Sequence $y(m)$ is also sinusoidal, as we should expect, but its frequency appears to be double the frequency of $x(n)$. Explain that apparent frequency difference.

Figure P10-2



10.3 Assume we collected 2048 samples of a sinewave whose frequency is 128 Hz using an f_s sample rate of 1024 Hz, and we call those samples $w(n)$. The first 20 samples of $w(n)$ are shown in [Figure P10-3](#). Next we perform a 2048-point FFT on $w(n)$ to produce a $W(m)$ sequence.

Figure P10-3



- (a) What is the m frequency index value, m_{\max} , of the FFT sample having the largest magnitude over the positive-frequency range of $|W(m)|$? Show how you arrived at your answer.

(b) Next, suppose we decimate $w(n)$ by a factor of two to generate the 1024-point sequence $x(n)$ defined by

$$x(n) = w(2n).$$

If we perform a 1024-point FFT of $x(n)$, what is the m frequency index value, $m_{\max, \text{dec}=2}$, of the FFT sample having the largest magnitude over the positive-frequency range of $|X(m)|$? Show how you arrived at your answer.

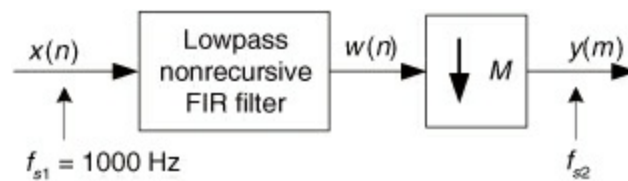
(c) Finally, assume we decimate $x(n)$ by a factor of two to generate the 512-point sequence $y(n)$ defined by

$$y(n) = x(2n).$$

If we perform a 512-point FFT of $y(n)$, what is the m frequency index value, $m_{\max, \text{dec}=2}$, of the FFT sample having the largest magnitude over the positive-frequency range of $|Y(m)|$? Show how you arrived at your answer.

10.4 In this chapter we've portrayed decimation by an integer factor M with the block diagram shown in [Figure P10-4](#), that is, a lowpass decimation filter followed by a downsampler (the “ $\downarrow M$ ” symbol) that discards all but every M th filter output sample. In this problem we explore the changes in signal time-domain amplitude and frequency-domain magnitude caused by decimation.

Figure P10-4



For this problem, our assumptions are:

- The lowpass filter in [Figure P10-4](#) has a passband gain of unity and passband width of 0 to 250 Hz.
- The $x(n)$ sequence contains a 100 Hz sinusoidal component whose time-domain peak amplitude is P .
- In the frequency domain, the 100 Hz $x(n)$ sinusoid is located exactly on a $4N$ -point discrete Fourier transform (DFT) bin center and its $4N$ -point DFT spectral magnitude is K .
- Finally, we apply exactly $4N$ samples of $w(n)$ to the $M = 4$ downsampler.

(a) What is the f_{s2} sample rate (in Hz) of the $y(m)$ time-domain sequence?

(b) What is the peak time-domain amplitude of the 100 Hz sinusoid in the $w(n)$ sequence?

(c) What is the peak time-domain amplitude of the 100 Hz sinusoid in the $y(m)$ sequence? Justify your answer.

(d) What is the magnitude of the 100 Hz spectral component in an N -point DFT of $y(m)$? Justify your answer.

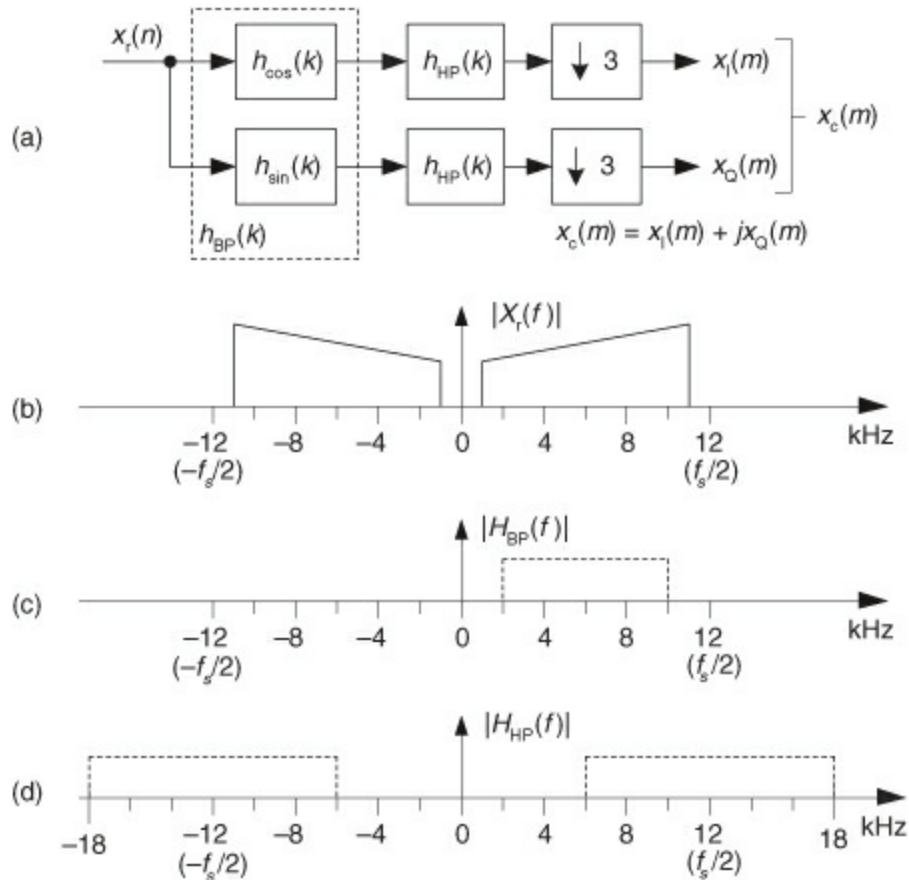
(e) What is the equation that defines [Figure P10-4](#)'s downsampled $y(m)$ sequence in terms of the $w(n)$ sequence?

Hint: Your solution to this part of the problem will take the form

$$y(m) = w(?).$$

10.5 Given the $x_r(n)$ input signal in [Figure P10-5\(a\)](#), whose $|X_r(f)|$ magnitude spectrum is shown in [Figure P10-5\(b\)](#), draw a rough sketch of the $|X_c(f)|$ spectrum of the system's complex $x_c(m) = x_I(m) + jx_Q(m)$ output sequence. The frequency magnitude responses of the complex bandpass $h_{BP}(k)$ filter and the real-valued highpass $h_{HP}(k)$ filters are provided in [Figures P10-5\(c\)](#) and [P10-5\(d\)](#).

Figure P10-5

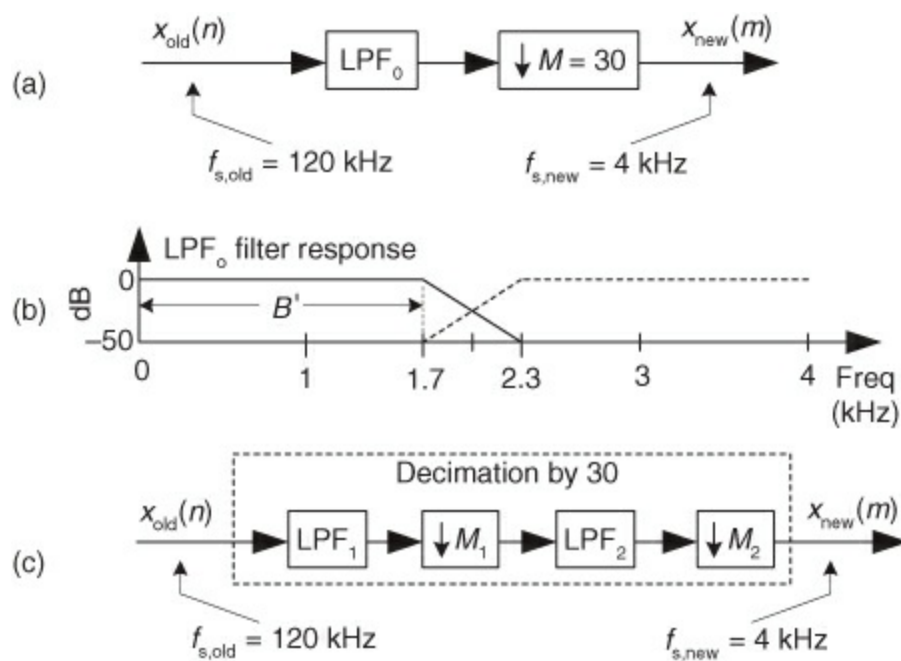


10.6 Assume we want to design the decimation by $M = 30$ system shown in [Figure P10-6\(a\)](#). The desired LPF_0 lowpass filter's frequency magnitude response is the solid lines shown in [Figure P10-6\(b\)](#). The filter's stopband attenuation is 50 dB. (The dashed lines are the spectral replication of the lowpass filter's frequency response.) The one-sided passband width of the lowpass filter is $B' = 1.7$ kHz.

(a) Using the text's [Eq. \(10-3\)](#), estimate the number of taps in the LPF_0 lowpass filter.

(b) Assuming we decide to implement our decimation by $M = 30$ system using two-stage decimation as shown in [Figure P10-6\(c\)](#), what are the optimum M_1 and M_2 decimation factors?

Figure P10-6

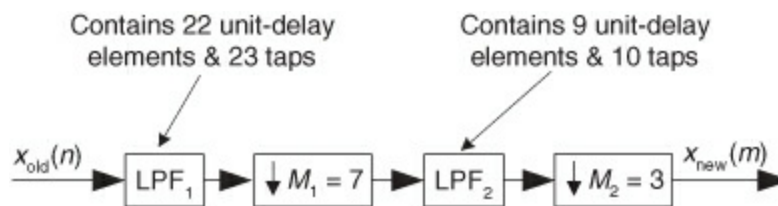


(c) Using the text's [Eq. \(10-3\)](#), estimate the number of taps in the LPF₁ and LPF₂ lowpass filters in [Figure P10-6\(c\)](#).

(d) What is the reduction in number of filter taps using the system in [Figure P10-6\(c\)](#) compared to the number of filter taps needed by the system in [Figure P10-6\(a\)](#)?

10.7 Here is an interesting problem. In [Chapter 5](#) we discussed the *transient response* of tapped-delay line FIR filters and stated that an FIR filter's output samples are not valid until the filter's delay line is filled with input data samples. Assuming that the 23rd output sample of LPF₁ is the first sample applied to LPF₂, how many $x_{\text{old}}(n)$ input samples must be applied to the two-stage decimation filter shown in [Figure P10-7](#) to fill the LPF₁ and LPF₂ lowpass filters with input data?

Figure P10-7



10.8 Assume we want to interpolate an $x(n)$ time-domain sequence by three.

(a) Should we perform upsampling (insertion of zero-valued samples) on the $x(n)$ sequence before or after implementing lowpass filtering?

(b) Draw the frequency magnitude response of an ideal lowpass filter used in this interpolation-by-three process. Label the frequency axis of your drawing in both Hz (in terms of the filter's input data sampling rate f_s Hz) and our "discrete-system" frequency notation of radians/sample.

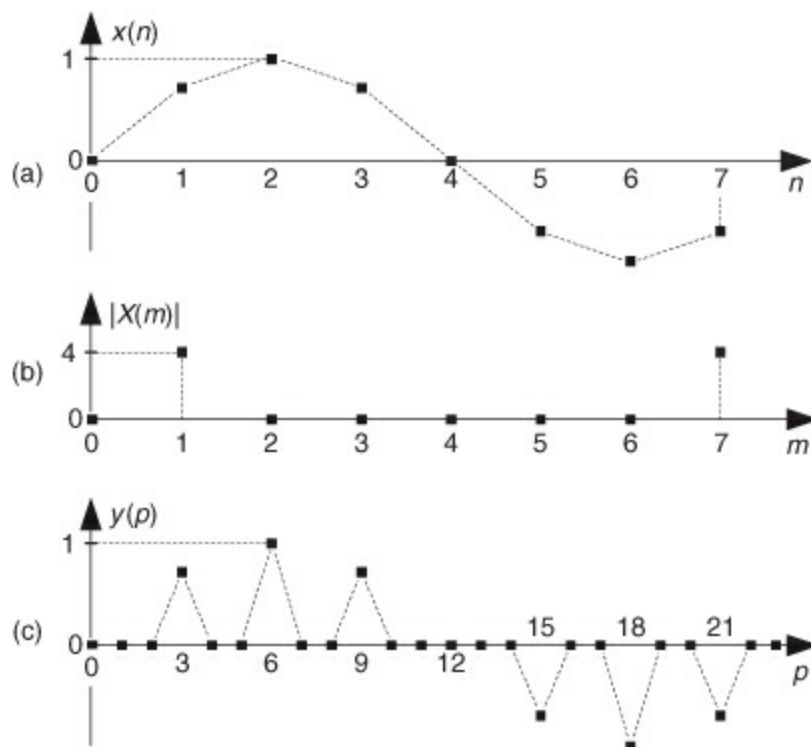
(c) What should be the lowpass filter's zero-Hz (DC) magnitude so that there is no time-domain amplitude gain or loss in our interpolation process?

10.9 Let's make sure we fully understand the spectral effects of interpolation by considering the 8-sample, single-cycle, $x(n)$ sinewave sequence in [Figure P10-9\(a\)](#). That sequence's $X(m)$ DFT spectral magnitude samples are shown in [Figure P10-9\(b\)](#). If we upsample $x(n)$ by a factor of three, by inserting two zero-valued samples between each $x(n)$ sample, we produce the 24-sample

$y(p)$ time sequence shown in [Figure P10-9\(c\)](#).

- (a) What is the time-domain equation that defines the upsampled $y(p)$ sequence in terms of the $x(n)$ sequence?

Figure P10-9



Hint: Your solution to this part of the problem will have two parts and look like

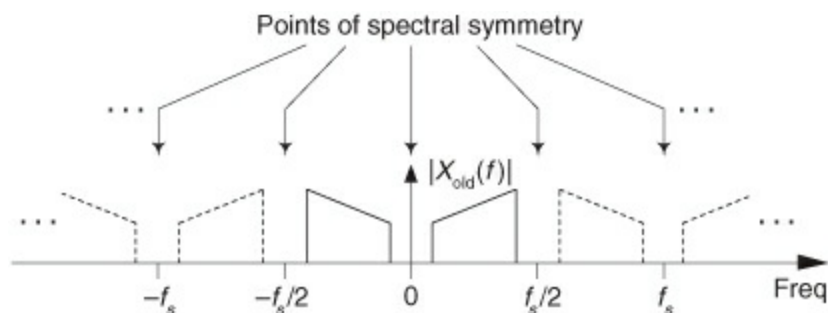
$$y(p) = \begin{cases} x(?), & \text{for } n = ? \\ ?, & \text{otherwise.} \end{cases}$$

- (b) Draw the spectral magnitude samples of the 24-point $Y(m)$ DFT of $y(p)$.

10.10 Assume we have a time-domain sequence of real-valued samples, $x_{\text{old}}(n)$, whose spectral magnitude is shown in [Figure P10-10](#). (We represent spectral replications by the dashed lines.) There we see that the frequency points of spectral symmetry of $|X_{\text{old}}(f)|$, represented by the bold down arrows, can be described by

$$y(p) = \begin{cases} x(?), & \text{for } n = ? \\ ?, & \text{otherwise} \end{cases}$$

Figure P10-10

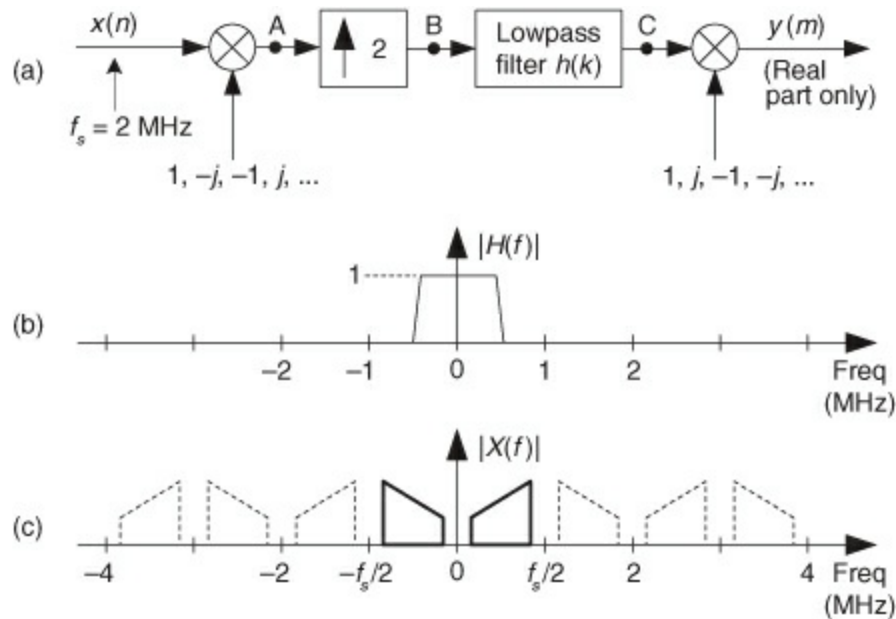


where k is an integer. If we upsample $x_{\text{old}}(n)$ by two, by inserting a zero-valued sample between each $x_{\text{old}}(n)$ sample, to generate a new time sequence $x_{\text{new}}(m)$, what is the expression for the

frequency points of spectral symmetry of $|X_{\text{new}}(f)|$?

10.11 Texas Instruments Inc. produces a digital filter chip, Part #GC2011A, used in cell phones for frequency up-conversion. The process, described in their AN9804 application note document, is depicted in [Figure P10-11\(a\)](#). The lowpass filter's 1 MHz-wide passband covers the frequency range shown in [Figure P10-11\(b\)](#). (The lowpass filter block comprises two separate real-valued 1 MHz-wide filters, filtering the real and imaginary parts of the complex signal at node B.) If the spectral magnitude of the $x(n)$ input is that shown by the solid curves in [Figure P10-11\(c\)](#), where we represent spectral replications by the dashed curves, draw the spectral magnitudes of the complex sequences at nodes A, B, C, and the real part of the $y(m)$ output sequence.

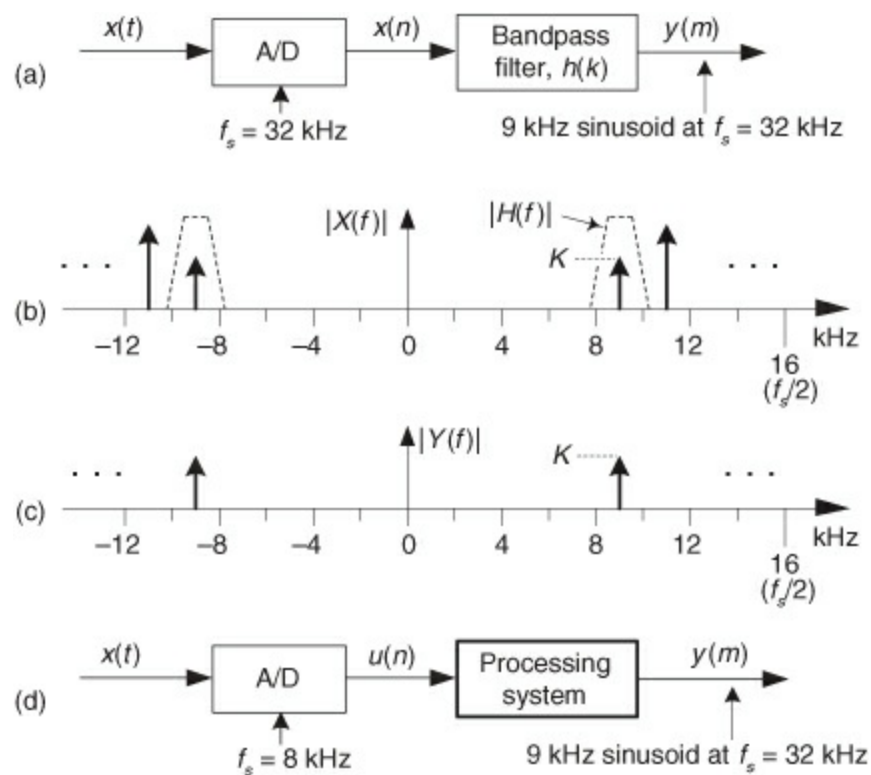
Figure P10-11



Hint: In [Chapter 8](#) we learned that multiplying a time sequence by $e^{-j2\pi n/4} = 1, -j, -1, j, \dots$, translates the signal's spectrum down in frequency.

10.12 Here is a fun interpolation problem. [Figure P10-12\(a\)](#) shows a simple digital filtering system. Assume that the analog $x(t)$ signal applied to the analog-digital (A/D) converter contains a 9 kHz sinusoid and an 11 kHz sinusoid. The spectral magnitude of the sampled $x(n)$ sequence is given in [Figure P10-12\(b\)](#). The system's function is to filter out the 11 kHz tone and provide a $y(m)$ output sequence that is a 9 kHz sinusoid at a sample rate of $f_s = 32$ kHz. The dashed curve in [Figure P10-12\(b\)](#) indicates the unity-gain bandpass filter's frequency magnitude response, while the spectrum of our desired filter output, whose magnitude is K , is given in [Figure P10-12\(c\)](#).

Figure P10-12



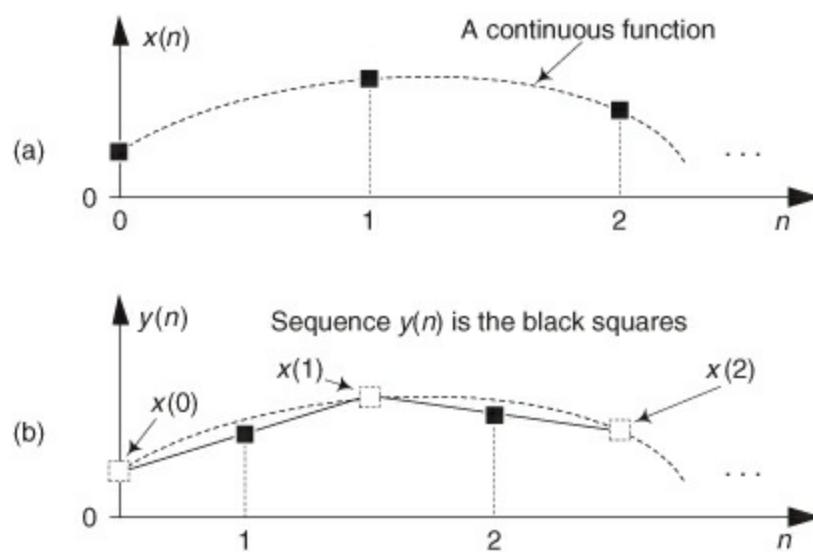
Now, assume that the system is constrained to use an A/D converter whose clock rate is 8 kHz (instead of 32 kHz), as shown in [Figure P10-12\(d\)](#).

- (a) Draw the block diagram of the *processing system* that provides the desired $y(m)$ output sequence at a sample rate of $f_s = 32$ kHz which is four times the $u(n)$ sample rate.
- (b) Draw spectral diagrams that justify your solution.

10.13 In this chapter we discussed various forms of interpolation. There is a well-known interpolation process called *linear interpolation*. It's an interpolation-by-two method for estimating sample values of a continuous function between some given $x(n)$ sample values of that function. For the $x(n)$ time samples in [Figure P10-13\(a\)](#), linear interpolation is the process of computing the intermediate $y(n)$ samples shown as the black squares in [Figure P10-13\(b\)](#). That is, the interpolated sample $y(1)$ is the value lying at the center of the straight line connecting $x(0)$ and $x(1)$, the interpolated sample $y(2)$ is the value lying at the center of the straight line connecting $x(1)$ and $x(2)$, and so on. Given this process of linear interpolation:

- (a) What is the z -domain expression for the $H(z) = Y(z)/X(z)$ transfer function of the linear interpolation process?
- (b) Draw a rough sketch of the frequency magnitude response of a linear interpolation filter over the frequency range of $\omega = \pm\pi$ radians/sample ($\pm f_s/2$ Hz).
- (c) Comment on the advantage of, and the disadvantage of, using linear interpolation to perform interpolation by a factor of two.

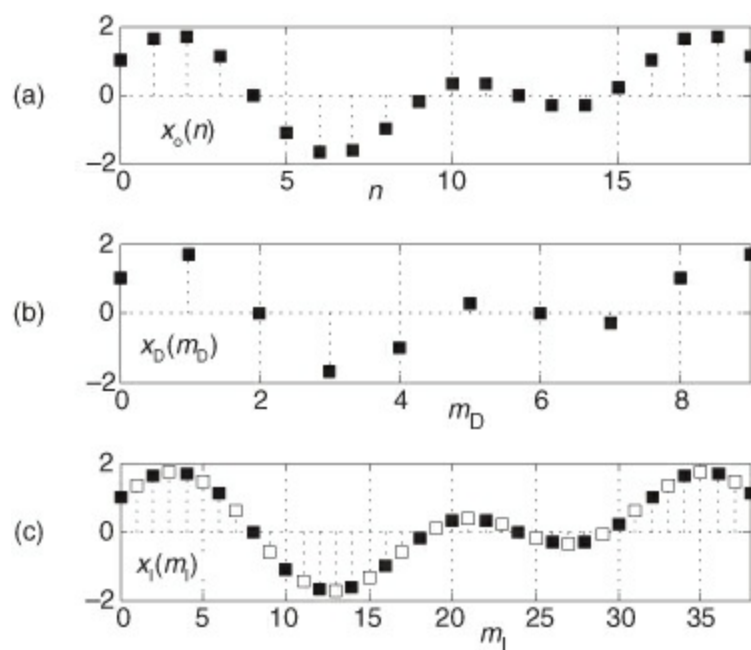
Figure P10-13



10.14 Assume we must convert a compact disc (CD) audio signal, whose sample rate is $f_{s,CD} = 44.1$ kHz, to a digital audio tape (DAT) signal whose sample rate is $f_{s,DAT} = 48$ kHz. If we interpolate that CD signal by a factor of $L = 160$, by what factor M must we decimate the interpolated signal to obtain a final sample rate of 48 kHz?

10.15 Consider the $x_o(n)$ time sequence in [Figure P10-15\(a\)](#), whose sample rate is $f_s = 1$ kHz. If we decimate $x_o(n)$ by two, we obtain the $x_D(m_D)$ sequence shown in [Figure P10-15\(b\)](#), where the odd- n samples of $x_o(n)$ have been discarded. Next, if we interpolate $x_o(n)$ by two, we obtain the $x_I(m_I)$ sequence shown in [Figure P10-15\(c\)](#), where the interpolated samples are shown as white dots. Comment on how decimation and interpolation affect the time duration of the decimated and interpolated sequences relative to the time duration of the original $x_o(n)$ sequence.

Figure P10-15



10.16 Fill in the following table. When complete and correct, the table shows the time-domain and frequency-domain gain of the two processes: decimation by M , and interpolation by L .

Here, decimation means lowpass filtering (by a unity-gain filter) NM time samples followed by the discarding of every M th filter output sample to obtain N time samples. By “interpolation” we mean upsampling by inserting $L-1$ zero-valued samples between adjacent samples of an N -length time-

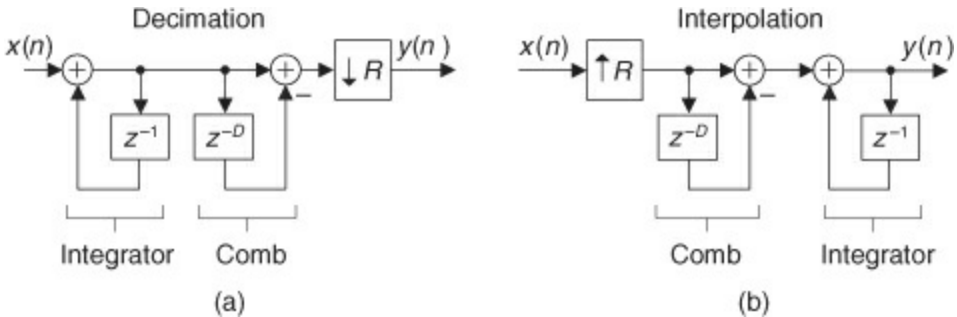
domain sequence followed by lowpass filtering using a unity-gain lowpass filter to obtain NL time samples. Assume the sample rate change factors M and L are integers.

Sample Rate Conversion Gain

	Time domain	Frequency domain
Decimation by M	Gain =	Gain =
Interpolation by L	Gain =	Gain =

10.17 Here is an interesting, and educational, problem because it shows the spectral effects of upsampling a downsampled sequence. Think about the sample rate change process in [Figure P10-17\(a\)](#). The upsampling operation “ $\uparrow 4$ ” means insert three zero-valued samples between each $q(m)$ sample. Assume the spectral magnitude of the $x(n)$ sequence is the $|X(f)|$ shown in [Figure P10-17\(b\)](#).

Figure P10-17



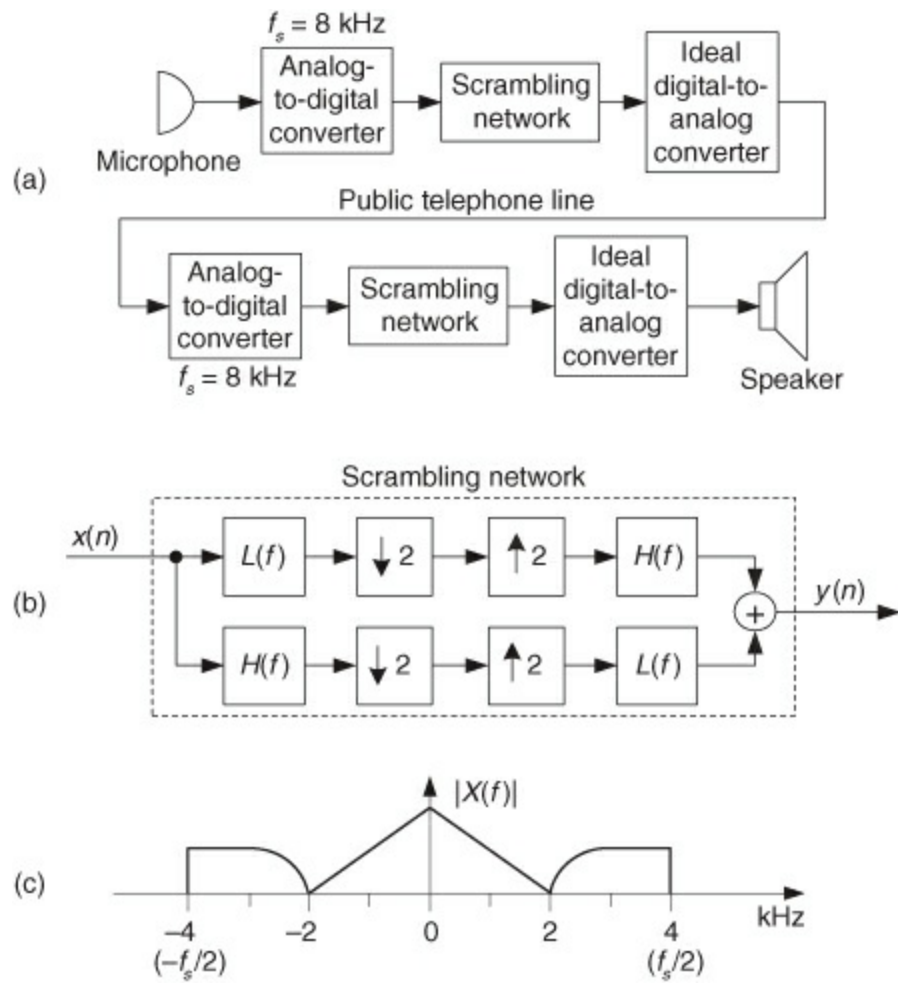
- (a) Draw the $|Q(f)|$ spectrum of sequence $q(m)$ including the peak spectral magnitude levels in terms of K . Show spectral replications (located at multiples of the $q(m)$ sample rate) as dashed curves as was done in [Figure P10-17\(b\)](#).

(b) Draw the $|W(f)|$ spectrum of sequence $w(p)$ including the peak spectral magnitude levels in terms of K . Show spectral replications as dashed curves.

(c) Draw the frequency magnitude response of the lowpass filter, including its passband gain value, that would produce a $y(p)$ output sequence whose $Y(f)$ spectral magnitude is equal to $|X(f)|$.

(d) When first learning the principles of sample rate change (multirate systems), it is easy to believe that following a “ $\downarrow 4$ ” decimation process with an “ $\uparrow 4$ ” upsampling process would mean the two processes cancel each other such that the overall cascaded effect would be *no change*. Is this correct?
- 10.18** One way to implement a secure telephone communications channel is shown in [Figure P10-18\(a\)](#). Anyone monitoring the telephone line will not be able to understand the audio speech signal on that line. The scrambling network is shown in [Figure P10-18\(b\)](#), where the two identical $L(f)$ digital lowpass filters have passbands that extend from -2 kHz to $+2$ kHz. The two identical $H(f)$ digital highpass filters have passbands that extend from -6 kHz to -2 kHz, and 2 kHz to 6 kHz.
- (a) If the $x(n)$ input to the first scrambling network has the spectrum shown in [Figure P10-18\(c\)](#), draw the spectrum, over the frequency range of $\pm f_s$, of the output sequence from the first scrambling network in [Figure P10-18\(a\)](#).

Figure P10-18



(b) Draw the spectrum, over the frequency range of $\pm f_s$, of the output sequence from the second scrambling network in [Figure P10-18\(a\)](#).

10.19 In [Section 10.7](#) we depicted a polyphase filter, used in an interpolation-by-four process, with the structure shown in [Figure P10-19-I](#). The $H_k(z)$ blocks represent tapped-delay line FIR polyphase subfilters containing unit-delay elements, multipliers, and adders.

- (a) Why are the polyphase subfilters useful when used in an interpolation process?
- (b) Determine how to replace the commutating (rotating) switch in [Figure P10-19-I](#) using only the delay and upsampler elements shown in [Figure P10-19-II\(a\)](#). That is, determine what's inside the mysterious block in [Figure P10-19-II\(b\)](#) to make that figure equivalent to [Figure P10-19-I](#).

Figure P10-19-I

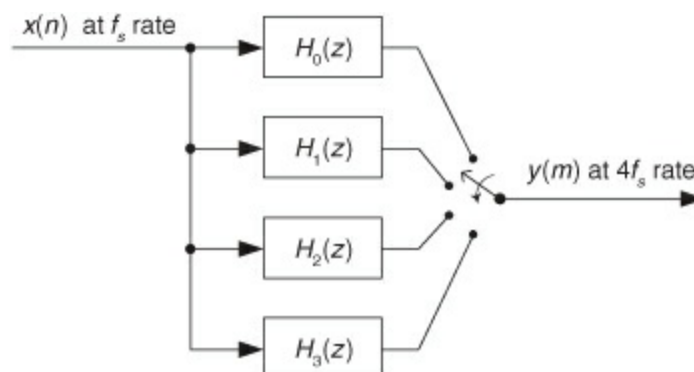
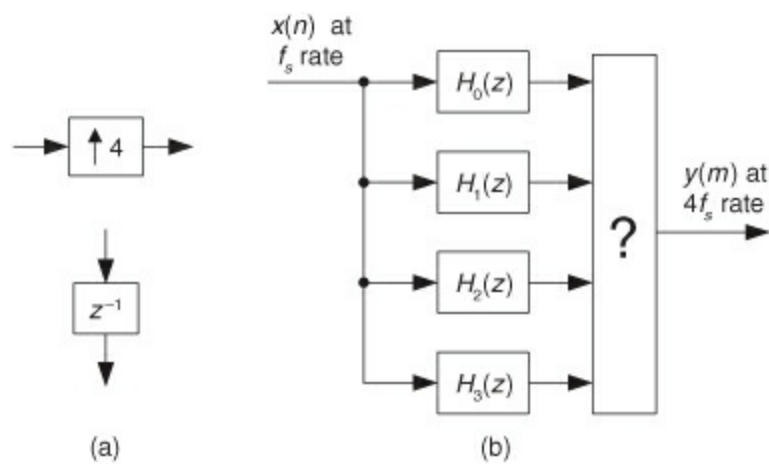


Figure P10-19-II

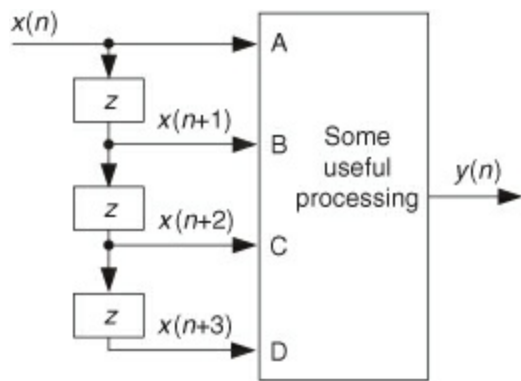


The correct solution to this problem will show a polyphase structure with which you should become familiar. That structure is often used in the DSP literature of multirate systems to depict polyphase interpolation filters.

Hint: Given some $x(n)$ sequence, write the sample sequences on the four output lines of the $H_k(z)$ subfilters, and $y(n)$ in [Figure P10-19-I](#). Then determine how to obtain that same $y(m)$ output sequence in [Figure P10-19-II\(b\)](#). The coefficients of polynomial $H_k(z)$ are not important to this problem. Assume the subfilters have no delay elements, a single multiplier, and a coefficient of one, if you wish.

10.20 Occasionally in the literature of DSP you'll encounter documentation that uses a drawing like that in [Figure P10-20](#) to illustrate some concept, or principle, regarding multirate systems. Notice that the cascaded elements are *not* our standard " z^{-1} " delay-by-one-sample elements but, instead, are *advance-by-one-sample* elements indicated by a " z " (z^{+1}).

Figure P10-20

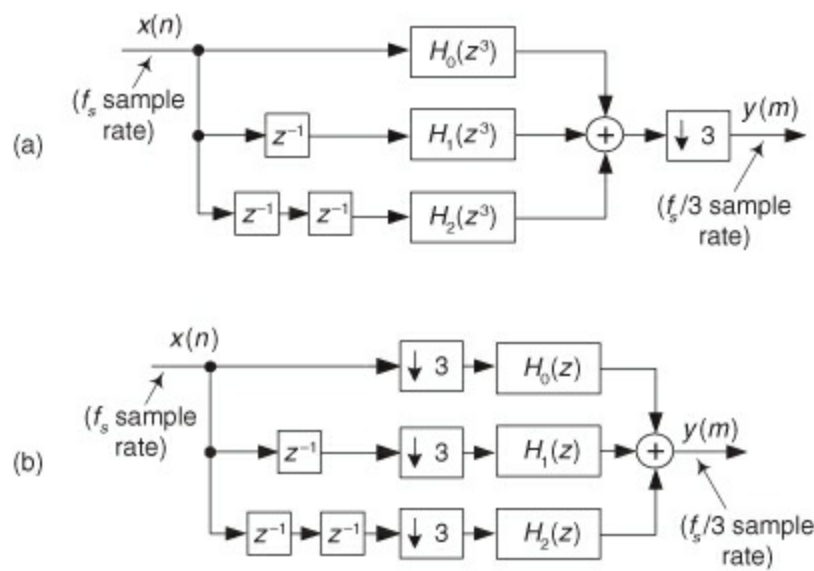


Show how you would implement the system in [Figure P10-20](#), in our universe where we cannot look forward in time, to provide the appropriate four time-domain sequences to the "Some useful processing" subsystem's input ports?

10.21 In the text we discussed decimation by $M = 3$ and showed two equivalent realizations of such a decimation process as those in [Figures P10-21\(a\)](#) and [P10-21\(b\)](#). Assume that all six subfilters in [Figure P10-21](#) are tapped-delay lines containing four multipliers, and that $f_s = 30$ samples/second.

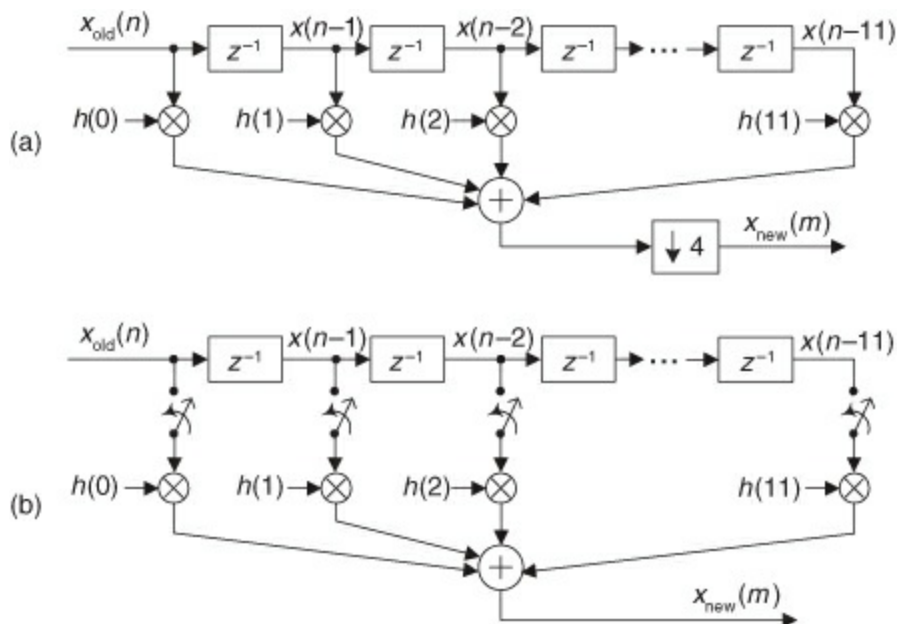
- (a) How many multiplications per second are performed in [Figure P10-21\(a\)](#)?
- (b) How many multiplications per second are performed in [Figure P10-21\(b\)](#)?

Figure P10-21



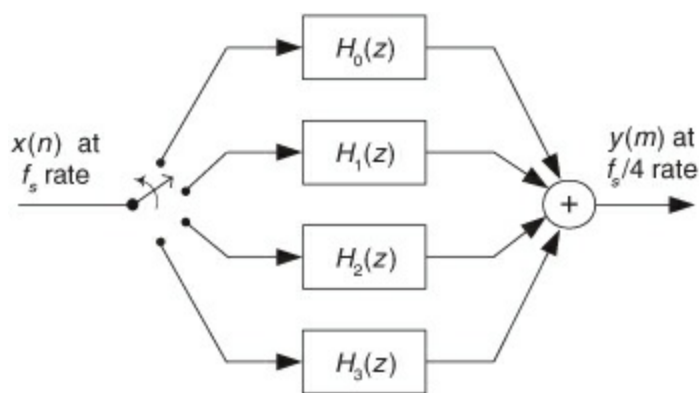
10.22 The decimation-by-four (lowpass filtering followed by downsampling) process shown in [Figure P10-22\(a\)](#) is inefficient because three out of every four computational results are discarded. A more efficient decimation process is shown in [Figure P10-22\(b\)](#), where the switches driving the multipliers close once, for one sample time period only, upon the arrival of every fourth $x_{\text{old}}(n)$ sample. This way, no unnecessary computations are performed. Likewise, in polyphase decimation filtering no unnecessary computations are performed. In real-time hardware implementations, explain the fundamental difference between the computations performed, from a time-domain standpoint, in the [Figure P10-22\(b\)](#) decimation filter and a polyphase decimation-by-four filter having 12 multipliers?

Figure P10-22



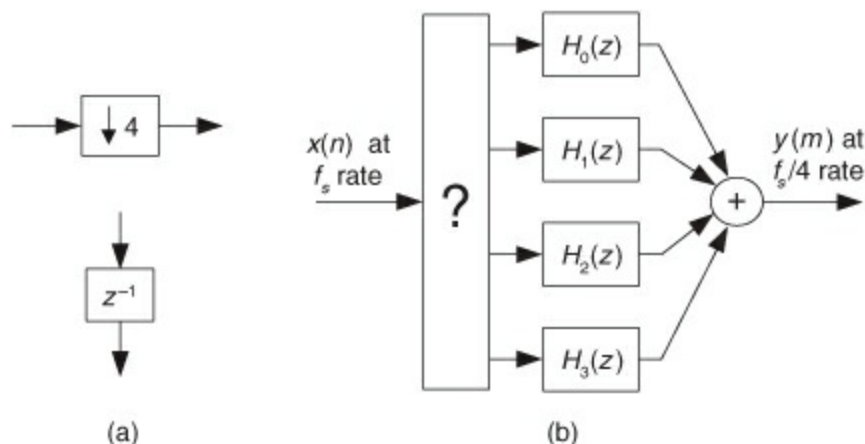
10.23 In [Section 10.7](#) we depicted a polyphase filter, used in a decimation-by-four process, with the structure shown in [Figure P10-23-I](#). The $H_k(z)$ blocks represent tapped-delay line FIR polyphase subfilters containing unit-delay elements, multipliers, and adders.

Figure P10-23-I



- (a) Why are polyphase subfilters useful when used in a decimation process?
- (b) Determine how to replace the commutating (rotating) input switch in [Figure P10-23-I](#) using only the delay and downsampler elements shown in [Figure P10-23-II\(a\)](#). That is, determine what interconnection of delay and downsampler elements must be inside the mysterious block in [Figure P10-23-II\(b\)](#) to make that figure equivalent to [Figure P10-23-I](#).

Figure P10-23-II



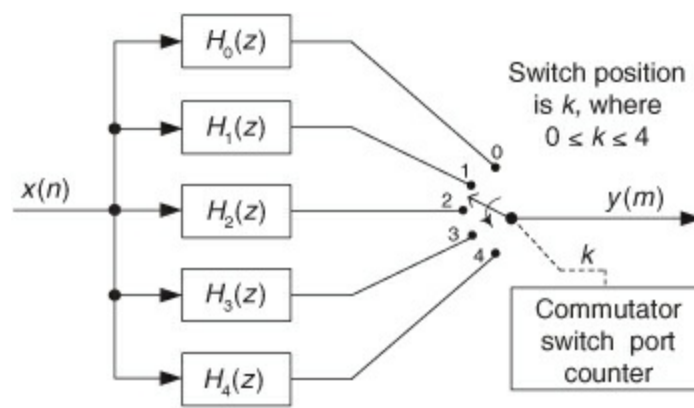
The correct solution to this problem will show a polyphase structure with which you should become familiar. That structure is often used in the DSP literature of multirate systems to depict polyphase decimation filters.

Hint: Given some $x(n)$ sequence, write the $x(n)$, $x(n-1)$, $x(n-2)$, etc., sample sequences on the four lines driving the $H_k(z)$ subfilters in [Figure P10-23-I](#). Then determine how to obtain those same sample sequences for routing to the subfilters in [Figure P10-23-II\(b\)](#).

10.24 This problem is related to the material in the text's [Section 10.10](#). Assume we are resampling a time sequence by the rational factor $5/4$ using a five-position commutating filter output switch as shown in [Figure P10-24](#).

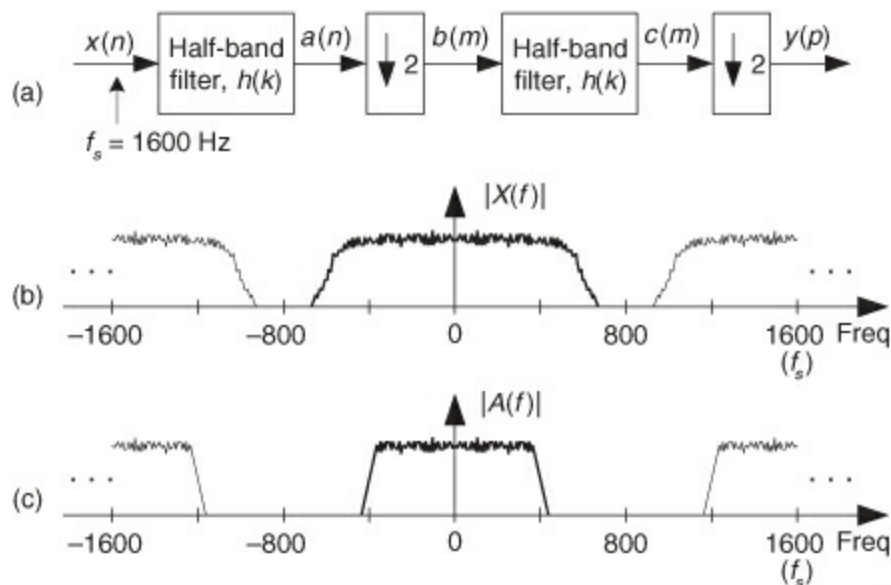
- (a) Determine the commutating switch's port position value (index) k , and the index n of the most recent input $x(n)$ sample applied to the subfilters, used to compute the resampler's $y(m)$ sample when output index $m = 7$. Show your work.
- (b) For the resampler in [Figure P10-24](#) to have a DC (zero Hz) gain of unity, what must be the DC gain of the original prototype lowpass FIR filter from which the five $H_k(z)$ subfilters were obtained?

Figure P10-24



10.25 Think about the multirate decimation system, employing lowpass half-band filters, in [Figure P10-25\(a\)](#). If the spectrum of the wideband $x(n)$ noise sequence is that shown in [Figure P10-25\(b\)](#), the spectrum of the $a(n)$ noise sequence is as shown in [Figure P10-25\(c\)](#). Draw the spectra, with appropriate frequency-axis labeling in Hz, of the $b(n)$, $c(m)$, and $y(p)$ sequences.

Figure P10-25

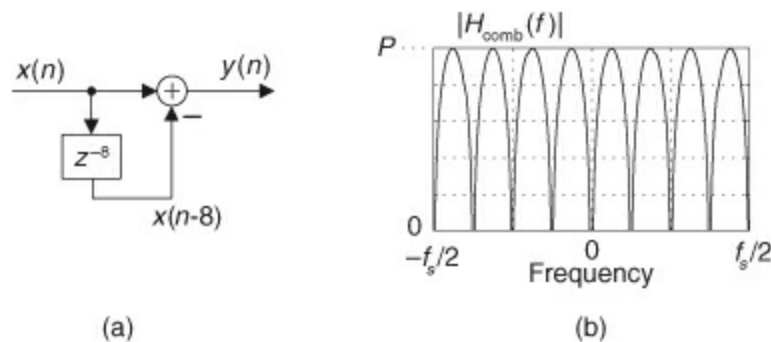


10.26 The z -domain transfer function of a CIC filter's comb subfilter having a delay line length of $N = 8$, shown in [Figure P10-26\(a\)](#), is

$$H_{\text{comb}}(z) = 1 - z^{-8},$$

and its frequency magnitude response is shown on a linear scale in [Figure P10-26\(b\)](#).

Figure P10-26



(a) Each of those multiple frequency magnitude passband curves in [Figure P10-26\(b\)](#) looks parabolic in shape. In terms of the frequency variable f , a single ideal downward-opening

parabola is described by the expression

$$|H_{\text{comb}}(f)| = -Kf^2$$

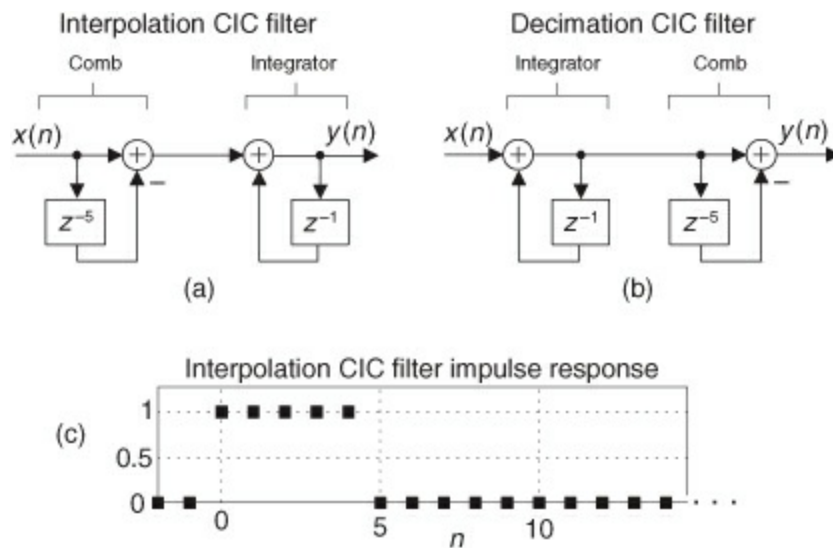
where K is some constant. Are the shapes of those passband curves in [Figure P10-26\(b\)](#) indeed a function of f^2 , making them parabolic? Show your work.

(b) What is the peak value, P , of the $|H(f)|$ frequency magnitude curve in P10-26(b)? Show your work. (The P value is important. It tells us what is the maximum gain of a comb subfilter.)

Hint: Deriving an equation for the $|H_{\text{comb}}(f)|$ frequency magnitude response will provide the solutions to Part (a) and Part (b) of this problem.

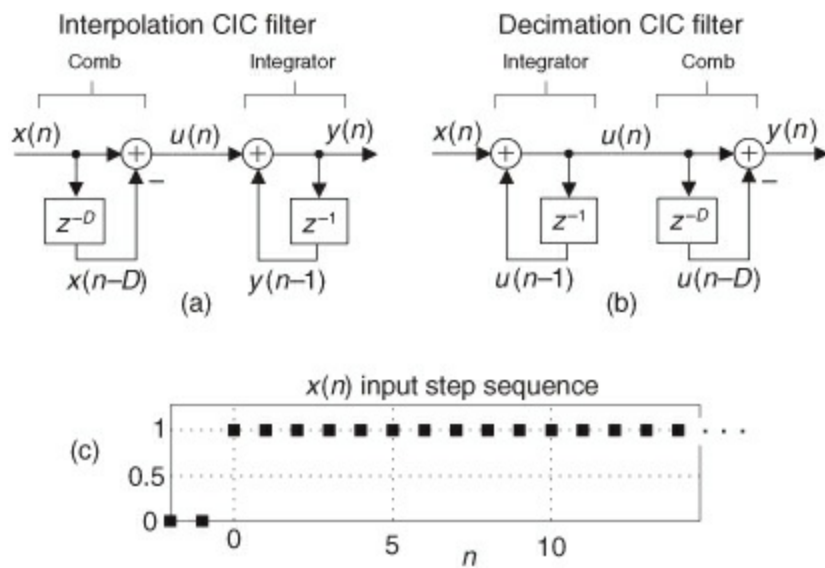
10.27 In the text we stated that the interpolation CIC filter in [Figure P10-27\(a\)](#) has an impulse response, when its differential delay $D = 5$, equal to that shown in [Figure P10-27\(c\)](#). We also stated that swapping [Figure P10-27\(a\)](#)'s comb and integrator resulted in a decimation CIC filter as shown in [Figure P10-27\(b\)](#). Prove that the decimation CIC filter in [Figure P10-27\(b\)](#) also has an impulse response equal to that shown in [Figure P10-27\(c\)](#).

Figure P10-27



10.28 Here is an important problem with regard to implementing two theoretically equivalent digital filters. We illustrate our point using the CIC filters shown in [Figures P10-28\(a\)](#) and [P10-28\(b\)](#). Because they are linear, we can swap the comb and integrator stages of the CIC filter used for interpolation to obtain a CIC filter used for decimation. The two CIC filters have identical time-domain impulse responses.

Figure P10-28



(a) However, to understand an important fundamental difference in the hardware implementation of the two filters, draw the $u(n)$ and $y(n)$ sequences for both filters when the $x(n)$ input to the filters is the *step* sequence shown in [Figure P10-28\(c\)](#). Assume a comb delay of $D = 4$ for both CIC filters. Your solution should comprise four separate drawings. (Also, assume that the $u(n)$ and $y(n)$ values are zero, for both CIC filters, at time index $n < 0$.)

(b) To appreciate the implementation difference between interpolation and decimation CIC filters, we need to determine the growth of the binary word width of the memory location, or hardware register, containing the $u(n)$ samples. To do so, fill in the following table, indicating how many binary bits are needed to accommodate the $u(n)$ and $y(n)$ samples for each CIC filter up to time index $n = 500$.

Hint: The number of binary bits needed to store $u(n)$ is the next integer greater than $\log_2[u(n)]$.

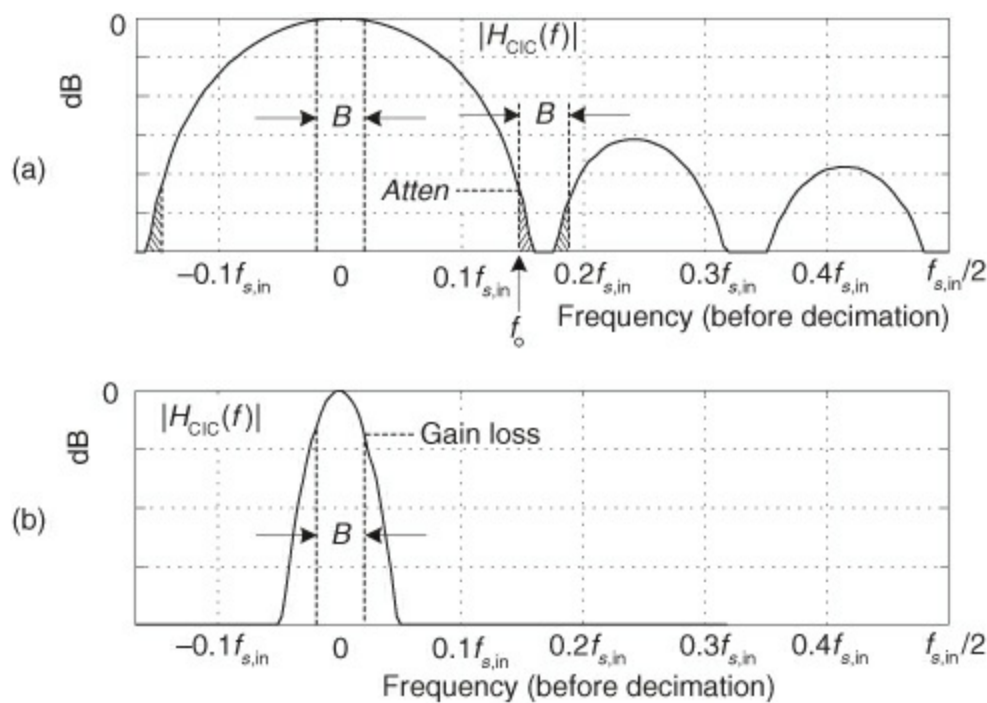
Memory, or Hardware Register, Bit-Width Requirements

Sequence	Interpolation CIC filter	Decimation CIC filter
$u(n)$		
$y(n)$		

(c) This question has great practical importance. What does your solution to Part (b) tell us about the binary-word-width requirements of the memory locations, or hardware registers, containing the integrators' $u(n)$ samples in CIC decimation and CIC interpolation filters?

10.29 Here is a typical problem faced by engineers who use CIC filters. As of this writing, Intersil Corp. makes a decimating digital filter chip (Part #HSP43220) that contains a 5th-order CIC filter. When used for decimation by a factor of $R = 6$, and the internal comb filters have a differential delay of $D = 6$, the CIC filter's frequency magnitude response is shown in [Figure P10-29\(a\)](#).

Figure P10-29

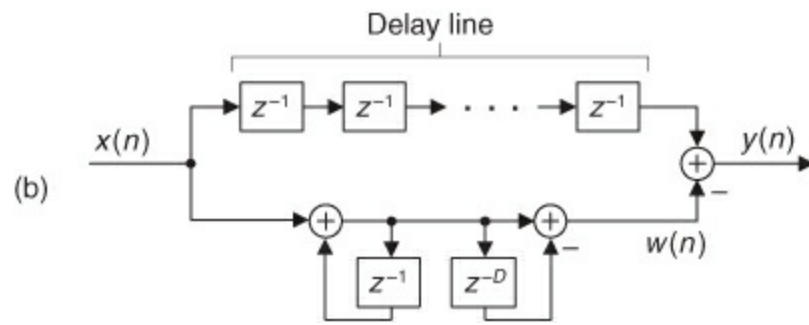
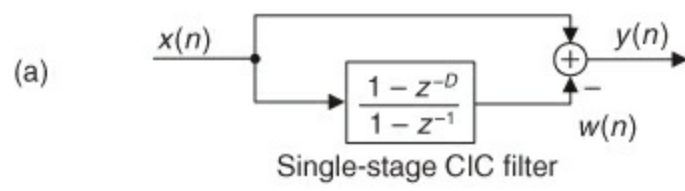


(a) After the decimation by 6, any spectral energy in the shaded area of the filter's response will alias into the B -width signal-of-interest passband centered at 0 Hz as was described in the text. For this commercial 5th-order CIC filter, what is the maximum level of the aliased spectral energy after the decimation by 6? (Stated in different words, what is the value of $Atten$ measured in dB for the HSP43220 CIC filter?) Assume $B = 0.04f_{s,in}$.

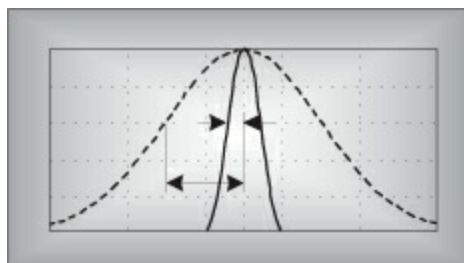
(b) Zooming in on the top portion of the CIC filter's passband, we show the droop in the passband gain in Figure P10-29(b). Measured in dB, what is the HSP43220's maximum passband gain loss at $B/2$ Hz?

10.30 There are digital filtering schemes that use the process conceptually shown in Figure P10-30(a). In that network the input is lowpass filtered to generate the sequence $w(n)$. The network's $y(n)$ output is the $x(n)$ input sequence minus the lowpass-filtered $w(n)$ sequence. The actual implementation of such a process is shown in Figure P10-30(b) where the multi-element delay line in the upper path of Figure P10-30(b) is needed for time alignment to compensate for the time (group) delay of the CIC filter. If we had to implement this parallel-path filter with a CIC filter whose differential delay is $D = 9$, how many unit-delay elements would we use in the upper path of Figure P10-30(b)? Show how you obtained your solution.

Figure P10-30



Chapter Eleven. Signal Averaging



How do we determine the typical amount, a valid estimate, or the true value of some measured parameter? In the physical world, it's not so easy to do because unwanted random disturbances contaminate our measurements. These disturbances are due to both the nature of the variable being measured and the fallibility of our measuring devices. Each time we try to accurately measure some physical quantity, we'll get a slightly different value. Those unwanted fluctuations in a measured value are called *noise*, and digital signal processing practitioners have learned to minimize noise through the process of averaging. In the literature, we can see not only how averaging is used to improve measurement accuracy, but that averaging also shows up in signal detection algorithms as well as in lowpass filter schemes. This chapter introduces the mathematics of averaging and describes how and when this important process is used. Accordingly, as we proceed to quantify the benefits of averaging, we're compelled to make use of the statistical measures known as the mean, variance, and standard deviation.

In digital signal processing, averaging often takes the form of summing a series of time-domain signal samples and then dividing that sum by the number of individual samples. Mathematically, the average of N samples of sequence $x(n)$, denoted x_{ave} , is expressed as

(11-1)

$$x_{\text{ave}} = \frac{1}{N} \sum_{n=1}^N x(n) = \frac{x(1) + x(2) + x(3) + \dots + x(N)}{N}.$$

(What we call the average, statisticians call the *mean*.) In studying averaging, a key definition that we must keep in mind is the variance of the sequence, σ^2 , defined as

(11-2)

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^N [x(n) - x_{\text{ave}}]^2,$$

(11-2')

$$= \frac{[x(1) - x_{\text{ave}}]^2 + [x(2) - x_{\text{ave}}]^2 + [x(3) - x_{\text{ave}}]^2 + \dots + [x(N) - x_{\text{ave}}]^2}{N}.$$

As explained in [Appendix D](#), the σ^2 variance in [Eqs. \(11-2\)](#) and [\(11-2'\)](#) gives us a well-defined quantitative measure of how much the values in a sequence fluctuate about the sequence's average. That's because the $x(1) - x_{\text{ave}}$ value in the bracket, for example, is the difference between the $x(1)$ value and the sequence average x_{ave} . The other important quantity that we'll use is the standard deviation, defined as the positive square root of the variance, or

(11-3)

$$\sigma = \sqrt{\frac{1}{N} \sum_{n=1}^N [x(n) - x_{\text{ave}}]^2} .$$

To reiterate our thoughts, the average value x_{ave} is the constant level about which the individual sequence values may vary. The variance σ^2 indicates the sum of the magnitudes squared of the noise fluctuations of the individual sequence values about the x_{ave} average value. If the sequence $x(n)$ represents a time series of signal samples, we can say that x_{ave} specifies the constant, or DC, value of the signal, the standard deviation σ reflects the amount of the fluctuating, or AC, component of the signal, and the variance σ^2 is an indication of the power in the fluctuating component. ([Appendix D](#) explains and demonstrates the nature of these statistical concepts for those readers who don't use them on a daily basis.)

We're now ready to investigate two kinds of averaging, *coherent* and *incoherent*, to learn how they're different from each other and to see under what conditions they should be used.

11.1 Coherent Averaging

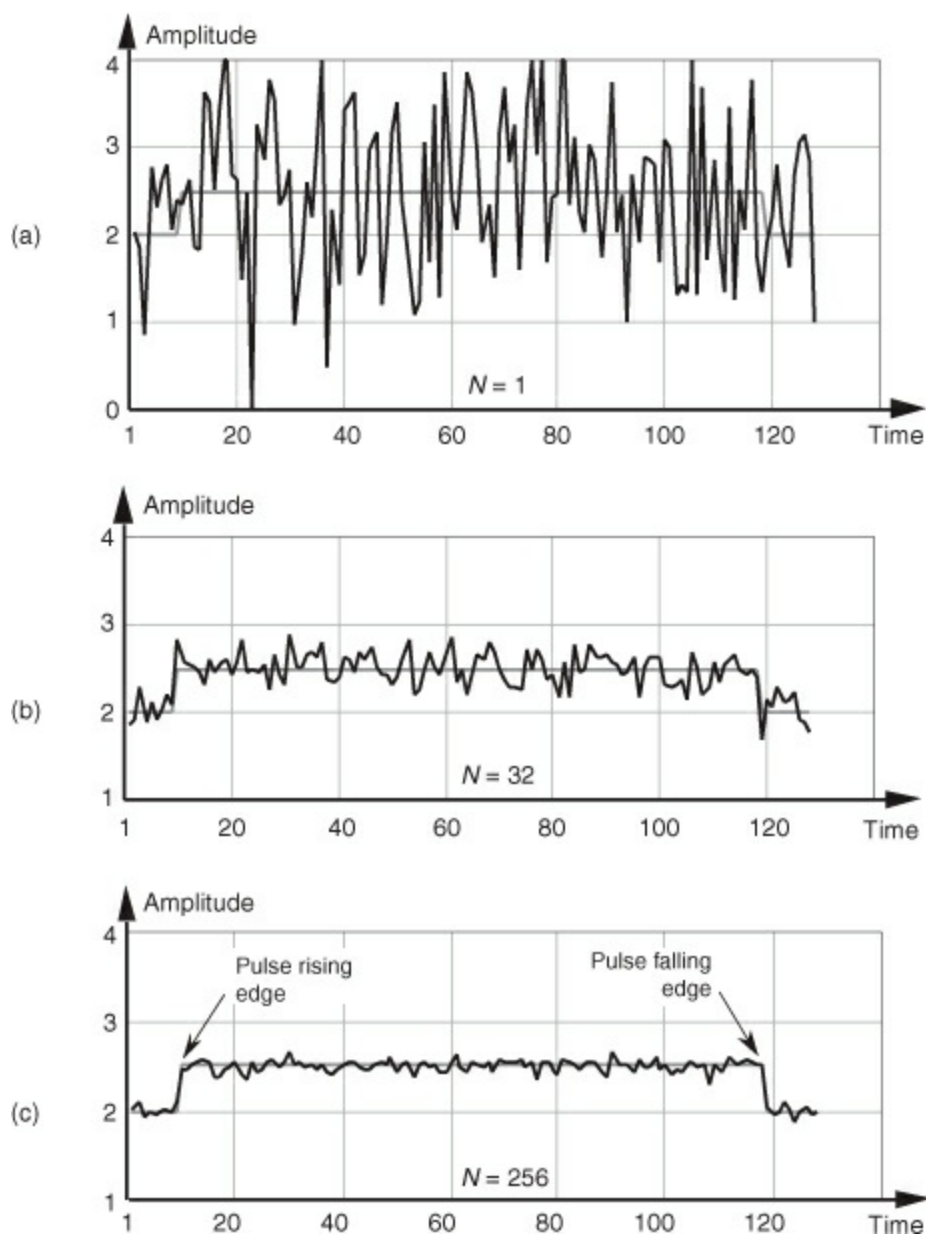
In the coherent averaging process (also known as *time-synchronous averaging*), the key feature is the timing used in sampling the original signal; that is, we collect multiple sets of signal-plus-noise samples, and we need the time phase of the signal in each set to be identical. For example, when averaging a sinewave embedded in noise, coherent averaging requires that the phase of the sinewave be the same at the beginning of each measured sample set. When this requirement is met, the sinewave will average to its true sinewave amplitude value. The noise, however, is different in each sample set and will average toward zero.[‡] The point is that coherent averaging reduces the variance of the noise, while preserving the amplitude of signals that are synchronous, or coherent, with the beginning of the sampling interval. With coherent averaging, we can actually improve the signal-to-noise ratio of a noisy signal. By way of example, consider the sequence of 128 data points plotted in [Figure 11-1\(a\)](#). Those data points represent the time-domain sampling of a single pulse contaminated with random noise. (For illustrative purposes the pulse, whose peak amplitude is 2.5, is shown in the background of [Figure 11-1](#).) It's very difficult to see a pulse in the bold pulse-plus-noise waveform in the foreground of [Figure 11-1\(a\)](#). Let's say we collect 32 sets of 128 pulse-plus-noise samples of the form

[‡] Noise samples are assumed to be uncorrelated with each other and uncorrelated with the sample rate. If some component of the noise is correlated with the sample rate, that noise component *will* be preserved after averaging.

(11-4)

$$\begin{aligned} \text{Sample set}_1 &= x_1(1), x_1(2), x_1(3), \dots, x_1(128), \\ \text{Sample set}_2 &= x_2(1), x_2(2), x_2(3), \dots, x_2(128), \\ \text{Sample set}_3 &= x_3(1), x_3(2), x_3(3), \dots, x_3(128), \\ &\dots \\ &\dots \\ \text{Sample set}_{32} &= x_{32}(1), x_{32}(2), x_{32}(3), \dots, x_{32}(128). \end{aligned}$$

Figure 11-1 Signal pulse plus noise: (a) one sample set; (b) average of 32 sample sets; (c) average of 256 sample sets.



Here's where the coherent part comes in: the signal measurement times must be synchronized, in some manner, with the beginning of the pulse, so that the pulse is in a constant time relationship with the first sample of each sample set. Coherent averaging of the 32 sets of samples, adding up the columns of [Eq. \(11-4\)](#), takes the form of

$$x_{\text{ave}}(k) = \frac{1}{32} \sum_{n=1}^{32} x_n(k) = [x_1(k) + x_2(k) + x_3(k) + \dots + x_{32}(k)] / 32 ,$$

or

$$(11-5)$$

$$x_{\text{ave}}(1) = [x_1(1) + x_2(1) + x_3(1) + \dots + x_{32}(1)] / 32$$

$$x_{\text{ave}}(2) = [x_1(2) + x_2(2) + x_3(2) + \dots + x_{32}(2)] / 32$$

$$x_{\text{ave}}(3) = [x_1(3) + x_2(3) + x_3(3) + \dots + x_{32}(3)] / 32$$

...

...

$$x_{\text{ave}}(128) = [x_1(128) + x_2(128) + x_3(128) + \dots + x_{32}(128)] / 32 .$$

If we perform 32 averages indicated by [Eq. \(11-5\)](#) on a noisy pulse like that in [Figure 11-1\(a\)](#), we'd get the 128-point $x_{\text{ave}}(k)$ sequence plotted in [Figure 11-1\(b\)](#). Here, we've reduced the noise

fluctuations riding on the pulse, and the pulse shape is beginning to become apparent. The coherent average of 256 sets of pulse measurement sequences results in the plot shown in [Figure 11-1\(c\)](#), where the pulse shape is clearly visible now. We've reduced the noise fluctuations while preserving the pulse amplitude. (An important concept to keep in mind is that summation and averaging both reduce noise variance. Summation is merely implementing [Eq. \(11-5\)](#) without dividing the sum by $N = 32$. If we perform summations and don't divide by N , we merely change the vertical scales for the graphs in [Figures 11-1\(b\)](#) and [11-1\(c\)](#). However, the noise fluctuations will remain unchanged relative to true pulse amplitude on the new scale.)

The mathematics of this averaging process in [Eq. \(11-5\)](#) is both straightforward and important. What we'd like to know is the signal-to-noise improvement gained by coherent averaging as a function of N , the number of sample sets averaged. Let's say that we want to measure some constant time signal with amplitude A , and each time we actually make a measurement we get a slightly different value for A . We realize that our measurements are contaminated with noise such that the n th measurement result $r(n)$ is

$$(11-6)$$

$$r(n) = A + \text{noise}(n)$$

where $\text{noise}(n)$ is the noise contribution. Our goal is to determine A when the $r(n)$ sequence of noisy measurements is all we have to work with. For a more accurate estimate of A , we average N separate $r(n)$ measurement samples and calculate a single average value r_{ave} . To get a feeling for the accuracy of r_{ave} , we decide to take a series of averages, $r_{\text{ave}}(k)$, to see how that series fluctuates with each new average; that is,

$$(11-7)$$

$$\begin{aligned} r_{\text{ave}}(1) &= [r(1) + r(2) + r(3) + \dots + r(N)]/N, && \leftarrow \text{1st } N\text{-point average} \\ r_{\text{ave}}(2) &= [r(N+1) + r(N+2) + r(N+3) + \dots + r(2N)]/N, && \leftarrow \text{2nd } N\text{-point average} \\ r_{\text{ave}}(3) &= [r(2N+1) + r(2N+2) + r(2N+3) + \dots + r(3N)]/N, && \leftarrow \text{3rd } N\text{-point average} \\ &\dots && \\ &\dots && \\ r_{\text{ave}}(k) &= [r([k-1] \cdot N + 1) + r([k-1] \cdot N + 2) + r([k-1] \cdot N + 3) + \dots + r(k \cdot N)]/N, \end{aligned}$$

or, more concisely,

$$(11-8)$$

$$r_{\text{ave}}(k) = \frac{1}{N} \sum_{n=1}^N r([k-1] \cdot N + n).$$

To see how averaging reduces our measurement uncertainty, we need to compare the standard deviation of our $r_{\text{ave}}(k)$ sequence of averages with the standard deviation of the original $r(n)$ sequence.

If the standard deviation of our original series of measurements $r(n)$ is σ_{in} , it has been shown[\[1–5\]](#) that the standard deviation of our $r_{\text{ave}}(k)$ sequence of N -point averages, σ_{ave} , is given by

$$(11-9)$$

$$\sigma_{\text{ave}} = \frac{\sigma_{\text{in}}}{\sqrt{N}}.$$

Likewise, we can relate the variance of our $r_{\text{ave}}(k)$ sequence of N -point averages to the variance of the original series of $r(n)$ measurements as

(11-9')

$$\sigma_{\text{ave}}^2 = \frac{\sigma_{\text{in}}^2}{N}.$$

[Equation \(11-9\)](#) is significant because it tells us that the $r_{\text{ave}}(k)$ series of averages will not fluctuate as much around A as the original $r(n)$ measurement values did; that is, the $r_{\text{ave}}(k)$ sequence will be less noisy than any $r(n)$ sequence, and the more we average by increasing N , the more closely an individual $r_{\text{ave}}(k)$ estimate will approach the true value of A .[‡]

[‡] [Equation \(11-9\)](#) is based on the assumptions that the average of the original noise is zero and that neither A nor σ_{in} changes during the time we're performing our averages.

In a different way, we can quantify the noise reduction afforded by averaging. If the quantity A represents the amplitude of a signal and σ_{in} represents the standard deviation of the noise riding on that signal amplitude, we can state that the original signal-amplitude-to-noise ratio is

(11-10)

$$SNR_{\text{in}} = \frac{A}{\sigma_{\text{in}}}.$$

Likewise, the signal-amplitude-to-noise ratio at the output of an averaging process, SNR_{ave} , is defined as

(11-11)

$$SNR_{\text{ave}} = \frac{r_{\text{ave}}}{\sigma_{\text{ave}}} = \frac{A}{\sigma_{\text{ave}}}.$$

Continuing, the signal-to-noise ratio *gain*, SNR_{coh} gain, that we've realized through coherent averaging is the ratio of SNR_{ave} over SNR_{in} , or

(11-12)

$$SNR_{\text{coh}} \text{ gain} = \frac{SNR_{\text{ave}}}{SNR_{\text{in}}} = \frac{A / \sigma_{\text{ave}}}{A / \sigma_{\text{in}}} = \frac{\sigma_{\text{in}}}{\sigma_{\text{ave}}}.$$

Substituting σ_{ave} from [Eq. \(11-9\)](#) in [Eq. \(11-12\)](#), the SNR gain becomes

(11-13)

$$SNR_{\text{coh}} \text{ gain} = \frac{\sigma_{\text{in}}}{\sigma_{\text{in}} / \sqrt{N}} = \sqrt{N}.$$

Through averaging, we can realize a signal-to-noise ratio improvement proportional to the square root of the number of signal samples averaged. In terms of signal-to-noise ratio measured in dB, we have a coherent averaging, or *integration*, gain of

(11-14)

$$SNR_{\text{coh}} \text{ gain(dB)} = 20 \cdot \log_{10}(SNR_{\text{coh}}) = 20 \cdot \log_{10}(\sqrt{N}) = 10 \cdot \log_{10}(N).$$

Again, [Eqs. \(11-13\)](#) and [\(11-14\)](#) are valid if A represents the *amplitude* of a signal and σ_{in} represents the original noise standard deviation.

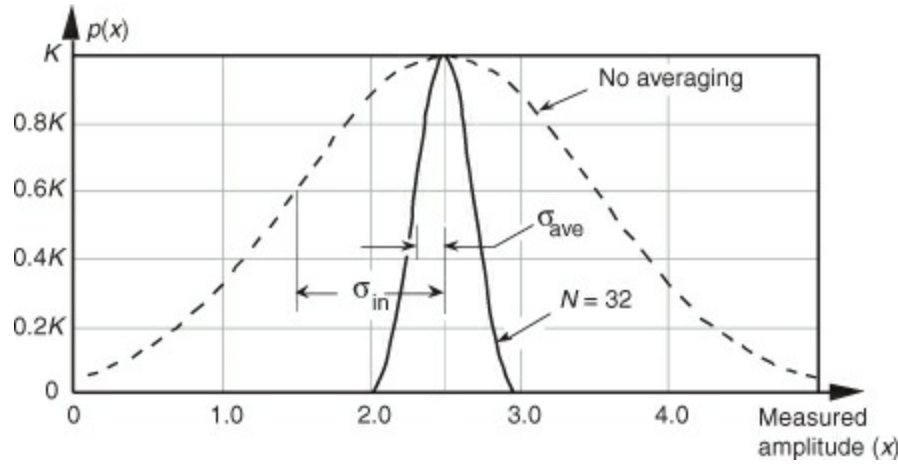
Another way to view the integration gain afforded by coherent averaging is to consider the standard deviation of the input noise, σ_{in} , and the probability of measuring a particular value for the [Figure 11-](#)

1 pulse amplitude. Assume that we made many individual measurements of the pulse amplitude and created a fine-grained histogram of those measured values to get the dashed curve in [Figure 11-2](#). The vertical axis of [Figure 11-2](#) represents the probability of measuring a pulse-amplitude value corresponding to the values on the horizontal axis. If the noise fluctuations follow the well-known *normal*, or Gaussian, distribution, that dashed probability distribution curve is described by

$$(11-15)$$

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2} = K e^{-(x-\mu)^2 / 2\sigma^2},$$

Figure 11-2 Probability density curves of measured pulse amplitudes with no averaging ($N = 1$) and with $N = 32$ averaging.



where $\sigma = \sigma_{in}$ and the true pulse amplitude is represented by $\mu = 2.5$. We see from that dashed curve that any given measured value will most likely (with highest probability) be near the actual pulse-amplitude value of 2.5. Notice, however, that there's a nonzero probability that the measured value could be as low as 1.0 or as high as 4.0. Let's say that the dashed curve represents the probability curve of the pulse-plus-noise signal in [Figure 11-1\(a\)](#). If we averaged a series of 32 pulse-amplitude values and plotted a probability curve of our averaged pulse-amplitude measurements, we'd get the solid curve in [Figure 11-2](#). This curve characterizes the pulse-plus-noise values in [Figure 11-1\(b\)](#). From this solid curve, we see that there's a very low likelihood (probability) that a measured value, after 32-point averaging, will be less than 2.0 or greater than 3.0.

From [Eq. \(11-9\)](#), we know that the standard deviation of the result of averaging 32 signal sample sets is

$$(11-16)$$

$$\sigma_{ave} = \frac{\sigma_{in}}{\sqrt{32}} = \frac{\sigma_{in}}{5.65}.$$

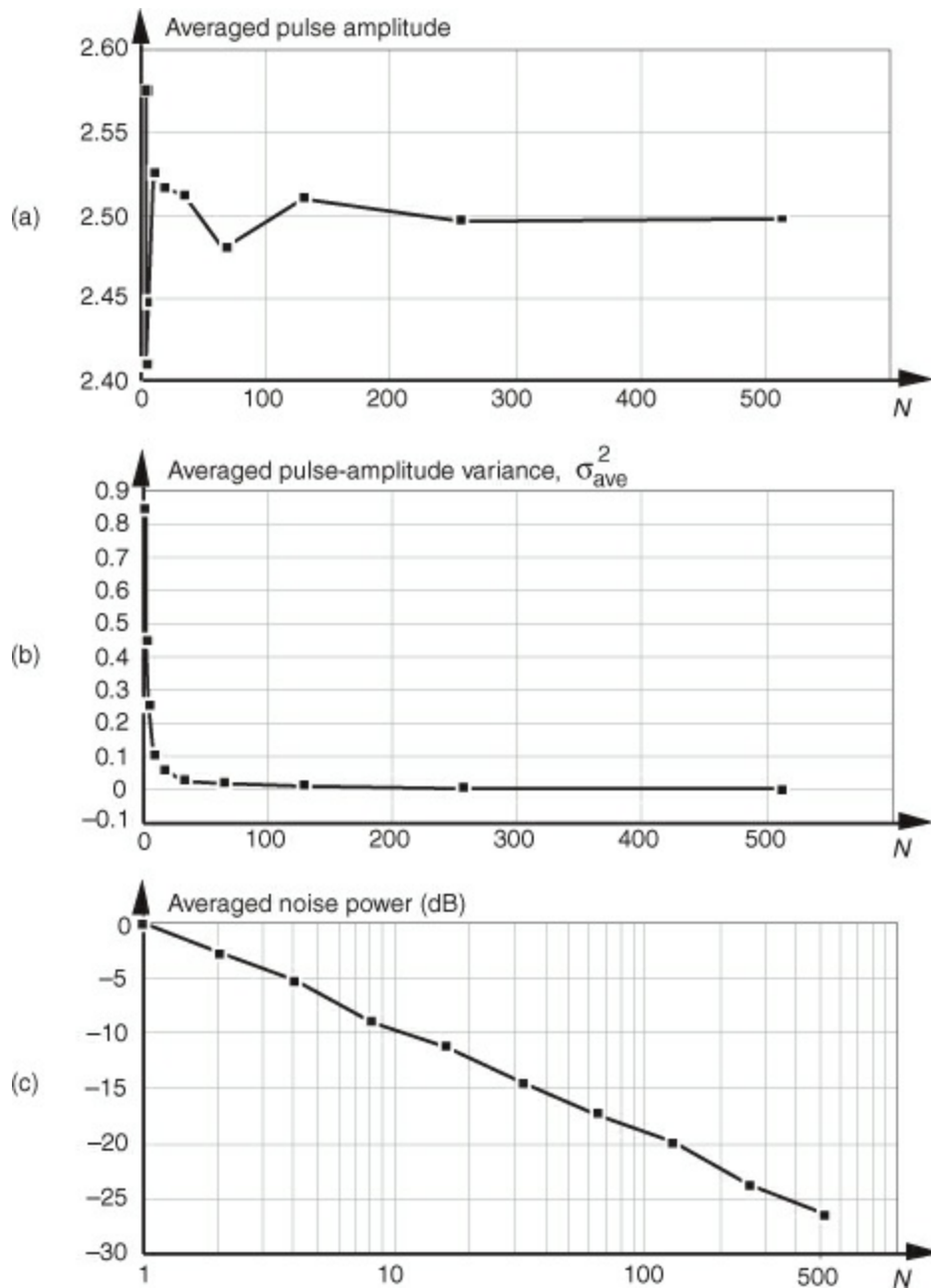
In [Figure 11-2](#), we can see a statistical view of how an averager's output standard deviation is reduced from the averager's input standard deviation. Taking larger averages by increasing N beyond 32 would squeeze the solid curve in [Figure 11-2](#) even more toward its center value of 2.5, the true pulse amplitude.[‡]

[‡] The curves in [Figure 11-2](#) are normalized for convenient illustration. From [Eq. \(11-15\)](#) and assuming that $\sigma = 1$ when $N = 1$, then $K = 0.3989$. When $N = 32$, the new standard deviation is $\sigma' = \sigma / \sqrt{N} = 1 / \sqrt{32}$ and $K = 0.3989 \cdot \sqrt{32} = 2.23$.

Returning to the noisy pulse signal in [Figure 11-1](#), and performing coherent averaging for various numbers of sample sets N , we see in [Figure 11-3\(a\)](#) that as N increases, the averaged pulse amplitude

approaches the true amplitude of 2.5. [Figure 11-3\(b\)](#) shows how rapidly the variance of the noise riding on the pulse falls off as N is increased. An alternate way to see how the noise variance decreases with increasing N is the noise power plotted on a logarithmic scale as in [Figure 11-3\(c\)](#). In this plot, the noise variance is normalized to that noise variance when no averaging is performed, i.e., when $N = 1$. Notice that the slope of the curve in [Figure 11-3\(c\)](#) closely approximates that predicted by [Eqs. \(11-13\)](#) and [\(11-14\)](#); that is, as N increases by a factor of ten, we reduce the average noise power by 10 dB. Although the test signal in this discussion was a pulse signal, had the signal been sinusoidal, [Eqs. \(11-13\)](#) and [\(11-14\)](#) would still apply.

Figure 11-3 Results of averaging signal pulses plus noise: (a) measured pulse amplitude versus N ; (b) measured variance of pulse amplitude versus N ; (c) measured pulse-amplitude noise power versus N on a logarithmic scale.



11.2 Incoherent Averaging

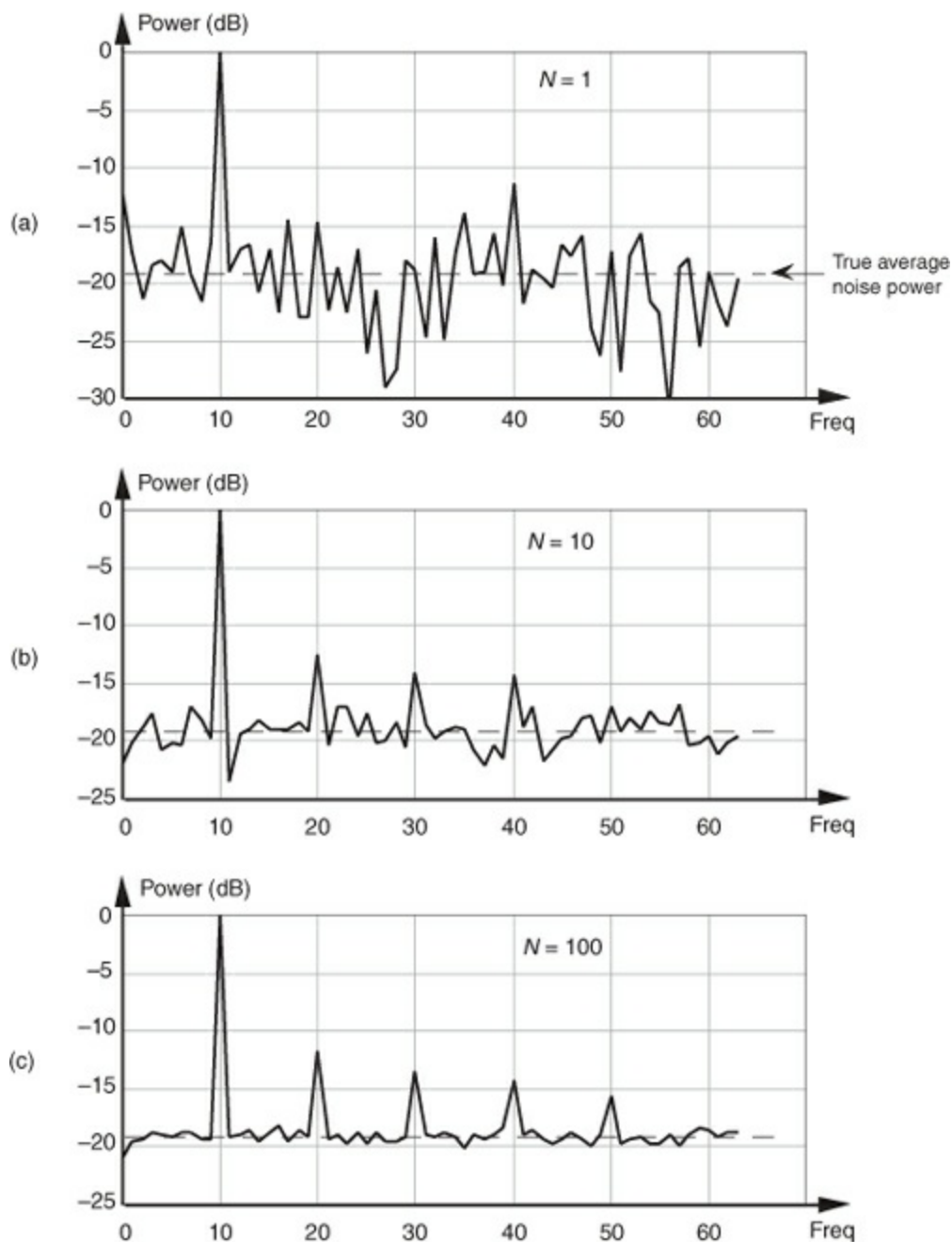
The process of incoherent averaging (also known as *rms*, *postdetection*, *scalar*, or *video averaging*) is the averaging of signal samples where no sample timing constraints are used; that is, signal measurement time intervals are not synchronized in any way with the phase of the signal being

measured. Think for a moment what the average would be of the noisy pulse signal in [Figure 11-1\(a\)](#) if we didn't in some way synchronize the beginning of the collection of the individual signal sample sets with the beginning of the pulse. The result would be pulses that begin at a different time index in each sample set. The averaging of multiple sample sets would then smear the pulse across the sample set, or just “average the pulse signal away.” (For those readers familiar with using oscilloscopes, incoherent averaging would be like trying to view the pulse when the beginning of the scope sweep was not triggered by the signal.) As such, incoherent averaging is not so useful in the time domain.[‡] In the frequency domain, however, it's a different story because incoherent averaging can provide increased accuracy in measuring relative signal powers. Indeed, incoherent averaging is used in many test instruments, such as spectrum, network, and signal analyzers.

[‡] The term *incoherent averaging* is a bit of a misnomer. Averaging a set of data is just that, averaging—we add up a set of data values and divide by the number of samples in the set. Incoherent averaging should probably be called *averaging data that's obtained incoherently*.

In some analog test equipment, time-domain signals are represented in the frequency domain using a narrowband sweeping filter followed by a power detector. These devices measure signal power as a function of frequency. The power detector is necessary because the sweeping measurement is not synchronized, in time, with the signal being measured. Thus the frequency-domain data represents power only and contains no signal phase information. Although it's too late to improve the input's signal-amplitude-to-noise ratio, incoherent averaging can improve the accuracy of signal power measurements in the presence of noise; that is, if the signal-power spectrum is very noisy, we can reduce the power estimation fluctuations and improve the accuracy of signal-power and noise-power measurements. [Figure 11-4\(a\)](#) illustrates this idea where we see the power (magnitude squared) output of an FFT of a fundamental tone and several tone harmonics buried in background noise. Notice that the noise-power levels in [Figure 11-4\(a\)](#) fluctuate by almost 20 dB about the true average noise power indicated by the dashed line at -19 dB.

Figure 11-4 Results of averaging signal tones plus noise-power spectra: (a) no averaging, $N = 1$; (b) $N = 10$; (c) $N = 100$.



If we take 10 FFTs, average the square of their output magnitudes, and normalize those squared values, we get the power spectrum shown in [Figure 11-4\(b\)](#). Here, we've reduced the variance of the noise in the power spectrum but have not improved the tones' signal-power-to-noise-power ratios; that is, the average noise-power level remains unchanged. Averaging the output magnitudes squared of 100 FFTs results in the spectrum in [Figure 11-4\(c\)](#), which provides a more accurate measure of the relative power levels of the fundamental tone's harmonics.

Just as we arrived at a coherent integration SNR gain expression in [Eq. \(11-14\)](#), we can express an incoherent integration gain, SNR_{incoh} gain, in terms of SNR measured in dB as

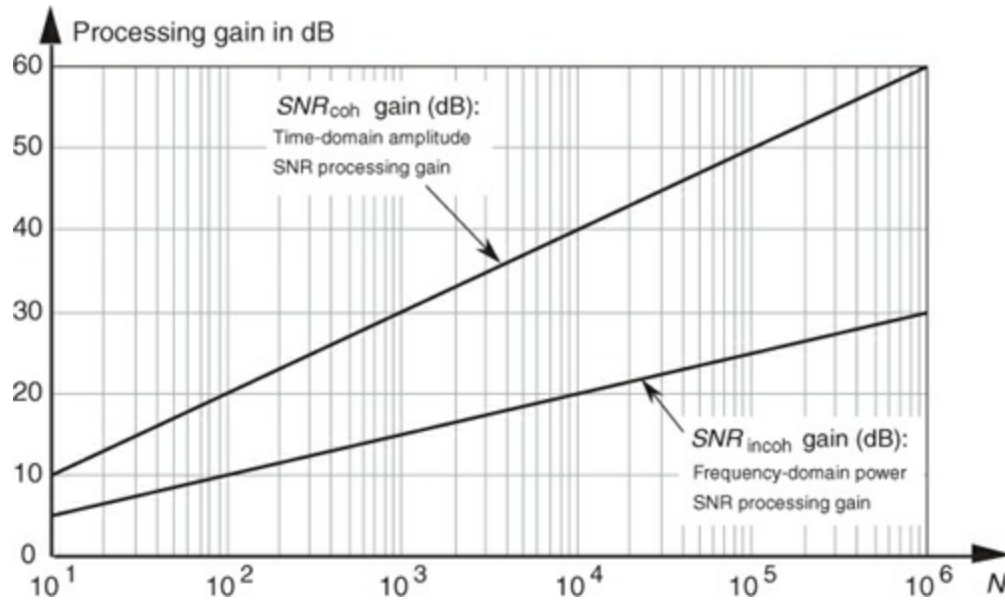
$$(11-17)$$

$$SNR_{\text{incoh}} \text{ gain(dB)} = 10 \cdot \log_{10}(\sqrt{N}).$$

[Equation \(11-17\)](#) applies when the quantity being averaged represents the *power* of a signal. That's why we used the factor of 10 in [Eq. \(11-17\)](#) as opposed to the factor of 20 used in [Eq. \(11-14\)](#).[‡] We can relate the processing gain effects of [Eqs. \(11-14\)](#) and [\(11-17\)](#) by plotting those expressions in [Figure 11-5](#).

[†] [Section E.1](#) of [Appendix E](#) explains why the multiplying factor is 10 for signal-power measurements and 20 when dealing with signal-amplitude values.

Figure 11-5 Time-domain amplitude SNR processing gain from [Eq. \(11-14\)](#), and the frequency-domain power SNR processing gain from [Eq. \(11-17\)](#), as functions of N .



11.3 Averaging Multiple Fast Fourier Transforms

We discussed the processing gain associated with a single DFT in [Section 3.12](#) and stated that we can realize further processing gain by increasing the point size of any given N -point DFT. Let's discuss this issue when the DFT is implemented using the FFT algorithm. The problem is that large FFTs require a lot of number crunching. Because addition is easier and faster to perform than multiplication, we can average the outputs of multiple FFTs to obtain further FFT signal detection sensitivity; that is, it's easier and typically faster to average the outputs of four 128-point FFTs than it is to calculate one 512-point FFT. The increased FFT sensitivity, or noise variance reduction, due to multiple FFT averaging is also called *integration gain*. So the random noise fluctuations in an FFT's output bins will decrease, while the magnitude of the FFT's signal bin output remains constant when multiple FFT outputs are averaged. (Inherent in this argument is the assumption that the signal is present throughout the observation intervals for all of the FFTs that are being averaged and that the noise sample values are independent of the original sample rate.) There are two types of FFT averaging integration gain: incoherent and coherent.

Incoherent integration, relative to FFTs, is averaging the corresponding bin magnitudes of multiple FFTs; that is, to incoherently average k FFTs, the zeroth bin of the incoherent FFT average $F_{\text{incoh}}(0)$ is given by

(11-18)

$$F_{\text{incoh}}(0) = \frac{|F_1(0)| + |F_2(0)| + |F_3(0)| + \dots + |F_k(0)|}{k}$$

where $|F_n(0)|$ is the magnitude of the zeroth bin from the n th FFT. Likewise, the first bin of the incoherent FFT average, $F_{\text{incoh}}(1)$, is given by

(11-18')

$$F_{\text{incoh}}(1) = \frac{|F_1(1)| + |F_2(1)| + |F_3(1)| + \dots + |F_k(1)|}{k},$$

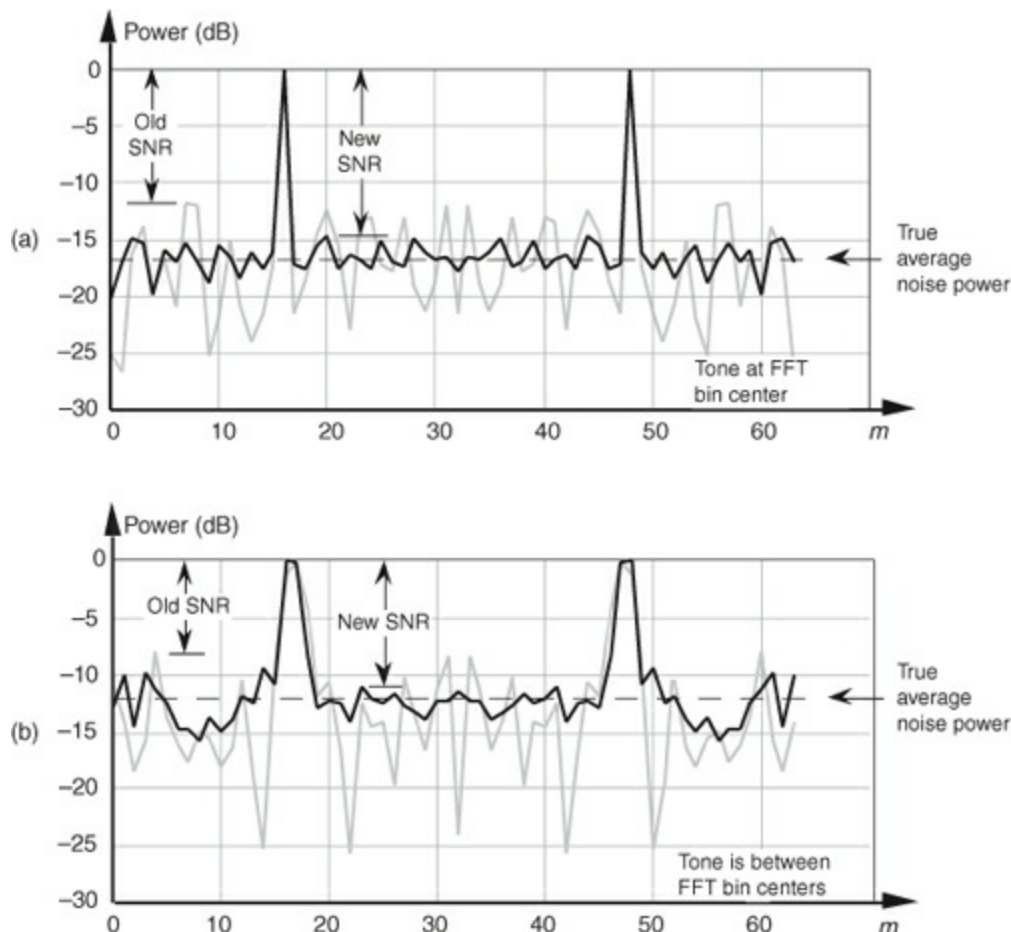
and so on, out to the last bin of the FFT average, $F_{\text{incoh}}(N-1)$, which is

(11-18'')

$$F_{\text{incoh}}(N-1) = \frac{|F_1(N-1)| + |F_2(N-1)| + |F_3(N-1)| + \dots + |F_k(N-1)|}{k}.$$

Incoherent integration provides additional reduction in background noise variation to augment a single FFT's inherent processing gain. We can demonstrate this in [Figure 11-6\(a\)](#), where the shaded curve is a single FFT output of random noise added to a tone centered in the 16th bin of a 64-point FFT. The solid curve in [Figure 11-6\(a\)](#) is the incoherent integration of ten individual 64-point FFT magnitudes. Both curves are normalized to their peak values, so that the vertical scales are referenced to 0 dB. Notice how the variations in the noise power in the solid curve have been reduced by the averaging of the ten FFTs. The noise-power values in the solid curve don't fluctuate as much as the shaded noise-power values. By averaging, we haven't raised the power of the tone in the 16th bin, but we have reduced the peaks of the noise-power values. The larger the number of FFTs averaged, the closer the individual noise-power bin values will approach the true average noise power indicated by the dashed horizontal line in [Figure 11-6\(a\)](#).

Figure 11-6 Single FFT output magnitudes (shaded) and the average of ten FFT output magnitudes (solid): (a) tone at bin center; (b) tone between bin centers.



When the signal tone is not at a bin center, incoherent integration still reduces fluctuations in the FFT's noise-power bins. The shaded curve in [Figure 11-6\(b\)](#) is a single FFT output of random noise added to a tone whose frequency is halfway between the 16th and 17th bins of the 64-point FFT. Likewise, the solid curve in [Figure 11-6\(b\)](#) is the magnitude average of ten FFTs. The variations in

the noise power in the solid curve have again been reduced by the integration of the ten FFTs. So incoherent integration gain reduces noise-power fluctuations regardless of the frequency location of any signals of interest. As we would expect, the signal peaks are wider, and the true average noise power is larger in [Figure 11-6\(b\)](#) relative to [Figure 11-6\(a\)](#) because leakage raises the average noise-power level and scalloping loss reduces the FFT bin's output power level in [Figure 11-6\(b\)](#). The thing to remember is that incoherent averaging of FFT output magnitudes reduces the *variations* in the background noise power but does not reduce the average background noise power. Equivalent to the incoherent averaging results in [Section 11.2](#), the reduction in the output noise variance[6] of the incoherent average of k FFTs relative to the output noise variance of a single FFT is expressed as

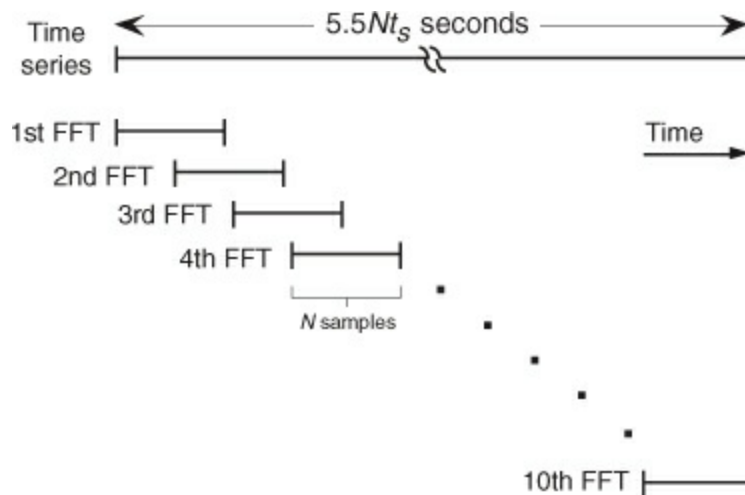
(11-19)

$$\frac{\sigma^2_{k \text{ FFTs}}}{\sigma^2_{\text{single FFT}}} = \frac{1}{k}.$$

Accordingly, if we average the magnitudes of k separate FFTs, we reduce the noise variance by a factor of k .

In practice, when multiple FFTs are averaged and the FFT inputs are windowed, an overlap in the time-domain sampling process is commonly used. [Figure 11-7](#) illustrates this concept with $5.5Nt_s$ seconds, worth of time series data samples, and we wish to average ten separate N -point FFTs where t_s is the sample period ($1/f_s$). Because the FFTs have a 50 percent overlap in the time domain, some of the input noise in the N time samples for the first FFT will also be contained in the second FFT. The question is “What’s the noise variance reduction when some of the noise is common to two FFTs in this averaging scheme?” Well, the answer depends on the window function used on the data before the FFTs are performed. It has been shown that for the most common window functions using an overlap of 50 percent or less, [Eq. \(11-19\)](#) still applies as the level of noise variance reduction[7].

Figure 11-7 Time relationship of multiple FFTs with 50 percent overlap.



Coherent FFT integration gain is possible when we average the real parts of multiple FFT bin outputs separately from computing the average of the imaginary parts. We can then combine the single real average and the single imaginary average into a single complex bin output average value. While this process may be useful for people who use analog sinewave signals to test the performance of A/D converters using the FFT, it only works for periodic time-domain signal sequences that have been obtained through careful synchronous sampling. Coherent integration of multiple FFT results is of no value in reducing spectral measurement noise for nonperiodic, real-world, information-carrying

signals.

11.4 Averaging Phase Angles

So far we've discussed averaging time-domain signal amplitude samples and averaging frequency-domain magnitude samples. It's prudent now to briefly discuss the *tricky* aspect of averaging phase-angle samples. We say tricky because, as Peter Kootsookos points out, the circular (wraparound) nature of angles can lead us into trouble when computing phase averages[\[8\]](#).

Consider computing the average of two phase angles, $\alpha = 7\pi/8$ radians and $\beta = -7\pi/8$ radians. Due to the directional nature of phase angles, we know the average of α and β is an angle exactly halfway between $7\pi/8$ radians and $-7\pi/8$ radians, or $\pm\pi$ radians (± 180 degrees). However, standard numerical averaging of the two scalar radian values $7\pi/8$ and $-7\pi/8$ results in zero radians (0 degrees), which is obviously incorrect.

The solution to this dilemma is to treat the two phase angles as the arguments of two complex numbers, add the two complex numbers, and determine the sum's argument (angle) to obtain the desired average phase angle. That is,

(11-20)

$$\text{Average phase of } \alpha \text{ and } \beta = \arg[e^{j\alpha} + e^{j\beta}],$$

where the notation " $\arg[e^{jq}]$ " means the phase angle of complex number e^{jq} . Of course, the complex addition in [Eq. \(11-20\)](#) is performed in rectangular form.

As an example, the average of phase angles $\alpha = 7\pi/8$ radians and $\beta = -7\pi/8$ radians is found by first computing the sum:

(11-20')

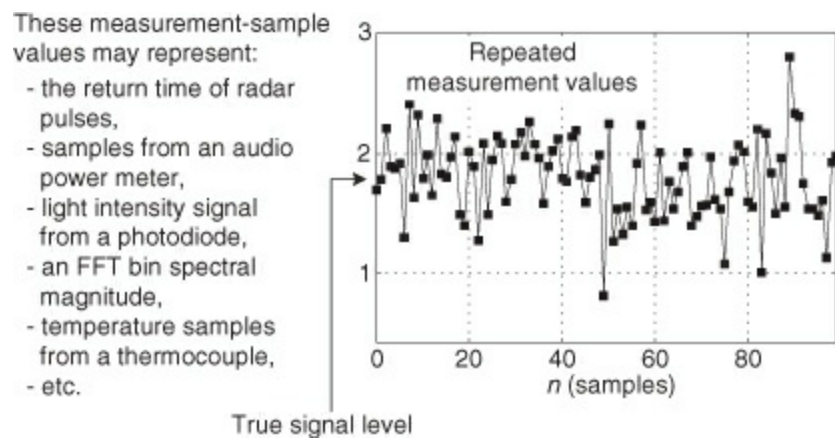
$$\begin{aligned}\text{Average phase of } \alpha \text{ and } \beta &= \arg[e^{j7\pi/8} + e^{-j7\pi/8}] \\ &= \arg[(-0.9239 + j0.3827) + (-0.9239 + j0.3827)] \\ &= \arg[-1.8478] = \arg[1.8478e^{\pm j\pi}].\end{aligned}$$

So, from [Eq. \(11-20'\)](#), our average phase angle is $\pm\pi$ radians (± 180 degrees).

11.5 Filtering Aspects of Time-Domain Averaging

To reinforce our concept of signal averaging, let's reiterate that we want to improve the accuracy (the *correctness*) of our measurement of some physical quantity, but our repeated measurements (signal level samples) are contaminated by random noise as shown in [Figure 11-8](#). That random noise can be inherent in the physical quantity that we're measuring, or it could be caused by an imperfect measurement device (transducer). Sadly, both of these sources of random noise are usually present in our real-world signal measurement activities.

Figure 11-8 A constant-level signal contaminated by random noise.



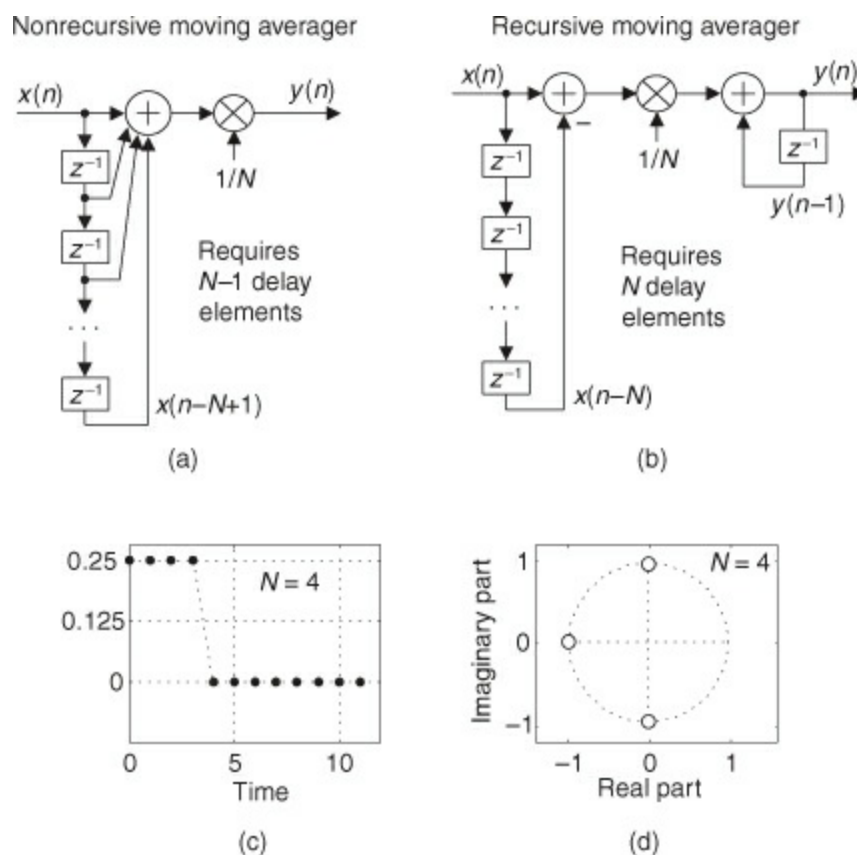
Of course, we can improve the accuracy of our estimation of the *true signal level* in [Figure 11-8](#) by summing a block of 100 measurement values and dividing that sum by 100, which gives us a single 100-point average estimate of the true signal level. However, in a real-time scenario we'd have to wait another 100-sample time interval ($100/f_s$) before we could compute a new estimated true signal level. To compute real-time signal averages at a sample rate of f_s Hz (computing a new average value upon the arrival of each new measurement value), we use digital filters.

In [Section 5.2](#) we introduced nonrecursive FIR filters with a moving average example, and there we learned that time-domain averaging performs lowpass filtering. [Figure 11-9\(a\)](#) shows an N -point nonrecursive moving averager implemented with an N -tap FIR filter structure. The N -point nonrecursive moving averager's output in time is expressed as

$$(11-21)$$

$$y(n) = \frac{1}{N} [x(n) + x(n-1) + x(n-2) + x(n-3) + \dots + x(n-N+1)]$$

Figure 11-9 N -point moving averagers: (a) nonrecursive; (b) recursive; (c) $N = 4$ impulse response; (d) $N = 4$ z -plane zeros locations.



while its z -domain transfer function is

(11-22)

$$H_{\text{ma}}(z) = \frac{Y(z)}{X(z)} = \frac{1}{N} \sum_{n=0}^{N-1} z^{-n} = \frac{1}{N} [1 + z^{-1} + z^{-2} + \dots + z^{-(N-1)}],$$

where the “ma” subscript means moving average.

[Figure 11-9\(b\)](#) illustrates an N -point recursive moving averager. The recursive moving averager has the sweet advantage that only two additions are required per output sample, regardless of the number of delay elements. (So a 100-point moving averager, for example, can be built that requires only two adds per output sample.) (Some people refer to both of our moving averagers as “boxcar averagers.”) An N -point recursive moving averager’s difference equation is

(11-23)

$$y(n) = \frac{1}{N} [x(n) - x(n - N)] + y(n - 1)$$

while its z -domain transfer function is

(11-24)

$$H_{\text{rma}}(z) = \frac{1}{N} \cdot \frac{1 - z^{-N}}{1 - z^{-1}},$$

where the “rma” subscript means recursive moving average.

The nonrecursive and recursive moving averagers have identical time-domain impulse responses and identical linear-phase frequency responses. As such, $H_{\text{ma}}(z) = H_{\text{rma}}(z)$. The nonrecursive and recursive moving averagers are merely two different implementations of the process known as an “ N -point moving average.” The unit impulse response and z -plane pole/zero plot of $N = 4$ moving averagers are provided in [Figures 11-9\(c\)](#) and [11-9\(d\)](#).

Please be aware of two issues regarding the nonrecursive and recursive moving averagers. First, the delay line of the nonrecursive moving averager will have $N-1$ delay elements, while the recursive moving averager will have N delay elements. Second, the feedback in the recursive moving averager means that, given certain $x(n)$ signals, the $y(n)$ output sequence can grow large in magnitude. This means that when implementing a recursive moving averager in fixed-point binary hardware we must test the process against our expected input signals to determine if binary overflow errors occur at the output of the second adder.

An agreeable feature of the moving averagers is that when N is an integer power of two, the multiplications by $1/N$ in [Figure 11-9](#) can be implemented with binary arithmetic right shifts, thus eliminating the multipliers altogether.

Both moving averagers have identical frequency magnitude responses, given by

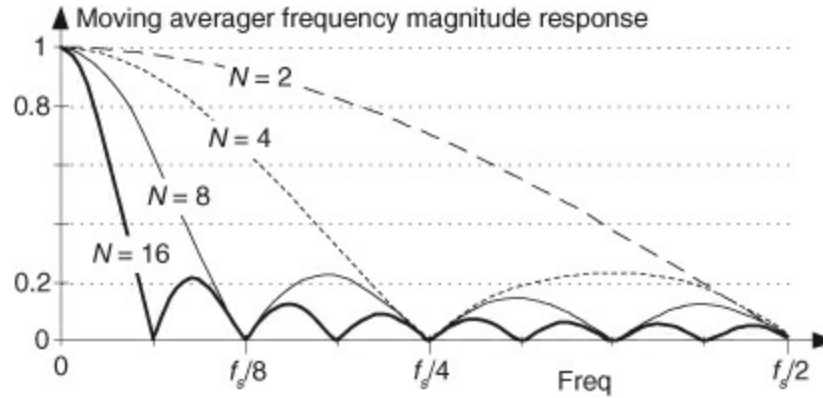
(11-25)

$$|H_{\text{ma}}(f)| = \frac{1}{N} \left| \frac{\sin(\pi f N)}{\pi f N} \right|$$

where the normalized frequency variable f is in the range of -0.5 to 0.5 corresponding to a continuous-time frequency range of $-f_s/2$ to $f_s/2$ Hz. (We derived [Eq. \(11-25\)](#) in [Section 7.5.1](#) when $k = 0$, and in [Section 10.14.2](#).) That response shows us why the averagers’ outputs have reduced noise

fluctuations. [Figure 11-10](#) depicts a moving averaging filter's frequency magnitude responses for various values of N . Those curves are approximated by the $\sin(x)/x$ -like curves we encountered so often in [Chapter 3](#) because they are the discrete Fourier transform (DFT) of an averager's rectangular time-domain impulse responses.

Figure 11-10 N -point moving averager frequency magnitude response as a function of N .



In [Figure 11-10](#) we see the moving average filter has a passband centered at zero Hz, and as N increases, the filter becomes more and more narrowband, attenuating more and more of the noise spectrum of an input signal. The frequencies of the response nulls in [Figure 11-10](#) for $N = 4$ ($\pm f_s/4$ and $f_s/2$) correspond to the locations of the z -plane zeros on the unit circle in [Figure 11-9\(d\)](#). In the general case, the z -plane zeros on the unit circle for an N -point moving averager will be located at angles

$$(11-26)$$

$$\theta_{\text{zeros}} = k \frac{2\pi}{N} \text{ radians}$$

corresponding to magnitude response nulls at frequencies

$$(11-26')$$

$$f_{\text{nulls}} = k \frac{f_s}{N} \text{ Hz}$$

where $k = 1, 2, 3, \dots, N-1$.

The output variance (noise power) properties of both moving averagers abide by the important relationship of

$$(11-27)$$

$$\sigma_{\text{out}}^2 = \frac{\sigma_{\text{in}}^2}{N}.$$

While used in many applications seeking noise reduction through real-time averaging, the above moving averagers have two shortcomings. First, the number of points in the average, N , must be an integer, so if we desired a noise-reducing frequency response somewhere between, say, $N = 4$ and $N = 5$ in [Figure 11-10](#), we're out of luck. Second, in real-time applications, these averagers are sluggish (slow) in their time response to abrupt amplitude changes in an input signal. One popular solution to these shortcomings is the computationally efficient *exponential averager*. Please read on.

11.6 Exponential Averaging

There is a kind of time-domain averaging that's used in many applications—it's called *exponential*

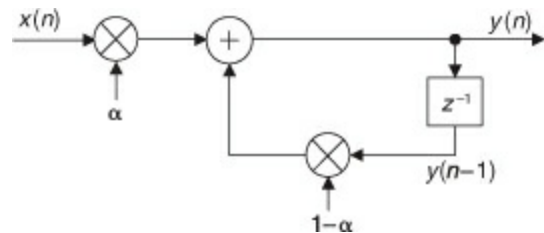
averaging[9–12]. This noise-reduction process, occasionally called *exponential smoothing*, uses a simple recursive lowpass filter described by the difference equation

$$(11-28)$$

$$y(n) = \alpha x(n) + (1 - \alpha)y(n-1)$$

where $y(n)$ is the current averager output sample, $y(n-1)$ is the previous averager output sample, and α is a constant *weighting factor* in the range $0 < \alpha < 1$. The process described by [Eq. \(11-28\)](#) is implemented as shown in [Figure 11-11](#).

Figure 11-11 Exponential averager.



With regard to noise-reduction filtering, the exponential averager has three very appealing properties. First, unlike the nonrecursive and recursive moving averagers described in the last section, the exponential averager permits meaningful control over its frequency response, i.e., its noise-reduction behavior. Second, the exponential averager requires fewer computations per output sample than standard nonrecursive moving averagers; and third, the exponential averager has greatly reduced memory requirements. Only one delay element, i.e., one memory location, is needed by the exponential averager to store the $y(n-1)$ sample.

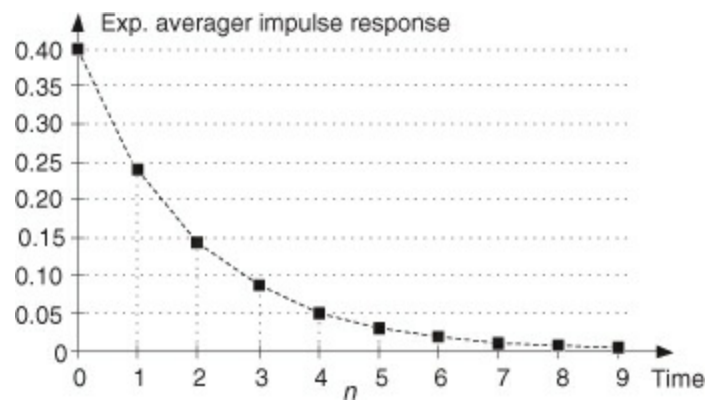
The multiply by α operation could be placed after rather than before the feedback network, if we chose to do so.

11.6.1 Time-Domain Filter Behavior

The exponential averager’s name stems from its time-domain impulse response. Let’s assume that the input to the averager is a long string of zeros, and we apply a single sample of value 1 at time $n = 0$. Then the input returns again to a string of zero-valued samples. Now if the weighting factor is $\alpha = 0.4$, the averager’s output is the impulse response sequence in [Figure 11-12](#). When $n = 0$, the input sample is multiplied by α , so the output is 0.4. On the next clock cycle, the input is zero, and the old value of 0.4 is multiplied by $(1 - 0.4)$, or 0.6 multiplied by $(1 - 0.4)$, or 0.6 to provide an output of 0.24. On the following clock cycle the input is zero and the previous output of 0.24 is multiplied by 0.6 to provide a new output of 0.144. This continues with the averager’s impulse response output falling off exponentially because of the successive multiplications by 0.6.[‡]

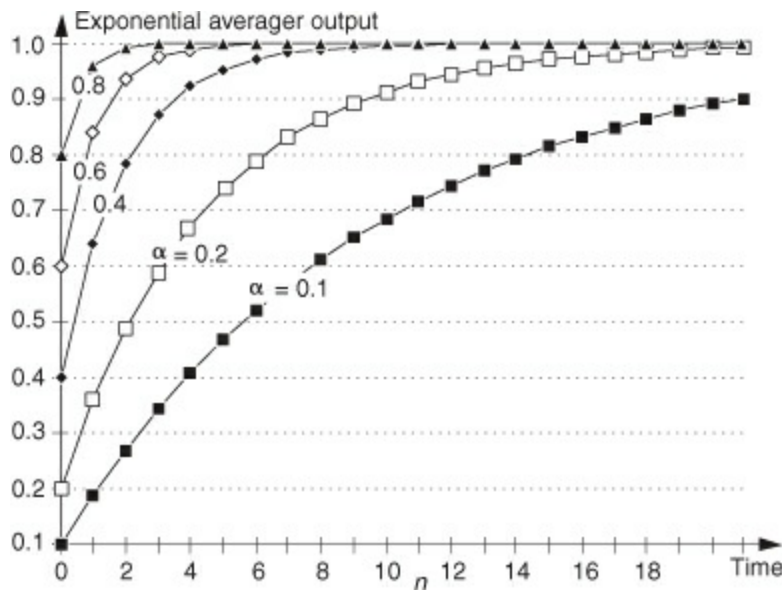
[‡] We often see exponential decay in nature—everywhere from a capacitor discharging through a resistor, the flow of heat, to the shrinkage of bubbles in a glass of beer. (See reference [\[13\]](#).)

Figure 11-12 Exponential averager impulse response with $\alpha = 0.4$.



A useful feature of the exponential averager is its capability to vary the amount of noise reduction by changing the value of the α weighting factor. If α equals one, input samples are not attenuated, past averager outputs are ignored, and no averaging takes place. In this case the averager output responds immediately to changes at the input. As α is decreased in value, input samples are attenuated and past averager outputs begin to affect the present output. These past values represent an exponentially weighted sum of recent inputs, and that summation tends to smooth out noisy signals. The smaller α gets, the more noise reduction is realized. However, with smaller values for α , the slower the averager is in responding to changes in the input. We can demonstrate this behavior by looking at the exponential averager's time-domain step response as a function of α as shown in [Figure 11-13](#).

Figure 11-13 Exponential averager output versus α when a step input is applied at time $n = 0$.



As so often happens in signal processing, we have a trade-off. The more the noise reduction, the more sluggish the averager will be in responding to abrupt changes at the input. We can see in [Figure 11-13](#) that as α gets smaller, affording better noise reduction, the averager's output takes longer to respond and stabilize. Some test instrumentation manufacturers use a clever scheme to resolve this noise reduction versus response time trade-off. They use a large value for α at the beginning of a measurement so the averager's output responds immediately with a nonzero value. Then as the measurement proceeds, the value of α is decreased in order to reduce the noise fluctuations at the input.

The exponential averager's noise variance reduction as a function of the weighting factor α has been shown to be [\[10,11\]](#)

$$(11-29)$$

$$\frac{\text{output noise variance}}{\text{input noise variance}} = \frac{\alpha}{2 - \alpha}.$$

[Equation \(11-29\)](#) is useful because it allows us to determine α given some desired averager noise variance (power) reduction. That is, if our desired noise variance reduction factor is R , where $R = (2 - \alpha)/\alpha$, we can write

$$(11-30)$$

$$\alpha = \frac{2}{R + 1}.$$

For example, if we want the output noise variance reduced by a factor of $R = 10$, then $\alpha = 2/(10+1) = 0.182$. The behavior of exponential averaging is such that to achieve noise reduction roughly equivalent to an N -point moving averager, we define α as

$$(11-31)$$

$$\alpha = \frac{2}{N + 1}, \text{ for } N > 3.$$

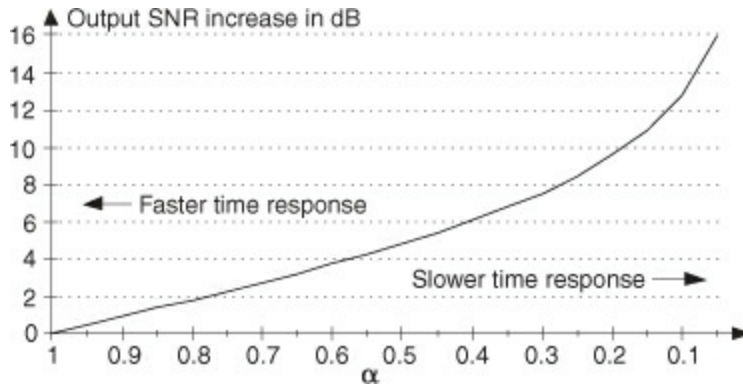
Considering the exponential averager's noise power reduction in [Eq. \(11-29\)](#) as an output signal-to-noise (SNR) increase, we can say the averager's output SNR increase (in dB) is

$$(11-32)$$

$$SNR_{\text{exp(dB)}} = 10 \cdot \log_{10} \left(\frac{\alpha}{2 - \alpha} \right).$$

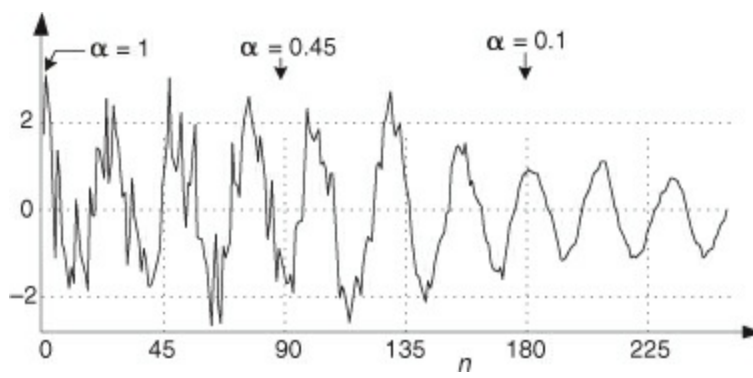
[Equation \(11-32\)](#) is plotted in [Figure 11-14](#) to illustrate the trade-off between output noise reduction and averager response times.

Figure 11-14 Exponential averager output SNR increase as a function of the weighting factor α .



To demonstrate the exponential averager's output noise power reduction capabilities, [Figure 11-15](#) shows the averager's output with a low-frequency (relative to the sample rate) cosine wave plus high-level noise as an input. The weighting factor α starts out with a value of 1 and decreases linearly to a final value of 0.1 at the 180th data input sample. Notice that the noise is reduced as α decreases.

Figure 11-15 Exponential averager output noise reduction as α decreases.



11.6.2 Frequency-Domain Filter Behavior

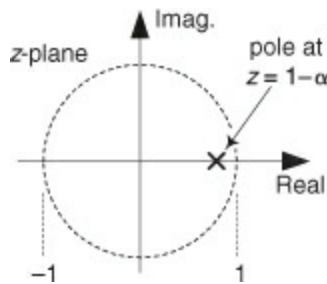
The reader may recognize the exponential averager as a 1st-order infinite impulse response (IIR) digital filter. It has a z -domain transfer function of

(11-33)

$$H_{\text{exp}}(z) = \frac{\alpha}{1 - (1 - \alpha)z^{-1}}.$$

Being a 1st-order IIR filter, the exponential averager has a single pole on the z -plane located at $z = 1 - \alpha$ as shown in [Figure 11-16](#). When α is reduced in value, the pole resides closer to the z -plane's unit circle, giving us a narrower lowpass passband width.

Figure 11-16 Exponential averager z -plane pole location.



Setting z in [Eq. \(11-33\)](#) equal to $e^{j\omega}$, we can write the frequency response of the exponential averager as

(11-34)

$$H_{\text{exp}}(\omega) = \frac{\alpha}{1 - (1 - \alpha)e^{-j\omega}} = \frac{\alpha}{1 - (1 - \alpha)\cos(\omega) + j(1 - \alpha)\sin(\omega)}.$$

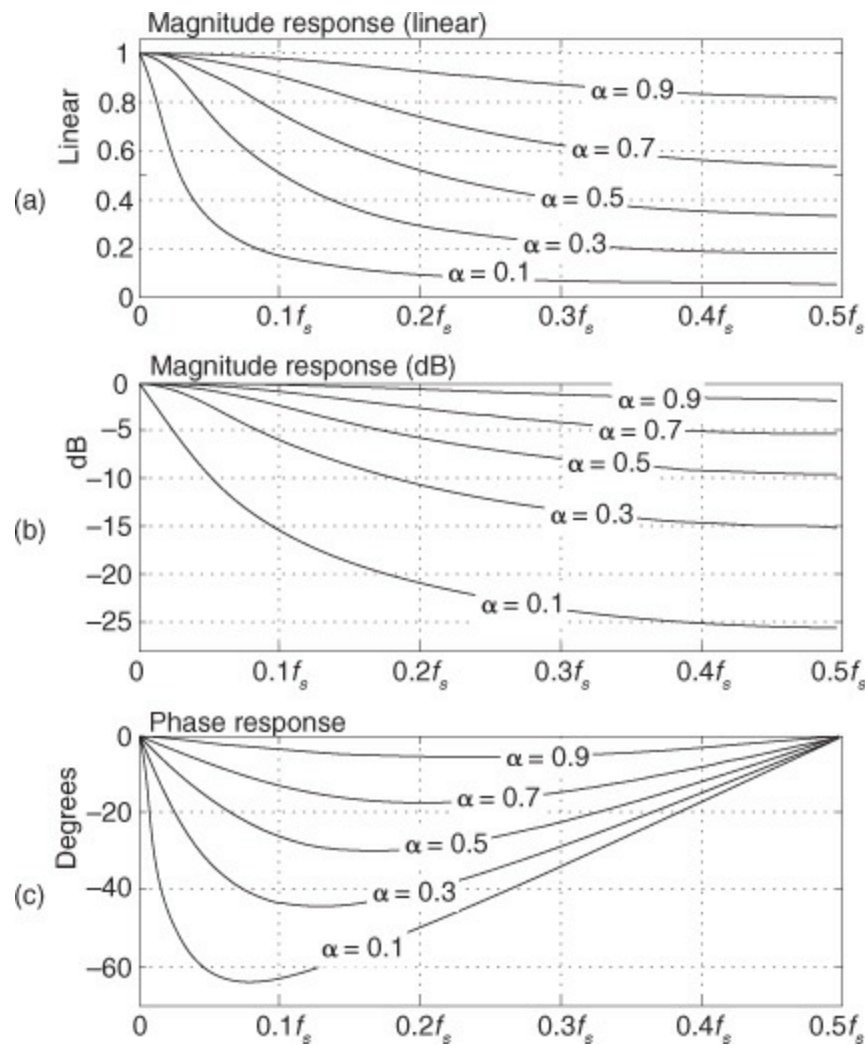
If we're interested in the magnitude response of our averager, we can express it as

(11-35)

$$|H_{\text{exp}}(\omega)| = \frac{\alpha}{\sqrt{1 - 2(1 - \alpha)\cos(\omega) + (1 - \alpha)^2}}.$$

Evaluating [Eq. \(11-35\)](#) over the normalized angular range of $0 \leq \omega \leq \pi$ (corresponding to a continuous-time frequency range of 0 to $f_s/2$ Hz), the frequency magnitude responses of our exponential averager for various values of α are shown in [Figure 11-17\(a\)](#). There we see that the averager's DC gain, its gain at zero Hz, is unity, which is just what we want for our noise-reduction applications. It's worth noting that if we can tolerate a DC gain of $1/\alpha$, the multiply by α in [Figure 11-11](#) can be eliminated to reduce the averager's computational workload.

Figure 11-17 Exponential averager frequency response versus α : (a) normalized magnitude response (linear); (b) normalized magnitude response in dB; (c) phase response in degrees.



The exponential averager's magnitude responses plotted on a logarithmic scale (dB) are provided in [Figure 11-17\(b\)](#). Notice as α decreases, the exponential averager behaves more and more like a lowpass filter. Again, it is from this behavior that the exponential averager's noise-reduction properties stem.

For those readers who prefer to think of a lowpass filter in terms of its 3 dB bandwidth, we can compute the appropriate value of the weighting factor α to achieve a desired exponential averaging filter 3 dB bandwidth. If f_c is the desired positive *cutoff* frequency in Hz, where the exponential averager's frequency magnitude response is 3 dB below the averager's zero-Hz response, the value of α needed to achieve such an f_c cutoff frequency is

$$(11-36)$$

$$\alpha = \cos(2f_c/f_s) - 1 + \sqrt{\cos^2(2f_c/f_s) - 4\cos(2f_c/f_s) + 3}$$

where f_s is the averager's input sample rate in Hz.

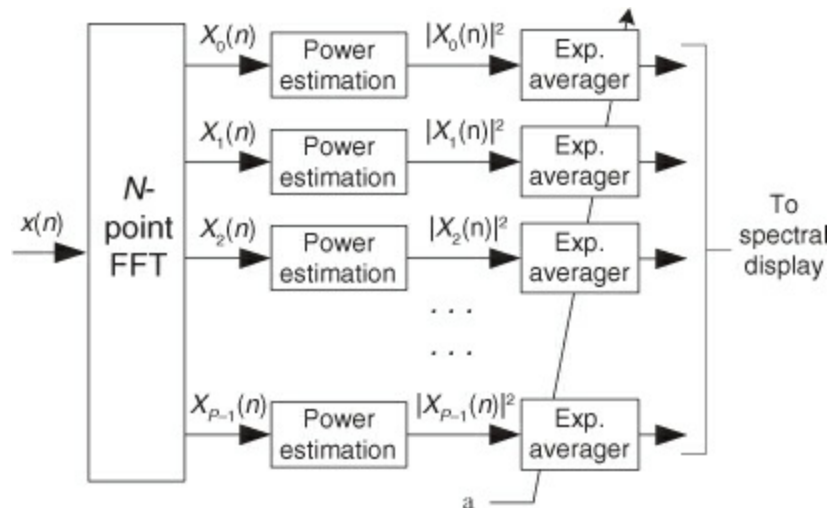
To comment on the exponential averager's nonlinear phase response: We're primarily concerned with the averager's frequency response at zero Hz. We want the averager to pass a zero-Hz (constant-amplitude) signal and attenuate noise fluctuations riding on the constant-amplitude signal of interest. As such, the exponential averager's phase nonlinearity is usually of little consequence.

11.6.3 Exponential Averager Application

I first encountered the exponential averager as lowpass filter in a specialized real-time hardware spectrum analyzer application. That analyzer, using the fast Fourier transform (FFT) shown in [Figure](#)

[11-18](#), was similar in operation to a graphics equalizer in home stereo systems. As P spectral power estimates were displayed on a computer monitor in real time, the common weighting factor α (used by each exponential averager) could be increased to speed the display's response to abrupt changes in the spectral content in $x(n)$. Then again, α could be reduced to minimize abrupt fluctuations (reduced variance) in the P power samples, yielding a slowly changing (sluggish) spectral display. The notation in [Figure 11-18](#) is such that $X_p(n)$ represents the p th FFT bin's complex sample value at the n th instant in time.

Figure 11-18 An application of exponential averaging.



In this application the exponential averagers were referred to as “leaky integrators” and, again, their nonlinear phase was unimportant. Their only purpose in life was to reduce the fluctuations in the real-time $|X_p(n)|^2$ power samples by means of lowpass filtering.

As an example of their utility, exponential averagers are used when we swipe our charge cards through a magnetic stripe reader (MSR). The analog signal from the magnetic read head is digitized with an A/D converter, and the discrete samples are exponentially averaged before the binary data (ones and zeros) detection process is performed[\[14\]](#).

To conclude this section, we inform the reader that [Section 13.33](#) presents computationally efficient implementations of exponential averagers.

References

- [1] Miller, I., and Freund, J. *Probability and Statistics for Engineers*, 2nd ed., Prentice Hall, Englewood Cliffs, New Jersey, 1977, p. 118.
- [2] Beller, J., and Pless, W. “A Modular All-Haul Optical Time-Domain Reflectometer for Characterizing Fiber Links,” *Hewlett-Packard Journal*, February 1993.
- [3] Spiegel, M. R. *Theory and Problems of Statistics*, Shaum's Outline Series, McGraw-Hill, New York, 1961, p. 142.
- [4] Papoulis, A. *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, New York, 1984, p. 245.
- [5] Davenport, W. B., Jr., and Root, W. L. *Random Signals and Noise*, McGraw-Hill, New York, 1958, pp. 81–84.
- [6] Welch, P. D. “The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method

Based on Time Averaging over Short, Modified Periodograms,” *IEEE Transactions on Audio and Electroacoust.*, Vol. AU-15, No. 2, June 1967.

- [7] Harris, F. J. “On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform,” *Proceedings of the IEEE*, Vol. 66, No. 1, January 1978.
- [8] Kootsookos, P. “The Nature of Circles,” DSPrelated blog, <http://www.dsprelated.com/showarticle/57.php>.
- [9] Booster, D. H., et al. “Design of a Precision Optical Low-Coherence Reflectometer,” *Hewlett-Packard Journal*, February 1993.
- [10] Witte, R. A. “Averaging Techniques Reduce Test Noise, Improve Accuracy,” *Microwaves & RF*, February 1988.
- [11] Oxaal, J. “Temporal Averaging Techniques Reduce Image Noise,” *EDN*, March 17, 1983.
- [12] Lymer, A. “Digital-Modulation Scheme Processes RF Broadcast Signals,” *Microwaves & RF*, April 1994.
- [13] Leike, A. “Demonstration of the Exponential Decay Law Using Beer Froth,” *European Journal of Physics*, Vol. 23, January 2002, pp. 21–26.
- [14] Silicon Laboratories Inc. “Magnetic Stripe Reader,” *Application Note: AN148*.

Chapter 11 Problems

11.1 Assume we have a four-sample $x(n)$ sequence, where index n is $1 \leq n \leq 4$, whose samples are

$$x(1) = 1, x(2) = 2, x(3) = 3, x(4) = 4.$$

- (a) What is the average of $x(n)$?
- (b) What is the variance of $x(n)$?
- (c) What is the standard deviation of $x(n)$?

11.2 This problem illustrates an important characteristic of the quantity known as the *average* (mean value) of a sequence of numbers. Suppose we have a six-sample $x(n)$ sequence, where index n is $1 \leq n \leq 6$, defined by

$$x(1) = 1, x(2) = -2, x(3) = 3, x(4) = -4, x(5) = 6, x(6) = \text{unspecified},$$

and the average of $x(n)$ is $x_{\text{ave}} = 4$. (Note that the sixth sample in $x(n)$ is not explicitly defined.) The difference between $x(n)$ and x_{ave} is the sequence $d_{\text{iff}}(n) = x(n) - x_{\text{ave}}$, given as

$$d_{\text{iff}}(1) = -3, d_{\text{iff}}(2) = -6, d_{\text{iff}}(3) = -1, d_{\text{iff}}(4) = -8, d_{\text{iff}}(5) = 2, d_{\text{iff}}(6) = \text{unspecified}.$$

- (a) What is the value of $d_{\text{iff}}(6)$? Justify your answer.

Hint: The discussion of sequence averages in [Appendix D](#)’s [Section D.1](#) will be helpful here.

- (b) What is the value of $x(6)$?

11.3 Let’s look at an important topic regarding averaging. Assume we have two N -point discrete sequences, $x(n)$ and $y(n)$, where index n is $1 \leq n \leq N$, and the N -point averages of the two sequences are

$$x_{\text{ave}} = \frac{1}{N} \sum_{n=1}^N x(n), \quad \text{and} \quad y_{\text{ave}} = \frac{1}{N} \sum_{n=1}^N y(n).$$

Next, let's add the two sequences, element for element, to obtain a new N -point sequence $z(n) = x(n) + y(n)$. Is it correct to say that the average of $z(n)$, defined as

$$z_{\text{ave}} = \frac{1}{N} \sum_{n=1}^N z(n),$$

is equal to the sum of x_{ave} and y_{ave} ? (In different words, we're asking, "Is the average of sums equal to the sum of averages?") Explain how you arrived at your answer.

Note: This problem is *not* "busy work." If the above statement $z_{\text{ave}} = x_{\text{ave}} + y_{\text{ave}}$ is true, it tells us that the average of a noisy signal is equal to the average of the noise-free signal plus the average of the noise.

11.4 Suppose we had three unity-magnitude complex numbers whose phase angles are $\pi/4$ radians, $-3\pi/4$ radians, and $-\pi/4$ radians. What is the average phase angle, measured in degrees, of the three phase angles? Show your work.

11.5 Assume we're averaging magnitude samples from multiple FFTs (fast Fourier transforms) and we want the variance of the *averaged FFT magnitudes* to be reduced below the variance of single-FFT magnitudes by a factor of 20. That is, we want

$$\sigma_{k \text{ FFTs}}^2 = \left(\frac{1}{20} \right) \cdot (\sigma_{\text{single FFT}}^2).$$

How many FFTs, k , must we compute and then average their magnitude samples?

11.6 Concerning the moving averager filters in the text's [Figure 11-9](#), we stated that their transfer functions are equal. Prove that $H_{\text{ma}}(z) = H_{\text{rma}}(z)$.

Hint: $H_{\text{ma}}(z)$ is a geometric series that we'd like to represent as a closed-form equation. To obtain a closed-form equation for a geometric series, start by looking up *geometric series* in the Index.

11.7 If we remove the $1/N$ multiplier from the recursive moving averager in the text's [Figure 11-9\(b\)](#), the remaining structure is called a *recursive running sum*. To exercise your digital network analysis skills, plot the frequency magnitude responses of a recursive running sum system for $N = 4, 8$, and 16 as we did in [Figure 11-10](#).

Hint: The frequency response of a recursive running sum network is, of course, the discrete Fourier transform (DFT) of the network's rectangular impulse response. Note that the recursive running sum network's magnitude response curves will be similar, but not equal, to the curves in [Figure 11-10](#).

11.8 In the text we said that the phase responses of both nonrecursive and recursive N -point moving averagers are linear. Why is it valid to make that statement?

11.9 Draw a rough sketch of the frequency magnitude response, over the positive-frequency range, of a three-point moving averager. Clearly show the frequency magnitude response at $f_s/2$ Hz.

Note: The locations of the frequency response nulls are defined by the locations of the averager's

transfer function zeros on its z -plane unit circle.

11.10 Think about building a two-stage filter comprising a four-point moving averager in cascade (series) with a two-point moving averager.

- (a) Draw a rough sketch of the frequency magnitude response of the two-stage filter.
- (b) Does the cascaded filter have a linear phase response? Justify your answer.

11.11 Let's assume we're measuring a constant-level, but very noisy, temperature signal from a thermocouple and we wish to reduce the noise variance (power) of our measurements by 13 dB.

- (a) What is the number of delay elements needed in a nonrecursive moving average filter to achieve the desired measurement-noise reduction?
- (b) What is the number of delay elements needed in a recursive moving average filter to achieve the desired measurement-noise reduction?
- (c) What is the value of the α weighting factor in a standard exponential averager to achieve the desired measurement-noise reduction?
- (d) Fill in the following table describing the implementation requirements to achieve measurement-noise variance reduction of 13 dB.

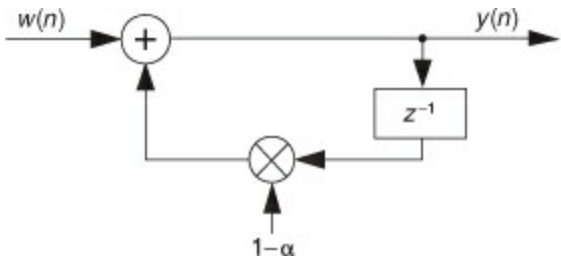
Implementation Requirements for 13 dB Noise Reduction

Computations per output sample and memory requirements	Nonrecursive moving averager	Recursive moving averager	Exponential averager
Multiplies			
Additions			
Data memory locations			

11.12 Regarding the exponential averaging filter, when $\alpha = 0$, the filter's single pole lies right on the z -plane unit circle. In [Chapter 6](#) and [Chapter 7](#) we discussed that having digital filter poles on the unit circle can lead to filter stability problems because quantizing a filter's coefficients to a fixed-width binary word representation can sometimes cause the poles to reside just outside the unit circle. Why does using $\alpha = 0$ cause no stability problems for us when we use exponential averagers?

11.13 In the text we stated that an alternate version of an exponential averager, shown in [Figure P11-13](#), has a DC (zero Hz) gain of $1/\alpha$. Prove that this DC gain factor of $1/\alpha$ is correct.

Figure P11-13

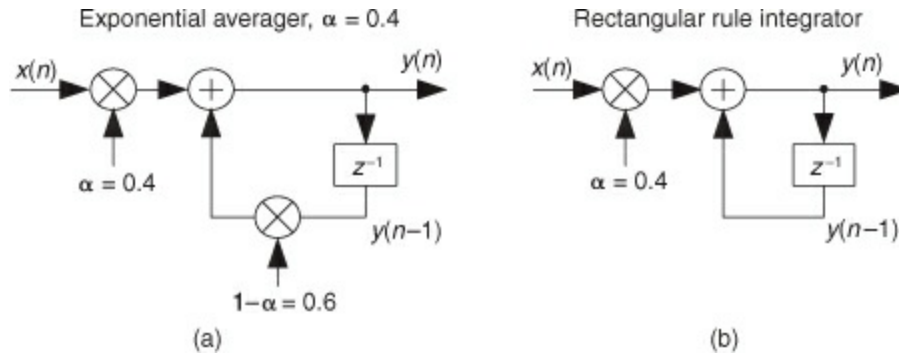


11.14 Show how to derive the equation for the frequency magnitude response of an exponential averager whose weighting factor is α .

11.15 Explain why it's valid to call the exponential averager in [Figure P11-15\(a\)](#), where for example

$\alpha = 0.4$, by the name *leaky integrator* compared to a standard (rectangular rule) integrator shown in [Figure P11-15\(b\)](#)?

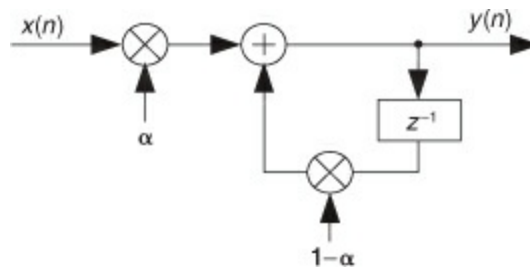
Figure P11-15



Hint: Compare the impulse responses of the two networks.

11.16 Here are (somewhat) challenging problems regarding the exponential averager in [Figure P11-16](#):

Figure P11-16



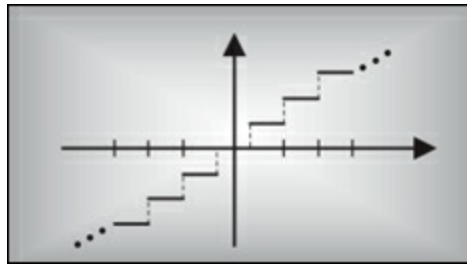
(a) Derive an algebraic expression for the exponential averager's time-domain response to a unity-valued input sample applied at time $n = 0$ followed by all zero-valued input samples. Use the term $h(n]$ to represent this [impulse response](#), where n is the time-domain index. (Assume the output of the z^{-1} delay element is zero at time $n = 0$.)

(b) Use your $h(n]$ expression from Part (a) to determine the exponential averager's gain at zero Hz (DC gain).

Hint: Recall the relationship between a filter's impulse response and its gain at zero Hz.

(c) Comment on how the value of the α weighting factor affects the averager's gain at zero Hz (DC gain).

Chapter Twelve. Digital Data Formats and Their Effects



In digital signal processing, there are many ways to represent numerical data in computing hardware. These representations, known as *data formats*, have a profound effect on the accuracy and ease of implementation of any given signal processing algorithm. The simpler data formats enable uncomplicated hardware designs to be used at the expense of a restricted range of number representation and susceptibility to arithmetic errors. The more elaborate data formats are somewhat difficult to implement in hardware, but they allow us to manipulate very large and very small numbers while providing immunity to many problems associated with digital arithmetic. The data format chosen for any given application can mean the difference between processing success and failure—it's where our algorithmic rubber meets the road.

In this chapter, we'll introduce the most common types of *fixed-point* digital data formats and show why and when they're used. Next, we'll use analog-to-digital (A/D) converter operations to establish the precision and dynamic range afforded by these fixed-point formats along with the inherent errors encountered with their use. Finally, we'll cover the interesting subject of *floating-point* binary formats.

12.1 Fixed-Point Binary Formats

Within digital hardware, numbers are represented by binary digits known as bits—in fact, the term *bit* originated from the words *Binary digIT*. A single bit can be in only one of two possible states: either a one or a zero.[‡] A six-bit binary number could, for example, take the form 101101, with the leftmost bit known as the *most significant bit* (msb); the rightmost bit is called the *least significant bit* (lsb). The number of bits in a binary number is known as the *word length*—hence 101101 has a word length of six. Like the decimal number system so familiar to us, the binary number system assumes a weight associated with each digit in the number. That weight is the base of the system (two for binary numbers and ten for decimal numbers) raised to an integral power. To illustrate this with a simple example, the decimal number 4631 is

[‡] Binary numbers are used because early electronic computer pioneers quickly realized that it was much more practical and reliable to use electrical devices (relays, vacuum tubes, transistors, etc.) that had only two states, *on* or *off*. Thus, the on/off state of a device could represent a single binary digit.

(12-1)

$$\begin{aligned} & (4 \cdot 10^3) + (6 \cdot 10^2) + (3 \cdot 10^1) + (1 \cdot 10^0) \\ & = 4000 + 600 + 30 + 1 = 4631. \end{aligned}$$

The factors 10^3 , 10^2 , 10^1 , and 10^0 are the digit weights in [Eq. \(12-1\)](#). Similarly, the six-bit binary number 101101 is equal to decimal 45 as shown by

(12-2)

$$(1 \cdot 2^5) + (0 \cdot 2^4) + (1 \cdot 2^3) + (1 \cdot 2^2) + (0 \cdot 2^1) + (1 \cdot 2^0) \\ = 32 + 8 + 4 + 1 = 45.$$

Using subscripts to signify the base of a number, we can write [Eq. \(12-2\)](#) as $101101_2 = 45_{10}$. [Equation \(12-2\)](#) shows us that, like decimal numbers, binary numbers use the *place value* system where the position of a digit signifies its weight. If we use B to denote a number system's base, the place value representation of the four-digit number $a_3a_2a_1a_0$ is

$$(12-3)$$

$$(a_3 \cdot B^3) + (a_2 \cdot B^2) + (a_1 \cdot B^1) + (a_0 \cdot B^0).$$

In [Eq. \(12-3\)](#), B^n is the weight multiplier for the digit a_n , where $0 \leq a_n \leq B-1$. (This place value system of representing numbers is very old—so old, in fact, that its origin is obscure. However, with its inherent positioning of the decimal or binary point, this number system is so convenient and powerful that its importance has been compared to that of the alphabet[\[1\]](#).)

12.1.1 Octal Numbers

As the use of minicomputers and microprocessors rapidly expanded in the 1960s, people grew tired of manipulating long strings of ones and zeros on paper and began to use more convenient ways to represent binary numbers. One way to express a binary number is an octal format, with its base of eight. (Of course, the only valid digits in the octal format are 0 to 7—the digits 8 and 9 have no meaning in octal representation.)

Converting from binary to octal is as simple as separating the binary number into three-bit groups starting from the right. For example, the binary number 10101001_2 can be converted to octal format as

$$10101001_2 \rightarrow 10 \mid 101 \mid 001 = 251_8.$$

Thus the octal format enables us to represent an eight-digit binary value with a simpler three-digit octal value. However, the relentless march of technology is pushing octal numbers, like wooden tennis rackets, into extinction.

12.1.2 Hexadecimal Numbers

Today the predominant binary number representation format is the hexadecimal number format using 16 as its base. Converting from binary to hexadecimal is done, this time, by separating the binary number into four-bit groups starting from the right. The binary number 10101001_2 is converted to hexadecimal format as

$$10101001_2 \rightarrow 1010 \mid 1001 = A9_{16}.$$

If you haven't seen the hexadecimal format used before, don't let the $A9$ digits confuse you. In this format, the characters A, B, C, D, E, and F represent the digits whose decimal values are 10, 11, 12, 13, 14, and 15 respectively. We convert the two groups of bits above to two hexadecimal digits by starting with the left group of bits, $1010_2 = 10_{10} = A_{16}$, and $1001_2 = 9_{10} = 9_{16}$. Hexadecimal format numbers also use the place value system, meaning that $A9_{16} = (A \cdot 16^1 + 9 \cdot 16^0)$. For convenience, then, we can represent the eight-digit 10101001_2 with the two-digit number $A9_{16}$. [Table 12-1](#) lists the permissible digit representations in the number systems discussed thus far.

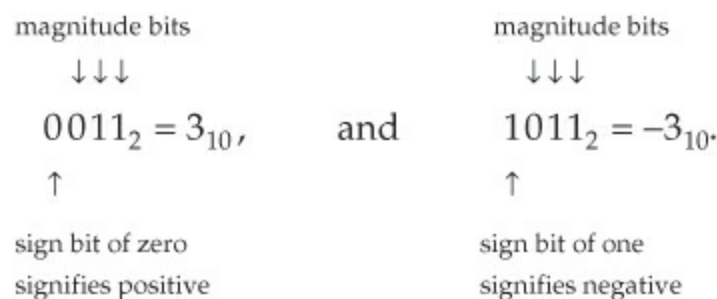
Table 12-1 Allowable Digit Representations versus Number System Base

Binary	Octal	Decimal	Hexadecimal	Decimal equivalent
0	0	0	0	0
1	1	1	1	1
	2	2	2	2
	3	3	3	3
	4	4	4	4
	5	5	5	5
	6	6	6	6
	7	7	7	7
		8	8	8
		9	9	9
			A	10
			B	11
			C	12
			D	13
			E	14
			F	15

In the above example we used a subscripted 16 to signify a hexadecimal number. Note that it's common, in the literature of binary number formats, to have hexadecimal numbers preceded by special characters to signify that indeed they are hexadecimal. You may see, for example, numbers like \$A9 or 0xA9 where the "\$" and "0x" characters specify the follow-on digits to be hexadecimal.

12.1.3 Sign-Magnitude Binary Format

For binary numbers to be at all useful in practice, they must be able to represent negative values. Binary numbers do this by dedicating one of the bits in a binary word to indicate the sign of a number. Let's consider a popular binary format known as *sign magnitude*. Here, we assume that a binary word's leftmost bit is a sign bit and the remaining bits represent the magnitude of a number that is always positive. For example, we can say that the four-bit number 0011_2 is $+3_{10}$ and the binary number 1011_2 is equal to -3_{10} , or



Of course, using one of the bits as a sign bit reduces the magnitude of the numbers we can represent. If an unsigned binary number's word length is b bits, the number of different values that can be represented is 2^b . An eight-bit word, for example, can represent $2^8 = 256$ different integral values. With zero being one of the values we have to express, a b -bit unsigned binary word can represent integers from 0 to $2^b - 1$. The largest value represented by an unsigned eight-bit word is $2^8 - 1 = 255_{10} = 1111111_2$. In the sign-magnitude binary format a b -bit word can represent only a magnitude of $\pm 2^{b-1} - 1$, so the largest positive or negative value we can represent by an eight-bit sign-magnitude word is $\pm 2^{8-1} - 1 = \pm 127$.

12.1.4 Two’s Complement Format

Another common binary number scheme, known as the *two’s complement* format, also uses the leftmost bit as a sign bit. The two’s complement format is the most convenient numbering scheme from a hardware design standpoint and has been used for decades. It enables computers to perform both addition and subtraction using the same hardware adder logic. To obtain the negative version of a positive two’s complement number, we merely complement (change a one to a zero, and change a zero to a one) each bit, add a binary one to the complemented word, and discard any bits carried beyond the original word length. For example, with 0011₂ representing a decimal 3 in two’s complement format, we obtain a negative decimal 3 through the following steps:

+3 in two’s complement →0 0 1 1

complement of +3 →1 1 0 0

add one →+ 0 0 0 1

−3 in two’s complement →1 1 0 1.

In the two’s complement format, a *b*-bit word can represent positive amplitudes as great as $2^{b-1}-1$, and negative amplitudes as large as -2^{b-1} . [Table 12-2](#) shows four-bit word examples of sign-magnitude and two’s complement binary formats.

Table 12-2 Integer Binary Number Formats

Decimal equivalent	Sign-magnitude	Two’s complement	Offset binary
7	0111	0111	1111
6	0110	0110	1110
5	0101	0101	1101
4	0100	0100	1100
3	0011	0011	1011
2	0010	0010	1010
1	0001	0001	1001
+0	0000	0000	1000
−0	1000	—	—
−1	1001	1111	0111
−2	1010	1110	0110
−3	1011	1101	0101
−4	1100	1100	0100
−5	1101	1011	0011
−6	1110	1010	0010
−7	1111	1001	0001
−8	—	1000	0000

While using two’s complement numbers, we have to be careful when adding two numbers of different word lengths. Consider the case where a four-bit number is added to an eight-bit number:

+15 in two’s complement →0 0 0 0 1 1 1 1

add +3 in two’s complement →+ 0 0 1 1

+18 in two’s complement →0 0 0 1 0 0 1 0.

No problem so far. The trouble occurs when our four-bit number is negative. Instead of adding a +3 to the +15, let’s try to add a −3 to the +15:

$$\begin{array}{rcl}
 +15 \text{ in two's complement} & \rightarrow & 00001111 \\
 \text{add a } -3 \text{ in two's complement} & \rightarrow & + \underline{1101} \\
 +28 \text{ in two's complement} & \rightarrow & 00011100. \quad \leftarrow \text{Wrong answer}
 \end{array}$$

The above arithmetic error can be avoided by performing what's called a *sign-extend* operation on the four-bit number. This process, typically performed automatically in hardware, extends the sign bit of the four-bit negative number to the left, making it an eight-bit negative number. If we sign-extend the -3 and then perform the addition, we'll get the correct answer:

$$\begin{array}{rcl}
 +15 \text{ in two's complement} & \rightarrow & 00001111 \\
 \text{add a sign-extended } -3 \text{ in two's complement} & \rightarrow & + \underline{11111101} \\
 +12 \text{ in two's complement} & \rightarrow & 100001100. \quad \leftarrow \text{That's better} \\
 & \uparrow & \\
 & \text{overflow bit is ignored} &
 \end{array}$$

12.1.5 Offset Binary Format

Another useful binary number scheme is known as the *offset binary* format. While this format is not as common as two's complement, it still shows up in some hardware devices. [Table 12-2](#) shows offset binary format examples for four-bit words. Offset binary represents numbers by subtracting 2^{b-1} from an unsigned binary value. For example, in the second row of [Table 12-2](#), the offset binary number is 1110_2 . When this number is treated as an unsigned binary number, it's equivalent to 14_{10} . For four-bit words $b = 4$ and $2^{b-1} = 8$, so $14_{10} - 8_{10} = 6_{10}$, which is the decimal equivalent of 1110_2 in offset binary. The difference between the unsigned binary equivalent and the actual decimal equivalent of the offset binary numbers in [Table 12-2](#) is always -8 . This kind of offset is sometimes referred to as a *bias* when the offset binary format is used. (It may interest the reader that we can convert back and forth between the two's complement and offset binary formats merely by complementing a word's most significant bit.)

The history, arithmetic, and utility of the many available number formats is a very broad field of study. A thorough and very readable discussion of the subject is given by Knuth in reference [\[2\]](#).

12.1.6 Fractional Binary Numbers

All of the binary numbers we've considered so far had integer decimal values. Noninteger decimal numbers, numbers with nonzero digits to the right of the decimal point, can also be represented with binary numbers if we use a *binary point*, also called a *radix point*, identical in function to our familiar decimal point. (As such, in the binary numbers we've discussed so far, the binary point is assumed to be fixed just to the right of the rightmost, lsb, bit.) For example, using the symbol \diamond to denote a binary point, the six-bit unsigned binary number $11\diamond 0101_2$ is equal to decimal 3.3125 as shown by

$$\begin{aligned}
 & (12-4) \\
 & (1 \cdot 2^1) + (1 \cdot 2^0) + (0 \cdot 2^{-1}) + (1 \cdot 2^{-2}) + (0 \cdot 2^{-3}) + (1 \cdot 2^{-4}) \\
 & = (1 \cdot 2) + (1 \cdot 1) + \left(0 \cdot \frac{1}{2}\right) + \left(1 \cdot \frac{1}{4}\right) + \left(0 \cdot \frac{1}{8}\right) + \left(1 \cdot \frac{1}{16}\right) \\
 & = 2 + 1 + 0 + 0.25 + 0 + 0.0625 = 3.3125_{10}.
 \end{aligned}$$

For our $11\diamond 0101_2$ example in [Eq. \(12-4\)](#) the binary point is set between the second and third most significant bits and we call that binary number a *fractional* number. Having a stationary position for

the binary point is why this binary number format is called *fixed-point binary*. The unsigned number 11_00101_2 has two integer bits and four fractional bits, so, in the parlance of binary numbers, such a number is said to have a 2.4, “two dot four,” format (two integer bits and four fractional bits).

Two’s complement binary numbers can also have this *integer plus fraction* format, and [Table 12-3](#) shows, for example, the decimal value ranges for all possible eight-bit two’s complement fractional binary numbers. Notice how the 8.0-format row in [Table 12-3](#) shows the decimal values associated with an eight-bit two’s complement binary number whose binary point is to the right of the lsb, signifying an all-integer binary number. On the other hand, the 1.7-format row in [Table 12-3](#) shows the decimal values associated with an eight-bit two’s complement binary number whose binary point is just to the right of the msb (the sign bit), signifying an all-fraction binary number.

Table 12-3 Eight-Bit, Two’s Complement, Fractional Format Values

Fract. binary format	Number of integer bits (including sign bit)	Number of fractional bits	Maximum positive decimal value	Maximum negative decimal value	Lsb decimal value
8.0	8	0	127.0	−128.0	1.0
7.1	7	1	63.5	−64.0	0.5
6.2	6	2	31.75	−32.0	0.25
5.3	5	3	15.875	−16.0	0.125
4.4	4	4	7.9375	−8.0	0.0625
3.5	3	5	3.96875	−4.0	0.03125
2.6	2	6	1.984375	−2.0	0.015625
1.7	1	7	0.9921875	−1.0	0.0078125

The decimal value range of a general fractional two’s complement binary number is

(12-5)

$$-2^{(\# \text{ of integer bits} - 1)} \leq \text{decimal value} \leq 2^{(\# \text{ of integer bits} - 1)} - 2^{-(\# \text{ of fraction bits})},$$

where the “# of integer bits” notation means the number of bits to the left of the binary point and “# of fraction bits” means the number of bits to the right of the binary point.

[Table 12-3](#) teaches us two important lessons. First, we can place the implied binary point anywhere we wish in the eight-bit word, just so long as everyone accessing the data agrees on that binary point placement and the designer keeps track of that placement throughout all of the system’s arithmetic computations. Binary arithmetic hardware behavior does not depend on the “agreed upon” binary point placement. Stated in different words, binary point placement does not affect two’s complement binary arithmetic operations. That is, adding or multiplying two binary numbers will yield the same binary result regardless of the implied binary point location within the data words. We leave an example of this behavior as a homework problem.

Second, for a fixed number of bits, fractional two’s complement binary numbers allow us to represent decimal numbers with poor precision over a wide range of values, or we can represent decimal numbers with fine precision but only over a narrow range of values. In practice you must “pick your poison” by choosing the position of the binary point based on what’s more important to you, number range or number precision.

Due to their 16-bit internal data paths, it's very common for programmable 16-bit DSP chips to use a 1.15 format (one integer bit to represent sign, and 15 fractional bits) to represent two's complement numbers. These 16-bit signed *all-fraction* binary numbers are particularly useful because multiplying two such numbers results in an *all-fraction* product, avoiding any unpleasant binary *overflow* problems, to be discussed shortly. (Be aware that this 1.15 format is also called *Q15 format*.) Because the 1.15-format is so commonly used in programmable hardware, we give examples of it and other 16-bit formats in [Table 12-4](#). In that table, the “resolution” is the decimal value of the format's lsb.

Table 12-4 16-Bit Format Values

16-bit binary format	Decimal value	Binary	Hex
Two's complement	0.9999694824...	0 _{111 1111 1111 1111}	7FFF
1.15 (Q15) format	0.5	0 _{100 0000 0000 0000}	4000
	0.25	0 _{010 0000 0000 0000}	2000
resolution →	0.0000305175...	0 _{000 0000 0000 0001}	0001
	0	0 _{000 0000 0000 0000}	0000
	−0.0000305175...	1 _{111 1111 1111 1111}	FFFF
	−0.25	1 _{110 0000 0000 0000}	E000
	−0.5	1 _{000 0000 0000 0000}	C000
	−1.0	1 _{000 0000 0000 0000}	8000
Standard two's complement integer	32767	0111 1111 1111 111 ₁	7FFF
(16.0 format)	16384	0100 0000 0000 000 ₀	4000
	8192	0010 0000 0000 000 ₀	2000
resolution →	1	0000 0000 0000 000 ₁	0001
	0	0000 0000 0000 000 ₀	0000
	−1	1111 1111 1111 111 ₁	FFFF
	−8192	1110 0000 0000 000 ₀	E000
	−16384	1100 0000 0000 000 ₀	C000
	−32767	1000 0000 0000 000 ₁	8001
	−32768	1000 0000 0000 000 ₀	8000
Unsigned integer	65535	1111 1111 1111 1111	FFFF
	32768	1000 0000 0000 0000	8000
	32767	0111 1111 1111 1111	7FFF
resolution →	1	0000 0000 0000 0001	0001
	0	0000 0000 0000 0000	0000

Multiplication of two 1.15 binary words results in a 2.30-format (also called a *Q30-format*) fractional number. That 32-bit product word contains two sign bits and 30 fractional bits, with the msb being called an extended sign bit. We have two ways to convert (truncate) such a 32-bit product to the 1.15 format so that it can be stored as a 16-bit word. They are

- shifting the 32-bit word left by one bit and storing the upper 16 bits, and
- shifting the 32-bit word right by 15 bits and storing the lower 16 bits.

To conclude this fractional binary discussion, we provide the steps to convert a decimal number whose magnitude is less than one, such as an FIR digital filter coefficient, to the 1.15 binary format. As an example, to convert the decimal value 0.452 to the two's complement 1.15 binary format:

1. Multiply the absolute value of the original decimal number 0.452 by 32768 (2^{15}), yielding a scaled decimal 14811.136.

2. Round the value 14811.136 to an integer, using your preferred rounding method, producing a scaled decimal value of 14811.
3. Convert the decimal 14811 to a binary integer and place the binary point to the right of the msb, yielding 0₀011 1001 1101 1011 (39DB₁₆).
4. If the original decimal value was positive, stop now. If the original decimal value was negative, implement a two's complement conversion by inverting Step 3's binary bits and add one.

If you, unfortunately, do not have software to perform the above positive decimal integer to 1.15 binary conversion in Step 3, here's how the conversion can be done (painfully) by hand:

- 3.1. Divide 14811 by 2, obtaining integer 7405 plus a remainder of 0.5. Because the remainder is not zero, place a one as the lsb of the desired binary number. Our binary number is 1.
- 3.2. Divide 7405 by 2, obtaining integer 3702 plus a remainder of 0.5. Because the remainder is not zero, place a one as the bit to the left of the lsb bit established in Step 3.1 above. Our binary number is now 11.
- 3.3. Divide 3702 by 2, obtaining integer 1851 plus a remainder of zero. Because the remainder is zero, place a zero as the bit to the left of the bit established in Step 3.2 above. Our binary number is now 011.
- 3.4. Continue this process until the integer portion of the divide-by-two quotient is zero. Append zeros to the left of the binary word to extend its length to 16 bits.

Using the above steps to convert decimal 14811₁₀ to binary 1.15 format proceeds as shown in [Table 12-5](#), producing our desired binary number of 0₀011 1001 1101 1011 (39DB₁₆).

Table 12-5 Decimal 14811 to Binary 1.15 Conversion Example

Operation	Binary word
14811 ÷ 2 = 7405 + remainder of 0.5 →	1
7405 ÷ 2 = 3702 + remainder of 0.5 →	11
3702 ÷ 2 = 1851 + remainder of 0 →	011
1851 ÷ 2 = 925 + remainder of 0.5 →	1011
925 ÷ 2 = 462 + remainder of 0.5 →	1 1011
462 ÷ 2 = 231 + remainder of 0 →	01 1011
231 ÷ 2 = 115 + remainder of 0.5 →	101 1011
115 ÷ 2 = 57 + remainder of 0.5 →	1101 1011
57 ÷ 2 = 28 + remainder of 0.5 →	1 1101 1011
28 ÷ 2 = 14 + remainder of 0 →	01 1101 1011
14 ÷ 2 = 7 + remainder of 0 →	001 1101 1011
7 ÷ 2 = 3 + remainder of 0.5 →	1001 1101 1011
3 ÷ 2 = 1 + remainder of 0.5 →	1 1001 1101 1011
1 ÷ 2 = 0 + remainder of 0.5 →	11 1001 1101 1011
Append two msb zeros (We're done!)	0 ₀ 011 1001 1101 1011

12.2 Binary Number Precision and Dynamic Range

As we implied earlier, for any binary number format, the number of bits in a data word is a key consideration. The more bits used in the word, the better the resolution of the number, and the larger the maximum value that can be represented.[‡] Assuming that a binary word represents the amplitude of

a signal, digital signal processing practitioners find it useful to quantify the dynamic range of various binary number schemes. For a signed integer binary word length of $b+1$ bits (one sign bit and b magnitude bits), the dynamic range is defined by

† Some computers use 64-bit words. Now, 2^{64} is approximately equal to $1.8 \cdot 10^{19}$ —that’s a pretty large number. So large, in fact, that if we started incrementing a 64-bit counter once per second at the beginning of the universe (≈ 20 billion years ago), the most significant four bits of this counter would *still* be all zeros today.

$$\begin{aligned} & \text{dynamic range}_{\text{linear}} = \frac{\text{largest positive word value}}{\text{smallest positive word value}} \\ & = \frac{2^b - 1}{1} = 2^b - 1. \end{aligned} \tag{12-6}$$

The dynamic range measured in dB is

$$\begin{aligned} \text{dynamic range}_{\text{dB}} &= 20 \cdot \log_{10}(\text{dynamic range}_{\text{linear}}) \\ &= 20 \cdot \log_{10}(2^b - 1). \end{aligned} \tag{12-6'}$$

When 2^b is much larger than 1, we can ignore the -1 in [Eq. \(12-6'\)](#) and state that

$$\begin{aligned} \text{dynamic range}_{\text{dB}} &= 20 \cdot \log_{10}(2^b) \\ &= 20 \cdot \log_{10}(2) \cdot b = 6.02 \cdot b \text{ dB}. \end{aligned} \tag{12-6''}$$

[Equation \(12-6''\)](#), dimensioned in dB, tells us that the dynamic range of our number system is directly proportional to the word length. Thus, an eight-bit two’s complement word, with seven bits available to represent signal magnitude, has a dynamic range of $6.02 \cdot 7 = 42.14$ dB. Most people simplify [Eq. \(12-6''\)](#) by using the rule of thumb that the dynamic range is equal to “6 dB per bit.”

12.3 Effects of Finite Fixed-Point Binary Word Length

The effects of finite binary word lengths touch all aspects of digital signal processing. Using finite word lengths prevents us from representing values with infinite precision, increases the background noise in our spectral estimation techniques, creates nonideal digital filter responses, induces noise in analog-to-digital (A/D) converter outputs, and can (if we’re not careful) lead to wildly inaccurate arithmetic results. The smaller the word lengths, the greater these problems will be. Fortunately, these finite, word-length effects are rather well understood. We can predict their consequences and take steps to minimize any unpleasant surprises. The first finite, word-length effect we’ll cover is the errors that occur during the A/D conversion process.

12.3.1 A/D Converter Quantization Errors

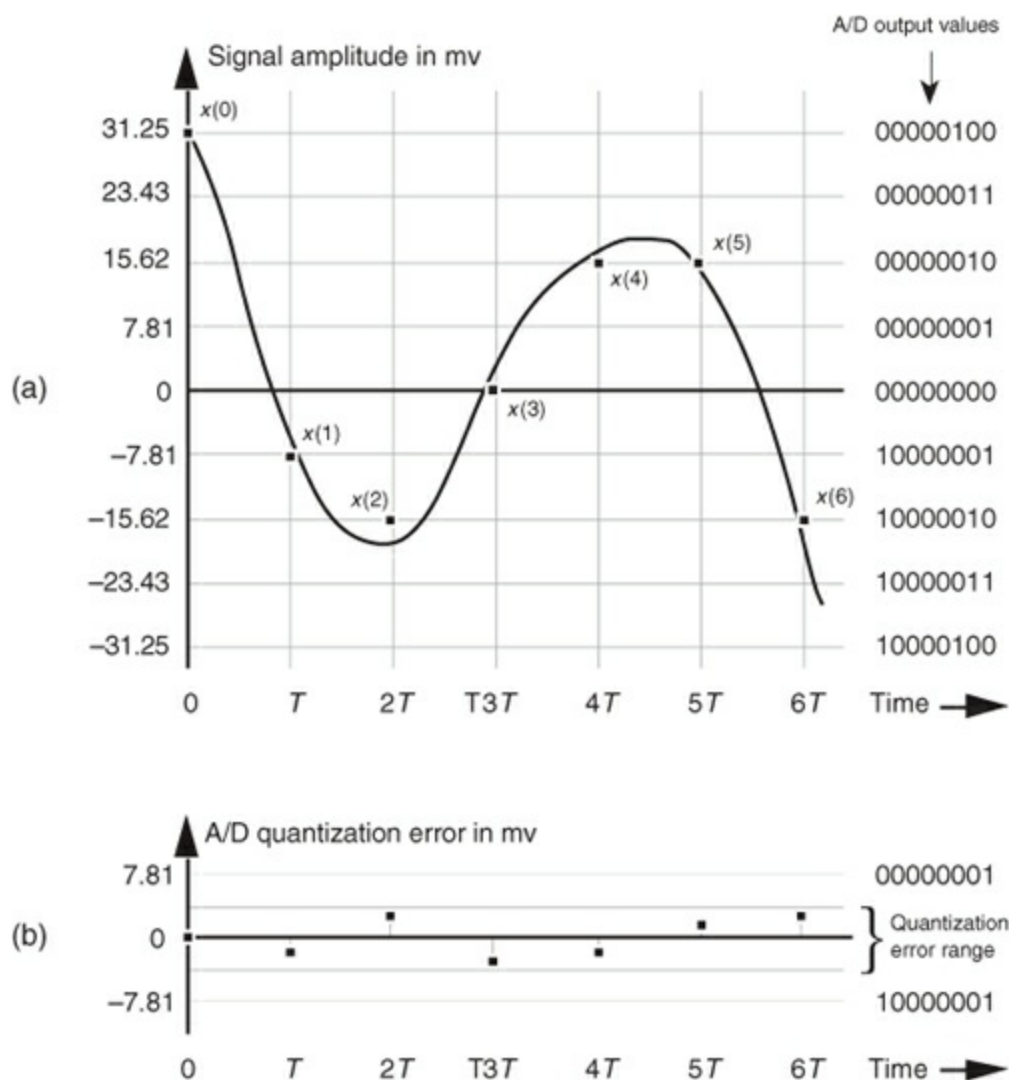
Practical A/D converters are constrained to have binary output words of finite length. Commercial A/D converters are categorized by their output word lengths, which are normally in the range from 8 to 16 bits. A typical A/D converter input analog voltage range is from -1 to $+1$ volt. If we used such an A/D converter having 8-bit output words, the least significant bit would represent

$$\tag{12-7}$$

$$\text{lsb value} = \frac{\text{full voltage range}}{2^{\text{word length}}} = \frac{2 \text{ volts}}{2^8} = 7.81 \text{ millivolts.}$$

What this means is that we can represent continuous (analog) voltages perfectly as long as they're integral multiples of 7.81 millivolts—any intermediate input voltage will cause the A/D converter to output a *best estimate* digital data value. The inaccuracies in this process are called *quantization errors* because an A/D output least significant bit is an indivisible quantity. We illustrate this situation in [Figure 12-1\(a\)](#), where the continuous waveform is being digitized by an 8-bit A/D converter whose output is in the sign-magnitude format. When we start sampling at time $t = 0$, the continuous waveform happens to have a value of 31.25 millivolts (mv), and our A/D output data word will be exactly correct for sample $x(0)$. At time T when we get the second A/D output word for sample $x(1)$, the continuous voltage is between 0 and -7.81 mv. In this case, the A/D converter outputs a sample value of 10000001, representing -7.81 mv, even though the continuous input was not quite as negative as -7.81 mv. The 10000001 A/D output word contains some quantization error. Each successive sample contains quantization error because the A/D's digitized output values must lie on the horizontal line in [Figure 12-1\(a\)](#). The difference between the actual continuous input voltage and the A/D converter's representation of the input is shown as the quantization error in [Figure 12-1\(b\)](#). For an ideal A/D converter, the quantization error, a kind of *roundoff* noise, can never be greater than $\pm 1/2$ an lsb, or ± 3.905 mv.

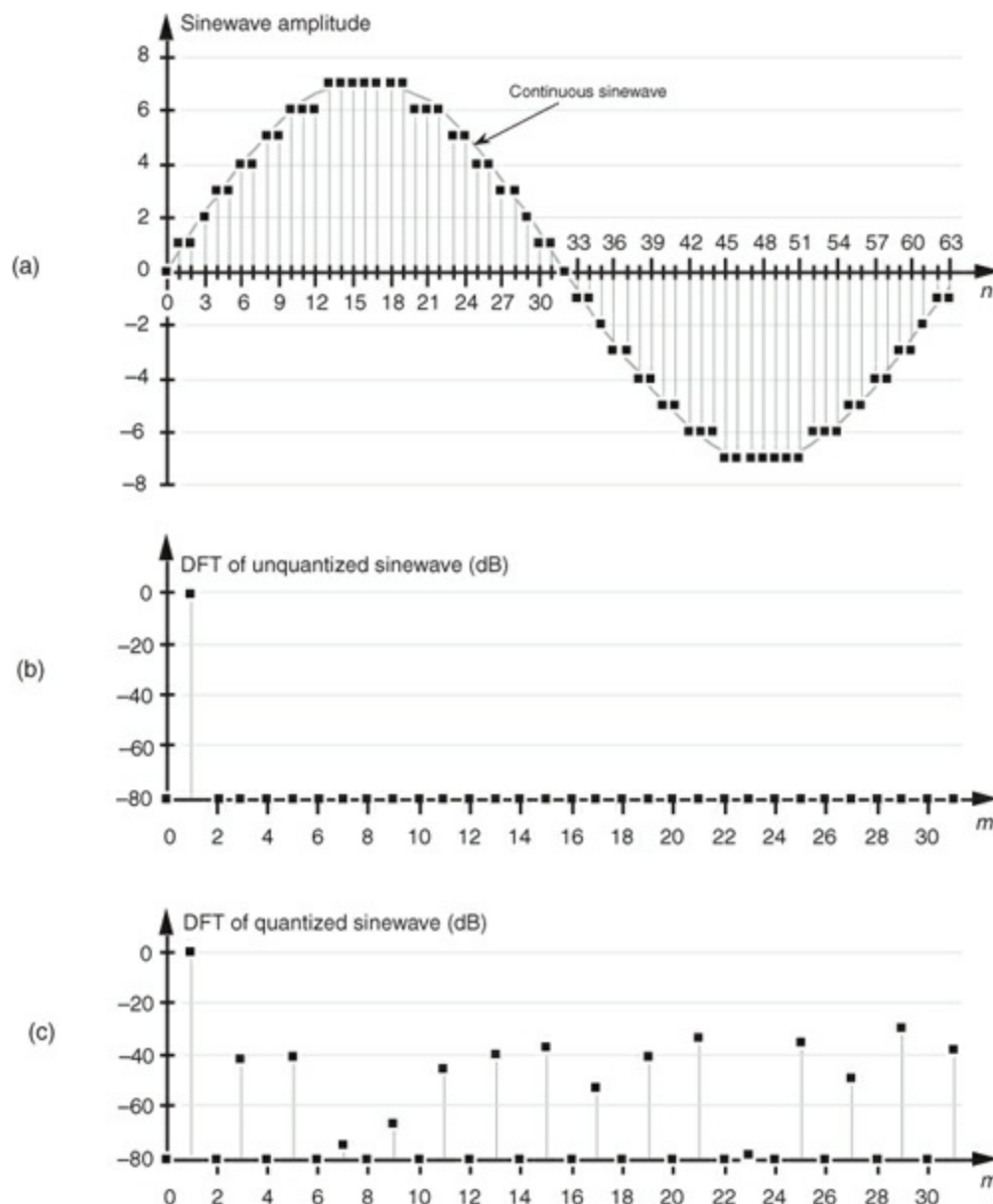
Figure 12-1 Quantization errors: (a) digitized $x(n)$ values of a continuous signal; (b) quantization error between the actual analog signal values and the digitized signal values.



While [Figure 12-1\(b\)](#) shows A/D quantization noise in the time domain, we can also illustrate this

noise in the frequency domain. [Figure 12-2\(a\)](#) depicts a continuous sinewave of one cycle over the sample interval shown as the dashed line and a quantized version of the time-domain samples of that wave as the dots. Notice how the quantized version of the wave is constrained to have only integral values, giving it a *stair-step* effect oscillating above and below the true unquantized sinewave. The quantization here is four bits, meaning that we have a sign bit and three bits to represent the magnitude of the wave. With three bits, the maximum peak values for the wave are ± 7 . [Figure 12-2\(b\)](#) shows the discrete Fourier transform (DFT) of a discrete version of the sinewave whose time-domain sample values are not forced to be integers but have high precision. Notice in this case that the DFT has a nonzero value only at $m = 1$. On the other hand, [Figure 12-2\(c\)](#) shows the spectrum of the four-bit quantized samples in [Figure 12-2\(a\)](#), where quantization effects have induced noise components across the entire spectral band. If the quantization noise depictions in [Figures 12-1\(b\)](#) and [12-2\(c\)](#) look random, that's because they are. As it turns out, even though A/D quantization noise is random, we can still quantify its effects in a useful way.

Figure 12-2 Quantization noise effects: (a) input sinewave applied to a 64-point DFT; (b) theoretical DFT magnitude of high-precision sinewave samples; (c) DFT magnitude of a sinewave quantized to four bits.



In the field of communications, people often use the notion of output signal-to-noise ratio, or $\text{SNR} = (\text{signal power})/(\text{noise power})$, to judge the usefulness of a process or device. We can do likewise and

obtain an important expression for the output SNR of an ideal A/D converter, $SNR_{A/D}$, accounting for finite word-length quantization effects. Because quantization noise is random, we can't explicitly represent its power level, but we can use its statistical equivalent of variance to define $SNR_{A/D}$ measured in dB as

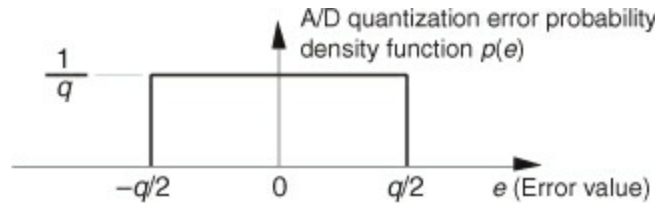
(12-8)

$$SNR_{A/D} = 10 \cdot \log_{10} \left(\frac{\text{input signal variance}}{\text{A/D quantization noise variance}} \right)$$

$$= 10 \cdot \log_{10} \left(\frac{\sigma^2_{\text{signal}}}{\sigma^2_{\text{A/D noise}}} \right).$$

Next, we'll determine an A/D converter's quantization noise variance relative to the converter's maximum input peak voltage V_p . If the full-scale ($-V_p$ to $+V_p$ volts) continuous input range of a b -bit A/D converter is $2V_p$, a single quantization level q is that voltage range divided by the number of possible A/D output binary values, or $q = 2V_p/2^b$. (In [Figure 12-1](#), for example, the quantization level q is the lsb value of 7.81 mv.) A depiction of the likelihood of encountering any given quantization error value, called the probability density function $p(e)$ of the quantization error, is shown in [Figure 12-3](#).

Figure 12-3 Probability density function of A/D conversion roundoff error (noise).



This simple rectangular function has much to tell us. It indicates that there's an equal chance that any error value between $-q/2$ and $+q/2$ can occur. By definition, because probability density functions have an area of unity (i.e., the probability is 100 percent that the error will be somewhere under the curve), the amplitude of the $p(e)$ density function must be the area divided by the width, or $p(e) = 1/q$. From [Figure D-7](#) and [Eq. \(D-29\)](#) in [Appendix D](#), the variance of our uniform $p(e)$ is

(12-9)

$$\sigma^2_{\text{A/D noise}} = \int_{-q/2}^{q/2} e^2 p(e) de = \frac{1}{q} \cdot \int_{-q/2}^{q/2} e^2 de = \frac{q^2}{12}.$$

We can now express the A/D noise error variance in terms of A/D parameters by replacing q in [Eq. \(12-9\)](#) with $q = 2V_p/2^b$ to get

(12-10)

$$\sigma^2_{\text{A/D noise}} = \frac{(2V_p)^2}{12 \cdot (2^b)^2} = \frac{V_p^2}{3 \cdot 2^{2b}}.$$

OK, we're halfway to our goal—with [Eq. \(12-10\)](#) giving us the denominator of [Eq. \(12-8\)](#), we need the numerator. To arrive at a general result, let's express the input signal in terms of its root mean

square (rms), the A/D converter's peak voltage, and a loading factor LF defined as

(12-11)

$$LF = \frac{\text{rms of the input signal}}{V_p} = \frac{\sigma_{\text{signal}}}{V_p}.$$

† As covered in [Appendix D, Section D.2](#), although the variance σ^2 is associated with the power of a signal, the standard deviation is associated with the rms value of a signal.

With the loading factor defined as the input rms voltage over the A/D converter's peak input voltage, we square and rearrange [Eq. \(12-11\)](#) to show the signal variance σ_{signal}^2 as

(12-12)

$$\sigma_{\text{signal}}^2 = (LF)^2 V_p^2.$$

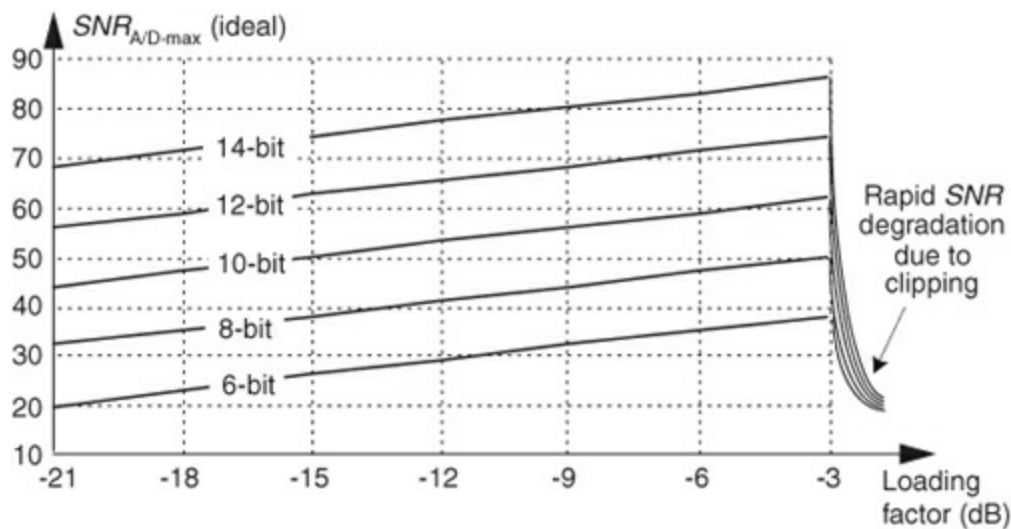
Substituting [Eqs. \(12-10\)](#) and [\(12-12\)](#) in [Eq. \(12-8\)](#),

(12-13)

$$\begin{aligned} \text{SNR}_{\text{A/D}} &= 10 \cdot \log_{10} \left(\frac{(LF)^2 V_p^2}{V_p^2 / (3 \cdot 2^{2b})} \right) = 10 \cdot \log_{10} [(LF)^2 (3 \cdot 2^{2b})] \\ &= 6.02 \cdot b + 4.77 + 20 \log_{10}(LF). \end{aligned}$$

[Eq. \(12-13\)](#) gives us the $\text{SNR}_{\text{A/D}}$ of an ideal b -bit A/D converter in terms of the loading factor and the number of bits b . [Figure 12-4](#) plots [Eq. \(12-13\)](#) for various A/D word lengths as a function of the loading factor. Notice that the loading factor in [Figure 12-4](#) is never greater than -3 dB, because the maximum continuous A/D input peak value must not be greater than V_p volts. Thus, for a sinusoid input, its rms value must not be greater than $V_p/\sqrt{2}$ volts (3 dB below V_p).

Figure 12-4 $\text{SNR}_{\text{A/D}}$ of ideal A/D converters as a function of loading factor in dB.



When the input sinewave's peak amplitude is equal to the A/D converter's full-scale voltage V_p , the full-scale LF is

(12-14)

$$LF_{\text{full scale}} = \frac{V_p / \sqrt{2}}{V_p} = \frac{1}{\sqrt{2}}.$$

Under this condition, the maximum A/D output SNR from [Eq. \(12-13\)](#) is

$$(12-15)$$

$$\begin{aligned} SNR_{A/D-\text{max}} &= 6.02 \cdot b + 4.77 + 20 \cdot \log_{10}(1/\sqrt{2}) \\ &= 6.02 \cdot b + 4.77 - 3.01 = 6.02 \cdot b + 1.76 \text{ dB.} \end{aligned}$$

This discussion of SNR relative to A/D converters means three important things to us:

1. An ideal A/D converter will have an $SNR_{A/D}$ defined by [Eq. \(12-13\)](#), so any discrete $x(n)$ signal produced by a b -bit A/D converter can never have an SNR greater than [Eq. \(12-13\)](#). ([Appendix D](#) dicusses methods for computing the SNR of discrete signals.) For example, let's say we want to digitize a continuous signal whose SNR is 55 dB. Using an ideal eight-bit A/D converter with its full-scale $SNR_{A/D}$ of $6.02 \cdot 8 + 1.76 = 49.9$ dB from [Eq. \(12-15\)](#), the quantization noise will contaminate the digitized values, and the resultant digital signal's SNR can be no better than 49.9 dB. We'll have lost signal SNR through the A/D conversion process. (A ten-bit A/D, with its ideal $SNR_{A/D} \approx 62$ dB, could be used to digitize a 55 dB SNR continuous signal to reduce the SNR degradation caused by quantization noise.) [Equations \(12-13\)](#) and [\(12-15\)](#) apply to ideal A/D converters and don't take into account such additional A/D noise sources as aperture jitter error, missing output bit patterns, and other nonlinearities. So actual A/D converters are likely to have SNRs that are lower than that indicated by theoretical [Eq. \(12-13\)](#). To be safe in practice, it's sensible to assume that $SNR_{A/D-\text{max}}$ is 3 to 6 dB lower than indicated by [Eq. \(12-15\)](#).
2. [Equation \(12-15\)](#) is often expressed in the literature, but it can be a little misleading because it's imprudent to force an A/D converter's input to full scale. It's wise to drive an A/D converter to some level below full scale because inadvertent overdriving will lead to signal clipping and will induce distortion in the A/D's output. So [Eq. \(12-15\)](#) is overly optimistic, and, in practice, A/D converter SNRs will be less than indicated by [Eq. \(12-15\)](#). The best approximation for an A/D's SNR is to determine the input signal's rms value that will never (or rarely) overdrive the converter input, and plug that value into [Eq. \(12-11\)](#) to get the loading factor value for use in [Eq. \(12-13\)](#).[‡] Again, using an A/D converter with a wider word length will alleviate this problem by increasing the available $SNR_{A/D}$.

[‡] By the way, some folks use the term *crest factor* to describe how hard an A/D converter's input is being driven. The crest factor is the reciprocal of the loading factor, or $CF = V_p/(\text{rms of the input signal})$.

3. Remember, now, real-world continuous signals always have their own inherent continuous SNR, so using an A/D converter whose $SNR_{A/D}$ is a great deal larger than the continuous signal's SNR serves no purpose. In this case, we would be wasting A/D converter bits by digitizing the analog signal's noise to a high degree of accuracy, which does not improve our digital signal's overall SNR. In general, we want the converter's $SNR_{A/D}$ value to be approximately 6 dB greater than an analog signal's SNR.

A word of caution is appropriate here concerning our analysis of A/D converter quantization errors. The derivations of [Eqs. \(12-13\)](#) and [\(12-15\)](#) are based upon three assumptions:

1. The cause of A/D quantization errors is a stationary random process; that is, the performance of the A/D converter does not change over time. Given the same continuous input voltage, we always expect an A/D converter to provide exactly the same output binary code.
2. The probability density function of the A/D quantization error is uniform. We're assuming that the A/D converter is ideal in its operation and all possible errors between $-q/2$ and $+q/2$ are equally likely. An A/D converter having stuck bits or missing output codes would violate this assumption. High-quality A/D converters being driven by continuous signals that cross many quantization levels will result in our desired uniform quantization noise probability density function.
3. The A/D quantization errors are uncorrelated with the continuous input signal. If we were to digitize a single continuous sinewave whose frequency was harmonically related to the A/D sample rate, we'd end up sampling the same input voltage repeatedly and the quantization error sequence would not be random. The quantization error would be predictable and repetitive, and our quantization noise variance derivation would be invalid. In practice, complicated continuous signals such as music or speech, with their rich spectral content, avoid this problem.

To conclude our discussion of A/D converters, let's consider one last topic. In the literature the reader is likely to encounter the expression

(12-16)

$$b_{\text{eff}} = \frac{\text{SNR} - 1.76}{6.02}.$$

[Equation \(12-16\)](#) is used by test equipment manufacturers to specify the sensitivity of test instruments using a b_{eff} parameter known as the number of *effective bits*, or effective number of bits (ENOB) [3–8]. [Equation \(12-16\)](#) is merely [Eq. \(12-15\)](#) solved for b and is based on the assumption that the A/D converter's analog input peak-peak voltage spans roughly 90 percent of the converter's full-scale voltage range. Test equipment manufacturers measure the actual SNR of their product, indicating its ability to capture continuous input signals relative to the instrument's inherent noise characteristics. Given this true SNR, they use [Eq. \(12-16\)](#) to determine the b_{eff} value for advertisement in their product literature. The larger the b_{eff} , the greater the continuous voltage that can be accurately digitized relative to the equipment's intrinsic quantization noise.

12.3.2 Data Overflow

The next finite, word-length effect we'll consider is called *overflow*. Overflow is what happens when the result of an arithmetic operation has too many bits, or digits, to be represented in the hardware registers designed to contain that result. We can demonstrate this situation to ourselves rather easily using a simple four-function, eight-digit pocket calculator. The sum of a decimal 9.999999 plus 1.0 is 10.999999, but on an eight-digit calculator the sum is 10.999999 as

$$\begin{array}{r} 9.999999 \\ + 1.000000 \\ \hline 10.999999. \end{array}$$

↑
this digit gets discarded

The hardware registers, which contain the arithmetic result and drive the calculator's display, can hold only eight decimal digits; so the least significant digit is discarded (of course). Although the above error is less than one part in ten million, overflow effects can be striking when we work with

large numbers. If we use our calculator to add 99,999,999 plus 1, instead of getting the correct result of 100 million, we'll get a result of 1. Now that's an authentic overflow error!

Let's illustrate overflow effects with examples more closely related to our discussion of binary number formats. First, adding two unsigned binary numbers is as straightforward as adding two decimal numbers. The sum of 42 plus 39 is 81, or

+42 in unsigned binary →

+39 in unsigned binary →

+81 in unsigned binary →

1 0 1 0 1 0

+ 1 0 0 1 1 1

1 0 1 0 0 0 1.

← carry bits

In this case, two 6-bit binary numbers required 7 bits to represent the results. The general rule is *the sum of m individual b-bit binary numbers can require as many as [b + log₂(m)] bits to represent the results*. So, for example, a 24-bit result register (accumulator) is needed to accumulate the sum of sixteen 20-bit binary numbers, or 20 + log₂(16) = 24. The sum of 256 eight-bit words requires an accumulator whose word length is [8 + log₂(256)], or 16 bits, to ensure that no overflow errors occur.

In the preceding example, if our accumulator word length was six bits, an overflow error occurs as

+42 in unsigned binary →

+39 in unsigned binary →

+17 in unsigned binary →

1 0 1 0 1 0

+ 1 0 0 1 1 1

1 0 1 0 0 0 1.

← overflow error

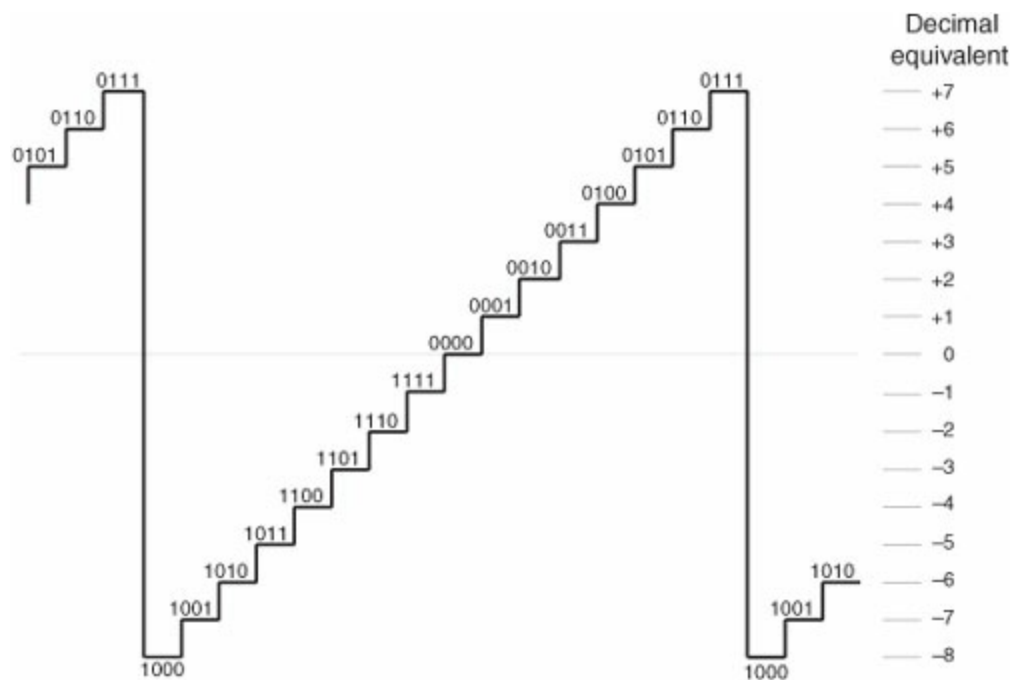
↑

an overflow out of the sign bit is ignored, causing an overflow error

Here, the most significant bit of the result overflowed the six-bit accumulator, and an error occurred.

With regard to overflow errors, the two's complement binary format has two interesting characteristics. First, under certain conditions, overflow during the summation of two numbers causes no error. Second, with multiple summations, intermediate overflow errors cause no problems if the final magnitude of the sum of the *b*-bit two's complement numbers is less than 2^{*b*-1}. Let's illustrate these properties by considering the four-bit two's complement format in [Figure 12-5](#), whose binary values are taken from [Table 12-2](#).

Figure 12-5 Four-bit two's complement binary numbers.



The first property of two's complement overflow, which sometimes causes no errors, can be shown by the following examples:

	0 1 0	← carry bits
-5 in two's complement →	1 0 1 1	
+2 in two's complement →	+ 0 0 1 0	
-3 in two's complement →	0 1 1 0 1	← valid negative result
	↑	
zero overflow out of the sign bit		

	1 1 0	← carry bits
-2 in two's complement →	1 1 1 0	
+6 in two's complement →	+ 0 1 1 0	
+4 in two's complement →	1 0 1 0 0	← valid positive result
	↑	
overflow out of the sign bit ignored, no harm done		

Then again, the following examples show how two's complement overflow sometimes does cause errors:

	0 0 0	← carry bits
-7 in two's complement →	1 0 0 1	
-6 in two's complement →	+ 1 0 1 0	
+3 in two's complement →	1 0 0 1 1	← invalid positive result
	↑	
overflow out of the sign bit ignored, causing overflow error		

	1 1 1	← carry bits
+7 in two's complement →	0 1 1 1	
+7 in two's complement →	0 1 1 1	
-2 in two's complement →	0 1 1 1 0	← invalid negative result
	↑	
zero overflow out of the sign bit		

The rule with two's complement addition is *if the carry bit into the sign bit is the same as the overflow bit out of the sign bit, the overflow bit can be ignored, causing no errors; if the carry bit into the sign bit is different from the overflow bit out of the sign bit, the result is invalid*. An even more interesting property of two's complement numbers is that a series of b -bit word summations can

be performed where intermediate sums are invalid, but the final sum will be correct if its magnitude is less than 2^{b-1} . We show this by the following example. If we add a +6 to a +7, and then add a -7, we'll encounter an intermediate overflow error but our final sum will be correct, as

+7 in two's complement →	0 1 1 1	
+6 in two's complement →	+ 0 1 1 0	
-3 in two's complement →	1 1 0 1	← overflow error here
-7 in two's complement →	+ 1 0 0 1	
+6 in two's complement →	1 0 1 1 0	← valid positive result
	↑	
overflow ignored, with no harm done		

The magnitude of the sum of the three four-bit numbers was less than 2^{4-1} (<8), so our result was valid. If we add a +6 to a +7, and next add a -5, we'll encounter an intermediate overflow error, and our final sum will also be in error because its magnitude is not less than 8.

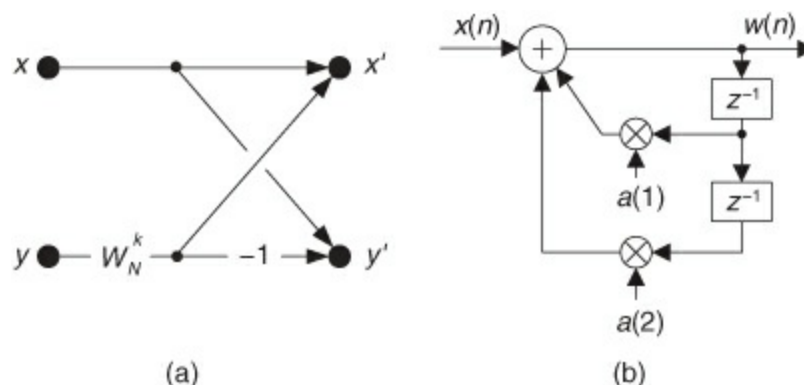
+7 in two's complement →	0 1 1 1	
+6 in two's complement →	+ 0 1 1 0	
-3 in two's complement →	1 1 0 1	← overflow error here
-5 in two's complement →	+ 1 0 1 1	
-8 in two's complement →	1 1 0 0 0	← invalid negative result

Another situation where overflow problems are conspicuous is during the calculation of the fast Fourier transform (FFT). It's difficult at first to imagine that multiplying complex numbers by sines and cosines can lead to excessive data word growth—particularly because sines and cosines are never greater than unity. Well, we can show how FFT data word growth occurs by considering a decimation-in-time FFT butterfly from [Figure 4-14\(c\)](#), repeated here as [Figure 12-6\(a\)](#), and grinding through a little algebra. The expression for the x' output of this FFT butterfly, from [Eq. \(4-26\)](#), is

(12-17)

$$x' = x + W_N^k \cdot y.$$

Figure 12-6 Data overflow scenarios: (a) single decimation-in-time FFT butterfly; (b) 2nd-order IIR filter.



Breaking up the butterfly's x and y inputs into their real and imaginary parts and remembering that $W_N^k = e^{-j2\pi k/N}$, we can express [Eq. \(12-17\)](#) as

(12-18)

$$x' = x_{\text{real}} + jx_{\text{imag}} + (e^{-j2\pi k/N}) \cdot (y_{\text{real}} + jy_{\text{imag}}).$$

If we let α be the twiddle factor angle of $2\pi k/N$, and recall that $e^{-j\alpha} = \cos(\alpha) - j\sin(\alpha)$, we can simplify [Eq. \(12-18\)](#) as

(12-19)

$$x' = x_{\text{real}} + jx_{\text{imag}} + [\cos(\alpha) - j\sin(\alpha)] \cdot (y_{\text{real}} + jy_{\text{imag}})$$

$$= x_{\text{real}} + \cos(\alpha)y_{\text{real}} + \sin(\alpha)y_{\text{imag}} + j(x_{\text{imag}} + \cos(\alpha)y_{\text{imag}} - \sin(\alpha)y_{\text{real}})$$

If we look, for example, at just the real part of the x' output, x'_{real} , it comprises the three terms

(12-20)

$$x'_{\text{real}} = x_{\text{real}} + \cos(\alpha)y_{\text{real}} + \sin(\alpha)y_{\text{imag}}$$

If x_{real} , y_{real} , and y_{imag} are of unity value when they enter the butterfly and the twiddle factor angle $\alpha = 2\pi k/N$ happens to be $\pi/4 = 45^\circ$, then, x'_{real} can be greater than 2 as

(12-21)

$$x'_{\text{real}} = 1 + \cos(45^\circ) \cdot 1 + \sin(45^\circ) \cdot 1$$

$$= 1 + 0.707 + 0.707 = 2.414$$

So we see that the real part of a complex number can more than double in magnitude in a single stage of an FFT. The imaginary part of a complex number is equally likely to more than double in magnitude in a single FFT stage. Without mitigating this word growth problem, overflow errors could render an FFT algorithm useless.

Overflow problems can also be troublesome for fixed-point systems containing feedback as shown in [Figure 12-6\(b\)](#). Examples of such networks are infinite impulse response (IIR) filters, cascaded integrator-comb (CIC) filters, and exponential averagers. The hardware register (accumulator) containing $w(n)$ must have a binary word width that will hold data values as large as the network's DC (zero Hz) gain G times the input signal, or $G \cdot x(n)$. To avoid data overflow, the number of bits in the $w(n)$ -results register must be at least

(12-22)

$$\text{accumulator bits} = \text{number of bits in } x(n) + \lceil \log_2(G) \rceil,$$

where $\lceil \log_2(G) \rceil$ means that if $\log_2(G)$ is not an integer, round it up to the next larger integer. (As a quick reminder, we can determine the DC gain of a digital network by substituting $z = 1$ in the network's z -domain transfer function.)

OK, overflow problems are handled in one of two ways—by truncation or rounding—each inducing its own individual kind of quantization errors, as we shall see.

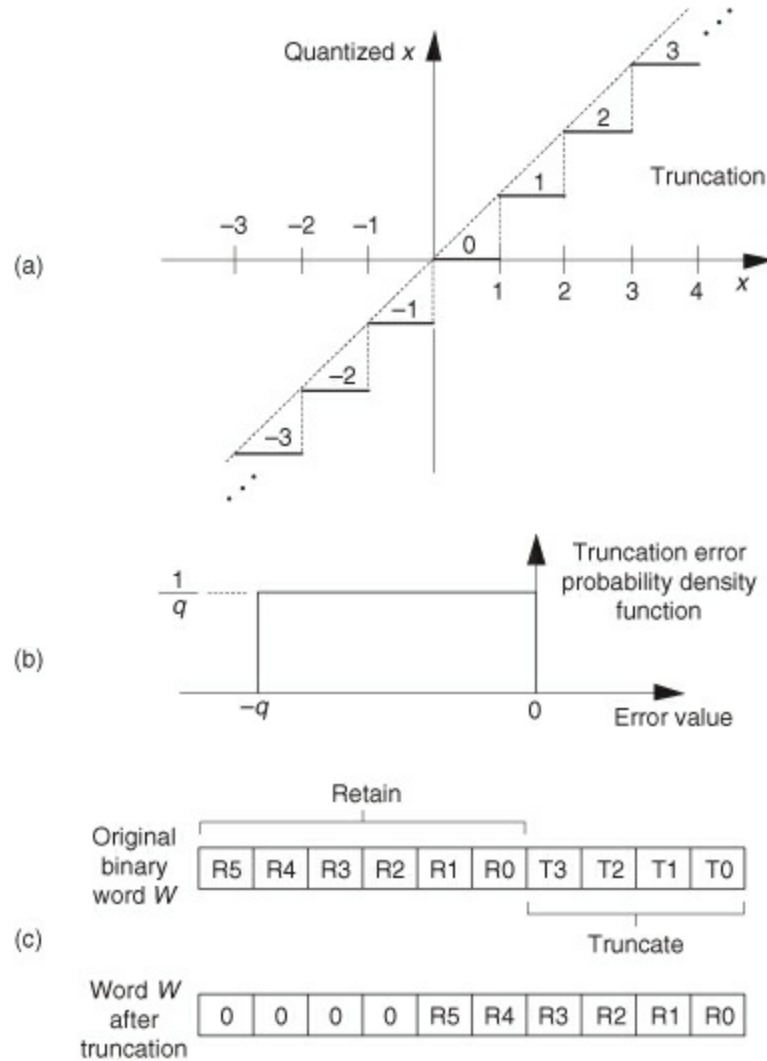
12.3.3 Truncation

Truncation is the process where some number of least significant bits are discarded from a binary number. A practical example of truncation is the situation where the results of a processing system are 16-bit signal samples that must be passed on to a 12-bit digital-to-analog converter. To avoid overflowing the converter's 12-bit input register, the least significant 4 bits of the 16-bit signal samples must be discarded. Thinking about decimal numbers, if we're quantizing to decimal integer values, for example, the real value 1.2 would be quantized to 1.

An example of truncation to integer values is shown in [Figure 12-7\(a\)](#), where all values of x in the

range of $0 \leq x < 1$ are set equal to 0, values of x in the range of $1 \leq x < 2$ are set equal to 1, and so on. The quantization level (value), in that figure, is $q = 1$. The quantization error induced by this truncation is the vertical distance between the horizontal bold lines and the dashed diagonal line in [Figure 12-7\(a\)](#).

Figure 12-7 Truncation: (a) quantization nonlinearities; (b) error probability density function; (c) binary truncation.



As we did with A/D converter quantization errors, we can call upon the concept of probability density functions to characterize the quantization errors induced by truncation. The probability density function of truncation errors, in terms of the quantization level q , is shown in [Figure 12-7\(b\)](#). In [Figure 12-7\(a\)](#) the quantization level q is 1, so in this case we can have truncation errors as great as -1 . Drawing upon our results from [Eqs. \(D-11\)](#) and [\(D-12\)](#) in [Appendix D](#), the mean and variance of our uniform truncation error probability density function are expressed as

$$(12-23)$$

$$\mu_{\text{trunc}} = \frac{-q}{2}$$

and

$$(12-24)$$

$$\sigma^2_{\text{trunc}} = \frac{q^2}{12}.$$

The notion of binary number truncation is shown in [Figure 12-7\(c\)](#), where the ten-bit binary word W

is to be truncated to six bits by discarding the four Truncate bits. So in this binary truncation situation, q in [Figure 12-7\(b\)](#) is equal to the least significant bit (lsb) value (bit R0) of the retained binary word.

In a sense, truncation error is the price we pay for the privilege of using integer binary arithmetic. One aspect of this is the error introduced when we use truncation to implement division by some integer power of two. A quick way of dividing a binary value by 2^K is to shift a binary word K bits to the right; that is, we're truncating the data value (not the binary word width) by discarding the rightmost K bits after the right shift.

For example, let's say we have the value 31 represented by the six-bit binary number 011111_2 , and we want to divide it by 16 through shifting the bits $K = 4$ places to the right and discarding those shifted bits. After the right shift we have a binary quotient of 000001_2 . Well, we see the significance of the problem because this type of division gave us a result of one instead of the correct quotient $31/16 = 1.9375$. Our division-by-truncation error here is roughly 50 percent of the correct quotient. Had our original dividend been 63 represented by the six-bit binary number 111111_2 , dividing it by 16 through a four-bit shift would give us an answer of binary 000011_2 , or decimal three. The correct answer, of course, is $63/16 = 3.9375$. In this case the percentage error is $0.9375/3.9375$, or about 23.8 percent. So, the larger the dividend, the lower the truncation error.

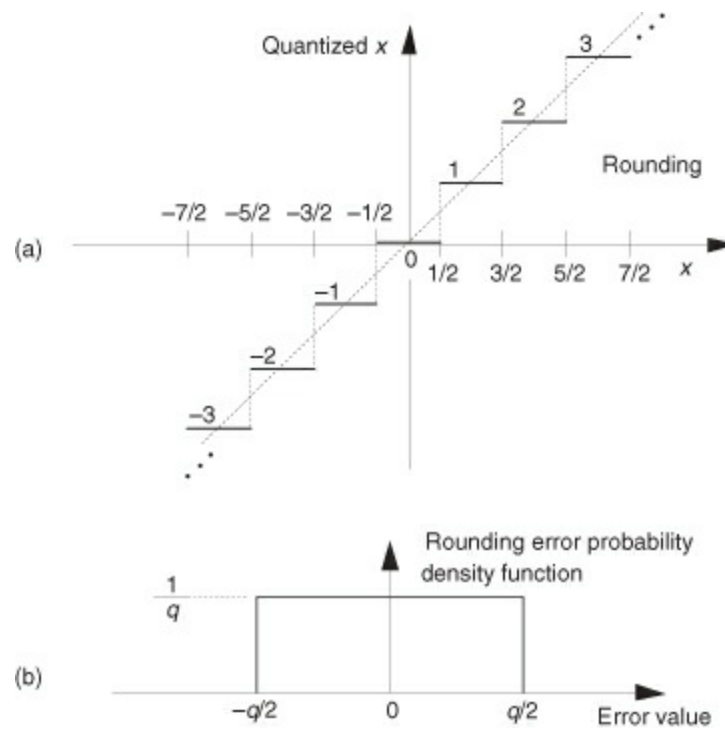
If we study these kinds of errors, we'll find that truncation error depends on three things: the number of value bits shifted and discarded, the values of the discarded bits (were those dropped bits ones or zeros?), and the magnitude of the binary number left over after shifting. Although a complete analysis of these truncation errors is beyond the scope of this book, a practical example of how division by truncation can cause serious numerical errors is given in reference [\[9\]](#).

Unfortunately, truncation induces a DC bias (an error whose average is a nonzero negative number) on the truncated signal samples, as predicted by [Eq. \(12-23\)](#). We see this behavior in [Figure 12-7\(b\)](#) where the truncation error is always negative. Inducing a constant (DC) error to a signal sequence can be troublesome in many applications because the always-negative truncation error can grow to an unacceptable level in subsequent computations. So, in an effort to avoid overflow errors, rounding (discussed in the next section) is often preferred over truncation.

12.3.4 Data Rounding

Rounding is where a binary number requiring truncation is slightly modified before the truncation operation is performed. Let's review the behavior of rounding by first defining rounding as the process wherein a number is modified such that it is subsequently represented by, or *rounded off to*, its nearest quantization level. For example, if we're quantizing to integer values, the decimal number 1.2 would be quantized to 1, and the number 1.6 would be quantized to 2. This is shown in [Figure 12-8\(a\)](#), where all values of x in the range of $-0.5 \leq x < 0.5$ are set equal to 0, values of x in the range of $0.5 \leq x < 1.5$ are set equal to 1, and so on.

Figure 12-8 Rounding: (a) quantization nonlinearities; (b) error probability density function.



The quantization error induced by such a rounding operation is the vertical distance between the bold horizontal lines and the dashed diagonal line in [Figure 12-8\(a\)](#). The probability density function of the error induced by rounding, in terms of the quantization level q , is shown in [Figure 12-8\(b\)](#). In [Figure 12-8\(a\)](#) the quantization level is $q = 1$, so in this case we can have quantization error magnitudes no greater than $q/2$, or $1/2$. Using our [Eqs. \(D-11\)](#) and [\(D-12\)](#) results from [Appendix D](#), the mean and variance of our uniform rounding probability density function are expressed as

$$(12-25)$$

$$\mu_{\text{round}} = 0$$

and

$$(12-26)$$

$$\sigma^2_{\text{round}} = \frac{q^2}{12}.$$

The notion of binary number rounding can be described using [Figure 12-7\(c\)](#), where the binary word W is to be truncated by discarding the four Truncate bits. With rounding, the binary word W is modified before the Truncate bits are discarded. So with binary rounding, q in [Figure 12-8\(b\)](#) is equal to the lsb value of the preserved binary word $R0$.

Let's not forget: the purpose of rounding, its goal in life, is to avoid data overflow errors while reducing the DC bias error (an error whose average is not zero) induced by simple truncation. Rounding achieves this goal because, in theory, its average error is zero as shown by [Eq. \(12-25\)](#). Next we discuss two popular methods of data rounding.

A common form of binary data rounding is straightforward to implement. Called *round-to-nearest*, it comprises the two-step process of adding one to the most significant (leftmost) of the lsb bits to be discarded, bit T3 of word W in [Figure 12-7\(c\)](#), and then discarding the appropriate Truncate bits. For an example of this rounding method, let's say we have 16-bit signal samples destined to be routed to a 12-bit digital-to-analog converter. To avoid overflowing the converter's 12-bit input register, we add a binary value of 1000_2 (decimal $8_{10} = 2^3$) to the original 16-bit sample value and then truncate

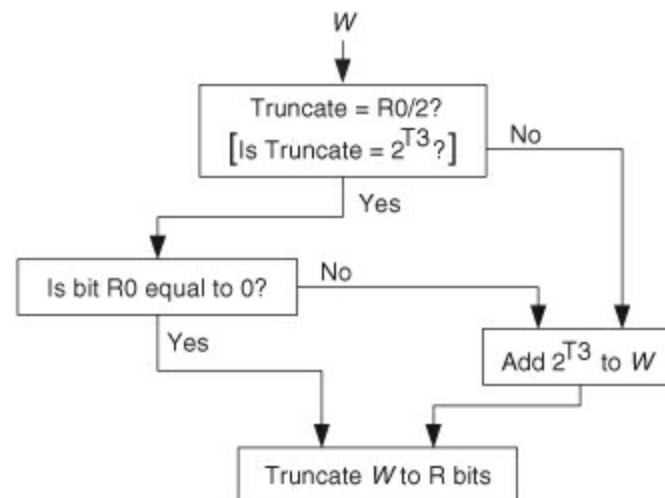
(discard) the sum's least significant 4 bits. As another example of round-to-nearest rounding, if a 32-bit "long" word is rounded to 16 bits, a value of 2^{15} is added to the long word before discarding the sum's 16 least significant bits.

Stated in different words, this round-to-nearest rounding method means: If the T3 bit is a one, increment the R bits by one. Then shift the R bits to the right, discarding the Truncate bits.

The round-to-nearest method does reduce the average (DC bias) of the quantization error induced by simple truncation; however the round-to-nearest method's average error bias is close to but not exactly equal to zero. (That's because the R bits, in [Figure 12-7\(c\)](#), are always incremented when the value of the Truncate bits is equal to the value $R0/2$. This means that over time the R bits are rounded up slightly more often than they are rounded down.) With additional bit checking we can force the average rounding error to be exactly zero using a scheme called *convergent rounding*.

Convergent rounding, also called *round to even*, is a slightly more complicated method of rounding, but one that yields zero-average rounding error on the rounded binary signal samples. Similar to the round-to-nearest method, convergent rounding does not always increment [Figure 12-7\(c\)](#)'s R bits (the value Retain) when the value of the Truncate bits is equal to $R0/2$. In the convergent rounding scheme, when $\text{Truncate} = R0/2$, the value Retain is only incremented if its original value was an odd number. This clever process is shown in [Figure 12-9](#).

Figure 12-9 Convergent rounding.



OK, here's what we've learned about rounding: Relative to simple truncation, rounding requires more computations, but rounding both minimizes the constant-level (DC bias) quantization error induced by truncation alone, and rounding has a lower maximum quantization error. So rounding is often the preferred method used to avoid binary data overflow errors. The above two rounding methods can, by the way, be used in two's complement number format systems.

As a practical rule, to retain maximum numerical precision, all necessary full-width binary arithmetic should be performed first and then rounding (or truncation) should be performed as the very last operation. For example, if we must add twenty 16-bit binary numbers followed by rounding the sum to 12 bits, we should perform the additions at full 16-bit precision and, as a final step, round the summation result to 12 bits.

In digital signal processing, statistical analysis of quantization error effects is complicated because quantization is a nonlinear process. Analytical results depend on the types of quantization errors, the magnitude of the data being represented, the numerical format used, and which of the many FFT or

digital filter structures we are implementing. Be that as it may, digital signal processing experts have developed simplified error models whose analysis has proved useful. Although discussion of these analysis techniques and their results is beyond the scope of this introductory text, many references are available for the energetic reader[10–18]. (Reference [11] has an extensive reference list of its own on the topic of quantization error analysis.)

Again, the overflow problems using fixed-point binary formats—which we try to alleviate with truncation or rounding—arise because so many digital signal processing algorithms comprise large numbers of additions or multiplications. This obstacle, particularly in hardware implementations of digital filters and the FFT, is avoided by hardware designers through the use of floating-point binary number formats.

12.4 Floating-Point Binary Formats

Floating-point binary formats allow us to overcome most of the limitations of precision and dynamic range mandated by fixed-point binary formats, particularly in reducing the ill effects of overflow[19]. Floating-point formats segment a data word into two parts: a mantissa m and an exponent e . Using these parts, the value of a binary floating-point number n is evaluated as

(12-27)

$$n = m \cdot 2^e,$$

that is, the number's value is the product of the mantissa and 2 raised to the power of the exponent. (*Mantissa* is a somewhat unfortunate choice of terms because it has a meaning here very different from that in the mathematics of logarithms. Mantissa originally meant the decimal fraction of a logarithm.[†] However, due to its abundance in the literature we'll continue using the term *mantissa* here.) Of course, both the mantissa and the exponent in Eq. (12-27) can be either positive or negative numbers.

[†] For example, the common logarithm (log to the base 10) of 256 is 2.4082. The 2 to the left of the decimal point is called the characteristic of the logarithm and the 4082 digits are called the mantissa. The 2 in 2.4082 does not mean that we multiply .4082 by 10^2 . The 2 means that we take the antilog of .4082 to get 2.56 and multiply that by 10^2 to get 256.

Let's assume that a b -bit floating-point number will use b_e bits for the fixed-point signed exponent and b_m bits for the fixed-point signed mantissa. The greater the number of b_e bits used, the larger the dynamic range of the number. The more bits used for b_m , the better the resolution, or precision, of the number. Early computer simulations conducted by the developers of b -bit floating-point formats indicated that the best trade-off occurred with $b_e \approx b/4$ and $b_m \approx 3b/4$. We'll see that for typical 32-bit floating-point formats used today, $b_e \approx 8$ bits and $b_m \approx 24$ bits.

To take advantage of a mantissa's full dynamic range, most implementations of floating-point numbers treat the mantissa as a fractional fixed-point binary number, shift the mantissa bits to the right or left, so that the most significant bit is a one, and adjust the exponent accordingly. The process of shifting a binary bit pattern so that the most significant bit is a one is called *bit normalization*. When normalized, the mantissa bits are typically called the *fraction* of the floating-point number, instead of the mantissa. For example, the decimal value 3.6875_{10} can be represented by the fractional binary number 11.1011_2 . If we use a two-bit exponent with a six-bit fraction floating-point word, we can just as well represent 11.1011_2 by shifting it to the right two places and setting the exponent to two as

(12-28)

$$11.1011_2 = \begin{array}{c} \text{exponent} \quad \text{fraction} \\ \downarrow \quad \downarrow \\ \boxed{10 \mid .111011} \\ \uparrow \\ \text{binary point} \end{array}$$

The floating-point word above can be evaluated to retrieve our decimal number again as

(12-29)

$$\begin{aligned} & [0(1 \cdot 2^{-1}) + (1 \cdot 2^{-2}) + (1 \cdot 2^{-3}) + (0 \cdot 2^{-4}) + (1 \cdot 2^{-5}) + (1 \cdot 2^{-6})] \cdot 2^2 \\ &= [0(1 \cdot \frac{1}{2}) + (1 \cdot \frac{1}{4}) + (1 \cdot \frac{1}{8}) + (0 \cdot \frac{1}{16}) + (1 \cdot \frac{1}{32}) + (1 \cdot \frac{1}{64})] \cdot 2^2 \\ &= [0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 + 0.015625] \cdot 2^2 \\ &= 0.921875 \cdot 4 = 3.6875. \end{aligned}$$

After some experience using floating-point normalization, users soon realized that always having a one in the most significant bit of the fraction was wasteful. That redundant one was taking up a single bit position in all data words and serving no purpose. So practical implementations of floating-point formats discard that one, assume its existence, and increase the useful number of fraction bits by one. This is why the term *hidden bit* is used to describe some floating-point formats. While increasing the fraction's precision, this scheme uses less memory because the hidden bit is merely accounted for in the hardware arithmetic logic. Using a hidden bit, the fraction in [Eq. \(12-28\)](#)'s floating-point number is shifted to the left one place and would now be

(12-30)

$$11.1011_2 = \begin{array}{c} \text{exponent} \quad \text{fraction} \\ \downarrow \quad \downarrow \\ \boxed{10 \mid .110110} \\ \uparrow \\ \text{binary point} \end{array}$$

Recall that the exponent and mantissa bits were fixed-point signed binary numbers, and we've discussed several formats for representing signed binary numbers, i.e., sign magnitude, two's complement, and offset binary. As it turns out, all three signed binary formats are used in industry-standard floating-point formats. The most common floating-point formats, all using 32-bit words, are listed in [Table 12-6](#).

Table 12-6 Floating-Point Number Formats

IEEE Standard P754 Format																
Bit	31	30	29	28	27	26	25	24	23	22	21	20	...	2	1	0
	S	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	...	2^{-21}	2^{-22}	2^{-23}
Sign (s)	← Exponent (e) →									← Fraction (f) →						

IBM Format																
Bit	31	30	29	28	27	26	25	24	23	22	21	20	...	2	1	0
	S	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	...	2^{-22}	2^{-23}	2^{-24}
Sign (s)	← Exponent (e) →									← Fraction (f) →						

DEC (Digital Equipment Corp.) Format																
Bit	31	30	29	28	27	26	25	24	23	22	21	20	...	2	1	0
	S	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-2}	2^{-3}	2^{-4}	...	2^{-22}	2^{-23}	2^{-24}
Sign (s)	← Exponent (e) →									← Fraction (f) →						

MIL-STD 1750A Format																
Bit	31	30	29	...	11	10	9	8	7	6	5	4	3	2	1	0
	2^0	2^{-1}	2^{-2}	...	2^{-20}	2^{-21}	2^{-22}	2^{-23}	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	← Fraction (f) →								← Exponent (e) →							

The IEEE P754 floating-point format is the most popular because so many manufacturers of floating-point integrated circuits comply with this standard[8,20–22]. Its exponent e is offset binary (biased exponent), and its fraction is a sign-magnitude binary number with a hidden bit that's assumed to be 2^0 . The decimal value of a normalized IEEE P754 floating-point number is evaluated as

(12-31)

$$\text{value}_{\text{IEEE}} = (-1)^s \cdot \underset{\substack{\uparrow \\ \text{hidden bit}}}{1} f \cdot 2^{e-127}$$

where f is the decimal-formatted value of the fractional bits divided by 2^{23} . Value e is the decimal value of the floating-point number's exponent bits.

The IBM floating-point format differs somewhat from the other floating-point formats because it uses a base of 16 rather than 2. Its exponent is offset binary, and its fraction is sign magnitude with no hidden bit. The decimal value of a normalized IBM floating-point number is evaluated as

(12-32)

$$\text{value}_{\text{IBM}} = (-1)^s \cdot 0 f \cdot 16^{e-64}.$$

The DEC floating-point format uses an offset binary exponent, and its fraction is sign magnitude with a hidden bit that's assumed to be 2^{-1} . The decimal value of a normalized DEC floating-point number is evaluated as

(12-33)

$$\text{value}_{\text{DEC}} = (-1)^s \cdot 0 \underset{\substack{\uparrow \\ \text{hidden bit}}}{1} f \cdot 2^{e-128}.$$

MIL-STD 1750A is a United States Military Airborne floating-point standard. Its exponent e is a two's complement binary number residing in the least significant eight bits. MIL-STD 1750A's fraction is also a two's complement number (with no hidden bit), and that's why no sign bit is specifically indicated in [Table 12-6](#). The decimal value of a MIL-STD 1750A floating-point number is evaluated as

(12-34)

$$\text{value}_{1750A} = f \cdot 2^e.$$

Notice how the floating-point formats in [Table 12-6](#) all have word lengths of 32 bits. This was not accidental. Using 32-bit words makes these formats easier to handle using 8-, 16-, and 32-bit hardware processors. That fact notwithstanding and given the advantages afforded by floating-point number formats, these formats do require a significant amount of logical comparisons and branching to correctly perform arithmetic operations. Reference [\[23\]](#) provides useful flow charts showing what procedural steps must be taken when floating-point numbers are added and multiplied.

12.4.1 Floating-Point Dynamic Range

Attempting to determine the dynamic range of an arbitrary floating-point number format is a challenging exercise. We start by repeating the expression for a number system's dynamic range from [Eq. \(12-6\)](#) as

(12-35)

$$\text{dynamic range}_{dB} = 20 \cdot \log_{10} \left(\frac{\text{largest possible word value}}{\text{smallest possible word value}} \right).$$

When we attempt to determine the largest and smallest possible values for a floating-point number format, we quickly see that they depend on such factors as

- the position of the binary point
- whether a hidden bit is used or not (If used, its position relative to the binary point is important.)
- the base value of the floating-point number format
- the signed binary format used for the exponent and the fraction (For example, recall from [Table 12-2](#) that the binary two's complement format can represent larger negative numbers than the sign-magnitude format.)
- how unnormalized fractions are handled, if at all (*Unnormalized*, also called *gradual underflow*, means a nonzero number that's less than the minimum normalized format but can still be represented when the exponent and hidden bit are both zero.)
- how exponents are handled when they're either all ones or all zeros. (For example, the IEEE P754 format treats a number having an all-ones exponent and a nonzero fraction as an invalid number, whereas the DEC format handles a number having a sign = 1 and a zero exponent as a special instruction instead of a valid number.)

Trying to develop a dynamic range expression that accounts for all the possible combinations of the above factors is impractical. What we can do is derive a rule-of-thumb expression for dynamic range that's often used in practice[\[8,22,24\]](#).

Let's assume the following for our derivation: the exponent is a b_e -bit offset binary number, the

fraction is a normalized sign-magnitude number having a sign bit and b_m magnitude bits, and a hidden bit is used just left of the binary point. Our hypothetical floating-point word takes the following form:

Bit	$b_m + b_e - 1$	$b_m + b_e - 2$	\dots	$b_m + 2$	b_m	$b_m - 1$	$b_m - 2$	\dots	1	0
	2^{b_e-1}	2^{b_e-2}	\dots	2^1	2^0	2^{-1}	2^{-2}	\dots	2^{-b_m+1}	2^{-b_m}
Sign (s)	← Exponent (e) →					← Fraction (f) →				

First we'll determine what the largest value can be for our floating-point word. The largest fraction is a one in the hidden bit, and the remaining b_m fraction bits are all ones. This would make fraction $f = [1 + (1 - 2^{-b_m})]$. The first 1 in this expression is the hidden bit to the left of the binary point, and the value in parentheses is all b_m bits equal to ones to the right of the binary point. The greatest positive value we can have for the b_e -bit offset binary exponent is $2^{(b_e-1)}$. So the largest value that can be represented with the floating-point number is the largest fraction raised to the largest positive exponent, or

$$(12-36)$$

$$\text{largest possible word value} = [1 + (1 - 2^{-b_m})] \cdot 2^{(2^{b_e-1}-1)}.$$

The smallest value we can represent with our floating-point word is a one in the hidden bit times two raised to the exponent's most negative value, $2^{-(2^{b_e-1})}$, or

$$(12-37)$$

$$\text{smallest possible word value} = 1 \cdot 2^{-(2^{b_e-1})}.$$

Plugging [Eqs. \(12-36\)](#) and [\(12-37\)](#) into [Eq. \(12-35\)](#),

$$(12-38)$$

$$\text{dynamic range}_{\text{dB}} = 20 \cdot \log_{10} \left(\frac{[1 + (1 - 2^{-b_m})] \cdot 2^{(2^{b_e-1}-1)}}{1 \cdot 2^{-(2^{b_e-1})}} \right).$$

Now here's where the thumb comes in—when b_m is large, say over seven, the 2^{-b_m} value approaches zero; that is, as b_m increases, the all-ones fraction $(1 - 2^{-b_m})$ value in the numerator approaches 1. Assuming this, [Eq. \(12-38\)](#) becomes

$$(12-39)$$

$$\begin{aligned} \text{dynamic range}_{\text{dB}} &\approx 20 \cdot \log_{10} \left(\frac{[1 + 1] \cdot 2^{(2^{b_e-1}-1)}}{1 \cdot 2^{-(2^{b_e-1})}} \right) \\ &= 20 \cdot \log_{10} \left(\frac{2 \cdot 2^{(2^{b_e-1}-1)}}{2^{-(2^{b_e-1})}} \right) = 20 \cdot \log_{10} \left(\frac{2^{(2^{b_e-1})}}{2^{-(2^{b_e-1})}} \right) \\ &= 20 \cdot \log_{10}(2 \cdot 2^{(2^{b_e-1})}) = 20 \cdot \log_{10}(2^{(2^{b_e})}) = 6.02 \cdot 2^{b_e}. \end{aligned}$$

Using [Eq. \(12-39\)](#), we can estimate, for example, the dynamic range of the single-precision IEEE

P754 standard floating-point format with its eight-bit exponent:

(12-40)

$$\text{dynamic range}_{\text{IEEE P754}} = 6.02 \cdot 2^8 = 1529 \text{ dB.}$$

Although we've introduced the major features of the most common floating-point formats, there are still more details to learn about floating-point numbers. For the interested reader, the references given in this section provide a good place to start.

12.5 Block Floating-Point Binary Format

A marriage of fixed-point and floating-point binary formats is known as *block floating point*. This scheme is used, particularly in dedicated FFT integrated circuits, when large arrays, or blocks, of associated data are to be manipulated mathematically. Block floating-point schemes begin by examining all the words in a block of data, normalizing the largest-valued word's fraction, and establishing the correct exponent. This normalization takes advantage of the fraction's full dynamic range. Next, the fractions of the remaining data words are shifted appropriately, so that they can use the exponent of the largest word. In this way, all of the data words use the same exponent value to conserve hardware memory.

In FFT implementations, the arithmetic is performed treating the block normalized data values as fixed-point binary. However, when an addition causes an overflow condition, all of the data words are shifted one bit to the right (division by two), and the exponent is incremented by one. As the reader may have guessed, block floating-point formats have increased dynamic range and avoid the overflow problems inherent in fixed-point formats but do not reach the performance level of true floating-point formats[[8,25,26](#)].

References

- [1] Neugebauer, O. "The History of Ancient Astronomy," *Journal of Near Eastern Studies*, Vol. 4, 1945, p. 12.
- [2] Knuth, D. E. *The Art of Computer Programming: Seminumerical Methods*, Vol. 2, Addison-Wesley, Reading, Massachusetts, 1981, [Section 4.1](#), p. 179.
- [3] Kester, W. "Peripheral Circuits Can Make or Break Sampling-ADC Systems," *EDN Magazine*, October 1, 1992.
- [4] Grove, M. "Measuring Frequency Response and Effective Bits Using Digital Signal Processing Techniques," *Hewlett-Packard Journal*, February 1992.
- [5] Tektronix. "Effective Bits Testing Evaluates Dynamic Range Performance of Digitizing Instruments," *Tektronix Application Note*, No. 45W-7527, December 1989.
- [6] Ushani, R. "Subranging ADCs Operate at High Speed with High Resolution," *EDN Magazine*, April 11, 1991.
- [7] Demler, M. "Time-Domain Techniques Enhance Testing of High-Speed ADCs," *EDN Magazine*, March 30, 1992.
- [8] Hilton, H. "A 10-MHz Analog-to-Digital Converter with 110-dB Linearity," *Hewlett-Packard Journal*, October 1993.
- [9] Lyons, R. G. "Providing Software Flexibility for Optical Processor Noise Analysis," *Computer*

Design, July 1978, p. 95.

- [10] Knuth, D. E. *The Art of Computer Programming: Seminumerical Methods*, Vol. 2, Addison-Wesley, Reading, Massachusetts, 1981, [Section 4.2](#), p. 198.
- [11] Rabiner, L. R., and Gold, B. *Theory and Application of Digital Signal Processing*, [Chapter 5](#), Prentice Hall, Englewood Cliffs, New Jersey, 1975, p. 353.
- [12] Jackson, L. B. “An Analysis of Limit Cycles Due to Multiplicative Rounding in Recursive Digital Filters,” *Proc. 7th Allerton Conf. Circuit System Theory*, 1969, pp. 69–78.
- [13] Kan, E. P. F., and Aggarwal, J. K. “Error Analysis of Digital Filters Employing Floating Point Arithmetic,” *IEEE Trans. Circuit Theory*, Vol. CT-18, November 1971, pp. 678–686.
- [14] Crochiere, R. E. “Digital Ladder Structures and Coefficient Sensitivity,” *IEEE Trans. Audio Electroacoustics*, Vol. AU-20, October 1972, pp. 240–246.
- [15] Jackson, L. B. “On the Interaction of Roundoff Noise and Dynamic Range in Digital Filters,” *Bell System Technical Journal*, Vol. 49, February 1970, pp. 159–184.
- [16] Roberts, R. A., and Mullis, C. T. *Digital Signal Processing*, Addison-Wesley, Reading, Massachusetts, 1987, p. 277.
- [17] Jackson, L. B. “Roundoff Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form,” *IEEE Trans. Audio Electroacoustics*, Vol. AU-18, June 1970, pp. 107–122.
- [18] Oppenheim, A. V., and Schaffer, R. W. *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1989, Sections 9.7 and 9.8.
- [19] Larimer, J., and Chen, D. “Fixed or Floating? A Pointed Question in DSPs,” *EDN Magazine*, August 3, 1995.
- [20] Ashton, C. “Floating Point Math Handles Iterative and Recursive Algorithms,” *EDN Magazine*, January 9, 1986.
- [21] Windsor, B., and Wilson, J. “Arithmetic Duo Excels in Computing Floating Point Products,” *Electronic Design*, May 17, 1984.
- [22] Windsor, W. A. “IEEE Floating Point Chips Implement DSP Architectures,” *Computer Design*, January 1985.
- [23] Texas Instruments Inc. *Digital Signal Processing Applications with the TMS320 Family: Theory, Algorithms, and Implementations*, SPRA012A, Texas Instruments, Dallas, Texas, 1986.
- [24] Strauss, W. I. “Integer or Floating Point? Making the Choice,” *Computer Design Magazine*, April 1, 1990, p. 85.
- [25] Oppenheim, A. V., and Weinstein, C. J. “Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform,” *Proceedings of the IEEE*, August 1972, pp. 957–976.
- [26] Woods, R. E. “Transform-Based Processing: How Much Precision Is Needed?” *ESD: The Electronic System Design Magazine*, February 1987.

Chapter 12 Problems

12.1 Given their specified format, convert the following integer binary numbers to decimal format:

(a) 1100 0111, unsigned,

- (b) 1100 0111, sign magnitude,
- (c) 1100 0111, two's complement,
- (d) 1100 0111, offset binary.

12.2 Convert the following unsigned integer binary numbers, given here in hexadecimal format, to decimal:

- (a) \$A231,
- (b) 0x71F.

12.3 Given the hexadecimal integer numbers \$07 and \$E2 in two's complement format, what is the decimal value of \$07 minus \$E2? Show your work.

12.4 Sign-extend the following two's complement integer numbers, given in hexadecimal format, to 16 bits and express the results in hexadecimal format:

- (a) \$45,
- (b) \$B3.

12.5 Show that the binary addition operation

$$\begin{array}{r} 0000\ 1111 \\ +0000\ 1101 \\ \hline 0001\ 1100 \end{array}$$

gives the correct decimal results when the two binary addends and the sum are in the following two's complement fractional formats:

- (a) 7.1 (7 integer bits and 1 fractional bit),
- (b) 6.2 (6 integer bits and 2 fractional bits),
- (c) 4.4 (4 integer bits and 4 fractional bits).

12.6 Microchip Technology Inc. produces a microcontroller chip (Part #PIC24F) that accommodates 16-bit data words. When using a two's complement integer number format, what are the most positive and most negative decimal numbers that can be represented by the microcontroller's data word?

12.7 Consider four-bit unsigned binary words using a 2.2 ("two dot two") "integer plus fraction" format. List all 16 possible binary words in this format and give their decimal equivalents.

12.8 The annual residential property tax in California is 0.0165 times the assessed dollar value of the property. What is this 0.0165 tax rate factor in a two's complement 1.15 format? Give the answer in both binary and hexadecimal representations. Show how you arrived at your solution.

12.9 The decimal number $1/3$ cannot be represented exactly with a finite number of decimal digits, nor with a finite number of binary bits. What would be the base of a number system that would allow decimal $1/3$ to be exactly represented with a finite number of digits?

12.10 If the number 4273_6 is in a base 6 numbering system, what would be its decimal value?

12.11 Think about a 32-bit two's complement fixed-point binary number having 31 fractional bits (a "1.31" two's complement number). This number format is very common in today's high-

performance programmable DSP chips.

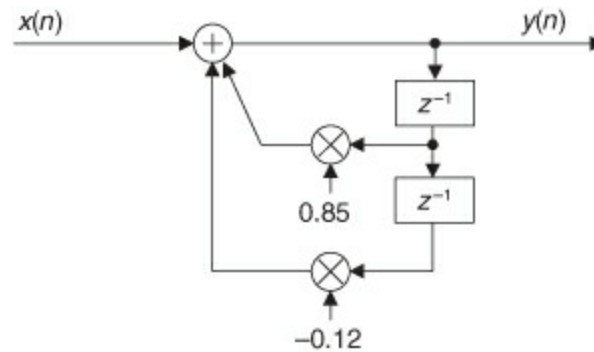
- (a) What is the most positive decimal value that can be represented by such a binary number? Show how you arrived at your solution.
- (b) What is the most negative decimal value?

12.12 As of this writing, Analog Devices Inc. produces an integrated circuit (Part #AD9958), called a *direct digital synthesizer*, that generates high-precision analog sinewaves. The AD9958 uses a 31-bit binary word to control the device's output frequency. When the control word is at its minimum value, the device's output frequency is zero Hz. When the control word is at its maximum value, the output frequency is 250 MHz. What is the frequency resolution (the frequency step size) of this sinusoidal signal generator in Hz?

12.13 The first commercial audio compact disc (CD) players used 16-bit samples to represent an analog audio signal. Their sample rate was $f_s = 44.1$ kHz. Those 16-bit samples were applied to a digital-to-analog (D/A) converter whose analog output was routed to a speaker. What is the combined data output rate of the digital portion, measured in bytes (8-bit binary words) per second, of a stereo CD player?

12.14 When implementing a digital filter using a fixed-point binary number format, care must be taken to avoid arithmetic overflow errors. With that notion in mind, if the $x(n)$ input samples in [Figure P12-14](#) are eight-bit binary words, how many bits are needed to represent the $y(n)$ output sequence to avoid any data overflow errors? Show how you arrived at your answer.

Figure P12-14

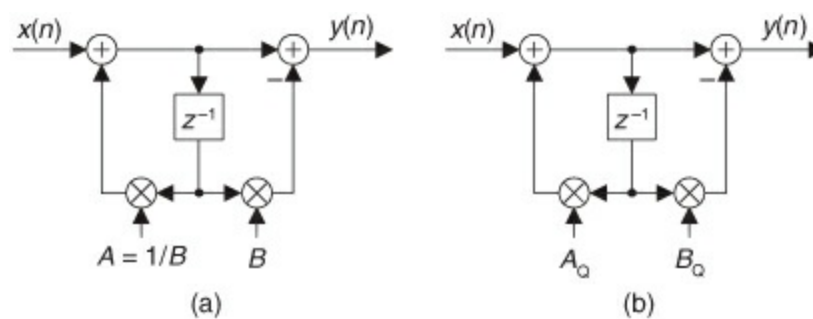


Hint: Review the last portion of the text's [Section 12.3.2](#).

12.15 Review the brief description of *allpass filters* in [Appendix F](#). One form of an allpass filter is shown in [Figure P12-15\(a\)](#). For the filter to have the desired constant magnitude response over its full operating frequency, coefficient A must be equal to

$$A = \frac{1}{B}.$$

Figure P12-15

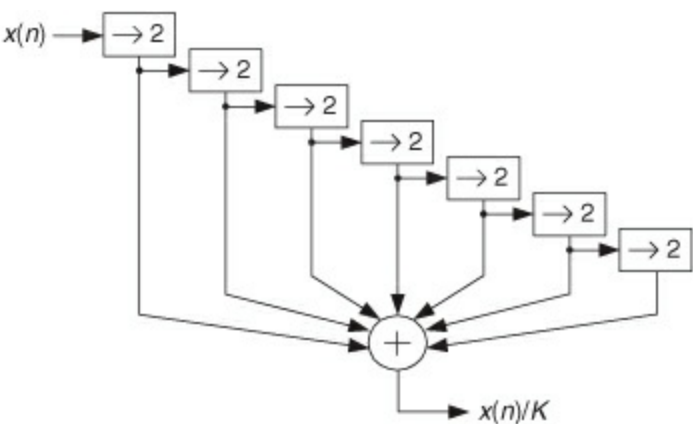


If the filter is designed such that $B = 2.5$, show why we cannot achieve the desired constant frequency magnitude response when coefficients A and B are quantized using four-bit unsigned binary words in a 2.2 (“two dot two”) “integer plus fraction” format, where A_Q and B_Q are the quantized coefficients as shown in [Figure P12-15\(b\)](#).

12.16 National Semiconductors Inc. produces a digital tuner chip (Part #CLC5903), used for building digital receivers, that has the capability to amplify its output signal by shifting its binary signal sample values to the left by as few as one bit to as many as seven bits. What is the maximum gain, measured in dB (decibels), of this tuner’s bit-shifting amplification capability?

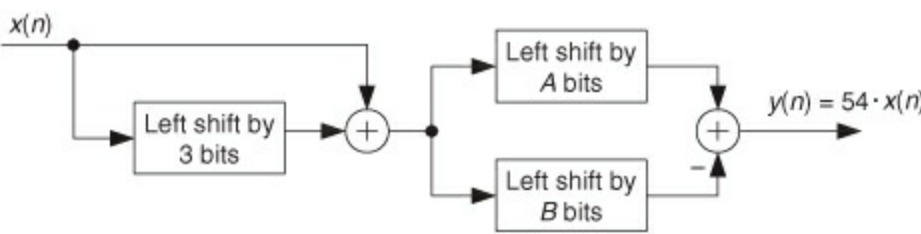
12.17 [Figure P12-17](#) shows an algorithm that approximates the operation of dividing a sign-magnitude binary number $x(n)$ by an integer value K . (A block containing the “ $\rightarrow 2$ ” symbol means truncation by way of a binary right shift by two bits.) What is the value of integer K ? Show your work.

Figure P12-17



12.18 When using programmable DSP chips, multiplication is a simple straightforward operation. However, when using field-programmable gate arrays (FPGAs), multiplier hardware is typically difficult to implement and should be avoided whenever possible. [Figure P12-18](#) shows how we can multiply a binary $x(n)$ input sequence by 54, without the need for multiplier hardware. What are the values for A and B in [Figure P12-18](#) so that $y(n)$ equals 54 times $x(n)$?

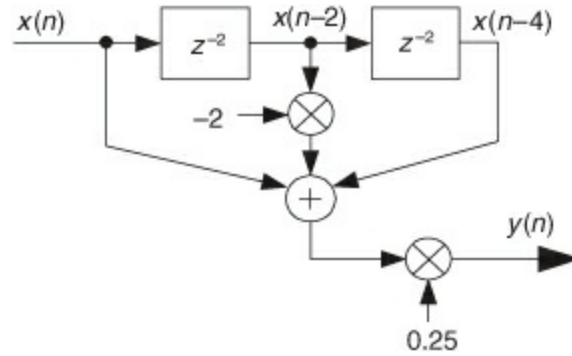
Figure P12-18



12.19 Consider the network shown in [Figure P12-19](#) which approximates a 2nd-order differentiation

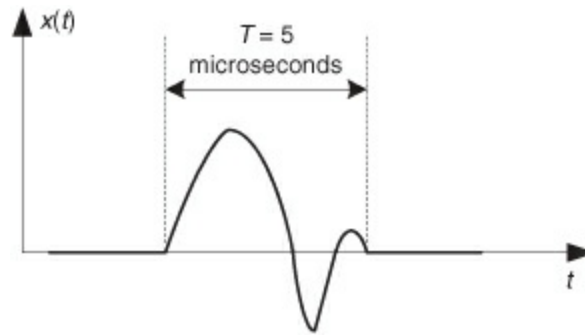
operation. In many DSP implementations (using field-programmable gate arrays, for example) it is advantageous to minimize the number of multiplications. Assuming that all the sequences in [Figure P12-19](#) use a binary two's complement integer number format, what data bit manipulations must be implemented to eliminate the two multipliers?

Figure P12-19



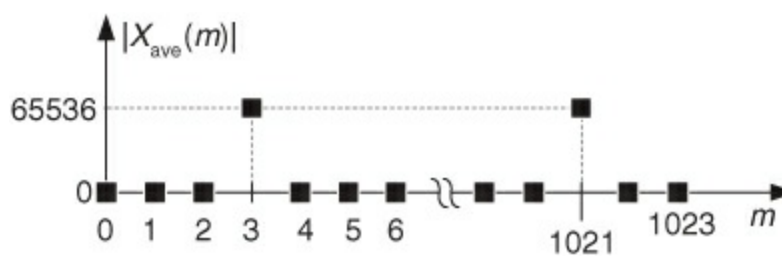
12.20 Agilent Inc. produces an A/D converter (Model #DP1400) whose sample rate is 2×10^9 samples/second ($f_s = 2$ GHz). This digitizer provides super-fine time resolution samples of analog signals whose durations are $T = 5 \times 10^{-6}$ seconds (5 microseconds) as shown in [Figure P12-20](#). If each converter output sample is stored in one memory location of a computer, how many memory locations are required to store the converter's $x(n)$ output sequence representing the 5-microsecond-duration $x(t)$ signal?

Figure P12-20



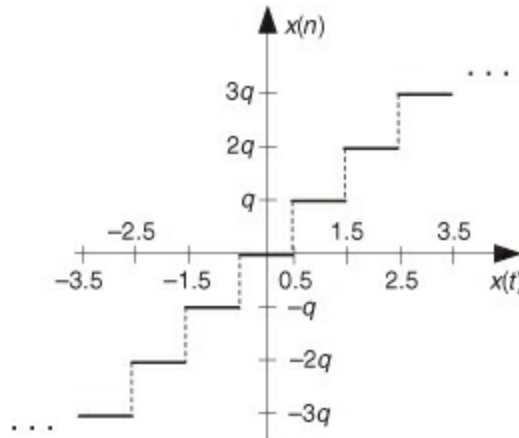
12.21 Here is a problem often encountered by DSP engineers. Assume we sample exactly three cycles of a continuous $x(t)$ sinewave resulting in a block of 1024 $x(n)$ time samples and compute a 1024-point fast Fourier transform (FFT) to obtain the FFT magnitude samples. Also assume that we repeat the sampling and FFT magnitude computations many times and average the FFT magnitude sequences to produce the average magnitude samples, $|X_{\text{ave}}(m)|$, shown in [Figure P12-21](#). (We averaged multiple FFT magnitude sequences to increase the accuracy, by reducing the variance, of our final $|X_{\text{ave}}(m)|$ sequence.) If the A/D converter produces ten-bit binary words in sign-magnitude format and has an input full-scale bipolar voltage range of ± 5 volts, what is the peak value of the continuous $x(t)$ sinewave? Justify your answer.

Figure P12-21



12.22 Suppose we have a 12-bit A/D converter that operates over an input voltage range of ± 5 volts (10 volts peak-peak). Assume the A/D converter is *ideal* in its operation and its transfer function is that shown in [Figure P12-22](#) where the tick mark spacing of the $x(t)$ and $x(n)$ axes is the converter's quantization-level q .

Figure P12-22



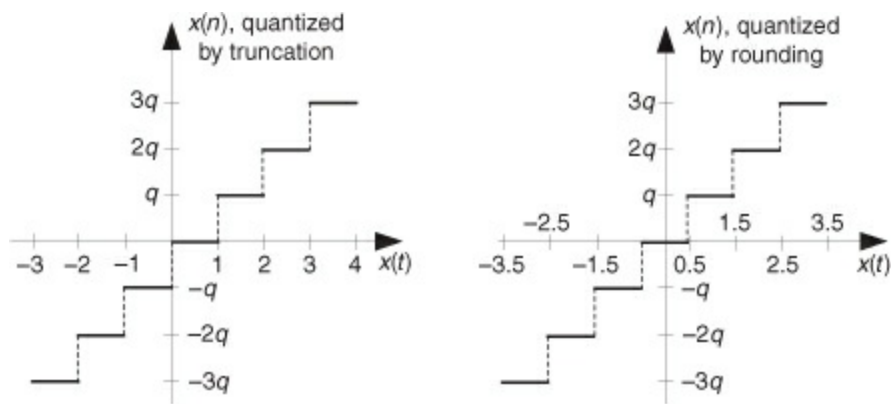
- What is the A/D converter's quantization-level q (least significant bit) voltage?
- What are the A/D converter's maximum positive and maximum negative quantization error voltages?
- If we apply a 7-volt peak-peak sinusoidal voltage to the converter's analog input, what A/D output signal-to-quantization noise value, $SNR_{A/D}$ in dB, should we expect? Show how you arrived at your answer.

12.23 Suppose an A/D converter manufacturer applies a 10-volt peak-peak sinusoidal voltage to their 12-bit converter's analog input, conducts careful testing, and measures the converter's overall signal-to-noise level to be 67 dB. What is the *effective number of bits* value, b_{eff} , for their A/D converter?

12.24 Let's reinforce our understanding of the quantization errors induced by typical A/D converters.

- [Figure P12-24](#) shows the quantized $x(n)$ output integer values of truncating and rounding A/D converters as a function of their continuous $x(t)$ input voltage. It's sensible to call those bold *stair-step* curves the "transfer functions" of the A/D converters. The curves are normalized to the A/D converter's quantization-level voltage q , such that an $x(t)$ value of 2 represents a voltage of $2q$ volts. Draw the curves of the quantization error as a function of the continuous $x(t)$ input for both truncating and rounding A/D converters.

Figure P12-24

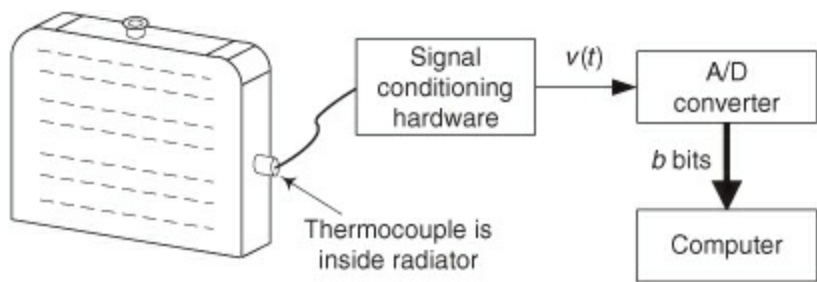


(b) Fill in the following table of important A/D converter quantization error properties in terms of the A/D converters' quantization-level voltage q .

A/D Type	Most negative quantization error	Most positive quantization error	Max peak-peak quantization error
Truncating			
Rounding			

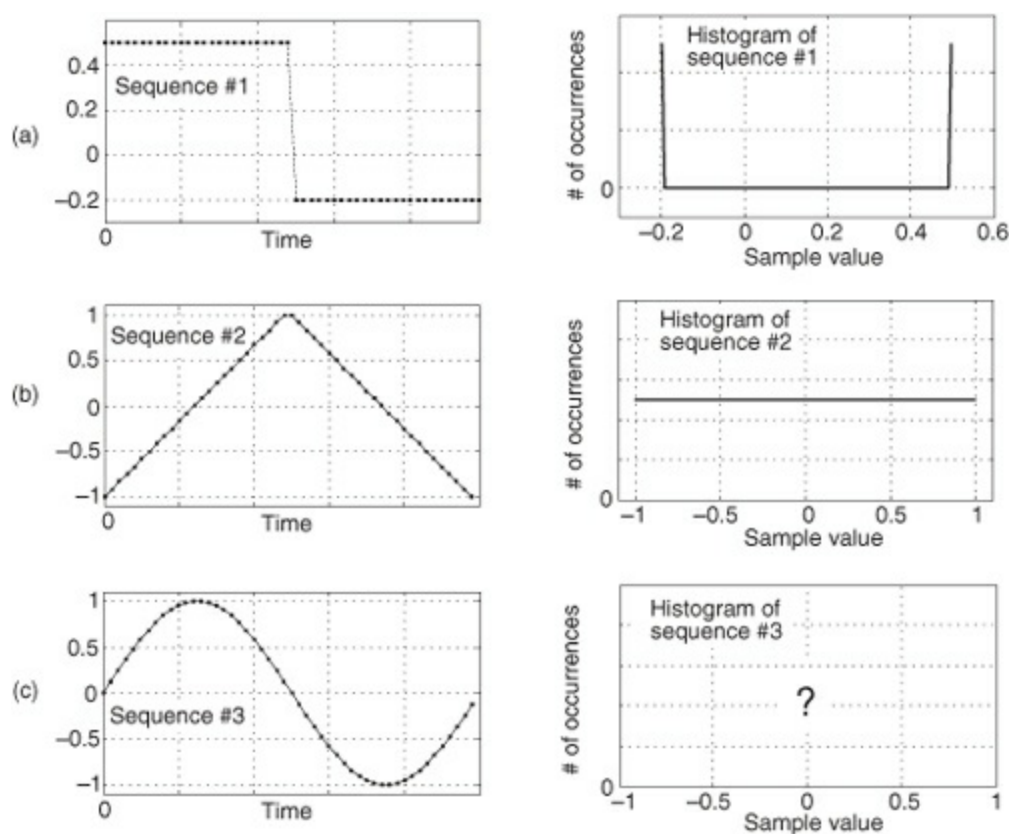
12.25 Assume we want to digitize the output voltage of a temperature measurement system, monitoring the internal temperature of an automobile radiator, as shown in [Figure P12-25](#). The system's manufacturer states that its output voltage $v(t)$ will represent the thermocouple's junction temperature with an accuracy of 2 degrees Fahrenheit (1.1 degrees Celsius), and its operating range covers temperatures as low as just-freezing water to twice the temperature of boiling water. To accommodate the precision and operating range of the temperature measurement system, how many bits, b , do we need for our A/D converter? Show your work.

Figure P12-25



12.26 One useful way to test the performance of A/D converters is to apply a specific analog signal to the A/D converter's analog input and perform a histogram of the converter's output samples. For example, if an analog squarewave-like signal is applied to an A/D converter, the converter's output sequence might be that shown in the left panel of [Figure P12-26\(a\)](#), and the histogram of the converter's output samples is shown in the right panel of [Figure P12-26\(a\)](#). That histogram shows that there are many converter output samples whose values are -0.2 , and many converter output samples whose values are 0.5 , and no sample values other than -0.2 and 0.5 . The shape of the histogram curve will indicate any severe defects in the converter's performance.

Figure P12-26

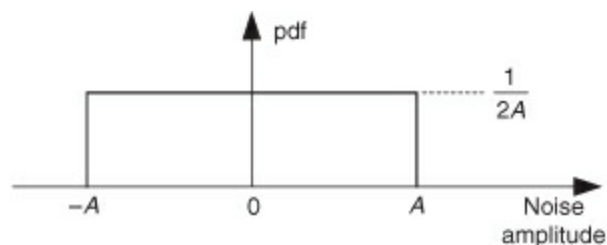


If a triangular analog signal is applied to an A/D converter, the converter's output sequence would be that shown in the left panel of [Figure P12-26\(b\)](#) and the histogram of the converter's output samples is shown in the right panel of [Figure P12-26\(b\)](#). This histogram shows that there are (ideally) an equal number of samples at all amplitudes between -1 and $+1$, which happens to indicate correct converter behavior.

In the testing of high-frequency A/D converters, high-frequency analog square and triangular waves are difficult to generate, so A/D converter engineers use high-frequency analog sinewaves to test their converters. Assuming that an analog sinewave is used as an input for A/D converter histogram testing and the converter output samples are those shown in the left panel of [Figure P12-26\(c\)](#), draw a rough sketch of the histogram of converter output samples.

- 12.27** In the text we discussed how to use the concept of a uniform probability density function (PDF), described in [Section D.3](#) of [Appendix D](#), to help us determine the variance (a measure of power) of random A/D-converter quantization noise. Sometimes we want to generate random noise samples, for testing purposes, that have a uniform PDF such as that shown in [Figure P12-27](#). What is the value of A for a uniform PDF random sequence whose variance is equal to 2?

Figure P12-27



- 12.28** Assume we have a single numerical data sample value in floating-point binary format. What two bit manipulation methods exist to multiply that sample by 4 without using any multiplier hardware circuitry?

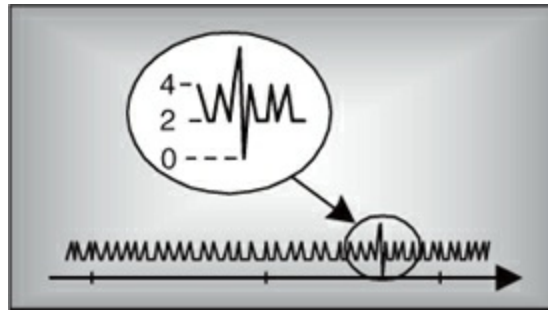
12.29 Convert the following IEEE P754 floating-point number, given here in hexadecimal format, to a decimal number:

\$C2ED0000

Show your work.

Hint: Don't forget to account for the hidden one in the IEEE P754 format.

Chapter Thirteen. Digital Signal Processing Tricks



As we study the literature of digital signal processing, we'll encounter some creative techniques that professionals use to make their algorithms more efficient. These practical techniques are straightforward examples of the philosophy "Don't work hard, work smart," and studying them will give us a deeper understanding of the underlying mathematical subtleties of DSP. In this chapter, we present a collection of these tricks of the trade, in no particular order, and explore several of them in detail because doing so reinforces the lessons we've learned in previous chapters.

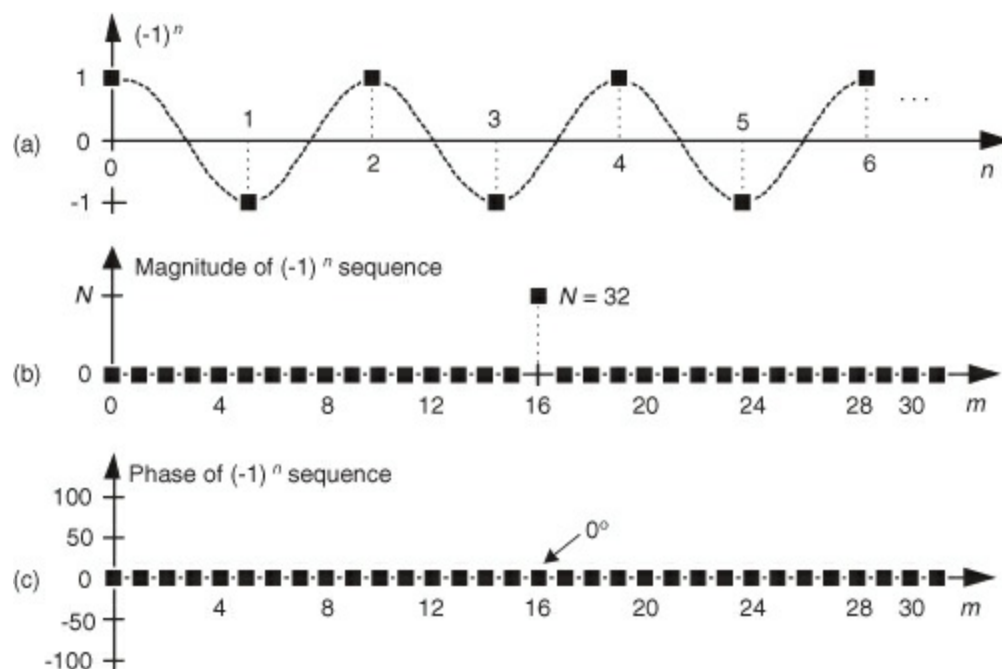
13.1 Frequency Translation without Multiplication

Frequency translation is often called for in digital signal processing algorithms. There are simple schemes for inducing frequency translation by $1/2$ and $1/4$ of the signal sequence sample rate. Let's take a look at these mixing schemes.

13.1.1 Frequency Translation by $f_s/2$

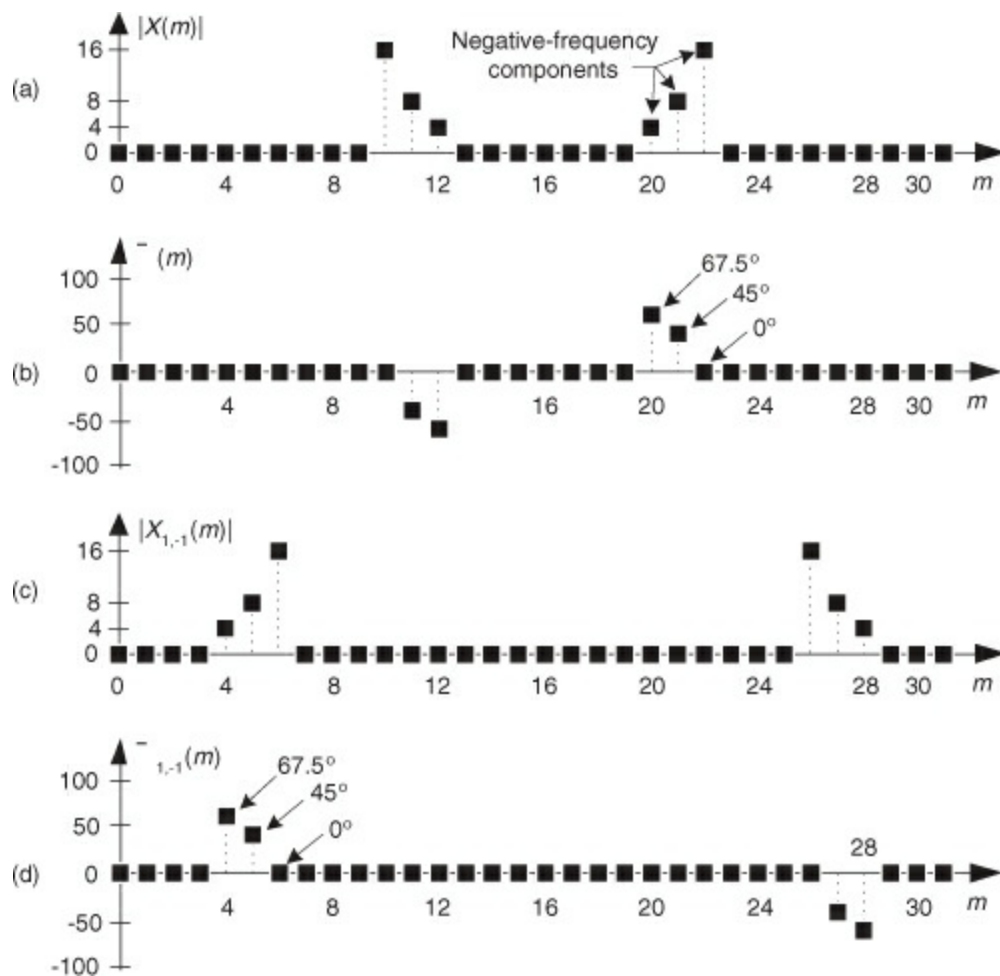
First we'll consider a technique for frequency translating an input sequence by $f_s/2$ by merely multiplying a sequence by $(-1)^n = 1, -1, 1, -1, \dots$, etc., where f_s is the signal sample rate in Hz. This process may seem a bit mysterious at first, but it can be explained in a straightforward way if we review [Figure 13-1\(a\)](#). There we see that multiplying a time-domain signal sequence by the $(-1)^n$ mixing sequence is equivalent to multiplying the signal sequence by a sampled sinusoid where the mixing sequence samples are shown as the dots in [Figure 13-1\(a\)](#). Because the mixing sequence's cosine repeats every two sample values, its frequency is $f_s/2$. [Figures 13-1\(b\)](#) and [13-1\(c\)](#) show the discrete Fourier transform (DFT) magnitude and phase of a 32-sample $(-1)^n$ sequence. As such, the right half of those figures represents the negative frequency range.

Figure 13-1 Mixing sequence comprising $(-1)^n = 1, -1, 1, -1$, etc.: (a) time-domain sequence; (b) frequency-domain magnitudes for 32 samples; (c) frequency-domain phase.



Let's demonstrate this $(-1)^n$ mixing with an example. Consider a real $x(n)$ signal sequence having 32 samples of the sum of three sinusoids whose $|X(m)|$ frequency magnitude and $\phi(m)$ phase spectra are as shown in [Figures 13-2\(a\)](#) and [13-2\(b\)](#). If we multiply that time signal sequence by $(-1)^n$, the resulting $x_{1,-1}(n)$ time sequence will have the magnitude and phase spectra that are shown in [Figures 13-2\(c\)](#) and [13-2\(d\)](#). Multiplying a time signal by our $(-1)^n$ cosine shifts half its spectral energy up by $f_s/2$ and half its spectral energy down by $-f_s/2$. Notice in these non-circular frequency depictions that as we count up, or down, in frequency, we wrap around the end points.

Figure 13-2 A signal and its frequency translation by $f_s/2$: (a) original signal magnitude spectrum; (b) original phase; (c) the magnitude spectrum of the translated signal; (d) translated phase.



Here's a terrific opportunity for the DSP novice to convolve the $(-1)^n$ spectrum in [Figure 13-1](#) with the $X(m)$ spectrum to obtain the frequency-translated $X_{1,-1}(m)$ signal spectrum. Please do so; that exercise will help you comprehend the nature of discrete sequences and their time- and frequency-domain relationships by way of the convolution theorem.

Remember, now, we didn't really perform any explicit multiplications—the whole idea here is to avoid multiplications; we merely changed the sign of alternating $x(n)$ samples to get $x_{1,-1}(n)$. One way to look at the $X_{1,-1}(m)$ magnitudes in [Figure 13-2\(c\)](#) is to see that multiplication by the $(-1)^n$ mixing sequence flips the positive-frequency band of $X(m)$ ($X(0)$ to $X(16)$) about the $f_s/4$ Hz point and flips the negative-frequency band of $X(m)$ ($X(17)$ to $X(31)$) about the $-f_s/4$ Hz sample. This process can be used to invert the spectra of real signals when bandpass sampling is used as described in [Section 2.4](#). By the way, in the DSP literature be aware that some clever authors may represent the $(-1)^n$ sequence with its equivalent expressions of

$$(13-1)$$

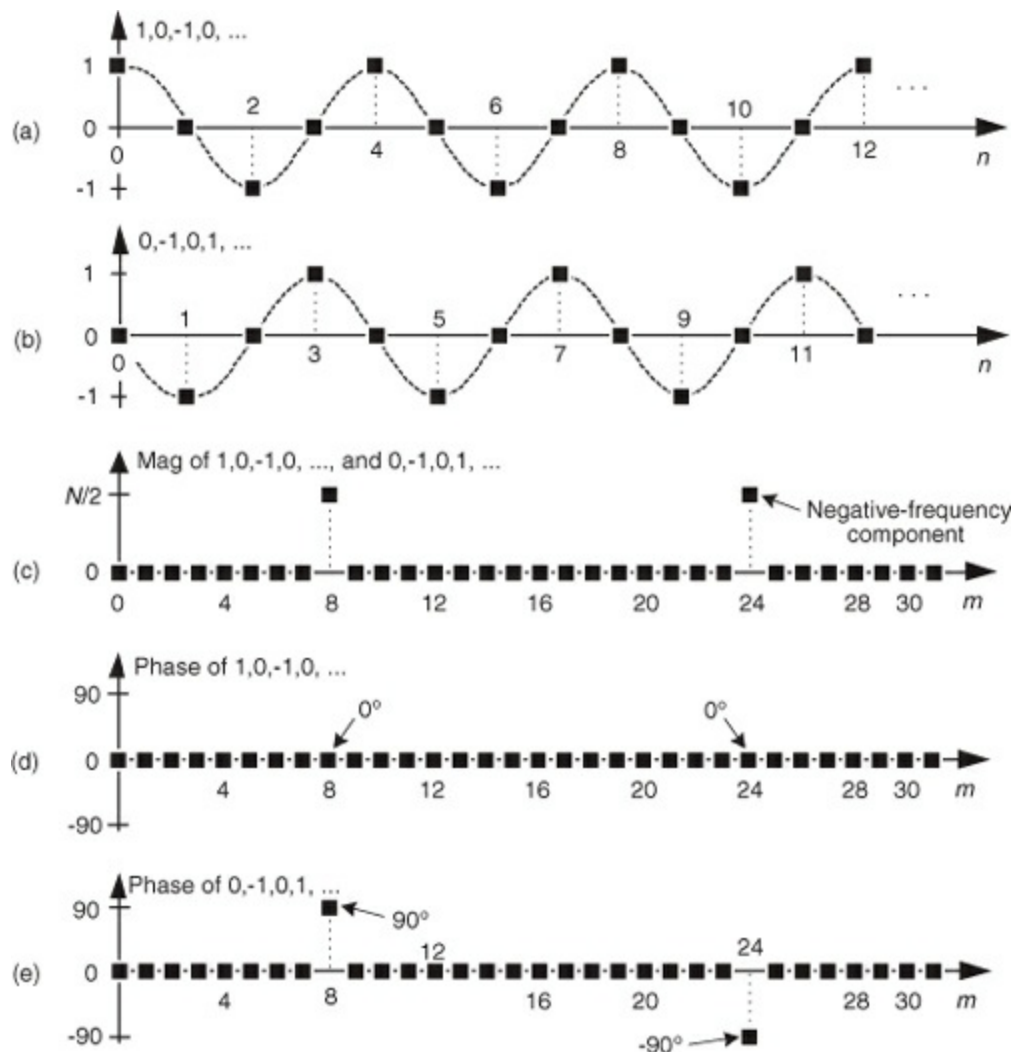
$$(-1)^n = \cos(\pi n) = e^{j\pi n}.$$

13.1.2 Frequency Translation by $-f_s/4$

Two other simple mixing sequences form the real and imaginary parts of a complex $-f_s/4$ oscillator used for frequency down-conversion to obtain a quadrature version (complex and centered at 0 Hz) of a real bandpass signal originally centered at $f_s/4$. The real (in-phase) mixing sequence is $\cos(\pi n/2) = 1, 0, -1, 0$, etc., shown in [Figure 13-3\(a\)](#). That mixing sequence's quadrature companion is $-\sin(\pi n/2) = 0, -1, 0, 1$, etc., as shown in [Figure 13-3\(b\)](#). The spectral magnitudes of those two sequences are

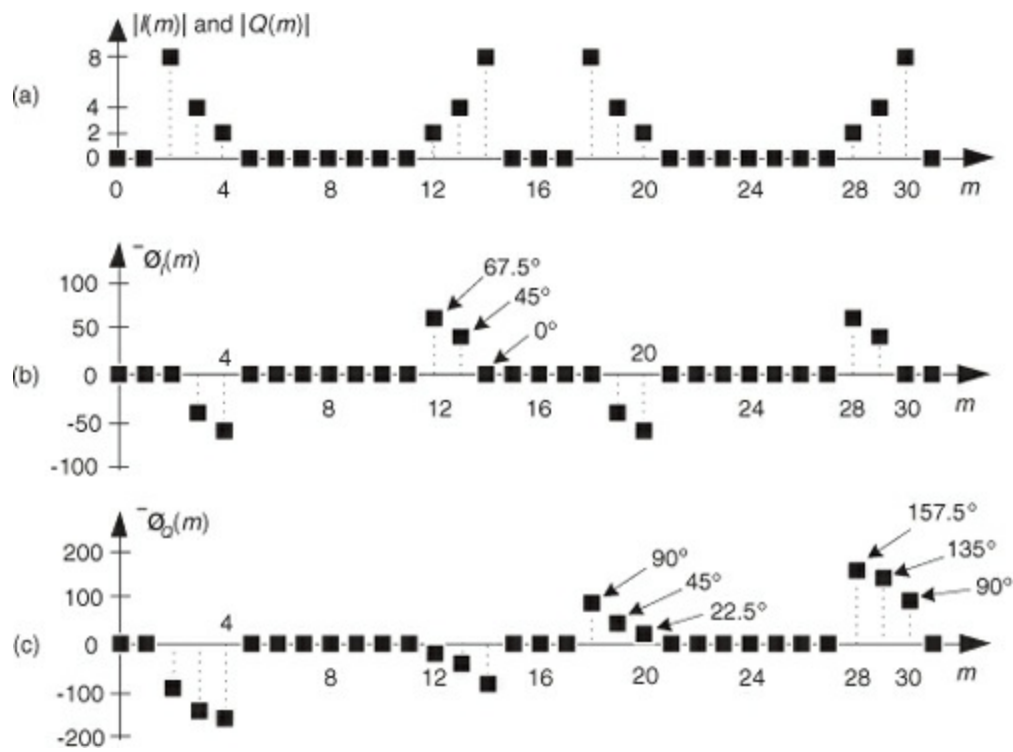
identical as shown in [Figure 13-3\(c\)](#), but their phase spectrum has a 90-degree shift relationship (what we call *quadrature*).

Figure 13-3 Quadrature mixing sequences for down-conversion by $f_s/4$: (a) in-phase mixing sequence; (b) quadrature-phase mixing sequence; (c) the frequency magnitudes of both sequences for $N = 32$ samples; (d) the phase of the cosine sequence; (e) phase of the sine sequence.



If we multiply the $x(n)$ sequence whose spectrum is that shown in [Figures 13-2\(a\)](#) and [13-2\(b\)](#) by the in-phase (cosine) mixing sequence, the product will have the $I(m)$ spectrum shown in [Figures 13-4\(a\)](#) and [13-4\(b\)](#). Again, $X(m)$'s spectral energy is translated up and down in frequency, only this time the translation is by $\pm f_s/4$. Multiplying $x(n)$ by the quadrature-phase (sine) sequence yields the $Q(m)$ spectrum in [Figures 13-4\(a\)](#) and [13-4\(c\)](#).

Figure 13-4 Spectra after translation down by $f_s/4$: (a) $I(m)$ and $Q(m)$ spectral magnitudes; (b) phase of $I(m)$; (c) phase of $Q(m)$.



Because their time sample values are merely 1, -1, and 0, the quadrature mixing sequences are useful because down-conversion by $f_s/4$ can be implemented without multiplication. That's why these mixing sequences are of so much interest: down-conversion of an input time sequence is accomplished merely with data assignment, or signal routing.

To down-convert a general $x(n) = x_{\text{real}}(n) + jx_{\text{imag}}(n)$ sequence by $f_s/4$, the value assignments are

(13-2)

$$\begin{aligned} x_{\text{new}}(0) &= x_{\text{real}}(0) + jx_{\text{imag}}(0) \\ x_{\text{new}}(1) &= x_{\text{imag}}(1) - jx_{\text{real}}(1) \\ x_{\text{new}}(2) &= -x_{\text{real}}(2) - jx_{\text{imag}}(2) \\ x_{\text{new}}(3) &= -x_{\text{imag}}(3) + jx_{\text{real}}(3) \\ &\text{repeat for down-conversion ...} \end{aligned}$$

If your implementation is hardwired gates, the above data assignments are performed by means of routing signals (and their negatives). Although we've focused on down-conversion so far, it's worth mentioning that up-conversion of a general $x(n)$ sequence by $f_s/4$ can be performed with the following data assignments:

(13-3)

$$\begin{aligned} x_{\text{new}}(0) &= x_{\text{real}}(0) + jx_{\text{imag}}(0) \\ x_{\text{new}}(1) &= -x_{\text{imag}}(1) + jx_{\text{real}}(1) \\ x_{\text{new}}(2) &= -x_{\text{real}}(2) - jx_{\text{imag}}(2) \\ x_{\text{new}}(3) &= x_{\text{imag}}(3) - jx_{\text{real}}(3) \\ &\text{repeat for up-conversion ...} \end{aligned}$$

We notify the reader, at this point, that [Section 13.29](#) presents an interesting trick for performing frequency translation using decimation rather than multiplication.

13.1.3 Filtering and Decimation after $f_s/4$ Down-Conversion

There's an efficient way to perform the complex down-conversion, by $f_s/4$, and filtering of a real signal process that we discussed for the quadrature sampling scheme in [Section 8.9](#). We can use a

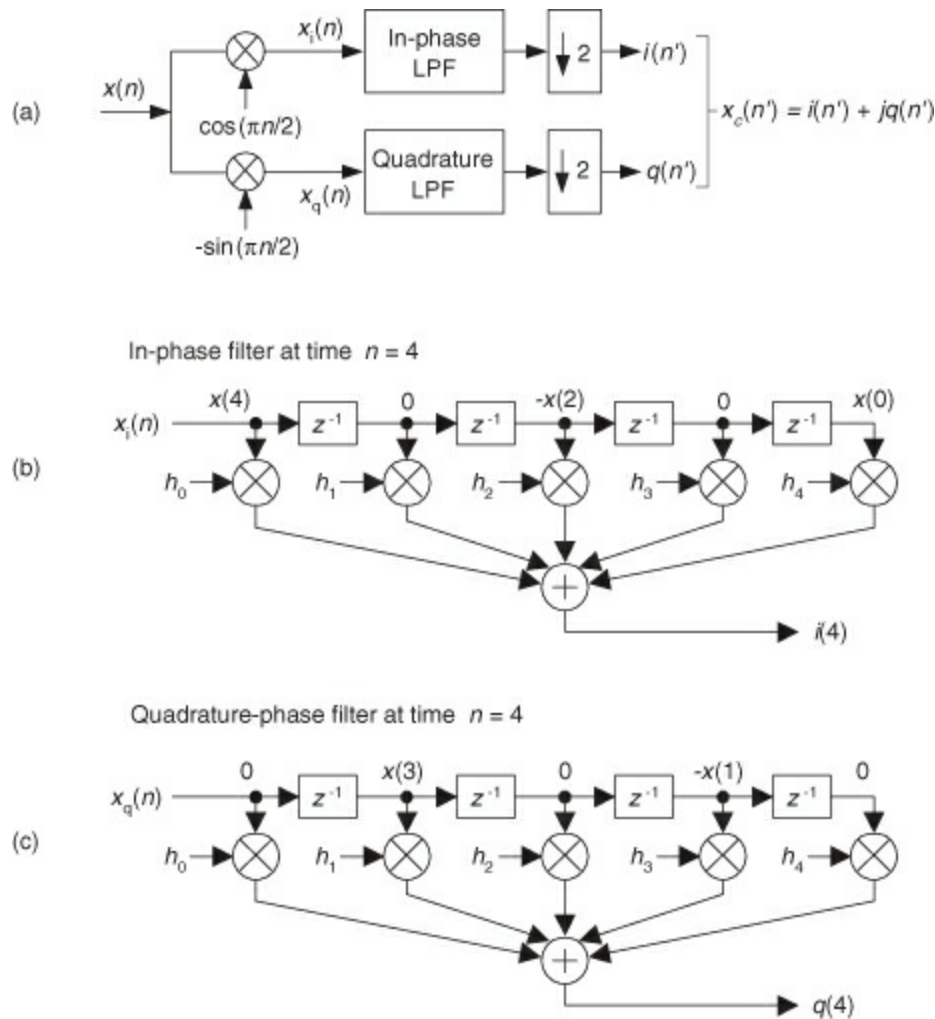
novel technique to greatly reduce the computational workload of the linear-phase lowpass filters[1–3]. In addition, decimation of the complex down-converted sequence by a factor of two is inherent, with no effort on our part, in this process.

Considering [Figure 13-5\(a\)](#), notice that if an original $x(n)$ sequence was real-only, and its spectrum is centered at $f_s/4$, multiplying $x(n)$ by $\cos(\pi n/2) = 1, 0, -1, 0$, for the in-phase path and $-\sin(\pi n/2) = 0, -1, 0, 1$, for the quadrature-phase path to down-convert $x(n)$'s spectrum to 0 Hz yields the new complex sequence $x_{\text{new}}(n) = x_i(n) + jx_q(n)$, or

$$(13-4)$$

$$\begin{aligned} x_{\text{new}}(0) &= x(0) + j0 \\ x_{\text{new}}(1) &= 0 - jx(1) \\ x_{\text{new}}(2) &= -x(2) + j0 \\ x_{\text{new}}(3) &= 0 + jx(3) \\ x_{\text{new}}(4) &= x(4) + j0 \\ x_{\text{new}}(5) &= 0 - jx(5) \\ &\text{repeat ...} \end{aligned}$$

Figure 13-5 Complex down-conversion by $f_s/4$ and filtering by a 5-tap LPF: (a) the process; (b) in-phase filter data; (c) quadrature-phase filter data.



Next, we want to lowpass filter (LPF) both the $x_i(n)$ and $x_q(n)$ sequences followed by decimation by a factor of two.

Here's the trick. Let's say we're using 5-tap FIR filters and at the $n = 4$ time index the data residing in the two lowpass filters would be that shown in [Figures 13-5\(b\)](#) and [13-5\(c\)](#). Due to the alternating

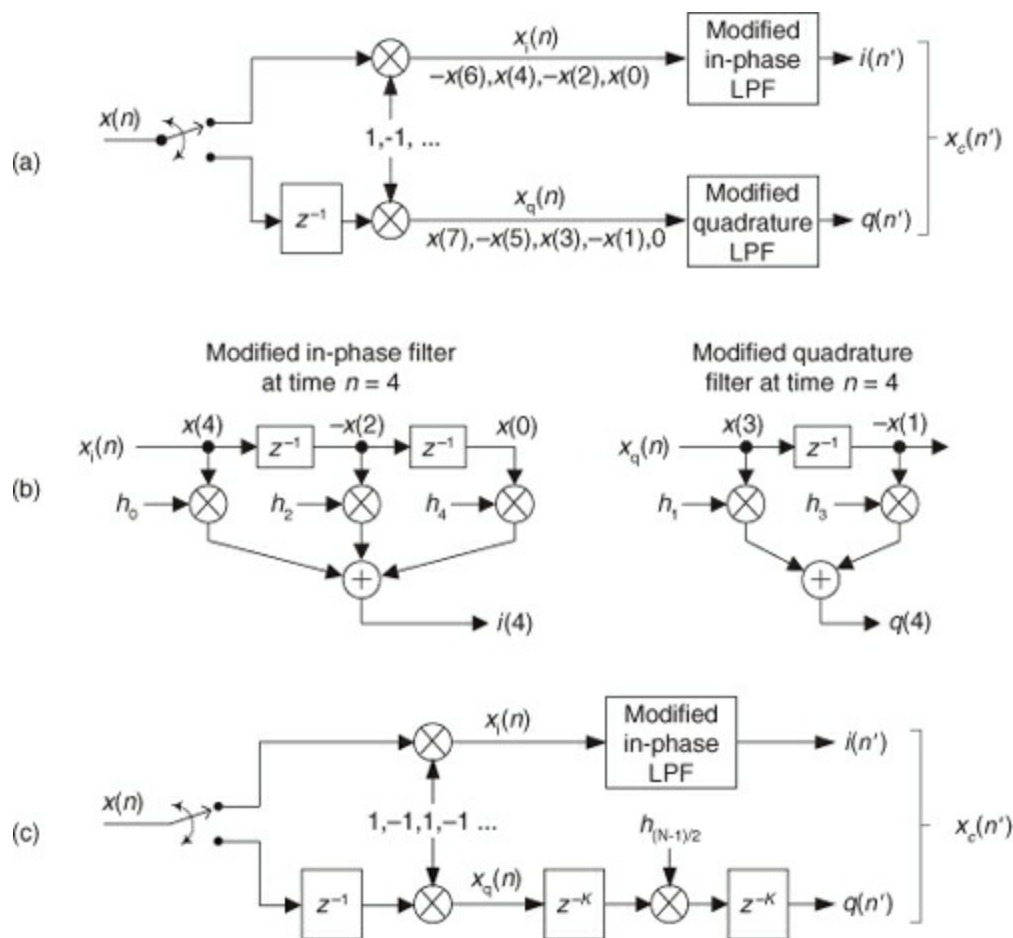
zero-valued samples in the $x_i(n)$ and $x_q(n)$ sequences, we see that only five nonzero multiplies are being performed at this time instant. Those computations, at time index $n = 4$, are shown in the third row of the rightmost column in [Table 13-1](#). Because we're decimating by two, we ignore the time index $n = 5$ computations. The necessary computations during the next time index ($n = 6$) are given in the fourth row of [Table 13-1](#), where again only five nonzero multiplies are computed.

Table 13-1 Filter Data and Necessary Computations after Decimation by Two

Time	Data in the filters					Necessary computations
$n = 0$	$x(0)$	—	—	—	—	$i(0) = x(0)h_0$
	h_0	h_1	h_2	h_3	h_4	
	0	—	—	—	—	$q(0) = 0$
$n = 2$	$-x(2)$	0	$x(0)$	—	—	$i(2) = x(0)h_2 - x(2)h_0$
	h_0	h_1	h_2	h_3	h_4	
	0	$-x(1)$	0	—	—	$q(2) = -x(1)h_1$
$n = 4$	$x(4)$	0	$-x(2)$	0	$x(0)$	$i(4) = x(0)h_4 - x(2)h_2 + x(4)h_0$
	h_0	h_1	h_2	h_3	h_4	
	0	$x(3)$	0	$-x(1)$	0	$q(4) = -x(1)h_3 + x(3)h_1$
$n = 6$	$-x(6)$	0	$x(4)$	0	$-x(2)$	$i(6) = -x(2)h_4 + x(4)h_2 - x(6)h_0$
	h_0	h_1	h_2	h_3	h_4	
	0	$-x(5)$	0	$x(3)$	0	$q(6) = x(3)h_3 - x(5)h_1$
$n = 8$	$x(8)$	0	$-x(6)$	0	$x(4)$	$i(8) = x(4)h_4 - x(6)h_2 + x(8)h_0$
	h_0	h_1	h_2	h_3	h_4	
	0	$x(7)$	0	$-x(5)$	0	$q(8) = -x(5)h_3 + x(7)h_1$

A review of [Table 13-1](#) tells us we can multiplex the real-valued $x(n)$ sequence, multiply the multiplexed sequences by the repeating mixing sequence 1, -1, ..., etc., and apply the resulting $x_i(n)$ and $x_q(n)$ sequences to two filters, as shown in [Figure 13-6\(a\)](#). Those two filters have *decimated* coefficients in the sense that their coefficients are the alternating $h(k)$ coefficients from the original lowpass filter in [Figure 13-5](#). The two new filters are depicted in [Figure 13-6\(b\)](#), showing the necessary computations at time index $n = 4$. Using this new process, we've reduced our multiplication workload by a factor of two. The original data multiplexing in [Figure 13-6\(a\)](#) is what implemented our desired decimation by two.

Figure 13-6 Efficient down-conversion, filtering by a 5-tap LPF, and decimation: (a) process block diagram; (b) the modified filters and data at time $n = 4$; (c) process when a half-band filter is used.



Here's another feature of this efficient down-conversion structure. If half-band filters are used in [Figure 13-5\(a\)](#), then only one of the coefficients in the modified quadrature lowpass filter is nonzero. This means we can implement the quadrature-path filtering as K unit delays, a single multiply by the original half-band filter's center coefficient, followed by another K delay as depicted in [Figure 13-6\(c\)](#). For an original N -tap half-band filter, K is the integer part of $N/4$. If the original half-band filter's $h(N-1)/2$ center coefficient is 0.5, as is often the case, we can implement its multiply by an arithmetic right shift of the delayed $x_q(n)$.

This down-conversion process is indeed slick. Here's another attribute. If the original lowpass filter in [Figure 13-5\(a\)](#) has an odd number of taps, the coefficients of the modified filters in [Figure 13-6\(b\)](#) will be symmetrical, and we can use the *folded* FIR filter scheme ([Section 13.7](#)) to reduce the number of multipliers by almost another factor of two!

Finally, if we need to invert the output $x_c(n')$ spectrum, there are two ways to do so. We can negate the 1, -1, sequence driving the mixer in the quadrature path, or we can swap the order of the single unit delay and the mixer in the quadrature path.

13.2 High-Speed Vector Magnitude Approximation

The quadrature processing techniques employed in spectrum analysis, computer graphics, and digital communications routinely require high-speed determination of the magnitude of a complex number (vector V) given its real and imaginary parts, i.e., the in-phase part I and the quadrature-phase part Q . This magnitude calculation requires a square root operation because the magnitude of V is

$$(13-5) \quad |V| = \sqrt{I^2 + Q^2}.$$

Assuming that the sum $I^2 + Q^2$ is available, the problem is to efficiently perform the square root computation.

There are several ways to obtain square roots, but the optimum technique depends on the capabilities of the available hardware and software. For example, when performing a square root using a high-level software language, we employ whatever software square root function is available. Accurate software square root routines, however, require many floating-point arithmetic computations. In contrast, if a system must accomplish a square root operation in just a few system clock cycles, high-speed magnitude approximations are required[4,5]. Let's look at a neat magnitude approximation scheme that avoids the dreaded square root operation.

There is a technique called the $\alpha\text{Max} + \beta\text{Min}$ (read as “alpha max plus beta min”) algorithm for estimating the magnitude of a complex vector.[‡] It's a linear approximation to the vector magnitude problem that requires the determination of which orthogonal vector, I or Q , has the greater absolute value. If the maximum absolute value of I or Q is designated by Max, and the minimum absolute value of either I or Q is Min, an approximation of $|V|$ using the $\alpha\text{Max} + \beta\text{Min}$ algorithm is expressed as

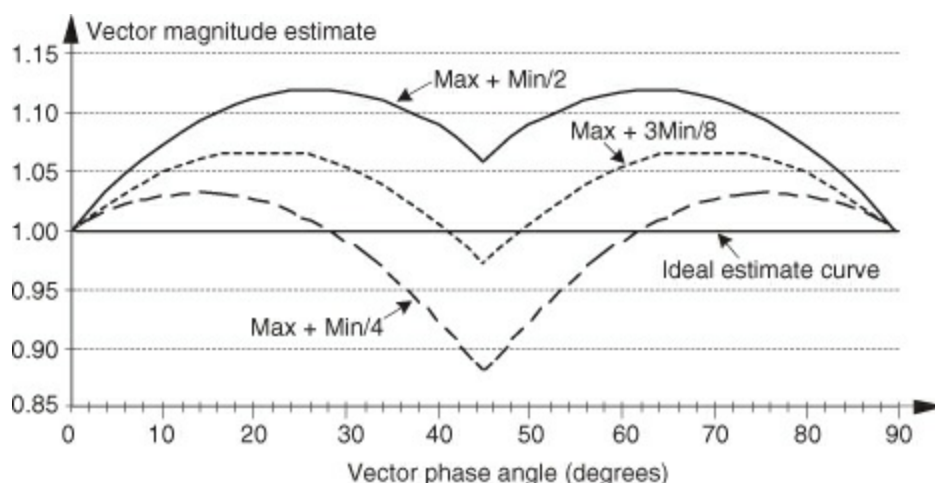
[‡]A “Max+ β Min” algorithm had been in use, but in 1988 this author suggested expanding it to the $\alpha\text{Max} + \beta\text{Min}$ form where α could be a value other than unity[6].

$$(13-6)$$

$$|V| \approx \alpha\text{Max} + \beta\text{Min}.$$

There are several pairs for the α and β constants that provide varying degrees of vector magnitude approximation accuracy to within 0.1 dB[4,7]. The $\alpha\text{Max} + \beta\text{Min}$ algorithms in reference [8] determine a vector magnitude at whatever speed it takes a system to perform a magnitude comparison, two multiplications, and one addition. But those algorithms require, as a minimum, a 16-bit multiplier to achieve reasonably accurate results. If, however, hardware multipliers are not available, all is not lost. By restricting the α and β constants to reciprocals of integer powers of two, Eq. (13-6) lends itself well to implementation in binary integer arithmetic. A prevailing application of the $\alpha\text{Max} + \beta\text{Min}$ algorithm uses $\alpha = 1.0$ and $\beta = 0.5$. The 0.5 multiplication operation is performed by shifting the value Min to the right by one bit. We can gauge the accuracy of any vector magnitude estimation algorithm by plotting its $|V|$ as a function of vector phase angle. Let's do that. The Max + 0.5Min estimate for a complex vector of unity magnitude, over the vector angular range of 0 to 90 degrees, is shown as the solid curve in Figure 13-7. (The curves in Figure 13-7 repeat every 90 degrees.)

Figure 13-7 $\alpha\text{Max} + \beta\text{Min}$ estimation performance.

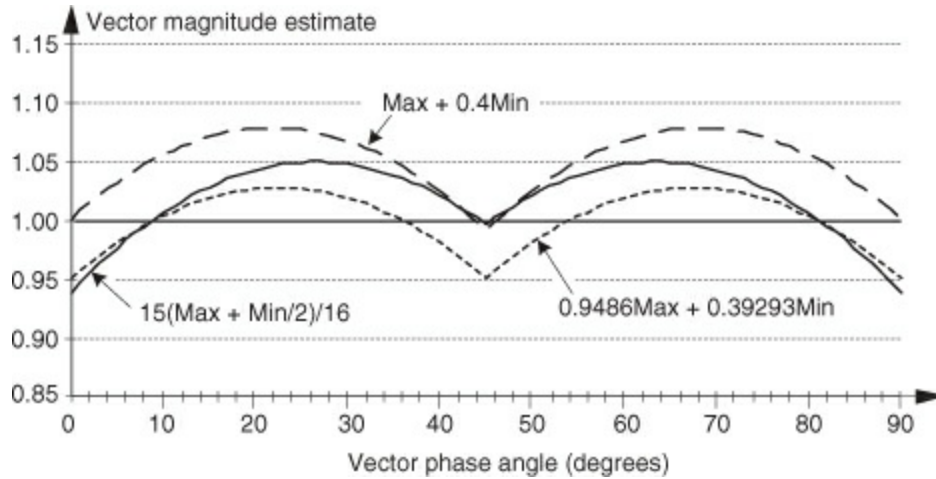


An ideal estimation curve for a unity magnitude vector would have a value of one, and we'll use this

ideal curve as a yardstick to measure the merit of various $\alpha\text{Max}+\beta\text{Min}$ algorithms. Let's make sure we know what the solid curve in [Figure 13-7](#) is telling us. That curve indicates that a unity magnitude vector oriented at an angle of approximately 26 degrees will be estimated by [Eq. \(13-6\)](#) to have a magnitude of 1.118 instead of the correct magnitude of one. The error then, at 26 degrees, is 11.8 percent. For comparison, two other magnitude approximation curves for various values of α and β are shown in [Figure 13-7](#).

Although the values for α and β in [Figure 13-7](#) yield somewhat accurate vector magnitude estimates, there are other values for α and β that deserve our attention because they result in smaller magnitude estimation errors. The $\alpha = 15/16$ and $\beta = 15/32$ solid curve in [Figure 13-8](#) is an example of a reduced-error algorithm. Multiplications by those values of α and β can be performed by multiplying by 15 and using binary right shifts to implement the divisions by 16 and 32. A mathematically simple, single-multiply, $\alpha = 1$ and $\beta = 0.4$ algorithm is also shown as the dashed curve[\[9\]](#). For the interested reader, the performance of the optimum values for α and β is shown as the dotted curve in [Figure 13-8](#). (The word *optimum*, as used here, means minimizing the magnitude estimation error fluctuations both above and below the ideal unity line.)

Figure 13-8 Alternate $\alpha\text{Max}+\beta\text{Min}$ algorithm performance.



To add to our catalog of magnitude estimation algorithms, at the expense of an additional multiply/shift and a compare operation, an accurate magnitude estimation scheme is that defined by [Eq. \(13-7\)](#)[\[10\]](#):

(13-7)

$$|V| = \begin{cases} \text{Max} + \text{Min}/8, & \text{if } \text{Min} < 3\text{Max}/8 \\ 27\text{Max}/32 + 19\text{Min}/16, & \text{if } \text{Min} \geq 3\text{Max}/8. \end{cases}$$

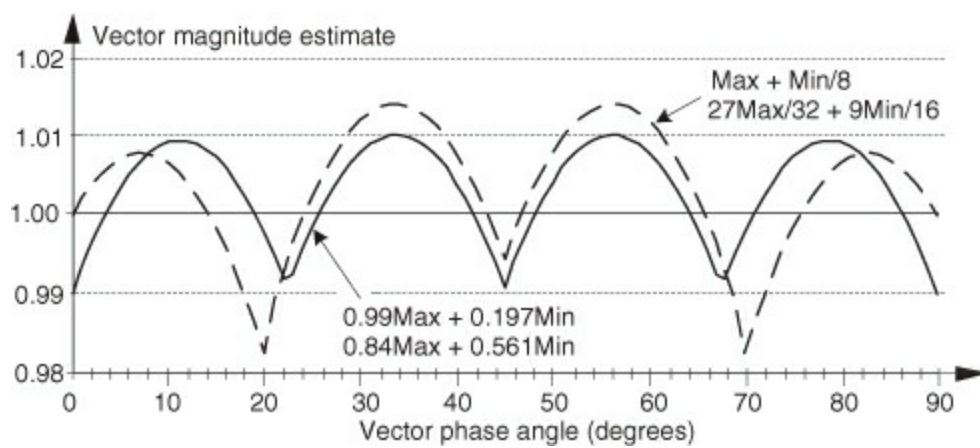
Again, the divisions in [Eq. \(13-7\)](#) are implemented as binary right shifts. In a similar vein we mention an algorithm that exhibits a maximum error of a mere 1 percent, when floating-point arithmetic is used, as defined by [Eq. \(13-7'\)](#)[\[11\]](#):

(13-7')

$$|V| = \begin{cases} 0.99\text{Max} + 0.197\text{Min}, & \text{if } \text{Min} < 0.4142135\text{Max} \\ 0.84\text{Max} + 0.561\text{Min}, & \text{if } \text{Min} \geq 0.4142135\text{Max}. \end{cases}$$

The performance curves of the last two magnitude estimation algorithms are shown in [Figure 13-9](#).

Figure 13-9 Additional $\alpha\text{Max}+\beta\text{Min}$ algorithm performance.



To summarize the behavior of the magnitude estimation algorithms we just covered so far, the relative performances of the various algorithms are shown in [Table 13-2](#). The table lists the magnitude of the algorithms' maximum error in both percent and decibels. The rightmost column of [Table 13-2](#) is the mean squared error (MSE) of the algorithms. That MSE value indicates how much the algorithms' results fluctuate about the ideal result of one, and we'd like to have that MSE value be as close to zero (a flat line) as possible.

Table 13-2 $\alpha\text{Max}+\beta\text{Min}$ Algorithm Performance Comparisons

Algorithm	Maximum error (%)	Maximum error (dB)	Mean squared error ($\times 10^{-3}$)
Max + Min/2	11.8%	0.97 dB	8.4851
Max + Min/4	11.6%	0.95 dB	1.7302
Max + 3Min/8	6.8%	0.57 dB	2.2682
15(Max + Min/2)/16	6.3%	0.53 dB	1.2412
0.9486Max + 0.39293Min	5.1%	0.43 dB	0.5773
Max + 0.4Min	7.7%	0.64 dB	3.0573
Max + Min/8, 27Max/32 + 19Min/16	1.8%	0.16 dB	0.0744
0.99Max + 0.197Min, 0.84Max + 0.561Min	1.0%	0.09 dB	0.0456

So, the $\alpha\text{Max}+\beta\text{Min}$ algorithms enable high-speed vector magnitude computation without the need for performing square root operations. Of course, with the availability of floating-point multiplier integrated circuits—with their ability to multiply in one or two clock cycles—the α and β coefficients need not always be restricted to multiples of reciprocals of integer powers of two.

13.3 Frequency-Domain Windowing

There's an interesting technique for minimizing the calculations necessary to implement windowing of FFT input data to reduce spectral leakage. There are times when we need the FFT of unwindowed time-domain data, while at the same time we also want the FFT of that same time-domain data with a window function applied. In this situation, we don't have to perform two separate FFTs. We can perform the FFT of the unwindowed data, and then we can perform frequency-domain windowing on that FFT result to reduce leakage. Let's see how.

Recall from [Section 3.9](#) that the expressions for the Hanning and the Hamming windows were $w_{\text{Han}}(n) = 0.5 - 0.5\cos(2\pi n/N)$ and $w_{\text{Ham}}(n) = 0.54 - 0.46\cos(2\pi n/N)$, respectively, where N is a

window sequence length. They both have the general cosine function form of

$$(13-8)$$

$$w(n) = \alpha - \beta \cos(2\pi n/N)$$

for $n = 0, 1, 2, \dots, N-1$. Looking at the frequency response of the general cosine window function, using the definition of the DFT, the transform of [Eq. \(13-8\)](#) is

$$(13-9)$$

$$W(m) = \sum_{n=0}^{N-1} [\alpha - \beta \cos(2\pi n/N)] e^{-j2\pi nm/N}$$

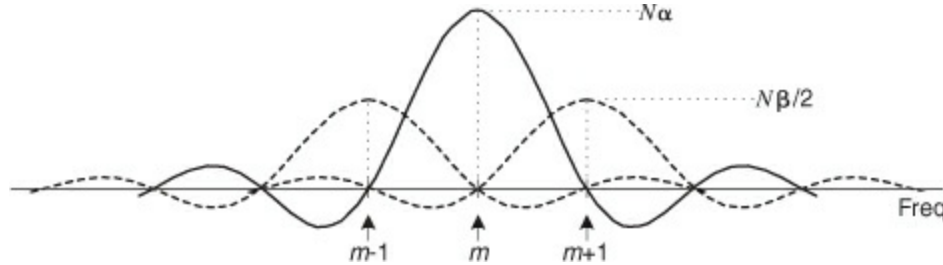
Because $\cos(2\pi n/N) = \frac{e^{j2\pi n/N} + e^{-j2\pi n/N}}{2}$, [Eq. \(13-9\)](#) can be written as

$$(13-10)$$

$$\begin{aligned} W(m) &= \sum_{n=0}^{N-1} \alpha e^{-j2\pi nm/N} - \frac{\beta}{2} \sum_{n=0}^{N-1} e^{j2\pi n/N} e^{-j2\pi nm/N} - \frac{\beta}{2} \sum_{n=0}^{N-1} e^{-j2\pi n/N} e^{-j2\pi nm/N} \\ &= \alpha \sum_{n=0}^{N-1} e^{-j2\pi nm/N} - \frac{\beta}{2} \sum_{n=0}^{N-1} e^{j2\pi n(m-1)/N} - \frac{\beta}{2} \sum_{n=0}^{N-1} e^{-j2\pi n(m+1)/N} \end{aligned}$$

[Equation \(13-10\)](#) looks pretty complicated, but using the derivation from [Section 3.13](#) for expressions like those summations, we find that [Eq. \(13-10\)](#) merely results in the superposition of three $\sin(x)/x$ functions in the frequency domain. Their amplitudes are shown in [Figure 13-10](#).

Figure 13-10 General cosine window frequency response amplitude.



Notice that the two translated $\sin(x)/x$ functions have sidelobes with opposite phase from that of the center $\sin(x)/x$ function. This means that $N\alpha$ times the m th bin output, minus $N\beta/2$ times the $(m-1)$ th bin output, minus $\beta/2$ times the $(m+1)$ th bin output will minimize the sidelobes of the m th bin. This frequency-domain convolution process is equivalent to multiplying the input time data sequence by the N -valued window function $w(n)$ in [Eq. \(13-8\)](#) [[12–14](#)].

For example, let's say the output of the m th FFT bin is $X(m) = a_m + jb_m$, and the outputs of its two neighboring bins are $X(m-1) = a_{-1} + jb_{-1}$ and $X(m+1) = a_{+1} + jb_{+1}$. Then frequency-domain windowing for the m th bin of the unwindowed $X(m)$ is as follows:

$$(13-11)$$

$$\begin{aligned} X_{\text{three-term}}(m) &= \alpha X(m) - \frac{\beta}{2} X(m-1) - \frac{\beta}{2} X(m+1) \\ &= \alpha(a_m + jb_m) - \frac{\beta}{2}(a_{-1} + jb_{-1}) - \frac{\beta}{2}(a_{+1} + jb_{+1}) \\ &= \alpha a_m - \frac{\beta}{2}(a_{-1} + a_{+1}) + j[\alpha b_m - \frac{\beta}{2}(b_{-1} + b_{+1})] \end{aligned}$$

To compute a windowed N -point FFT, $X_{\text{three-term}}(m)$, we can apply [Eq. \(13-11\)](#), requiring $4N$ additions and $3N$ multiplications, to the unwindowed N -point FFT result $X(m)$ and avoid having to perform the N multiplications of time-domain windowing and a second FFT with its $M\log_2(N)$ additions and $2M\log_2(N)$ multiplications. (In this case, we called our windowed results $X_{\text{three-term}}(m)$ because we're performing a convolution of a three-term $W(m)$ sequence with the $X(m)$ sequence.)

To accommodate the $m = 0$ beginning and the $m = N-1$ end of our N -point FFT, we effectively wrap the FFT samples back on themselves. That is, due to the circular nature of FFT samples based on real-valued time sequences, we use

$$(13-11')$$

$$X_{\text{three-term}}(0) = \alpha X(0) - \frac{\beta}{2} X(N-1) - \frac{\beta}{2} X(1)$$

and

$$(13-11'')$$

$$X_{\text{three-term}}(N-1) = \alpha X(N-1) - \frac{\beta}{2} X(N-2) - \frac{\beta}{2} X(0).$$

Now if the FFT's $x(n)$ input sequence is real-only, then $X(0) = a_0$, and [Eq. \(13-11'\)](#) simplifies to a real-only $X_{\text{three-term}}(0) = \alpha a_0 - \beta a_1$.

The neat situation here is the frequency-domain coefficients, values, α and β , for the Hanning window. They're both 0.5, and the multiplications in [Eq. \(13-11\)](#) can be performed in hardware with two binary right shifts by a single bit for $\alpha = 0.5$ and two shifts for each of the two $\beta/2 = 0.25$ factors, for a total of six binary shifts. If a gain of four is acceptable, we can get away with only two left shifts (one for the real and one for the imaginary parts of $X(m)$) using

$$(13-12)$$

$$X_{\text{Hanning, gain=4}}(m) = 2X(m) - X(m-1) - X(m+1).$$

In *application-specific integrated circuit* (ASIC) and *field-programmable gate array* (FPGA) hardware implementations, where multiplies are to be avoided, the binary shifts can be eliminated through hardwired data routing. Thus only additions are necessary to implement frequency-domain Hanning windowing. The issues we need to consider are which window function is best for the application, and the efficiency of available hardware in performing the frequency-domain multiplications. Frequency-domain Hamming windowing can be implemented but, unfortunately, not with simple binary shifts.

Along with the Hanning and Hamming windows, reference [\[14\]](#) describes a family of windows known as *Blackman* windows that provide further FFT spectral leakage reduction when performing frequency-domain windowing. (Note: Reference [\[14\]](#) reportedly has two typographical errors in the 4-Term (-74 dB) window coefficients column on its page 65. Reference [\[15\]](#) specifies those coefficients to be 0.40217, 0.49703, 0.09892, and 0.00188.) Blackman windows have five nonzero frequency-domain coefficients, and their use requires the following five-term convolution:

$$(13-13)$$

$$X_{\text{five-term}}(m) = \alpha X(m) + \frac{\gamma}{2} X(m-2) - \frac{\beta}{2} X(m-1) - \frac{\beta}{2} X(m+1) + \frac{\gamma}{2} X(m+2).$$

[Table 13-3](#) provides the frequency-domain coefficients for several common window functions.

Table 13-3 Frequency-Domain Windowing Coefficients

Window function	α	β	γ
Rectangular	1.0	—	—
Hanning	0.5	0.5	—
Hamming	0.54	0.46	—
Blackman	0.42	0.5	0.08
Exact Blackman	$\frac{7938}{18608}$	$\frac{9240}{18608}$	$\frac{1430}{18608}$
3-term Blackman-Harris	0.42323	0.49755	0.07922

Let's end our discussion of the frequency-domain windowing trick by saying this scheme can be efficient because we don't have to window the entire set of FFT data; windowing need only be performed on those FFT bin outputs of interest to us. An application of frequency-domain windowing is presented in [Section 13.18](#).

13.4 Fast Multiplication of Complex Numbers

The multiplication of two complex numbers is one of the most common functions performed in digital signal processing. It's mandatory in all discrete and fast Fourier transformation algorithms, necessary for graphics transformations, and used in processing digital communications signals. Be it in hardware or software, it's always to our benefit to streamline the processing necessary to perform a complex multiply whenever we can. If the available hardware can perform three additions faster than a single multiplication, there's a way to speed up a complex multiply operation[\[16\]](#).

The multiplication of two complex numbers, $a + jb$ and $c + jd$, results in the complex product

(13-14)

$$R + jI = (a + jb)(c + jd) = (ac - bd) + j(bc + ad).$$

We can see that [Eq. \(13-14\)](#) requires four multiplications and two additions. (From a computational standpoint we'll assume a subtraction is equivalent to an addition.) Instead of using [Eq. \(13-14\)](#), we can calculate the following intermediate values:

(13-15)

$$\begin{aligned} k_1 &= a(c + d), \\ k_2 &= d(a + b), \text{ and} \\ k_3 &= c(b - a). \end{aligned}$$

We then perform the following operations to get the final R and I :

(13-16)

$$\begin{aligned} R &= k_1 - k_2, \text{ and} \\ I &= k_1 + k_3. \end{aligned}$$

The reader is invited to plug the k values from [Eq. \(13-15\)](#) into [Eq. \(13-16\)](#) to verify that the expressions in [Eq. \(13-16\)](#) are equivalent to [Eq. \(13-14\)](#). The intermediate values in [Eq. \(13-15\)](#) required three additions and three multiplications, while the results in [Eq. \(13-16\)](#) required two more additions. So we traded one of the multiplications required in [Eq. \(13-14\)](#) for three addition operations needed by [Eqs. \(13-15\)](#) and [\(13-16\)](#). If our hardware uses fewer clock cycles to perform three additions than a single multiplication, we may well gain overall processing speed by using [Eqs. \(13-15\)](#) and [\(13-16\)](#) instead of [Eq. \(13-14\)](#) for complex multiplication.

13.5 Efficiently Performing the FFT of Real Sequences

Upon recognizing its linearity property and understanding the odd and even symmetries of the transform's output, the early investigators of the fast Fourier transform (FFT) realized that two separate, real N -point input data sequences could be transformed using a single N -point complex FFT. They also developed a technique using a single N -point complex FFT to transform a $2N$ -point real input sequence. Let's see how these two techniques work.

13.5.1 Performing Two N -Point Real FFTs

The standard FFT algorithms were developed to accept complex inputs; that is, the FFT's normal input $x(n)$ sequence is assumed to comprise real and imaginary parts, such as

(13-17)

$$\begin{aligned}x(0) &= x_r(0) + jx_i(0), \\x(1) &= x_r(1) + jx_i(1), \\x(2) &= x_r(2) + jx_i(2), \\&\dots \\&\dots \\x(N-1) &= x_r(N-1) + jx_i(N-1).\end{aligned}$$

In typical signal processing schemes, FFT input data sequences are usually real. The most common example of this is the FFT input samples coming from an A/D converter that provides real integer values of some continuous (analog) signal. In this case the FFT's imaginary $x_i(n)$'s inputs are all zero. So initial FFT computations performed on the $x_i(n)$ inputs represent wasted operations. Early FFT pioneers recognized this inefficiency, studied the problem, and developed a technique where two independent N -point, *real* input data sequences could be transformed by a single N -point complex FFT. We call this scheme the Two N -Point Real FFTs algorithm. The derivation of this technique is straightforward and described in the literature[17–19]. If two N -point, real input sequences are $a(n)$ and $b(n)$, they'll have discrete Fourier transforms represented by $X_a(m)$ and $X_b(m)$. If we treat the $a(n)$ sequence as the real part of an FFT input and the $b(n)$ sequence as the imaginary part of the FFT input, then

(13-18)

$$\begin{aligned}x(0) &= a(0) + jb(0), \\x(1) &= a(1) + jb(1), \\x(2) &= a(2) + jb(2), \\&\dots \\&\dots \\x(N-1) &= a(N-1) + jb(N-1).\end{aligned}$$

Applying the $x(n)$ values from [Eq. \(13-18\)](#) to the standard DFT,

(13-19)

$$X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N},$$

we'll get a DFT output $X(m)$ where m goes from 0 to $N-1$. (We're assuming, of course, that the DFT is implemented by way of an FFT algorithm.) Using the superscript “*” symbol to represent the complex conjugate, we can extract the two desired FFT outputs $X_a(m)$ and $X_b(m)$ from $X(m)$ by using the following:

(13-20)

$$X_a(m) = \frac{X^*(N-m) + X(m)}{2}$$

and

$$(13-21)$$

$$X_b(m) = \frac{j[X^*(N-m) - X(m)]}{2}.$$

Let's break [Eqs. \(13-20\)](#) and [\(13-21\)](#) into their real and imaginary parts to get expressions for $X_a(m)$ and $X_b(m)$ that are easier to understand and implement. Using the notation showing $X(m)$'s real and imaginary parts, where $X(m) = X_r(m) + jX_i(m)$, we can rewrite [Eq. \(13-20\)](#) as

$$(13-22)$$

$$X_a(m) = \frac{X_r(N-m) + X_r(m) + j[X_i(m) - X_i(N-m)]}{2}$$

where $m = 1, 2, 3, \dots, N-1$. What about the first $X_a(m)$, when $m = 0$? Well, this is where we run into a bind if we actually try to implement [Eq. \(13-20\)](#) directly. Letting $m = 0$ in [Eq. \(13-20\)](#), we quickly realize that the first term in the numerator, $X^*(N-0) = X^*(N)$, isn't available because the $X(N)$ sample does not exist in the output of an N -point FFT! We resolve this problem by remembering that $X(m)$ is periodic with a period N , so $X(N) = X(0)$.[‡] When $m = 0$, [Eq. \(13-20\)](#) becomes

[‡] This fact is illustrated in [Section 3.8](#) during the discussion of spectral leakage in DFTs.

$$(13-23)$$

$$X_a(0) = \frac{X_r(0) - jX_i(0) + X_r(0) + jX_i(0)}{2} = X_r(0).$$

Next, simplifying [Eq. \(13-21\)](#),

$$(13-24)$$

$$\begin{aligned} X_b(m) &= \frac{j[X_r(N-m) - jX_i(N-m) - X_r(m) - jX_i(m)]}{2} \\ &= \frac{X_i(N-m) + X_i(m) + j[X_r(N-m) - X_r(m)]}{2} \end{aligned}$$

where, again, $m = 1, 2, 3, \dots, N-1$. By the same argument used for [Eq. \(13-23\)](#), when $m = 0$, $X_b(0)$ in [Eq. \(13-24\)](#) becomes

$$(13-25)$$

$$X_b(0) = \frac{X_i(0) + X_i(0) + j[X_r(0) - X_r(0)]}{2} = X_i(0).$$

This discussion brings up a good point for beginners to keep in mind. In the literature [Eqs. \(13-20\)](#) and [\(13-21\)](#) are often presented without any discussion of the $m = 0$ problem. So, whenever you're grinding through an algebraic derivation or have some equations tossed out at you, be a little skeptical. Try the equations out on an example—see if they're true. (After all, both authors and book typesetters are human and sometimes make mistakes. We had an old saying in Ohio for this situation: "Trust everybody, but cut the cards.") Following this advice, let's prove that this Two N -Point Real FFTs algorithm really does work by applying the 8-point data sequences from [Chapter 3](#)'s DFT examples to [Eqs. \(13-22\)](#) through [\(13-25\)](#). Taking the 8-point input data sequence from [Section 3.1](#)'s DFT Example 1 and denoting it $a(n)$,

$$(13-26)$$

$$\begin{aligned}
 a(0) &= 0.3535, & a(1) &= 0.3535, \\
 a(2) &= 0.6464, & a(3) &= 1.0607, \\
 a(4) &= 0.3535, & a(5) &= -1.0607, \\
 a(6) &= -1.3535, & a(7) &= -0.3535.
 \end{aligned}$$

Taking the 8-point input data sequence from [Section 3.6](#)'s DFT Example 2 and calling it $b(n)$,

(13-27)

$$\begin{aligned}
 b(0) &= 1.0607, & b(1) &= 0.3535, \\
 b(2) &= -1.0607, & b(3) &= -1.3535, \\
 b(4) &= -0.3535, & b(5) &= 0.3535, \\
 b(6) &= 0.3535, & b(7) &= 0.6464.
 \end{aligned}$$

Combining the sequences in [Eqs. \(13-26\)](#) and [\(13-27\)](#) into a single complex sequence $x(n)$,

(13-28)

$$\begin{array}{rcl}
 & \begin{array}{c} a(n) \\ \downarrow \end{array} & \begin{array}{c} b(n) \\ \downarrow \end{array} \\
 x(n) = & \begin{array}{l} 0.3535 \\ + 0.3535 \\ + 0.6464 \\ + 1.0607 \\ + 0.3535 \\ - 1.0607 \\ - 1.3535 \\ - 0.3535 \end{array} & \begin{array}{l} + j 1.0607 \\ + j 0.3535 \\ - j 1.0607 \\ - j 1.3535 \\ - j 0.3535 \\ + j 0.3535 \\ + j 0.3535 \\ + j 0.6464. \end{array}
 \end{array}$$

Now, taking the 8-point FFT of the complex sequence in [Eq. \(13-28\)](#), we get

(13-29)

$$\begin{array}{rcl}
 & \begin{array}{c} X_r(m) \\ \downarrow \end{array} & \begin{array}{c} X_i(m) \\ \downarrow \end{array} & \\
 X(m) = & \begin{array}{l} 0.0000 \\ - 2.8283 \\ + 2.8282 \\ + 0.0000 \\ + 0.0000 \\ + 0.0000 \\ + 0.0000 \\ + 2.8283 \end{array} & \begin{array}{l} + j 0.0000 \\ - j 1.1717 \\ + j 2.8282 \\ + j 0.0000 \\ + j 0.0000 \\ + j 0.0000 \\ + j 0.0000 \\ + j 6.8282 \end{array} & \begin{array}{l} \leftarrow m = 0 \text{ term} \\ \leftarrow m = 1 \text{ term} \\ \leftarrow m = 2 \text{ term} \\ \leftarrow m = 3 \text{ term} \\ \leftarrow m = 4 \text{ term} \\ \leftarrow m = 5 \text{ term} \\ \leftarrow m = 6 \text{ term} \\ \leftarrow m = 7 \text{ term.} \end{array}
 \end{array}$$

So from [Eq. \(13-23\)](#),

$$X_a(0) = X_r(0) = 0.$$

To get the rest of $X_a(m)$, we have to plug the FFT output's $X(m)$ and $X(N-m)$ values into [Eq. \(13-22\)](#).[‡]

Doing so,

[‡] Remember, when the FFT's input is complex, the FFT outputs may not be conjugate symmetric; that is, we can't assume that $F(m)$ is equal to $F^*(N-m)$ when the FFT input sequence's real and imaginary parts are both nonzero.

$$\begin{aligned}
X_a(1) &= \frac{X_r(7) + X_r(1) + j[X_i(1) - X_i(7)]}{2} = \frac{2.8283 - 2.8283 + j[-1.1717 - 6.8282]}{2} \\
&= \frac{0 - j7.9999}{2} = 0 - j4.0 = 4 \angle -90^\circ, \\
X_a(2) &= \frac{X_r(6) + X_r(2) + j[X_i(2) - X_i(6)]}{2} = \frac{0.0 + 2.8282 + j[2.8282 - 0.0]}{2} \\
&= \frac{2.8282 + j2.8282}{2} = 1.414 + j1.414 = 2 \angle 45^\circ, \\
X_a(3) &= \frac{X_r(5) + X_r(3) + j[X_i(3) - X_i(5)]}{2} = \frac{0.0 + 0.0 + j[0.0 - 0.0]}{2} = 0 \angle 0^\circ, \\
X_a(4) &= \frac{X_r(4) + X_r(4) + j[X_i(4) - X_i(4)]}{2} = \frac{0.0 + 0.0 + j[0.0 - 0.0]}{2} = 0 \angle 0^\circ, \\
X_a(5) &= \frac{X_r(3) + X_r(5) + j[X_i(5) - X_i(3)]}{2} = \frac{0.0 + 0.0 + j[0.0 - 0.0]}{2} = 0 \angle 0^\circ, \\
X_a(6) &= \frac{X_r(2) + X_r(6) + j[X_i(6) - X_i(2)]}{2} = \frac{2.8282 + 0.0 + j[0.0 - 2.8282]}{2} \\
&= \frac{2.8282 - j2.8282}{2} = 1.414 - j1.414 = 2 \angle -45^\circ, \text{ and} \\
X_a(7) &= \frac{X_r(1) + X_r(7) + j[X_i(7) - X_i(1)]}{2} = \frac{-2.8282 + 2.8282 + j[6.8282 + 1.1717]}{2} \\
&= \frac{0.0 + j7.9999}{2} = 0 + j4.0 = 4 \angle 90^\circ.
\end{aligned}$$

So [Eq. \(13-22\)](#) really does extract $X_a(m)$ from the $X(m)$ sequence in [Eq. \(13-29\)](#). We can see that we need not solve [Eq. \(13-22\)](#) when m is greater than 4 (or $N/2$) because $X_a(m)$ will always be conjugate symmetric. Because $X_a(7) = X_a(1)$, $X_a(6) = X_a(2)$, etc., only the first $N/2$ elements in $X_a(m)$ are independent and need be calculated.

OK, let's keep going and use [Eqs. \(13-24\)](#) and [\(13-25\)](#) to extract $X_b(m)$ from the FFT output. From [Eq. \(13-25\)](#),

$$X_b(0) = X_i(0) = 0.$$

Plugging the FFT's output values into [Eq. \(13-24\)](#) to get the next four $X_b(m)$ s, we have

$$\begin{aligned}
X_b(1) &= \frac{X_i(7) + X_i(1) + j[X_r(7) - X_r(1)]}{2} = \frac{6.8282 - 1.1717 + j[2.8283 + 2.8283]}{2} \\
&= \frac{5.656 + j5.656}{2} = 2.828 + j2.828 = 4 \angle 45^\circ, \\
X_b(2) &= \frac{X_i(6) + X_i(2) + j[X_r(6) - X_r(2)]}{2} = \frac{0.0 + 2.8282 + j[0.0 - 2.8282]}{2} \\
&= \frac{2.8282 - j2.8282}{2} = 1.414 - j1.414 = 2 \angle -45^\circ, \\
X_b(3) &= \frac{X_i(5) + X_i(3) + j[X_r(5) - X_r(3)]}{2} = \frac{0.0 + 0.0 + j[0.0 - 0.0]}{2} = 0 \angle 0^\circ, \text{ and} \\
X_b(4) &= \frac{X_i(4) + X_i(4) + j[X_r(4) - X_r(4)]}{2} = \frac{0.0 + 0.0 + j[0.0 - 0.0]}{2} = 0 \angle 0^\circ.
\end{aligned}$$

The question arises “With the additional processing required by [Eqs. \(13-22\)](#) and [\(13-24\)](#) after the initial FFT, how much computational saving (or loss) is to be had by this Two N -Point Real FFTs algorithm?” We can estimate the efficiency of this algorithm by considering the number of arithmetic operations required relative to two separate N -point radix-2 FFTs. First, we estimate the number of arithmetic operations in two separate N -point complex FFTs.

From [Section 4.6](#), we know that a standard radix-2 N -point complex FFT comprises $(N/2) \cdot \log_2 N$ butterfly operations. If we use the optimized butterfly structure, each butterfly requires one complex multiplication and two complex additions. Now, one complex multiplication requires two real additions and four real multiplications, and one complex addition requires two real additions.[‡] So a single FFT butterfly operation comprises four real multiplications and six real additions. This means that a single N -point complex FFT requires $(4N/2) \cdot \log_2 N$ real multiplications, and $(6N/2) \cdot \log_2 N$ real additions. Finally, we can say that two separate N -point complex radix-2 FFTs require

$$(13-30)$$

$$\text{two } N\text{-point complex FFTs} \rightarrow 4N \cdot \log_2 N \text{ real multiplications, and}$$

$$(13-30')$$

$$6N \cdot \log_2 N \text{ real additions.}$$

Next, we need to determine the computational workload of the Two N -Point Real FFTs algorithm. If we add up the number of real multiplications and real additions required by the algorithm’s N -point complex FFT, plus those required by [Eq. \(13-22\)](#) to get $X_a(m)$, and those required by [Eq. \(13-24\)](#) to get $X_b(m)$, the Two N -Point Real FFTs algorithm requires

$$(13-31)$$

$$\text{two } N\text{-Point Real FFTs algorithm} \rightarrow 2N \cdot \log_2 N + N \text{ real multiplications, and}$$

$$(13-31')$$

$$3N \cdot \log_2 N + 2N \text{ real additions.}$$

[Equations \(13-31\)](#) and [\(13-31'\)](#) assume that we’re calculating only the first $N/2$ independent elements

of $X_a(m)$ and $X_b(m)$. The single N term in [Eq. \(13-31\)](#) accounts for the $N/2$ divide by 2 operations in [Eq. \(13-22\)](#) and the $N/2$ divide by 2 operations in [Eq. \(13-24\)](#).

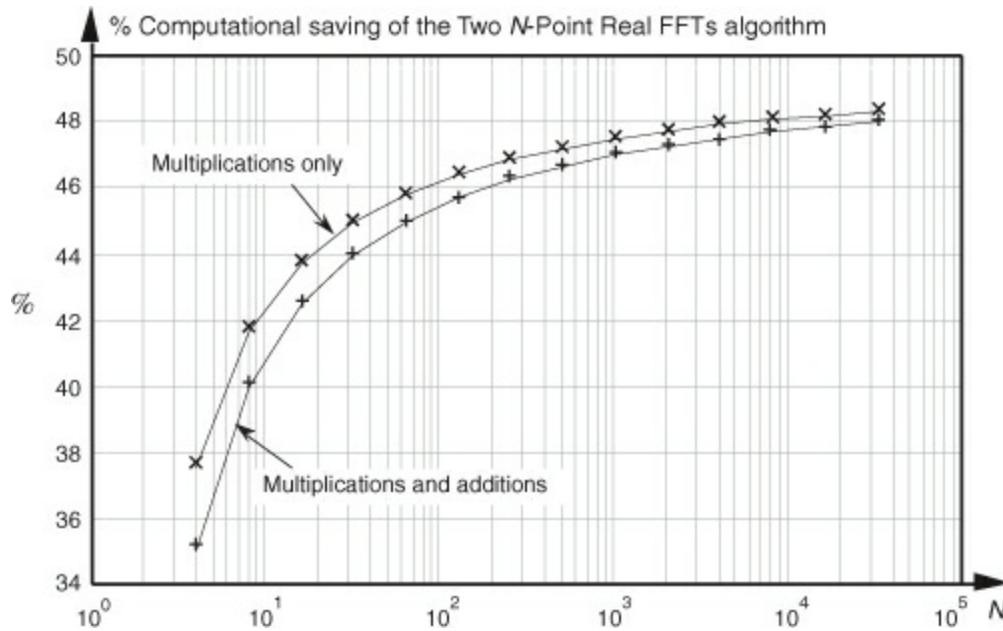
OK, now we can find out how efficient the Two N -Point Real FFTs algorithm is compared to two separate complex N -point radix-2 FFTs. This comparison, however, depends on the hardware used for the calculations. If our arithmetic hardware takes many more clock cycles to perform a multiplication than an addition, then the difference between multiplications in [Eqs. \(13-30\)](#) and [\(13-31\)](#) is the most important comparison. In this case, the percentage gain in computational saving of the Two N -Point Real FFTs algorithm relative to two separate N -point complex FFTs is the difference in their necessary multiplications over the number of multiplications needed for two separate N -point complex FFTs, or

$$(13-32)$$

$$\frac{4N \cdot \log_2 N - (2N \cdot \log_2 N + N)}{4N \cdot \log_2 N} \cdot 100\% = \frac{2 \cdot \log_2 N - 1}{4 \cdot \log_2 N} \cdot 100\%.$$

The computational (multiplications only) saving from [Eq. \(13-32\)](#) is plotted as the top curve of [Figure 13-11](#). In terms of multiplications, for $N \geq 32$, the Two N -Point Real FFTs algorithm saves us over 45 percent in computational workload compared to two separate N -point complex FFTs.

Figure 13-11 Computational saving of the Two N -Point Real FFTs algorithm over that of two separate N -point complex FFTs. The top curve indicates the saving when only multiplications are considered. The bottom curve is the saving when both additions and multiplications are used in the comparison.



For hardware using high-speed multiplier integrated circuits, multiplication and addition can take roughly equivalent clock cycles. This makes addition operations just as important and time consuming as multiplications. Thus the difference between those combined arithmetic operations in [Eqs. \(13-30\)](#) plus [\(13-30'\)](#) and [Eqs. \(13-31\)](#) plus [\(13-31'\)](#) is the appropriate comparison. In this case, the percentage gain in computational saving of our algorithm over two FFTs is their total arithmetic operational difference over the total arithmetic operations in two separate N -point complex FFTs, or

$$(13-33)$$

$$\frac{(4N \cdot \log_2 N + 6N \cdot \log_2 N) - (2N \cdot \log_2 N + N + 3N \cdot \log_2 N + 2N)}{4N \cdot \log_2 N + 6N \cdot \log_2 N} \cdot 100\%$$

$$= \frac{5 \cdot \log_2 N - 3}{10 \cdot \log_2 N} \cdot 100\%.$$

The full computational (multiplications and additions) saving from [Eq. \(13-33\)](#) is plotted as the bottom curve of [Figure 13-11](#). This concludes our discussion and illustration of how a single N -point complex FFT can be used to transform two separate N -point real input data sequences.

13.5.2 Performing a $2N$ -Point Real FFT

Similar to the scheme above where two separate N -point real data sequences are transformed using a single N -point FFT, a technique exists where a $2N$ -point real sequence can be transformed with a single complex N -point FFT. This $2N$ -Point Real FFT algorithm, whose derivation is also described in the literature, requires that the $2N$ -sample real input sequence be separated into two parts[[19,20](#)]
—not broken in two, but unzipped—separating the even and odd sequence samples. The N even-indexed input samples are loaded into the real part of a complex N -point input sequence $x(n)$. Likewise, the input's N odd-indexed samples are loaded into $x(n)$'s imaginary parts. To illustrate this process, let's say we have a $2N$ -sample real input data sequence $a(n)$ where $0 \leq n \leq 2N-1$. We want $a(n)$'s $2N$ -point transform $X_a(m)$. Loading $a(n)$'s odd/even sequence values appropriately into an N -point complex FFT's input sequence, $x(n)$,

(13-34)

$$\begin{aligned} x(0) &= a(0) + ja(1), \\ x(1) &= a(2) + ja(3), \\ x(2) &= a(4) + ja(5), \\ &\dots \\ &\dots \\ x(N-1) &= a(2N-2) + ja(2N-1). \end{aligned}$$

Applying the N complex values in [Eq. \(13-34\)](#) to an N -point complex FFT, we'll get an FFT output $X(m) = X_r(m) + jX_i(m)$, where m goes from 0 to $N-1$. To extract the desired $2N$ -Point Real FFT algorithm output $X_a(m) = X_{a,\text{real}}(m) + jX_{a,\text{imag}}(m)$ from $X(m)$, let's define the following relationships:

(13-35)

$$X_r^+(m) = \frac{X_r(m) + X_r(N-m)}{2},$$

(13-36)

$$X_r^-(m) = \frac{X_r(m) - X_r(N-m)}{2},$$

(13-37)

$$X_i^+(m) = \frac{X_i(m) + X_i(N-m)}{2}, \text{ and}$$

(13-38)

$$X_i^-(m) = \frac{X_i(m) - X_i(N-m)}{2}.$$

For the reasons presented following [Eq. \(13-22\)](#) in the last section, in the above expressions recall that $X_r(N) = X_r(0)$, and $X_i(N) = X_i(0)$. The values resulting from [Eqs. \(13-35\)](#) through [\(13-38\)](#) are, then, used as factors in the following expressions to obtain the real and imaginary parts of our final

$X_a(m)$:

(13-39)

$$X_{a,\text{real}}(m) = X_r^+(m) + \cos\left(\frac{\pi m}{N}\right) \cdot X_i^+(m) - \sin\left(\frac{\pi m}{N}\right) \cdot X_r^-(m)$$

and

(13-40)

$$X_{a,\text{imag}}(m) = X_i^-(m) - \sin\left(\frac{\pi m}{N}\right) \cdot X_i^+(m) - \cos\left(\frac{\pi m}{N}\right) \cdot X_r^-(m).$$

Remember, now, the original $a(n)$ input index n goes from 0 to $2N-1$, and our N -point FFT output index m goes from 0 to $N-1$. We apply $2N$ real input time-domain samples to this algorithm and get back N complex frequency-domain samples representing the first half of the equivalent $2N$ -point complex FFT, $X_a(0)$ through $X_a(N-1)$. Because this algorithm's $a(n)$ input is constrained to be real, $X_a(N+1)$ through $X_a(2N-1)$ are merely the complex conjugates of their $X_a(1)$ through $X_a(N-1)$ counterparts and need not be calculated.

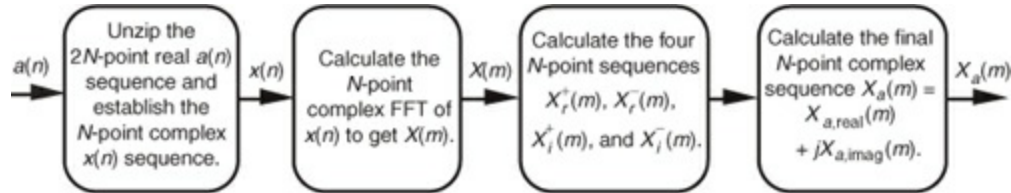
The above process does not compute the $X_a(N)$ sample. The $X_a(N)$ sample, which is real-only, is

(13-40')

$$X_a(N) = X_r(0) - X_i(0).$$

To help us keep all of this straight, [Figure 13-12](#) depicts the computational steps of the $2N$ -Point Real FFT algorithm.

Figure 13-12 Computational flow of the $2N$ -Point Real FFT algorithm.



To demonstrate this process by way of example, let's apply the 8-point data sequence from [Eq. \(13-26\)](#) to the $2N$ -Point Real FFT algorithm. Partitioning those [Eq. \(13-26\)](#), samples as dictated by [Eq. \(13-34\)](#), we have our new FFT input sequence:

(13-41)

$$\begin{aligned} x(0) &= 0.3535 + j 0.3535, \\ x(1) &= 0.6464 + j 1.0607, \\ x(2) &= 0.3535 - j 1.0607, \\ x(3) &= -1.3535 - j 0.3535. \end{aligned}$$

With $N = 4$ in this example, taking the 4-point FFT of the complex sequence in [Eq. \(13-41\)](#), we get

(13-42)

	$X_r(m)$	$X_i(m)$	
	↓	↓	
$X(m) =$	0.0000	$+ j 0.0000$	$\leftarrow m = 0 \text{ term}$
	$+ 1.4142$	$- j 0.5857$	$\leftarrow m = 1 \text{ term}$
	$+ 1.4141$	$- j 1.4141$	$\leftarrow m = 2 \text{ term}$
	$- 1.4142$	$+ j 3.4141$	$\leftarrow m = 3 \text{ term.}$

Using these values, we now get the intermediate factors from [Eqs. \(13-35\)](#) through [\(13-38\)](#).

Calculating our first $X_r^+(0)$ value, again we're reminded that $X(m)$ is periodic with a period N , so $X(4) = X(0)$, and $X_r^+(0) = [X_r(0) + X_r(0)]/2 = 0$. Continuing to use [Eqs. \(13-35\)](#) through [\(13-38\)](#),

$$(13-43)$$

$$\begin{aligned} X_r^+(0) &= 0, & X_r^-(0) &= 0, & X_i^+(0) &= 0, & X_i^-(0) &= 0, \\ X_r^+(1) &= 0, & X_r^-(1) &= 1.4142, & X_i^+(1) &= 1.4142, & X_i^-(1) &= -1.9999, \\ X_r^+(2) &= 1.4141, & X_r^-(2) &= 0, & X_i^+(2) &= -1.4144, & X_i^-(2) &= 0, \\ X_r^+(3) &= 0, & X_r^-(3) &= -1.4142, & X_i^+(3) &= 1.4142, & X_i^-(3) &= 1.9999. \end{aligned}$$

Using the intermediate values from [Eq. \(13-43\)](#) in [Eqs. \(13-39\)](#) and [\(13-40\)](#),

$$(13-44)$$

$$\begin{aligned} X_{a,\text{real}}(0) &= (0) + \cos\left(\frac{\pi \cdot 0}{4}\right) \cdot (0) - \sin\left(\frac{\pi \cdot 0}{4}\right) \cdot (0) \\ X_{a,\text{imag}}(0) &= (0) - \sin\left(\frac{\pi \cdot 0}{4}\right) \cdot (0) - \cos\left(\frac{\pi \cdot 0}{4}\right) \cdot (0) \\ X_{a,\text{real}}(1) &= (0) + \cos\left(\frac{\pi \cdot 1}{4}\right) \cdot (1.4142) - \sin\left(\frac{\pi \cdot 1}{4}\right) \cdot (1.4142) \\ X_{a,\text{imag}}(1) &= (-1.9999) - \sin\left(\frac{\pi \cdot 1}{4}\right) \cdot (1.4142) - \cos\left(\frac{\pi \cdot 1}{4}\right) \cdot (1.4142) \\ X_{a,\text{real}}(2) &= (1.4141) + \cos\left(\frac{\pi \cdot 2}{4}\right) \cdot (-1.4144) - \sin\left(\frac{\pi \cdot 2}{4}\right) \cdot (0) \\ X_{a,\text{imag}}(2) &= (0) - \sin\left(\frac{\pi \cdot 2}{4}\right) \cdot (-1.4144) - \cos\left(\frac{\pi \cdot 2}{4}\right) \cdot (0) \\ X_{a,\text{real}}(3) &= (0) + \cos\left(\frac{\pi \cdot 3}{4}\right) \cdot (1.4142) - \sin\left(\frac{\pi \cdot 3}{4}\right) \cdot (-1.4142) \\ X_{a,\text{imag}}(3) &= (1.9999) - \sin\left(\frac{\pi \cdot 3}{4}\right) \cdot (1.4142) - \cos\left(\frac{\pi \cdot 3}{4}\right) \cdot (-1.4142). \end{aligned}$$

Evaluating the sine and cosine terms in [Eq. \(13-44\)](#),

$$(13-45)$$

$$\begin{aligned} X_{a,\text{real}}(0) &= (0) + (1) \cdot (0) - (0) \cdot (0) = 0, \\ X_{a,\text{imag}}(0) &= (0) - (0) \cdot (0) - (1) \cdot (0) = 0, \\ X_{a,\text{real}}(1) &= (0) + (0.7071) \cdot (1.4142) - (0.7071) \cdot (1.4142) = 0, \\ X_{a,\text{imag}}(1) &= (-1.9999) - (0.7071) \cdot (1.4142) - (0.7071) \cdot (1.4142) = -3.9999, \\ X_{a,\text{real}}(2) &= (1.4141) + (0) \cdot (-1.4144) - (1) \cdot (0) = 1.4141, \\ X_{a,\text{imag}}(2) &= (0) - (1) \cdot (-1.4144) - (0) \cdot (0) = 1.4144, \\ X_{a,\text{real}}(3) &= (0) + (-0.7071) \cdot (1.4142) - (0.7071) \cdot (-1.4142) = 0, \text{ and} \\ X_{a,\text{imag}}(3) &= (1.9999) - (0.7071) \cdot (1.4142) - (-0.7071) \cdot (-1.4142) = 0. \end{aligned}$$

Combining the results of the terms in [Eq. \(13-45\)](#), we have our final correct answer of

$$(13-46)$$

$$\begin{aligned} X_a(0) &= X_{a,\text{real}}(0) + jX_{a,\text{imag}}(0) = 0 + j0 = 0 \angle 0^\circ, \\ X_a(1) &= X_{a,\text{real}}(1) + jX_{a,\text{imag}}(1) = 0 - j3.9999 = 4 \angle -90^\circ, \\ X_a(2) &= X_{a,\text{real}}(2) + jX_{a,\text{imag}}(2) = 1.4141 + j1.4144 = 2 \angle 45^\circ, \text{ and} \\ X_a(3) &= X_{a,\text{real}}(3) + jX_{a,\text{imag}}(3) = 0 + j0 = 0 \angle 0^\circ. \end{aligned}$$

After going through all the steps required by [Eqs. \(13-35\)](#) through [\(13-40\)](#), the reader might question the efficiency of this $2N$ -Point Real FFT algorithm. Using the same process as the above Two N -Point Real FFTs algorithm analysis, let's show that the $2N$ -Point Real FFT algorithm does provide some

modest computational saving. First, we know that a single $2N$ -point radix-2 FFT has $(2N/2) \cdot \log_2 2N = N \cdot (\log_2 N + 1)$ butterflies and requires

$$(13-47)$$

$$2N\text{-point complex FFT} \rightarrow 4N \cdot (\log_2 N + 1) \text{ real multiplications}$$

and

$$(13-47')$$

$$6N \cdot (\log_2 N + 1) \text{ real additions.}$$

If we add up the number of real multiplications and real additions required by the algorithm's N -point complex FFT, plus those required by [Eqs. \(13-35\)](#) through [\(13-38\)](#) and those required by [Eqs. \(13-39\)](#) and [\(13-40\)](#), the complete $2N$ -Point Real FFT algorithm requires

$$(13-48)$$

$$2N\text{-Point Real FFT algorithm} \rightarrow 2N \cdot \log_2 N + 8N \text{ real multiplications}$$

and

$$(13-48')$$

$$3N \cdot \log_2 N + 8N \text{ real additions.}$$

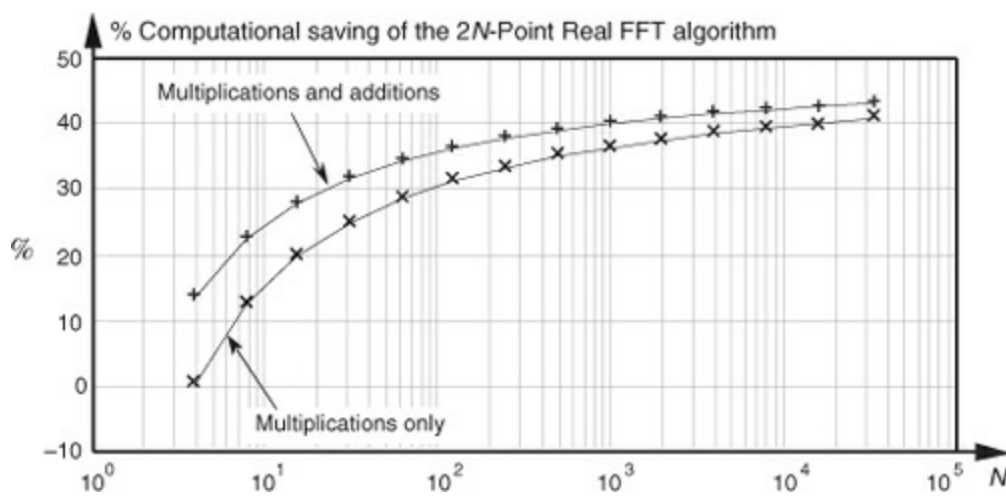
OK, using the same hardware considerations (multiplications only) we used to arrive at [Eq. \(13-32\)](#), the percentage gain in multiplication saving of the $2N$ -Point Real FFT algorithm relative to a $2N$ -point complex FFT is

$$(13-49)$$

$$\begin{aligned} & \frac{4N \cdot (\log_2 N + 1) - (2N \cdot \log_2 N + 8N)}{4N \cdot (\log_2 N + 1)} \cdot 100\% \\ &= \frac{2N \cdot \log_2 N + 2N - N \cdot \log_2 N - 4N}{2N \cdot \log_2 N + 2N} \cdot 100\% \\ &= \frac{\log_2 N - 2}{2 \cdot \log_2 N + 2} \cdot 100\%. \end{aligned}$$

The computational (multiplications only) saving from [Eq. \(13-49\)](#) is plotted as the bottom curve of [Figure 13-13](#). In terms of multiplications, the $2N$ -Point Real FFT algorithm provides a saving of >30 percent when $N \geq 128$ or whenever we transform input data sequences whose lengths are ≥ 256 .

Figure 13-13 Computational saving of the $2N$ -Point Real FFT algorithm over that of a single $2N$ -point complex FFT. The top curve is the saving when both additions and multiplications are used in the comparison. The bottom curve indicates the saving when only multiplications are considered.



Again, for hardware using high-speed multipliers, we consider both multiplication and addition operations. The difference between those combined arithmetic operations in [Eqs. \(13-47\)](#) plus [\(13-47'\)](#) and [Eqs. \(13-48\)](#) plus [\(13-48'\)](#) is the appropriate comparison. In this case, the percentage gain in computational saving of our algorithm is

(13-50)

$$\begin{aligned}
 & \frac{4N \cdot (\log_2 N + 1) + 6N \cdot (\log_2 N + 1) - (2N \cdot \log_2 N + 8N + 3N \cdot \log_2 N + 8N)}{4N \cdot (\log_2 N + 1) + 6N \cdot (\log_2 N + 1)} \cdot 100\% \\
 &= \frac{10 \cdot (\log_2 N + 1) - 5 \cdot \log_2 N - 16}{10 \cdot (\log_2 N + 1)} \cdot 100\% \\
 &= \frac{5 \cdot \log_2 N - 6}{10 \cdot (\log_2 N + 1)} \cdot 100\%.
 \end{aligned}$$

The full computational (multiplications and additions) saving from [Eq. \(13-50\)](#) is plotted as a function of N in the top curve of [Figure 13-13](#).

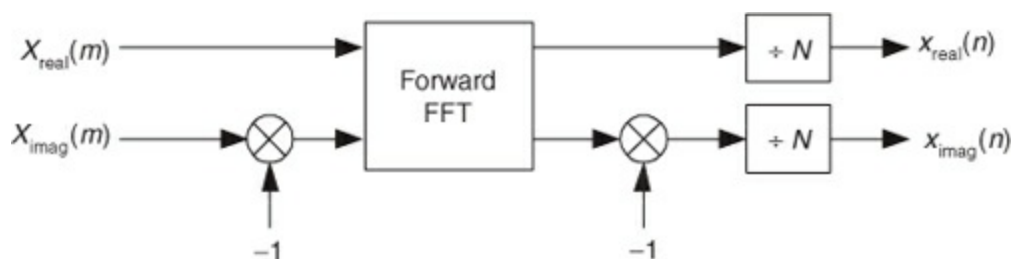
13.6 Computing the Inverse FFT Using the Forward FFT

There are many signal processing applications where the capability to perform the inverse FFT is necessary. This can be a problem if available hardware, or software routines, have only the capability to perform the *forward* FFT. Fortunately, there are two slick ways to perform the inverse FFT using the forward FFT algorithm.

13.6.1 Inverse FFT Method 1

The first inverse FFT calculation scheme is implemented following the processes shown in [Figure 13-14](#).

Figure 13-14 Processing for first inverse FFT calculation method.



To see how this works, consider the expressions for the forward and inverse DFTs. They are

(13-51)

$$\text{Forward DFT} \rightarrow X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi mn/N} \quad (13-52)$$

$$\text{Inverse DFT} \rightarrow x(n) = \frac{1}{N} \sum_{m=0}^{N-1} X(m)e^{j2\pi mn/N}$$

To reiterate our goal, we want to use the process in [Eq. \(13-51\)](#) to implement [Eq. \(13-52\)](#). The first step of our approach is to use complex conjugation. Remember, conjugation (represented by the superscript “*” symbol) is the reversal of the sign of a complex number’s imaginary exponent—if $x = e^{j\theta}$, then $x^* = e^{-j\theta}$. So, as a first step we take the complex conjugate of both sides of [Eq. \(13-52\)](#) to give us

$$(13-53)$$

$$x^*(n) = \frac{1}{N} \left[\sum_{m=0}^{N-1} X(m)e^{j2\pi mn/N} \right]^*$$

One of the properties of complex numbers, discussed in [Appendix A](#), is that the conjugate of a product is equal to the product of the conjugates. That is, if $c = ab$, then $c^* = (ab)^* = a^*b^*$. Using this, we can show the conjugate of the right side of [Eq. \(13-53\)](#) to be

$$(13-54)$$

$$x^*(n) = \frac{1}{N} \sum_{m=0}^{N-1} X(m)^* (e^{j2\pi mn/N})^* = \frac{1}{N} \sum_{m=0}^{N-1} X(m)^* e^{-j2\pi mn/N}$$

Hold on; we’re almost there. Notice the similarity of [Eq. \(13-54\)](#) to our original forward DFT expression, [Eq. \(13-51\)](#). If we perform a forward DFT on the conjugate of the $X(m)$ in [Eq. \(13-54\)](#), and divide the results by N , we get the conjugate of our desired time samples $x(n)$. Taking the conjugate of both sides of [Eq. \(13-54\)](#), we get a more straightforward expression for $x(n)$:

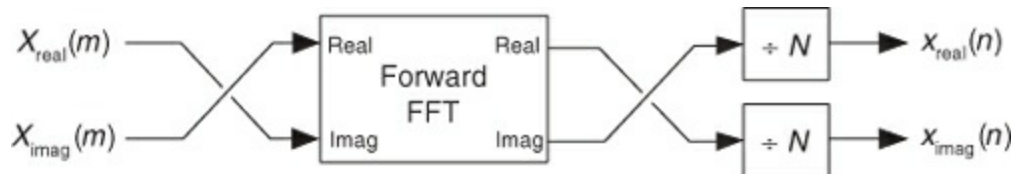
$$(13-55)$$

$$x(n) = \frac{1}{N} \left[\sum_{m=0}^{N-1} X(m)^* e^{-j2\pi mn/N} \right]^*$$

13.6.2 Inverse FFT Method 2

The second inverse FFT calculation technique is implemented following the interesting data flow shown in [Figure 13-15](#).

Figure 13-15 Processing for second inverse FFT calculation method.



In this clever inverse FFT scheme we don’t bother with conjugation. Instead, we merely swap the real and imaginary parts of sequences of complex data[\[21\]](#). To see why this process works, let’s look at the inverse DFT equation again while separating the input $X(m)$ term into its real and imaginary parts and remembering that $e^{j\theta} = \cos(\theta) + j\sin(\theta)$.

$$(13-56)$$

$$\begin{aligned} \text{Inverse DFT} \rightarrow x(n) &= \frac{1}{N} \sum_{m=0}^{N-1} X(m) e^{j2\pi mn/N} \\ &= \frac{1}{N} \sum_{m=0}^{N-1} [X_{\text{real}}(m) + jX_{\text{imag}}(m)] [\cos(2\pi mn/N) + j\sin(2\pi mn/N)]. \end{aligned}$$

Multiplying the complex terms in [Eq. \(13-56\)](#) gives us

(13-57)

$$\begin{aligned} x(n) &= \frac{1}{N} \sum_{m=0}^{N-1} [X_{\text{real}}(m)\cos(2\pi mn/N) - X_{\text{imag}}(m)\sin(2\pi mn/N) \\ &\quad + j[X_{\text{real}}(m)\sin(2\pi mn/N) + X_{\text{imag}}(m)\cos(2\pi mn/N)]. \end{aligned}$$

[Equation \(13-57\)](#) is the general expression for the inverse DFT, and we'll now quickly show that the process in [Figure 13-15](#) implements this equation. With $X(m) = X_{\text{real}}(m) + jX_{\text{imag}}(m)$, then swapping these terms gives us

(13-58)

$$X_{\text{swap}}(m) = X_{\text{imag}}(m) + jX_{\text{real}}(m).$$

The forward DFT of our $X_{\text{swap}}(m)$ is

(13-59)

$$\text{Forward DFT} \rightarrow \sum_{n=0}^{N-1} [X_{\text{imag}}(m) + jX_{\text{real}}(m)] [\cos(2\pi mn/N) - j\sin(2\pi mn/N)].$$

Multiplying the complex terms in [Eq. \(13-59\)](#) gives us

(13-60)

$$\begin{aligned} \text{Forward DFT} \rightarrow \sum_{n=0}^{N-1} [X_{\text{imag}}(m)\cos(2\pi mn/N) + X_{\text{real}}(m)\sin(2\pi mn/N) \\ + j[X_{\text{real}}(m)\cos(2\pi mn/N) - X_{\text{imag}}(m)\sin(2\pi mn/N)]. \end{aligned}$$

Swapping the real and imaginary parts of the results of this forward DFT gives us what we're after:

(13-61)

$$\begin{aligned} \text{Forward DFT}_{\text{swap}} \rightarrow \sum_{n=0}^{N-1} [X_{\text{real}}(m)\cos(2\pi mn/N) - X_{\text{imag}}(m)\sin(2\pi mn/N) \\ + j[X_{\text{imag}}(m)\cos(2\pi mn/N) + X_{\text{real}}(m)\sin(2\pi mn/N)]. \end{aligned}$$

If we divided [Eq. \(13-61\)](#) by N , it would be exactly equal to the inverse DFT expression in [Eq. \(13-57\)](#), and that's what we set out to show.

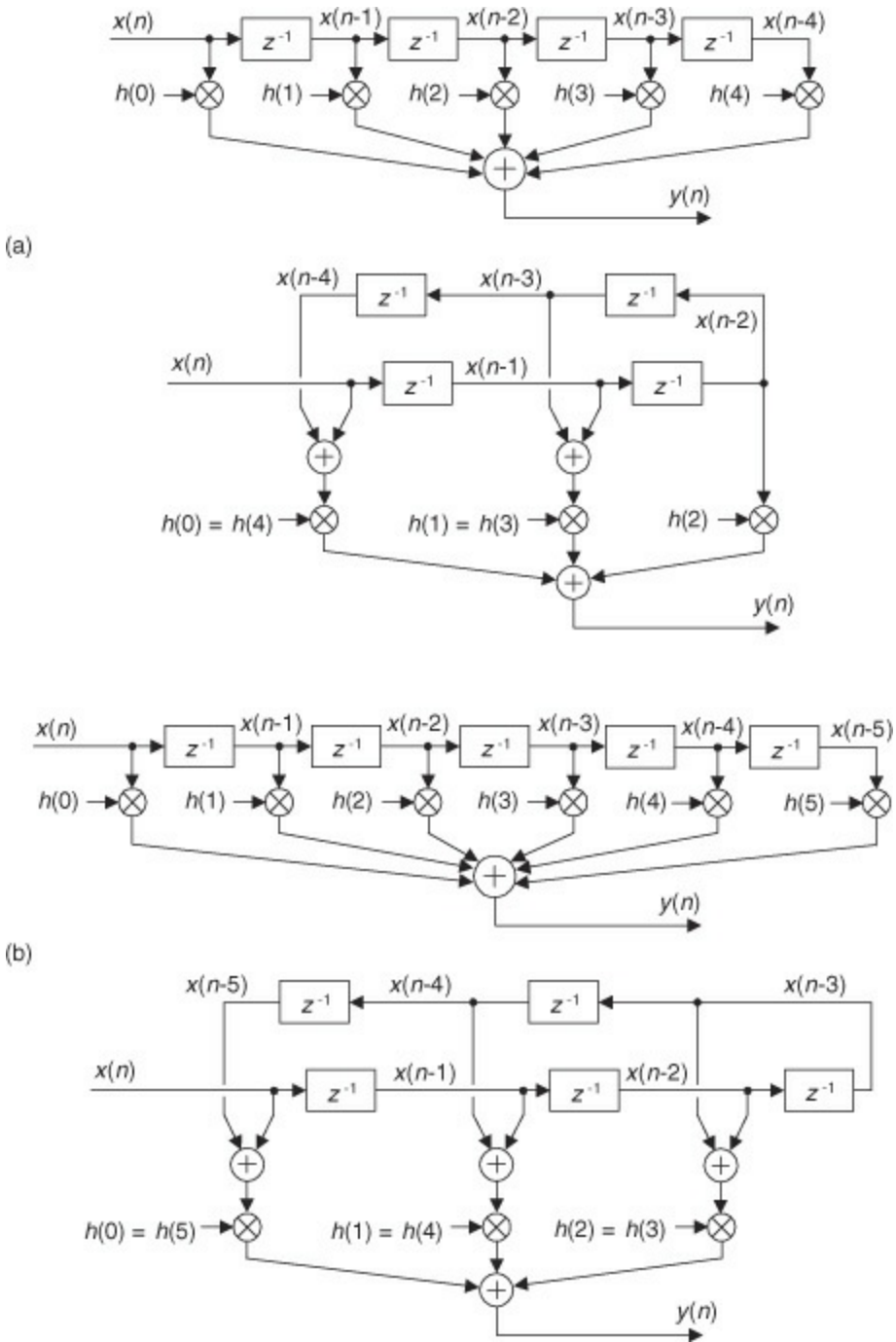
13.7 Simplified FIR Filter Structure

If we implement a linear-phase FIR digital filter using the standard structure in [Figure 13-16\(a\)](#), there's a way to reduce the number of multipliers when the filter has an odd number of taps. Let's look at the top of [Figure 13-16\(a\)](#) where the 5-tap FIR filter coefficients are $h(0)$ through $h(4)$ and the $y(n)$ output is

(13-62)

$$y(n) = h(4)x(n-4) + h(3)x(n-3) + h(2)x(n-2) + h(1)x(n-1) + h(0)x(n).$$

Figure 13-16 Conventional and simplified structures of an FIR filter: (a) with an odd number of taps; (b) with an even number of taps.



If the FIR filter’s coefficients are symmetrical, we can reduce the number of necessary multipliers. That is, if $h(4) = h(0)$, and $h(3) = h(1)$, we can implement [Eq. \(13-62\)](#) by

$$(13-63)$$

$$y(n) = h(4)[x(n-4)+x(n)] + h(3)[x(n-3)+x(n-1)] + h(2)x(n-2)$$

where only three multiplications are necessary as shown at the bottom of [Figure 13-16\(a\)](#). In our 5-tap filter case, we’ve eliminated two multipliers. This minimum-multiplier structure is called a *folded* FIR filter.

So in the case of an odd number of taps, we need only perform $(S-1)/2 + 1$ multiplications for each filter output sample. For an even number of symmetrical taps as shown in [Figure 13-16\(b\)](#), the saving afforded by this technique reduces the necessary number of multiplications to $S/2$. Some commercial programmable DSP chips have specialized instructions, and dual multiply-and-accumulate (MAC)

units, that take advantage of the folded FIR filter implementation.

13.8 Reducing A/D Converter Quantization Noise

In [Section 12.3](#) we discussed the mathematical details, and ill effects, of quantization noise in analog-to-digital (A/D) converters. DSP practitioners commonly use two tricks to reduce converter quantization noise. Those schemes are called *oversampling* and *dithering*.

13.8.1 Oversampling

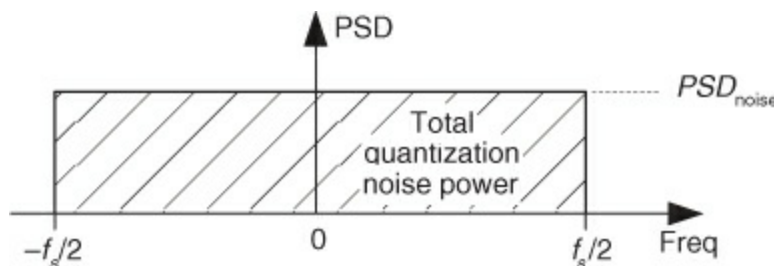
The process of oversampling to reduce A/D converter quantization noise is straightforward. We merely sample an analog signal at an f_s sample rate higher than the minimum rate needed to satisfy the Nyquist criterion (twice the analog signal's bandwidth), and then lowpass filter. What could be simpler? The theory behind oversampling is based on the assumption that an A/D converter's total quantization noise power (variance) is the converter's least significant bit (lsb) value squared over 12, or

(13-64)

$$\text{total quantization noise power} = \sigma^2 = \frac{(\text{lsb value})^2}{12}.$$

We derived that expression in [Section 12.3](#). The next assumptions are: The quantization noise values are truly random, and in the frequency domain the quantization noise has a flat spectrum. (These assumptions are valid if the A/D converter is being driven by an analog signal that covers most of the converter's analog input voltage range and is not highly periodic.) Next we consider the notion of quantization noise power spectral density (PSD), a frequency-domain characterization of quantization noise measured in noise power per hertz as shown in [Figure 13-17](#). Thus we can consider the idea that quantization noise can be represented as a certain amount of power (watts, if we wish) per unit bandwidth.

Figure 13-17 Frequency-domain power spectral density of an ideal A/D converter.



In our world of discrete systems, the flat noise spectrum assumption results in the total quantization noise (a fixed value based on the converter's lsb voltage) being distributed equally in the frequency domain, from $-f_s/2$ to $+f_s/2$ as indicated in [Figure 13-17](#). The amplitude of this quantization noise PSD is the rectangle area (total quantization noise power) divided by the rectangle width (f_s), or

(13-65)

$$PSD_{\text{noise}} = \frac{(\text{lsb value})^2}{12} \frac{1}{f_s} = \frac{(\text{lsb value})^2}{12f_s}$$

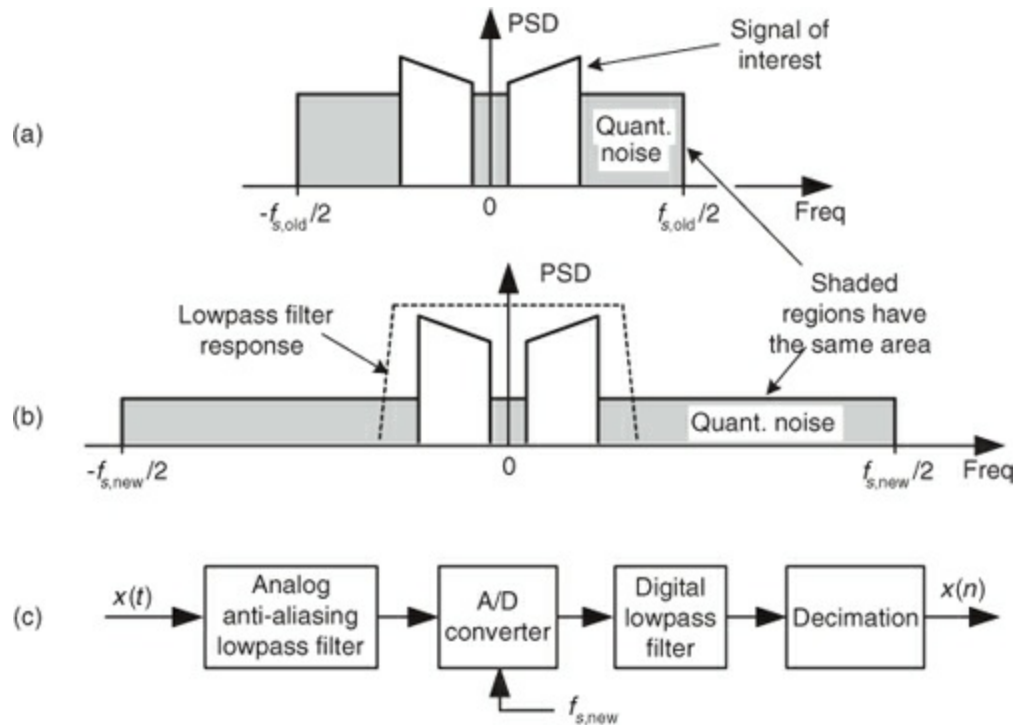
measured in watts/Hz.

The next question is: “How can we reduce the PSD_{noise} level defined by [Eq. \(13-65\)](#)?” We could reduce the lsb value (volts) in the numerator by using an A/D converter with additional bits. That

would make the lsb value smaller and certainly reduce PSD_{noise} , but that's an expensive solution. Extra converter bits cost money. Better yet, let's increase the denominator of [Eq. \(13-65\)](#) by increasing the sample rate f_s .

Consider a low-level discrete signal of interest whose spectrum is depicted in [Figure 13-18\(a\)](#). By increasing the $f_{s,\text{old}}$ sample rate to some larger value $f_{s,\text{new}}$ (oversampling), we spread the total noise power (a fixed value) over a wider frequency range as shown in [Figure 13-18\(b\)](#). The areas under the shaded curves in [Figures 13-18\(a\)](#) and [13-18\(b\)](#) are equal. Next we lowpass filter the converter's output samples. At the output of the filter, the quantization noise level contaminating our signal will be reduced from that at the input of the filter.

Figure 13-18 Oversampling example: (a) noise PSD at an $f_{s,\text{old}}$ samples rate; (b) noise PSD at the higher $f_{s,\text{new}}$ samples rate; (c) processing steps.



The improvement in signal-to-quantization-noise ratio, measured in dB, achieved by oversampling is

$$(13-66)$$

$$SNR_{A/D\text{-gain}} = 10\log_{10}(f_{s,\text{new}}/f_{s,\text{old}}).$$

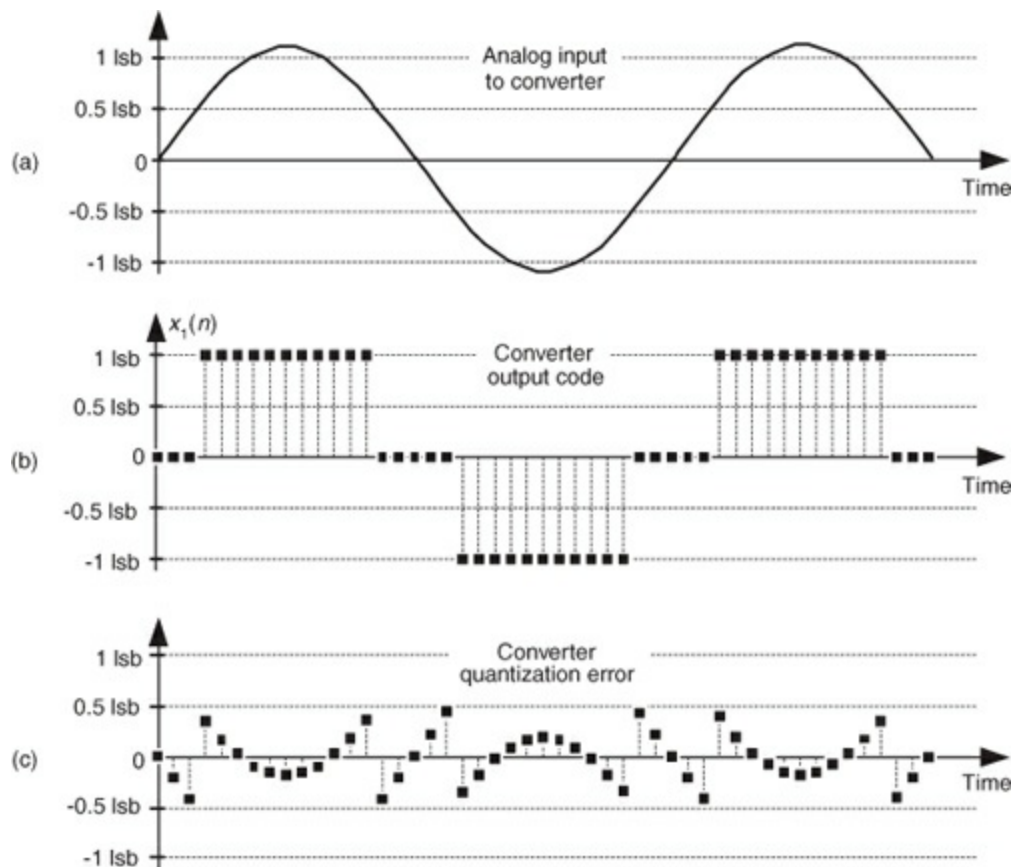
For example, if $f_{s,\text{old}} = 100$ kHz, and $f_{s,\text{new}} = 400$ kHz, the $SNR_{A/D\text{-gain}} = 10\log_{10}(4) = 6.02$ dB. Thus oversampling by a factor of four (and filtering), we gain a single bit's worth of quantization noise reduction. Consequently we can achieve $N+1$ -bit performance from an N -bit A/D converter, because we gain signal amplitude resolution at the expense of higher sampling speed. After digital filtering, we can decimate to the lower $f_{s,\text{old}}$ without degrading the improved SNR. Of course, the number of bits used for the lowpass filter's coefficients and registers must exceed the original number of A/D converter bits, or this oversampling scheme doesn't work.

With the use of a digital lowpass filter, depending on the interfering analog noise in $x(t)$, it's possible to use a lower-performance (simpler) analog anti-aliasing filter relative to the analog filter necessary at the lower sampling rate.

13.8.2 Dithering

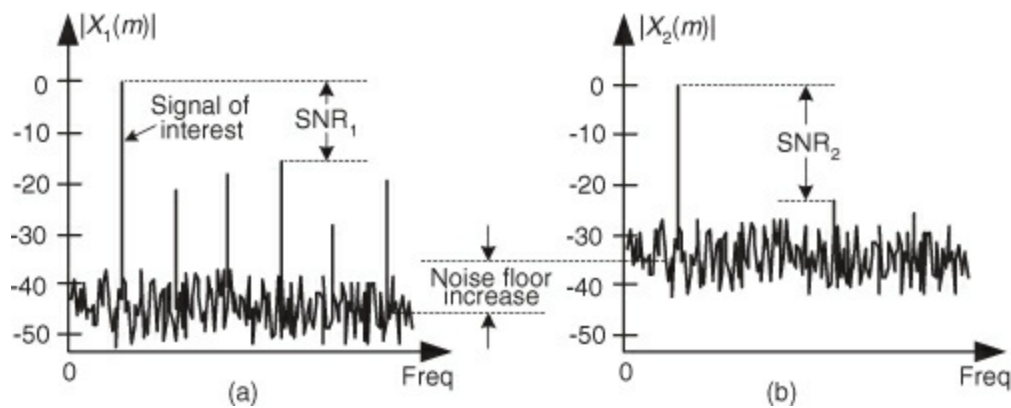
Dithering, another technique used to minimize the effects of A/D quantization noise, is the process of adding noise to our analog signal prior to A/D conversion. This scheme, which doesn't seem at all like a good idea, can indeed be useful and is easily illustrated with an example. Consider digitizing the low-level analog sinusoid shown in [Figure 13-19\(a\)](#), whose peak voltage just exceeds a single A/D converter least significant bit (lsb) voltage level, yielding the converter output $x_1(n)$ samples in [Figure 13-19\(b\)](#). The $x_1(n)$ output sequence is *clipped*. This generates all sorts of spectral harmonics. Another way to explain the spectral harmonics is to recognize the periodicity of the quantization noise in [Figure 13-19\(c\)](#).

Figure 13-19 Dithering: (a) a low-level analog signal; (b) the A/D converter output sequence; (c) the quantization error in the converter's output.



We show the spectrum of $x_1(n)$ in [Figure 13-20\(a\)](#) where the spurious quantization noise harmonics are apparent. It's worthwhile to note that averaging multiple spectra will not enable us to pull some spectral component of interest up above those spurious harmonics in [Figure 13-20\(a\)](#). Because the quantization noise is highly correlated with our input sinewave—the quantization noise has the same time period as the input sinewave—spectral averaging will also raise the noise harmonic levels. Dithering to the rescue.

Figure 13-20 Spectra of a low-level discrete sinusoid: (a) with no dithering; (b) with dithering.



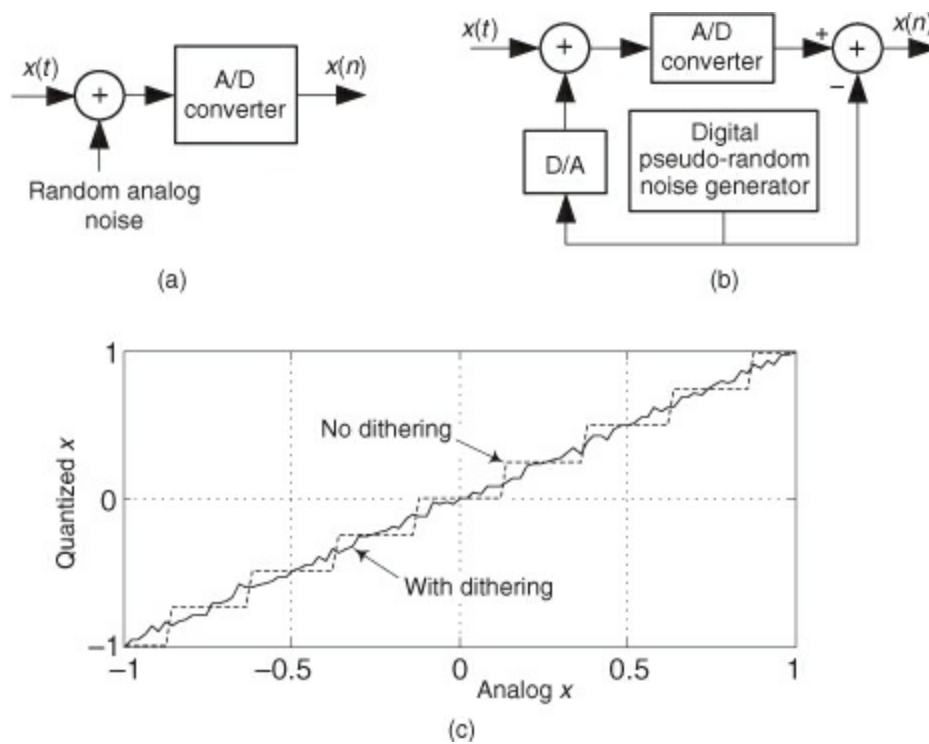
Dithering is the technique where random analog noise is added to the analog input sinusoid before it is digitized. This technique results in a noisy analog signal that crosses additional converter lsb boundaries and yields a quantization noise that's much more random, with a reduced level of undesirable spectral harmonics as shown in [Figure 13-20\(b\)](#). Dithering raises the average spectral noise floor but increases our signal-to-noise ratio SNR_2 . Dithering forces the quantization noise to lose its coherence with the original input signal, and we could then perform signal averaging if desired.

Dithering is indeed useful when we're digitizing

- low-amplitude analog signals,
- highly periodic analog signals (like a sinewave with an even number of cycles in the sample time interval), and
- slowly varying (very low frequency, including DC) analog signals.

The standard implementation of dithering is shown in [Figure 13-21\(a\)](#). The typical amount of random wideband analog noise used in this process, provided by a noise diode or noise generator ICs, has an rms (root mean squared) level equivalent to 1/3 to 1 lsb voltage level. The system-level effect of adding the analog dithering signal is to linearize the undithered *stair-step* transfer function of an A/D converter as shown in [Figure 13-21\(c\)](#).

Figure 13-21 Dithering implementations: (a) standard dithering process; (b) advanced dithering with noise subtraction; (c) improved transfer function due to dithering.



For high-performance audio applications, engineers have found that adding dither noise from two separate noise generators improves background audio low-level noise suppression. The probability density function (PDF) of the sum of two noise sources (having rectangular PDFs) is the convolution of their individual PDFs. Because the convolution of two rectangular functions is triangular, this dual-noise-source dithering scheme is called *triangular dither*. Typical triangular dither noise has rms levels equivalent to, roughly, 2 lsb voltage levels.

In the situation where our signal of interest occupies some well-defined portion of the full frequency band, injecting narrowband dither noise having an rms level equivalent to 4 to 6 lsb voltage levels, whose spectral energy is outside that signal band, would be advantageous. (Remember, though: the dither signal can't be too narrowband, like a sinewave. Quantization noise from a sinewave signal would generate more spurious harmonics!) That narrowband dither noise can then be removed by follow-on digital filtering.

One last note about dithering: To improve our ability to detect low-level signals, we could add the analog dither noise and then subtract that noise from the digitized data, as shown in [Figure 13-21\(b\)](#). This way, we randomize the quantization noise but reduce the amount of total noise power injected in the analog signal. This scheme is used in commercial analog test equipment[[22,23](#)].

13.9 A/D Converter Testing Techniques

We can take advantage of digital signal processing techniques to facilitate the testing of A/D converters. In this section we present two schemes for measuring converter performance: first, a technique using the FFT to estimate overall converter noise, and second, a histogram analysis scheme to detect missing converter output codes.

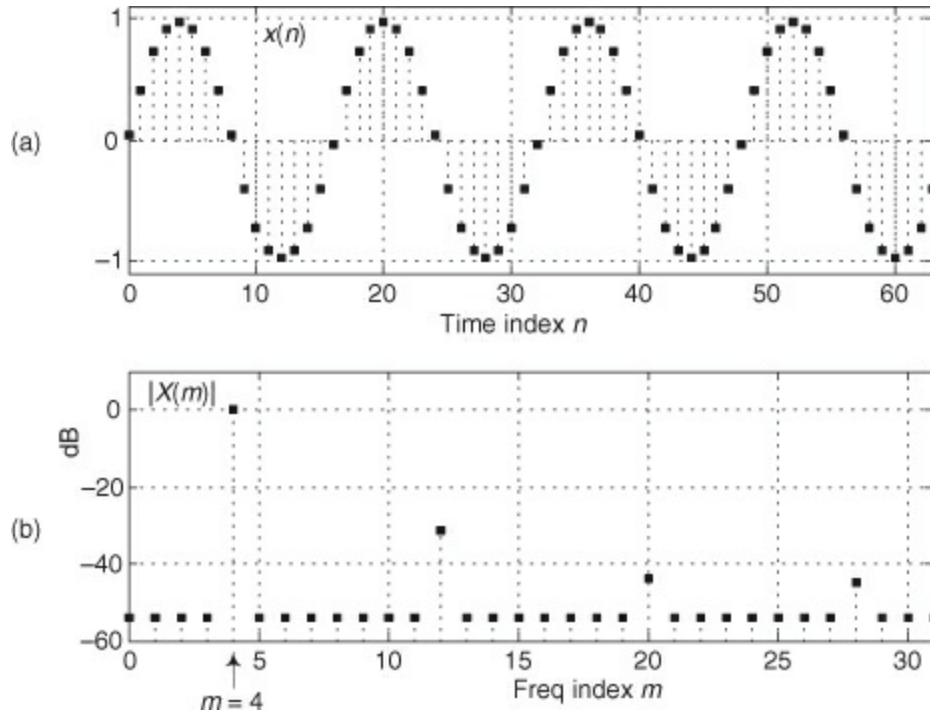
13.9.1 Estimating A/D Quantization Noise with the FFT

The combination of A/D converter quantization noise, missing bits, harmonic distortion, and other nonlinearities can be characterized by analyzing the spectral content of the converter's output. Converter performance degradation caused by these nonlinearities is not difficult to recognize because they show up as spurious spectral components and increased background noise levels in the

A/D converter's output samples. The traditional test method involves applying a sinusoidal analog voltage to an A/D converter's input and examining the spectrum of the converter's digitized time-domain output samples. We can use the FFT to compute the spectrum of an A/D converter's output samples, but we have to minimize FFT spectral leakage to improve the sensitivity of our spectral measurements. Traditional time-domain windowing, however, often provides insufficient FFT leakage reduction for high-performance A/D converter testing.

The trick to circumvent this FFT leakage problem is to use a sinusoidal analog input voltage whose frequency is a rational factor of the A/D converter's clock frequency as shown in [Figure 13-22\(a\)](#). That frequency is mf_s/N where m is an integer, f_s is the clock frequency (sample rate), and N is the FFT size. [Figure 13-22\(a\)](#) shows the $x(n)$ time-domain output of an ideal 5-bit A/D converter when its analog input is a sinewave having exactly $m = 4$ cycles over $N = 64$ converter output samples. In this case, the analog input frequency is $4f_s/64$ Hz. Recall from [Chapter 3](#) that the expression mf_s/N defined the analysis frequencies, or bin centers, of the DFT, and a DFT input sinusoid whose frequency is at a bin center causes no spectral leakage.

Figure 13-22 A/D converter (5-bit) output with an analog $4f_s/64$ Hz sinewave input: (a) $m = 4$ -cycle sinusoidal time samples; (b) spectral magnitude in dB.

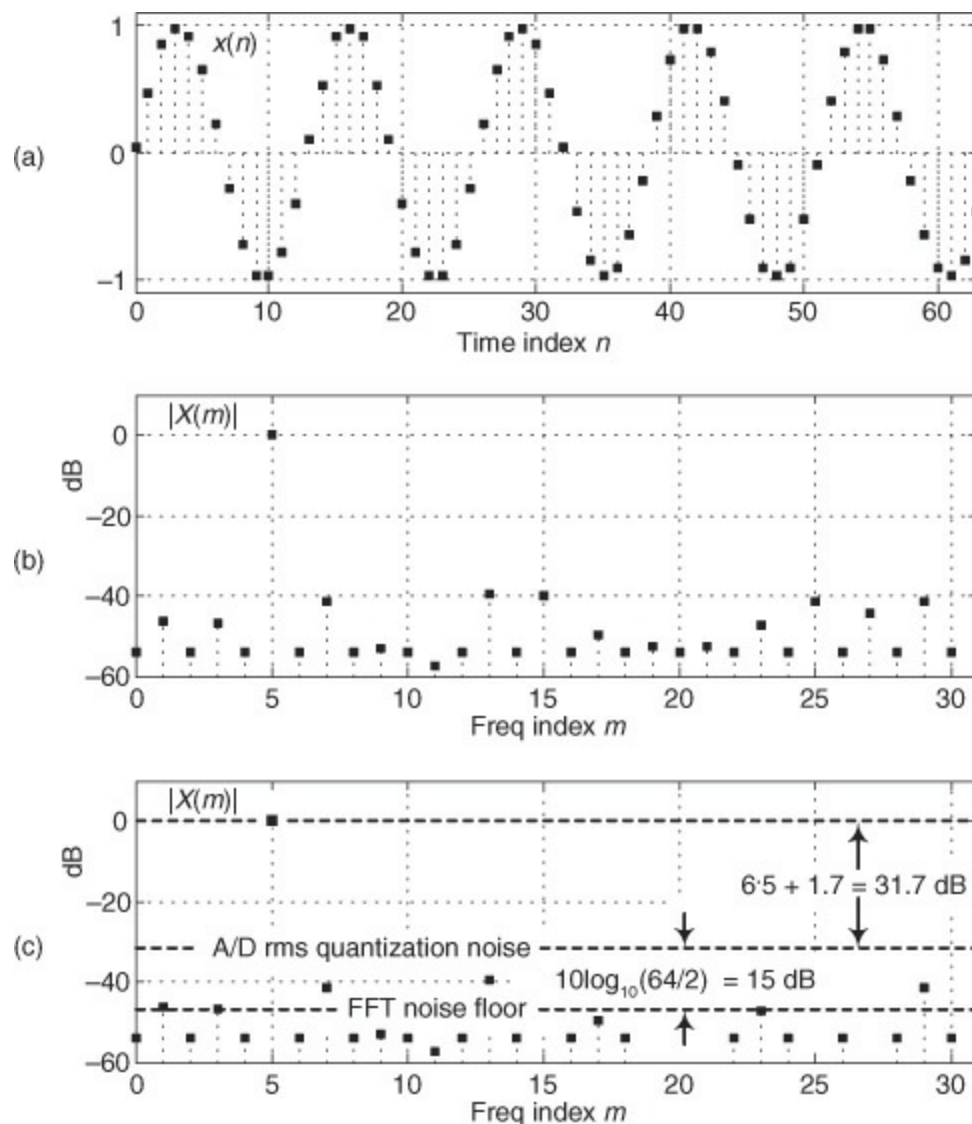


The magnitudes of the first half of an $N = 64$ -point FFT of $x(n)$ are shown in the logarithmic plot in [Figure 13-22\(b\)](#) where the analog input spectral component lies exactly at the $m = 4$ bin center. (The additional nonzero spectral samples are not due to FFT leakage; they represent A/D converter quantization noise.) Specifically, if the sample rate were 1 MHz, then the A/D's input analog sinewave's frequency is $4(10^6/64) = 62.5$ kHz. In order to implement this A/D testing scheme we must ensure that the analog test-signal generator is synchronized, exactly, with the A/D converter's clock frequency of f_s Hz. Achieving this synchronization is why this A/D converter testing procedure is referred to as *coherent sampling*[\[24–26\]](#). That is, the analog signal generator and the A/D clock generator providing f_s must not drift in frequency relative to each other—they must remain coherent. (Here we must take care from a semantic viewpoint because the quadrature sampling schemes described in [Chapter 8](#) are also sometimes called *coherent sampling*, and they are unrelated to this

A/D converter testing procedure.)

As it turns out, some values of m are more advantageous than others. Notice in [Figure 13-22\(a\)](#), that when $m = 4$, only ten different binary output values, output codes, are output by the A/D converter. Those values are repeated over and over, and the quantization noise is far from being random. As shown in [Figure 13-23\(a\)](#), when $m = 5$, we exercise more than ten different A/D output codes, and the quantization noise in [Figure 13-23\(b\)](#) is much more random than when $m = 4$.

Figure 13-23 A/D converter (5-bit) output with an analog $5f_s/64$ Hz sinewave input: (a) $m = 5$ -cycle time samples; (b) spectral magnitude in dB; (c) FFT results interpretation.



Because it's best to test as many A/D output codes as possible, while keeping the quantization noise sufficiently random, users of this A/D testing scheme have discovered another trick; they found making m an odd prime number (3, 5, 7, 11, etc.) minimizes the number of redundant A/D output code values and makes the quantization noise more random, which is what we want. The larger m is, the more codes that are exercised. (We can use *histogram testing*, discussed in the next section, to determine how many of a b -bit A/D converter's 2^b possible output codes have been exercised.)

While examining the quantization noise level in [Figure 13-23\(b\)](#), we might be tempted to say the A/D converter has a signal-to-quantization-noise ratio of 40 to 50 dB. As it turns out, the true A/D converter noise levels will be higher than those indicated by [Figure 13-23\(b\)](#). That's because the inherent processing gain of the FFT (discussed in [Section 3.12.1](#)) will *pull* the high-level $m = 5$ signal spectral component up out of the background converter noise, making that $m = 5$ spectral

magnitude sample appear higher above the background noise than is correct. Consequently, when viewing [Figure 13-23\(b\)](#), we must keep in mind an $N = 64$ -point FFT's processing gain of $10\log_{10}(64/2)$. Our interpretation of A/D performance based on the FFT magnitude results is given in [Figure 13-23\(c\)](#).

There is a technique used to characterize an A/D converter's true signal-to-noise ratio (including quantization noise, harmonic distortion, and other nonlinearities). That testing technique measures what is commonly called an A/D converter's SINAD—for signal-to-noise-and-distortion—and does not require us to consider FFT processing gain. The SINAD value for an A/D converter, based on spectral power samples, is

$$(13-66')$$

$$\text{SINAD} = 10\log_{10}\left(\frac{\text{Total signal power}}{\text{Total noise power}}\right), \text{ in dB.}$$

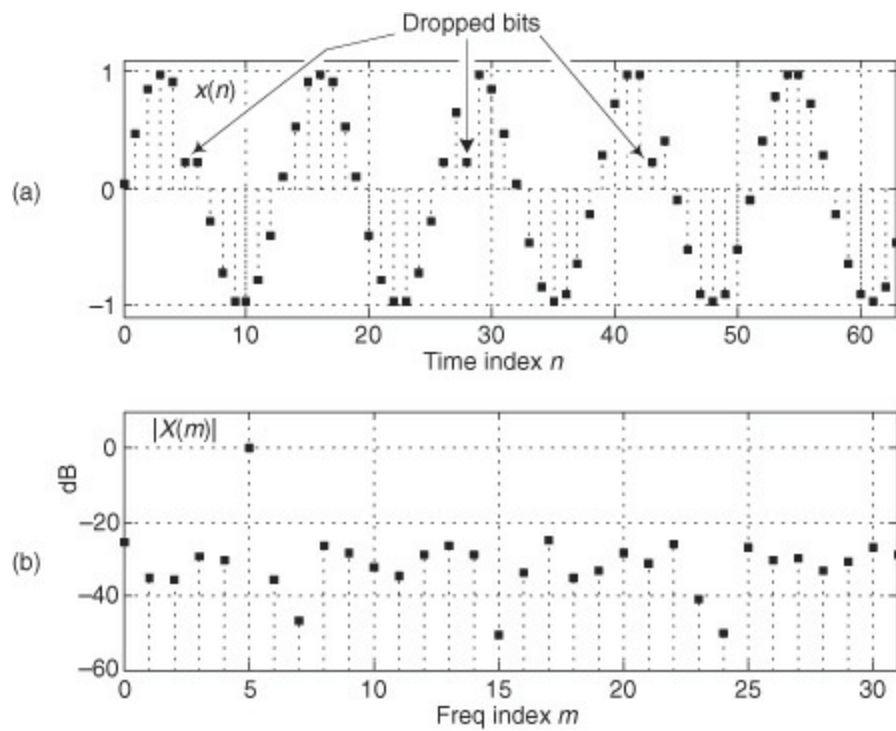
The SINAD value for an A/D converter is a good quantitative indicator of a converter's overall dynamic performance. The steps to compute SINAD are:

1. Compute an N -point FFT of an A/D converter's output sequence. Discard the negative-frequency samples of the FFT results.
2. Over the positive-frequency range of the FFT results, compute the total signal spectral power by summing the squares of all signal-only spectral magnitude samples. For our [Figure 13-23](#) example that's simply squaring the FFT's $|X(5)|$ magnitude value. (We square the linear $|X(5)|$ value and not the value of $|X(5)|$ in dB!)
3. Over the positive-frequency range of the FFT results, sum the squares of all noise-only spectral magnitude samples, including any signal harmonics, but excluding the zero-Hz $X(0)$ sample. This summation result represents total noise power, which includes harmonic distortion.
4. Perform the computation given in [Eq. \(13-66'\)](#).

Performing those steps on the spectrum in [Figure 13-23\(b\)](#) yields a SINAD value of 31.6 dB. This result is reasonable for our simulated 5-bit A/D converter because its signal-to-quantization-noise ratio would ideally be $6 \cdot 5 + 1.7 = 31.7$ dB.

[Figure 13-24\(a\)](#) illustrates an extreme example of nonlinear A/D converter operation with several binary output codes (words) having dropped bits in the time-domain $x(n)$ sequence with $m = 5$. The FFT magnitudes, provided in [Figure 13-24\(b\)](#), indicate severe A/D converter nonlinear distortion because we can see the increased background noise level compared to [Figure 13-23\(b\)](#). Performing [Eq. \(13-66'\)](#) for this noisy A/D gives us a measured SINAD value of 15.2 dB, which is drastically smaller than the ideal 5-bit A/D converter's $\text{SINAD} = 31.6$ dB. The point here is that we can quickly measure an A/D converter's performance using FFTs and [Eq. \(13-66'\)](#).

Figure 13-24 Nonideal A/D converter output showing several dropped bits: (a) time samples; (b) spectral magnitude in dB.



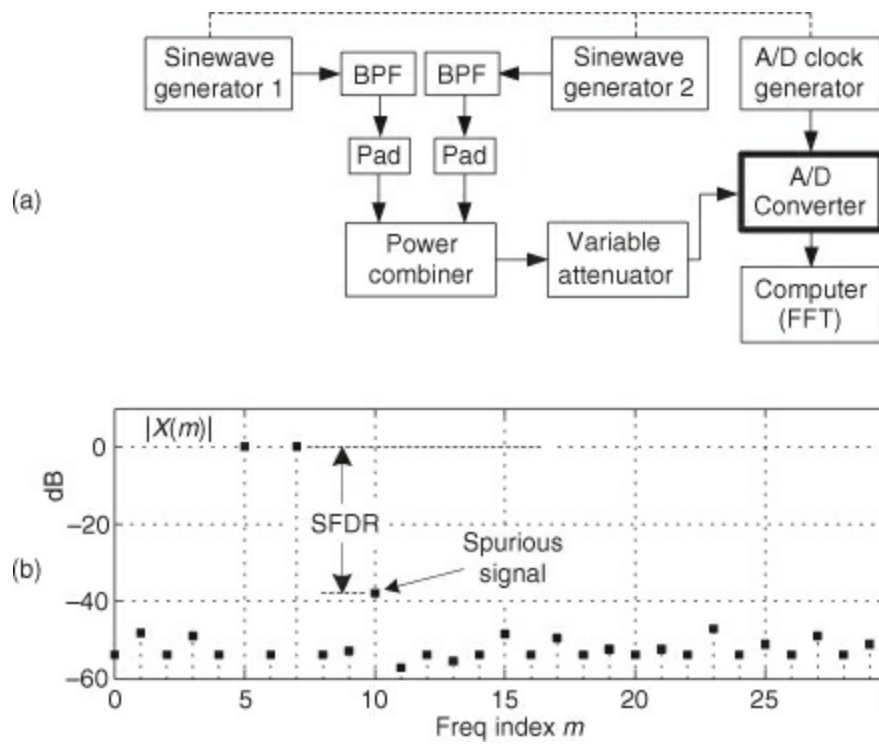
To fully characterize the dynamic performance of an A/D converter we'd need to perform this SINAD testing technique at many different input frequencies and amplitudes. (The analog sinewave applied to an A/D converter must, of course, be as *pure* as possible. Any distortion inherent in the analog signal will show up in the final FFT output and could be mistaken for A/D nonlinearity.) The key issue here is that when any input frequency is mf_s/N , where m is less than $N/2$ to satisfy the Nyquist sampling criterion, we can take full advantage of the FFT's processing capability while minimizing spectral leakage.

For completeness, we mention that what we called SINAD in [Eq. \(13-66'\)](#) is sometimes called SNDR. In addition, there is a measurement scheme called SINAD used by RF engineers to quantify the sensitivity of radio receivers. That receiver SINAD concept is quite different from our [Eq. \(13-66'\)](#) A/D converter SINAD estimation process and will not be discussed here.

13.9.2 Estimating A/D Dynamic Range

In this section we describe a technique of applying the sum of two analog sinewaves to an A/D converter's input to quantify the intermodulation distortion performance of a converter, which in turn measures the converter's dynamic range. That dynamic range is called the converter's *spurious free dynamic range* (SFDR). In this testing scheme both input sinewaves must comply with the mf_s/N restriction. [Figure 13-25\(a\)](#) shows the test configuration.

Figure 13-25 A/D converter SFDR testing: (a) hardware test configuration; (b) example test results.



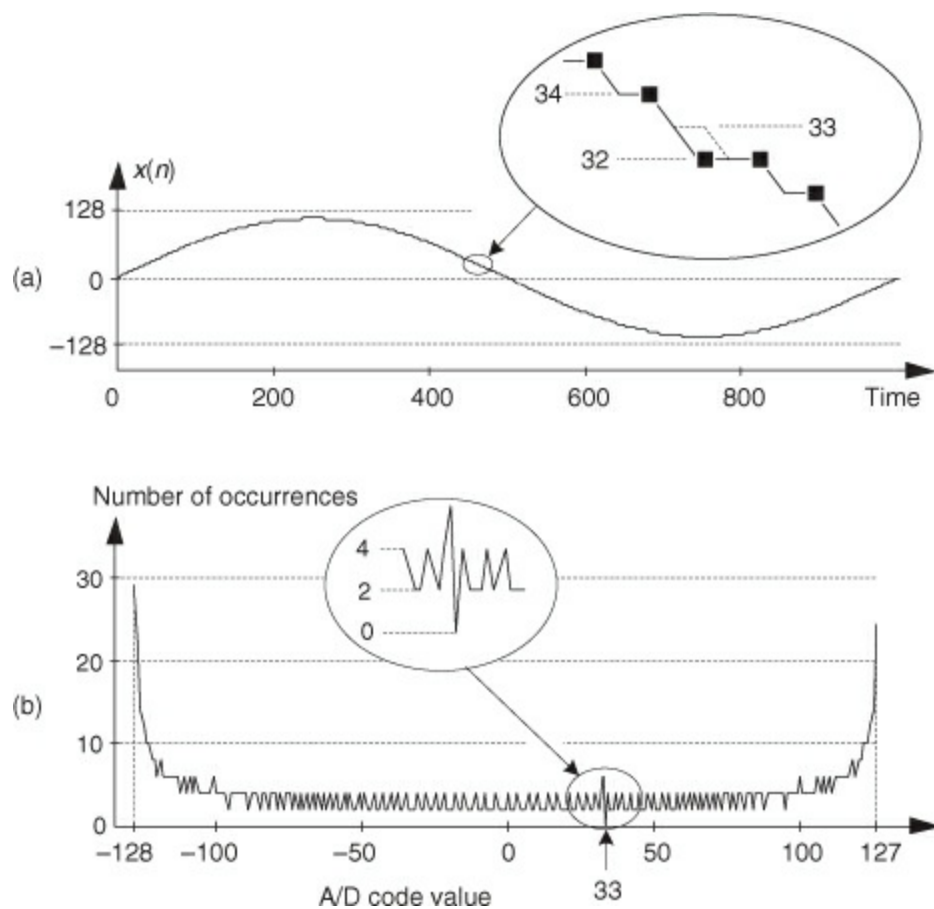
The SFDR test starts by applying the sum of two equal-amplitude analog sinewaves to an A/D converter and monitoring the spectrum of the converter's output samples. Next we increase both analog sinewaves' amplitudes until we see a spurious spectral component rising above the converter's background spectral noise as shown in [Figure 13-25\(b\)](#). Finally we measure the converter's SFDR as the dB difference between a high-level signal spectral magnitude sample and the spurious signal's spectral magnitude.

For this SFDR testing it's prudent to use bandpass filters (BPFs) to improve the spectral purity of the sinewave generators' outputs, and small-valued fixed attenuators (pads) are used to keep the generators from adversely interacting with each other. (I recommend 3 dB fixed attenuators for this.) The power combiner is typically an analog power splitter driven backward, and the A/D clock generator output is a squarewave. The dashed lines in [Figure 13-25\(a\)](#) indicate that all three generators are synchronized to the same reference frequency source.

13.9.3 Detecting Missing Codes

One problem that can plague A/D converters is *missing codes*. This defect occurs when a converter is incapable of outputting a specific binary word (a code). Think about driving an eight-bit converter with an analog sinusoid and the effect when its output should be the binary word 00100001 (decimal 33); its output is actually the word 00100000 (decimal 32) as shown in [Figure 13-26\(a\)](#). The binary word representing decimal 33 is a missing code. This subtle nonlinearity is very difficult to detect by examining time-domain samples or performing spectrum analysis. Fortunately there is a simple, reliable way to detect the missing 33 using histogram analysis.

Figure 13-26 Eight-bit converter missing codes: (a) missing code of binary 00100001, decimal 33; (b) histogram plot.



The histogram testing technique merely involves collecting many A/D converter output samples and plotting the number of occurrences of each sample value versus that sample value as shown in [Figure 13-26\(b\)](#). Any missing code (like our missing 33) would show up in the histogram as a zero value. That is, there were zero occurrences of the binary code representing a decimal 33.

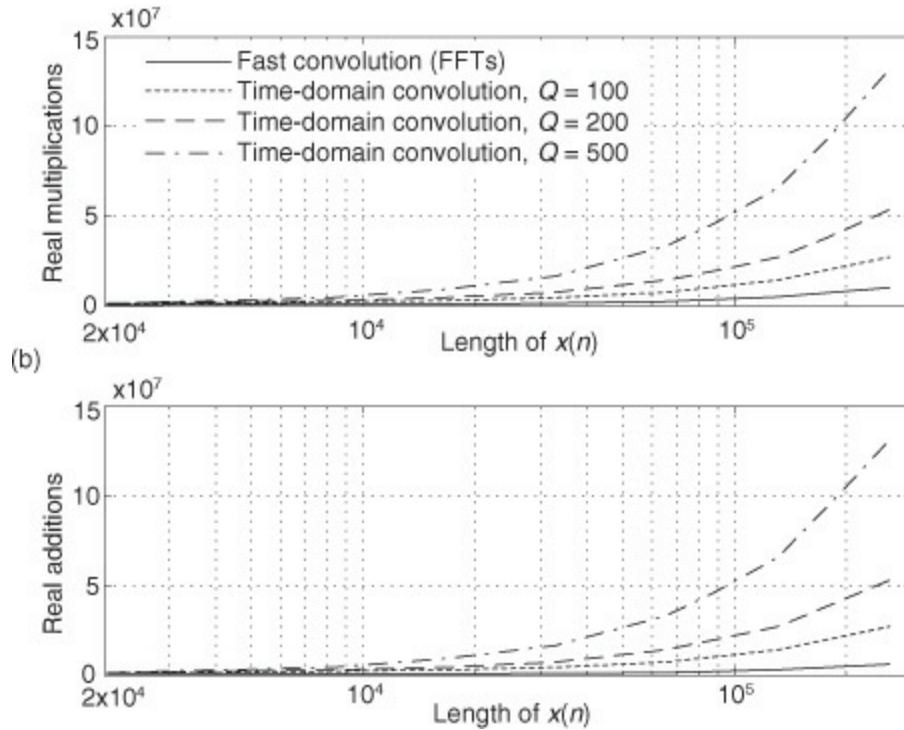
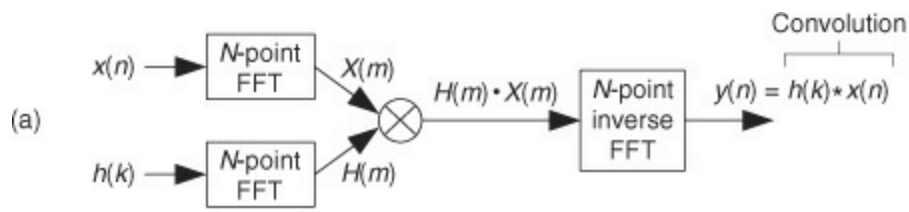
Additional useful information can be obtained from our histogram results. That is, counting the number of nonzero samples in [Figure 13-26\(b\)](#) tells us how many actual different A/D converter output codes (out of a possible 2^b codes) have been exercised.

In practice, the input analog sinewave must have an amplitude that's somewhat greater than the analog signal that we intend to digitize in an actual application, and a frequency that is unrelated to (incoherent with) the f_s sampling rate. In an effort to exercise (test) all of the converter's output codes, we digitize as many cycles of the input sinewave as possible for our histogram test.

13.10 Fast FIR Filtering Using the FFT

In the late 1960s, while contemplating the notion of time-domain convolution, DSP pioneer Thomas Stockham (digital audio expert and inventor of the compact disc) realized that time-domain convolution could sometimes be performed much more efficiently using fast Fourier transform (FFT) algorithms rather than using the direct convolution implemented with tapped-delay line FIR filters. The principle behind this FFT-based convolution scheme, called *fast convolution* (also called *block convolution* or *FFT convolution*), is diagrammed in [Figure 13-27\(a\)](#). In that figure $x(n)$ is an input signal sequence and $h(k)$ is the Q -length impulse response (coefficients) of a tapped-delay line FIR filter. [Figure 13-27\(a\)](#) is a graphical depiction of one form of the convolution theorem: Multiplication in the frequency domain is equivalent to convolution in the time domain.

Figure 13-27 Fast convolution: (a) basic process; (b) computational workloads for various FIR filter tap lengths Q .



The standard convolution equation, for a Q -tap FIR filter, given in [Eq. \(5-6\)](#) is repeated here for reference as

(13-67)

$$y(n) = \sum_{k=0}^{Q-1} h(k)x(n-k) = h(k) * x(n)$$

where the symbol “*” means convolution. When the filter’s $h(k)$ impulse response has a length greater than 40 to 80 (depending on the hardware and software being used), the process in [Figure 13-27\(a\)](#) requires fewer computations than directly implementing the convolution expression in [Eq. \(13-67\)](#). Consequently, this fast convolution technique is a computationally efficient signal processing tool, particularly when used for digital filtering. Fast convolution’s gain in computational efficiency becomes quite significant when the lengths of $h(k)$ and $x(n)$ are large.

[Figure 13-27\(b\)](#) indicates the reduction in the fast convolution algorithm’s computational workload relative to the standard (tapped-delay line) time-domain convolution method, [Eq. \(13-67\)](#), versus the length of the $x(n)$ sequence for various filter impulse response lengths Q . (Please do not view [Figure 13-27\(b\)](#) as any sort of *gospel truth*. That figure is merely an indicator of fast convolution’s computational efficiency.)

The necessary forward and inverse FFT sizes, N , in [Figure 13-27\(a\)](#) must of course be equal and are dependent upon the length of the original $h(k)$ and $x(n)$ sequences. Recall from [Eq. \(5-29\)](#) that if $h(k)$ is of length Q and $x(n)$ is of length P , the length of the final $y(n)$ sequence will be L where

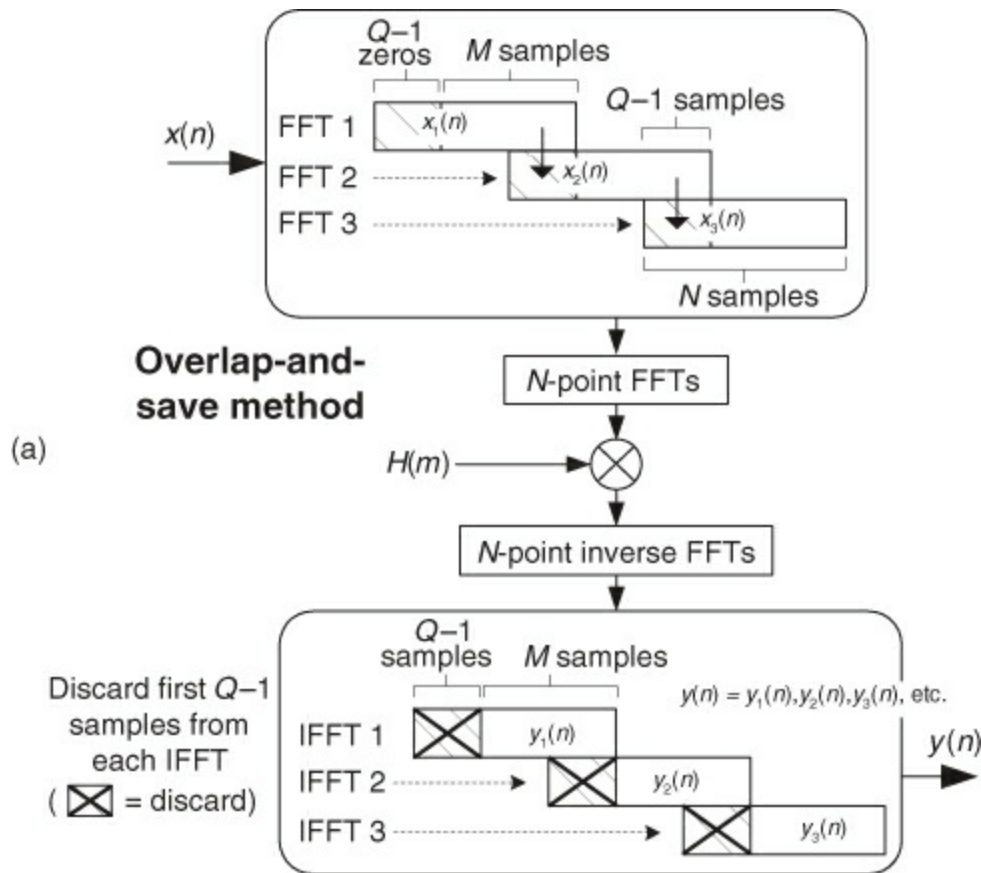
(13-67')

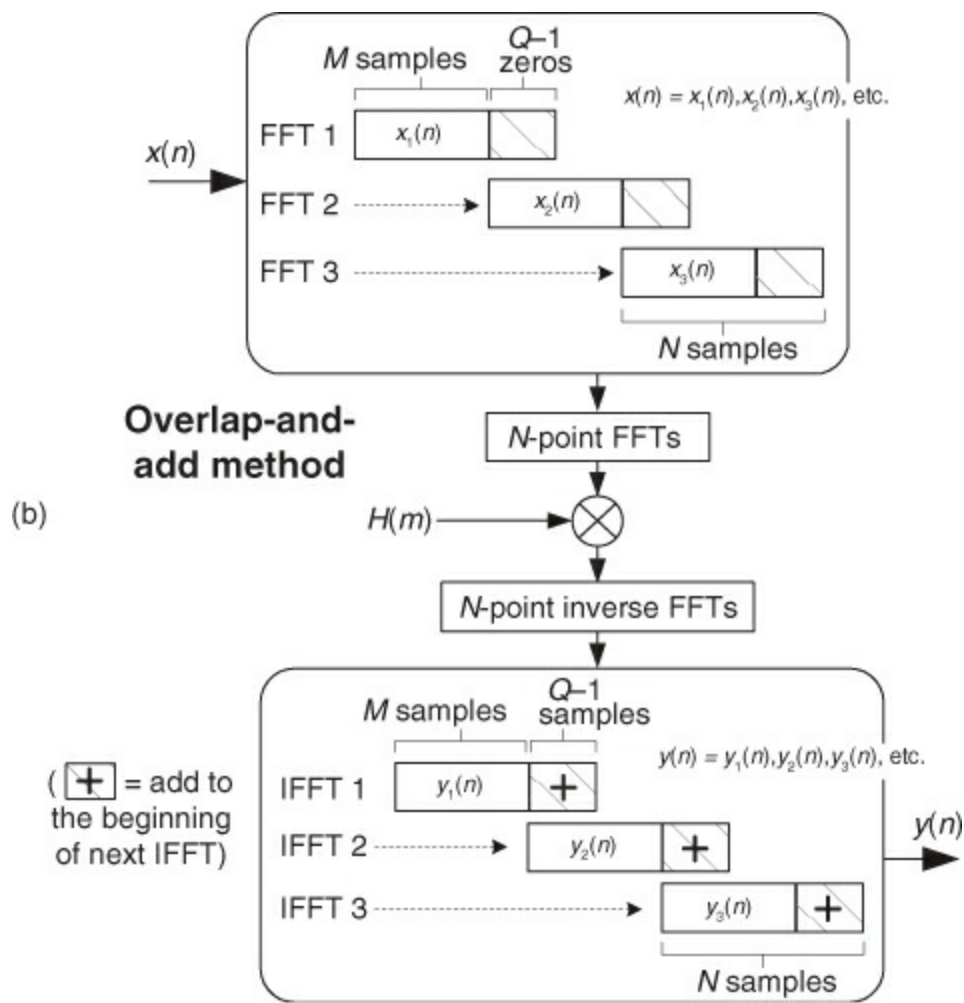
$$\text{Length of } y(n): L = Q + P - 1.$$

For this fast convolution technique to yield valid results, the forward and inverse FFT sizes *must* be equal to or greater than L . So, to implement fast convolution we must choose an N -point FFT size such that $N \geq L$, and zero-pad $h(k)$ and $x(n)$ so they have new lengths equal to N . The desired $y(n)$ output is the real part of the first L samples of the inverse FFT. Note that the $H(m)$ sequence, the FFT of the FIR filter's $h(k)$ impulse response, need only be computed once and stored in memory.

Now if the $x(n)$ input sequence length P is so large that FFT processing becomes impractical, or your hardware memory buffer can only hold small segments of the $x(n)$ time samples, then $x(n)$ must be partitioned into multiple blocks of samples and each sample block processed individually. If the partitioned- $x(n)$ block lengths are N , a straightforward implementation of [Figure 13-27\(a\)](#) leads to time-domain aliasing errors in $y(n)$ due to the circular nature (spectral wraparound) of the discrete Fourier transform (and the FFT). Two techniques are used to avoid that time-domain aliasing problem, the *overlap-and-save* method and the *overlap-and-add* method. Of these two methods, let's first have a look at the overlap-and-save fast convolution filtering technique shown in [Figure 13-28\(a\)](#).

Figure 13-28 Fast convolution block processing (continues).





Given that the desired FIR filter's $h(k)$ impulse response length is Q and the $x(n)$ filter input sequence is of length P , the steps to perform overlap-and-save fast convolution filtering are as follows:

1. Choose an FFT size of N , where N is an integer power of two equal to roughly four times Q .
2. Append $(N-Q)$ zero-valued samples to the end of the $h(k)$ impulse response and perform an N -point FFT on the extended sequence, producing the complex $H(m)$ sequence.
3. Compute integer M using $M = N-(Q-1)$.
4. Insert $(Q-1)$ zero-valued samples prior to the first M samples of $x(n)$, creating the first N -point FFT input sequence $x_1(n)$.
5. Perform an N -point FFT on $x_1(n)$, multiply that FFT result by the $H(m)$ sequence, and perform an N -point inverse FFT on the product. Discard the first $(Q-1)$ samples of the inverse FFT results to generate the first M -point output block of data $y_1(n)$.
6. Attach the last $(Q-1)$ samples of $x_1(n)$ to the beginning of the second M -length block of the original $x(n)$ sequence, creating the second N -point FFT input sequence $x_2(n)$ as shown in [Figure 13-28\(a\)](#).
7. Perform an N -point FFT on $x_2(n)$, multiply that FFT result by the $H(m)$ sequence, and perform an N -point inverse FFT on the product. Discard the first $(Q-1)$ samples of the inverse FFT results to generate the second M -point output block of data $y_2(n)$.
8. Repeat Steps 6 and 7 until we have gone through the entire original $x(n)$ filter input sequence. Depending on the length P of the original $x(n)$ input sequence and the chosen value for N , we must

append anywhere from $Q-1$ to $N-1$ zero-valued samples to the end of the original $x(n)$ input samples in order to accommodate the final block of forward and inverse FFT processing.

9. Concatenate the $y_1(n)$, $y_2(n)$, $y_3(n)$, . . . sequences shown in [Figure 13-28\(a\)](#), discarding any unnecessary trailing zero-valued samples, to generate your final linear-convolution filter output $y(n)$ sequence.
10. Finally, experiment with different values of N to see if there exists an *optimum* N that minimizes the computational workload for your hardware and software implementation. In any case, N must not be less than $(M+Q-1)$. (Smaller N means many small-sized FFTs are needed, and large N means fewer, but larger-sized, FFTs are necessary. Pick your poison.)

The second fast convolution method, the *overlap-and-add* technique, is shown in [Figure 13-28\(b\)](#). In this method, the $x(n)$ input sequence is partitioned (segmented) into data blocks of length M , and our data overlapping takes place in the inverse FFT time-domain sequences. Given that the desired FIR filter's $h(k)$ impulse response length is Q and the $x(n)$ filter input sequence is of length P , the steps to perform overlap-and-add fast convolution filtering are as follows:

1. Choose an FFT size of N , where N is an integer power of two equal to roughly two times Q .
2. Append $(N-Q)$ zero-valued samples to the end of the $h(k)$ impulse response and perform an N -point FFT on the extended sequence, producing the complex $H(m)$ sequence.
3. Compute integer M using $M = N-(Q-1)$.
4. Append $(Q-1)$ zero-valued samples to the end of the first M samples, $x_1(n)$, of the original $x(n)$ sequence, creating the first N -point FFT input sequence.
5. Perform an N -point FFT on the first N -point FFT input sequence, multiply that FFT result by the $H(m)$ sequence, and perform an N -point inverse FFT on the product. Retain the first M samples of the inverse FFT sequence, generating the first M -point output block of data $y_1(n)$.
6. Append $(Q-1)$ zero-valued samples to the end of the second M samples, $x_2(n)$, of the original $x(n)$ sequence, creating the second N -point FFT input sequence.
7. Perform an N -point FFT on the second N -point FFT input sequence, multiply that FFT result by the $H(m)$ sequence, and perform an N -point inverse FFT on the product. Add the last $(Q-1)$ samples from the previous inverse FFT to the first $(Q-1)$ samples of the current inverse FFT sequence. Retain the first M samples of the sequence resulting from the $(Q-1)$ -element addition process, generating the second M -point output block of data $y_2(n)$.
8. Repeat Steps 6 and 7 until we have gone through the entire original $x(n)$ filter input sequence. Depending on the length P of the original $x(n)$ input sequence and the chosen value for N , we must append anywhere from $Q-1$ to $N-1$ zero-valued samples to the end of the original $x(n)$ input samples in order to accommodate the final block of forward and inverse FFT processing.
9. Concatenate the $y_1(n)$, $y_2(n)$, $y_3(n)$, . . . sequences shown in [Figure 13-28\(b\)](#), discarding any unnecessary trailing zero-valued samples, to generate your final linear-convolution filter output $y(n)$ sequence.
10. Finally, experiment with different values of N to see if there exists an *optimum* N that minimizes the computational workload for your hardware and software implementation. N must not be less

than $(M+Q-1)$. (Again, smaller N means many small-sized FFTs are needed, and large N means fewer, but larger-sized, FFTs are necessary.)

It's useful to realize that the computational workload of these fast convolution filtering schemes does not change as Q increases in length up to a value of N . Another interesting aspect of fast convolution, from a hardware standpoint, is that the FFT indexing bit-reversal problem discussed in [Sections 4.5](#) and [4.6](#) is not an issue here. If the FFTs result in $X(m)$ and $H(m)$ having bit-reversed output sample indices, the multiplication can still be performed directly on the scrambled $H(m)$ and $X(m)$ sequences. Then an appropriate inverse FFT structure can be used that expects bit-reversed input data. That inverse FFT then provides an output sequence whose time-domain indexing is in the correct order. Neat!

By the way, it's worth knowing that there are no restrictions on the filter's finite-length $h(k)$ impulse response— $h(k)$ is not limited to being real-valued and symmetrical as is traditional with tapped-delay line FIR filters. Sequence $h(k)$ can be complex-valued, asymmetrical (to achieve nonlinear-phase filtering), or whatever you choose.

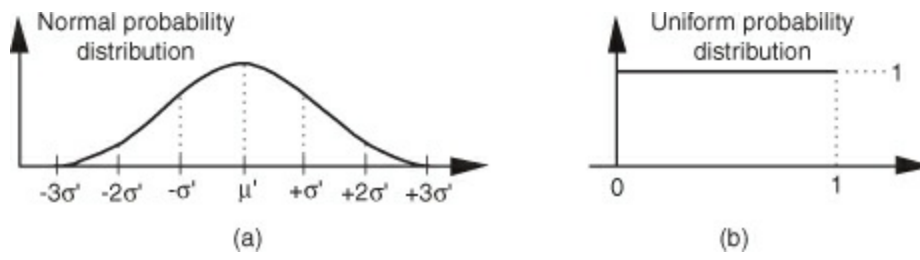
One last issue to bear in mind: the complex amplitudes of the standard radix-2 FFT's output samples are proportional to the FFT sizes, N , so the product of two FFT outputs will have a gain proportional to N^2 . The inverse FFT has a normalizing gain reduction of only $1/N$. As such, our fast convolution filtering methods will have an overall gain that is not unity. We suggest that practitioners give this gain normalization topic some thought during the design of their fast convolution system.

To summarize this frequency-domain filtering discussion, the two fast convolution filtering schemes can be computationally efficient, compared to standard tapped-delay line FIR convolution filtering, particularly when the $x(n)$ input sequence is large and high-performance filtering is needed (requiring many filter taps, i.e., $Q = 40$ to 80). As for which method, overlap-and-save or overlap-and-add, should be used in any given situation, there is no simple answer. Choosing a fast convolution method depends on many factors: the fixed/floating-point arithmetic used, memory size and access latency, computational hardware architecture, and specialized built-in filtering instructions, etc.

13.11 Generating Normally Distributed Random Data

[Section D.7](#) in [Appendix D](#) discusses the normal distribution curve as it relates to random data. A problem we may encounter is how actually to generate random data samples whose distribution follows that normal (Gaussian) curve. There's a straightforward way to solve this problem using any software package that can generate uniformly distributed random data, as most of them do[\[27\]](#). [Figure 13-29](#) shows our situation pictorially where we require random data that's distributed normally with a mean (average) of μ' and a standard deviation of σ' , as in [Figure 13-29\(a\)](#), and all we have available is a software routine that generates random data that's uniformly distributed between zero and one as in [Figure 13-29\(b\)](#).

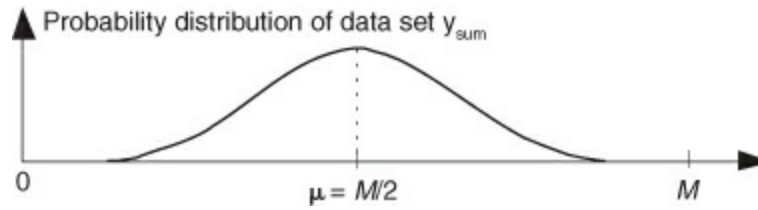
Figure 13-29 Probability distribution functions: (a) normal distribution with mean $= \mu'$ and standard deviation σ' ; (b) uniform distribution between zero and one.



As it turns out, there's a principle in advanced probability theory, known as the *Central Limit Theorem*, that says when random data from an arbitrary distribution is summed over M samples, the probability distribution of the sum begins to approach a normal distribution as M increases[28–30]. In other words, if we generate a set of N random samples that are uniformly distributed between zero and one, we can begin adding other sets of N samples to the first set. As we continue summing additional sets, the distribution of the N -element set of sums becomes more and more *normal*. We can sound impressive and state that “the sum becomes asymptotically normal.” Experience has shown that for practical purposes, if we sum $M \geq 30$ times, the summed data distribution is essentially normal. With this rule in mind, we're halfway to solving our problem.

After summing M sets of uniformly distributed samples, the summed set y_{sum} will have a distribution as shown in [Figure 13-30](#).

Figure 13-30 Probability distribution of the summed set of random data derived from uniformly distributed data.



Because we've summed M data sets whose mean values were all 0.5, the mean of y_{sum} is the sum of those M means, or $\mu = M/2$. From [Section D.6](#) of [Appendix D](#) we know the variance of a single data sample set, having the probability distribution in [Figure 13-29\(b\)](#), is $1/12$. Because the variance of the sum of M data sets is equal to the sum of their individual variances, we can say

$$(13-68)$$

$$\text{variance of } y_{\text{sum}} \text{ is: } \sigma^2 \approx \frac{M}{12}$$

and

$$(13-69)$$

$$\text{standard deviation of } y_{\text{sum}} \text{ is: } \sigma \approx \sqrt{\frac{M}{12}}.$$

So, here's the trick: To convert the y_{sum} data set to our desired data set having a mean of μ' and a standard deviation of σ' , we

1. subtract $M/2$ from each element of y_{sum} to shift its mean to zero;
2. scale y_{sum} so that its standard deviation is the desired σ' , by multiplying each sample in the shifted data set by σ'/σ ; and
3. finally, center the new data set at the desired μ' value by adding μ' to each sample of the scaled data set.

If we call our desired normally distributed random data set y_{desired} , then the n th element of that set is described mathematically as

(13-70)

$$y_{\text{desired}}(n) = \sqrt{\frac{12}{M}} \cdot \sigma' \cdot \left[\left(\sum_{k=1}^M x_k(n) \right) - \frac{M}{2} \right] + \mu'.$$

Our discussion thus far has had a decidedly software algorithm flavor, but hardware designers also occasionally need to generate normally distributed random data at high speeds in their designs. For you hardware designers, reference [30] presents an efficient hardware design technique to generate normally distributed random data using fixed-point arithmetic integrated circuits.

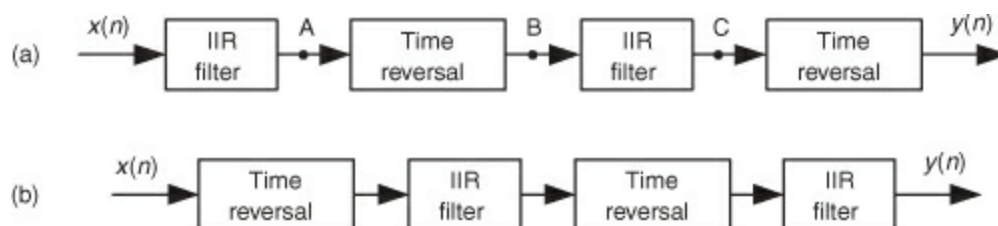
The above method for generating normally distributed random numbers works reasonably well, but its results are not perfect because the tails of the probability distribution curve in [Figure 13-30](#) are not perfectly Gaussian.[‡] An advanced, and more statistically correct (improved randomness), technique that you may want to explore is called the Ziggurat method[31–33].

[‡] I thank my DSP pal Dr. Peter Kootsookos, of UTC Fire and Security, Farmington, Connecticut, for his advice on this issue.

13.12 Zero-Phase Filtering

You can cancel the nonlinear phase effects of an IIR filter by following the process shown in [Figure 13-31\(a\)](#). The $y(n)$ output will be a filtered version of $x(n)$ with no filter-induced phase distortion. The same IIR filter is used twice in this scheme, and the time reversal step is a straight left-right flipping of a time-domain sequence. Consider the following. If some spectral component in $x(n)$ has an arbitrary phase of α degrees, and the first filter induces a phase shift of $-\beta$ degrees, that spectral component's phase at node A will be $\alpha - \beta$ degrees. The first time reversal step will conjugate that phase and induce an additional phase shift of $-\theta$ degrees. ([Appendix C](#) explains this effect.) Consequently, the component's phase at node B will be $-\alpha + \beta - \theta$ degrees. The second filter's phase shift of $-\beta$ degrees yields a phase of $-\alpha - \theta$ degrees at node C. The final time reversal step (often omitted in literary descriptions of this zero-phase filtering process) will conjugate that phase and again induce an additional phase shift of $-\theta$ degrees. Thankfully, the spectral component's phase in $y(n)$ will be $\alpha + \theta - \theta = \alpha$ degrees, the same phase as in $x(n)$. This property yields an overall filter whose phase response is zero degrees over the entire frequency range.

Figure 13-31 Two equivalent zero-phase filtering techniques.



An equivalent zero-phase filter is presented in [Figure 13-31\(b\)](#). Of course, these methods of zero-phase filtering cannot be performed in real time because we can't reverse the flow of time (at least not in our universe). This filtering is a *block processing*, or *off-line*, process, such as filtering an audio file stored in a computer. We must have all the time samples available before we start processing. The initial time reversal in [Figure 13-31\(b\)](#) illustrates this restriction.

There will be filter transient effects at the beginning and end of the filtered sequences. If transient

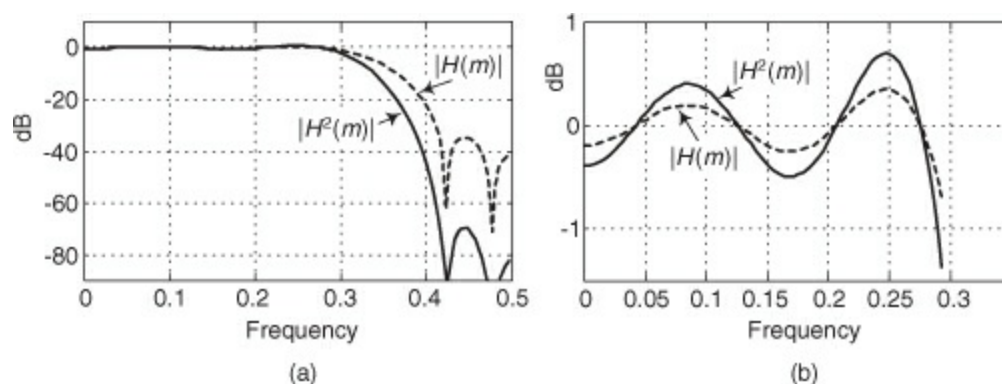
effects are bothersome in a given application, consider discarding L samples from the beginning and end of the final $y(n)$ time sequence, where L is four (or five) times the order of the IIR filter.

By the way, the final peak-to-peak passband ripple (in dB) of this zero-phase filtering process will be twice the peak-to-peak passband ripple of the single IIR filter. The final stopband attenuation will also be double that of the single filter.

13.13 Sharpened FIR Filters

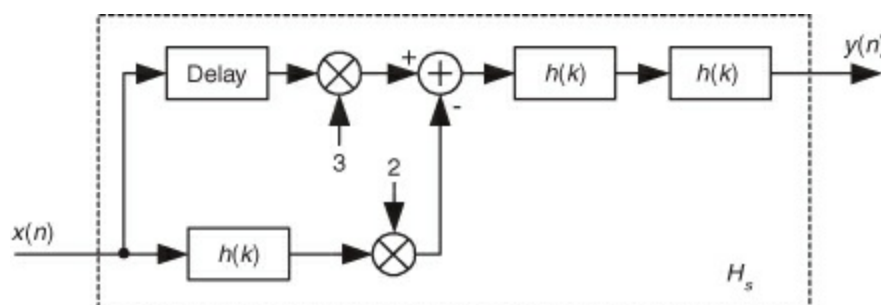
Here's an interesting technique for improving the stopband attenuation of a digital filter under the condition that we're unable, for whatever reason, to modify that filter's coefficients. Actually, we can double a filter's stopband attenuation by cascading the filter with itself. This works, as shown in [Figure 13-32\(a\)](#), where the frequency magnitude response of a single filter is a dashed curve $|H(m)|$ and the response of the filter cascaded with itself is represented by the solid curve $|H^2(m)|$. The problem with this simple cascade idea is that it also doubles the passband peak-to-peak ripple as shown in [Figure 13-32\(b\)](#). The frequency axis in [Figure 13-32](#) is normalized such that a value of 0.5 represents half the signal sample rate.

Figure 13-32 Frequency magnitude responses of a single filter and that filter cascaded with itself: (a) full response; (b) passband detail.



Well, there's a better scheme for improving the stopband attenuation performance of a filter and avoiding passband ripple degradation without actually changing the filter's coefficients. The technique is called *filter sharpening* [\[34\]](#) and is shown as H_s in [Figure 13-33](#).

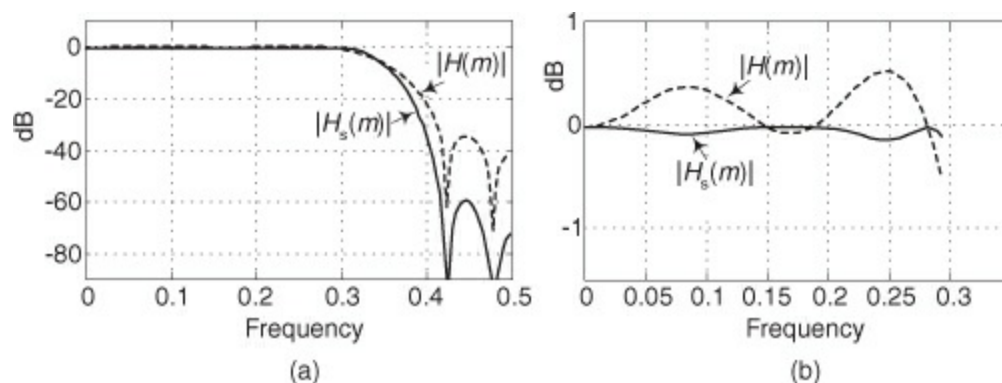
Figure 13-33 Filter sharpening process.



The delay element in [Figure 13-33](#) is equal to $(N-1)/2$ samples where N is the number of $h(k)$ coefficients, the unit-impulse response length, in the original $H(m)$ FIR filter. Using the sharpening process results in the improved $|H_s(m)|$ filter performance shown as the solid curve in [Figure 13-34](#), where we see the increased stopband attenuation and reduced passband ripple beyond that afforded by the original $H(m)$ filter. Because of the delayed time-alignment constraint, filter sharpening is not applicable to filters having non-constant group delay, such as minimum-phase FIR filters or IIR

filters.

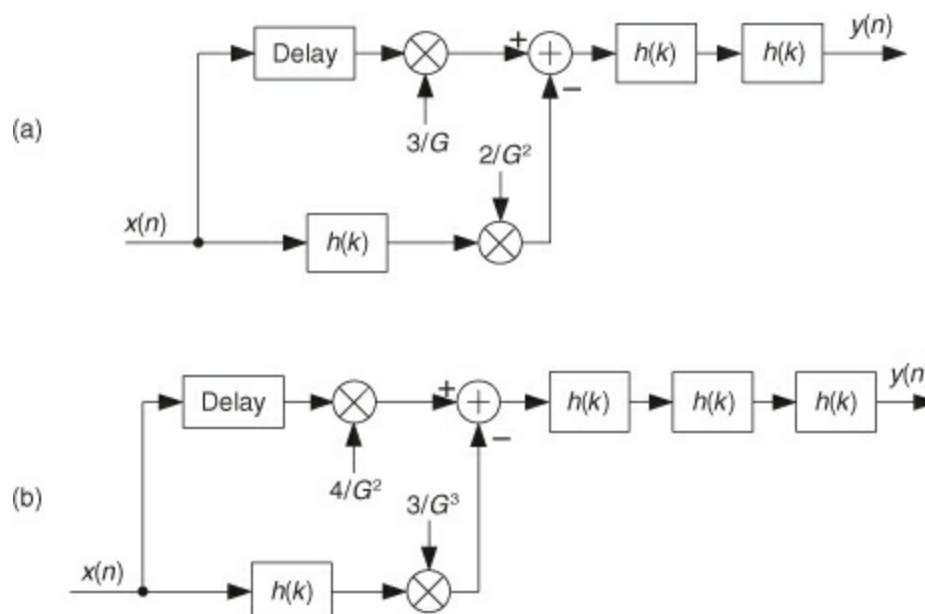
Figure 13-34 $|H(m)|$ and $|H_s(m)|$ performance: (a) full frequency response; (b) passband detail.



If need be, we can eliminate the multipliers shown in [Figure 13-33](#). The multiply by two operation can be implemented with an arithmetic left shift by one binary bit. The multiply by three operation can be implemented by adding the Delay output sample to a shifted-left-by-one-bit version of itself.

Be aware that the gain factors in [Figure 13-33](#) are based on the assumption that the original $h(k)$ filter to be sharpened has a passband gain of one. If the $h(k)$ filter has a non-unity passband gain of $G \neq 1$, then the network in [Figure 13-35\(a\)](#) should be used, where the alternate constant gain factors provide optimum filter sharpening. On the other hand, the [Figure 13-35\(a\)](#) gain factors can be modified to some extent if doing so simplifies the filter implementation. For example, if $2/G^2 = 1.7$, for ease of implementation, the practitioner should try using a factor of 2 in place of the factor 1.7. Using a gain factor of 2 will not be optimum but it may well be acceptable, depending on the characteristics of the filter to be sharpened. Software modeling will resolve this issue.

Figure 13-35 Non-unity gain filter sharpening: (a) low-order sharpening; (b) higher-order sharpening for increased stopband attenuation.



If additional stopband attenuation is needed, then the process shown in [Figure 13-35\(b\)](#) can be used, where again the Delay element is equal to $(N-1)/2$ unit delays.

In real-time applications, the filter sharpening networks we presented are straightforward and applicable to linear-phase lowpass, bandpass, and highpass FIR filters, just so long as the original filter's $H(f)$ has an integer group delay. (That restriction is necessary because the number of unit delays of the Delay element, needed for time synchronization in real-time systems, in the parallel path

must be an integer.) This sharpening procedure is particularly useful if the original filter hardware is constrained to have some fixed number of bits to represent its coefficients. If an FIR filter's coefficient bit width is b bits, the filter sharpening process in [Figure 13-33](#) can, luckily for us, achieve the performance of filters having $(b + 4)$ -bit coefficients. So, if our hardware forces us to use, say, 8-bit coefficients, we can achieve roughly 12-bit-coefficient filter performance.

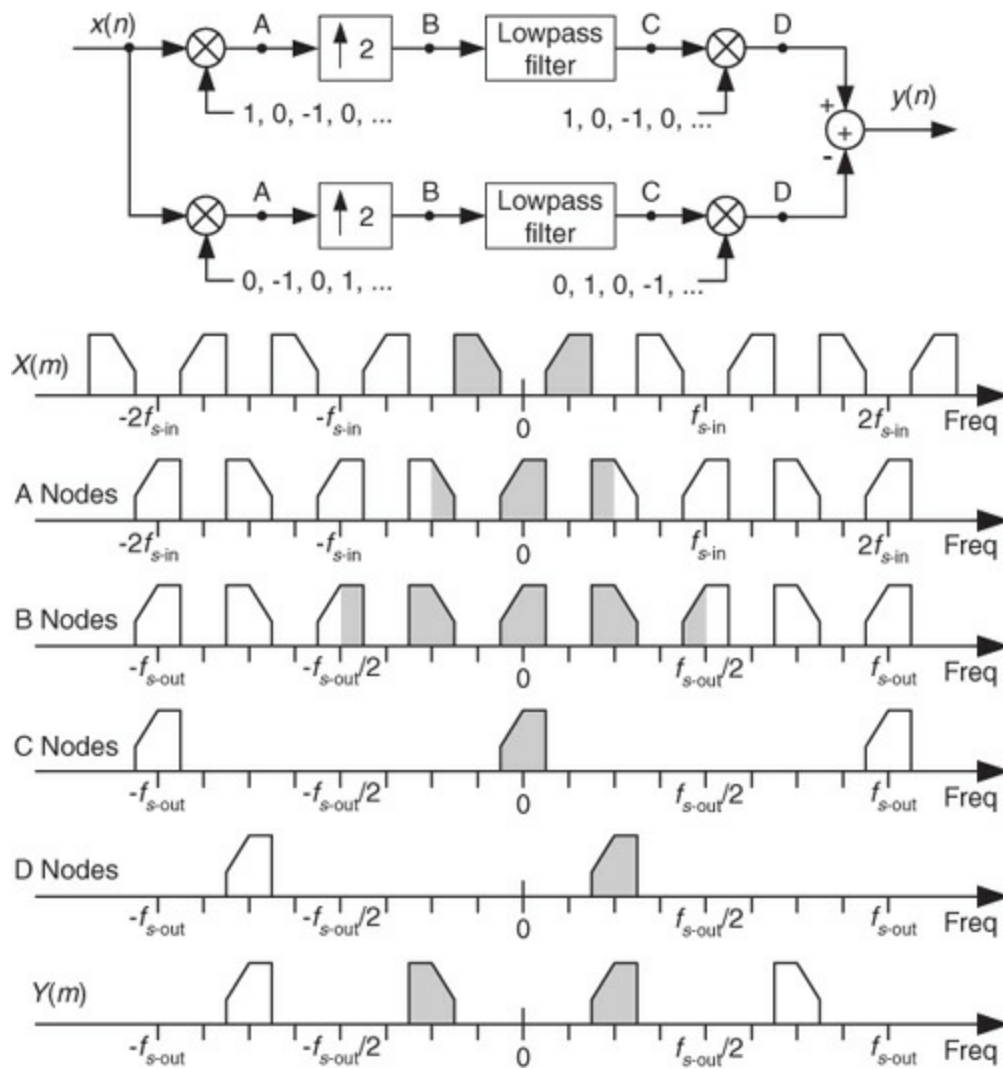
Filter sharpening can be used whenever a given filter response cannot be modified, such as an unchangeable software subroutine, and can even be applied to cascaded integrator-comb (CIC) filters to flatten their passband responses, as well as FIR fixed-point multiplierless filters where the coefficients are constrained to be powers of two[[35,36](#)].

As a historical aside, *filter sharpening* is a process refined and expanded by the accomplished R. Hamming (of Hamming window fame) based on an idea originally proposed by the great American mathematician John Tukey, the inventor of the radix-2 fast Fourier transform (FFT).

13.14 Interpolating a Bandpass Signal

There are many digital communications applications where a real signal is centered at one-fourth the sample rate, or $f_s/4$. This condition makes quadrature down-conversion particularly simple. (See [Sections 8.9](#) and [13.1](#).) In the event that you'd like to generate an interpolated (increased sample rate) version of the bandpass signal but maintain its $f_s/4$ center frequency, there's an efficient way to do so[[37](#)]. Suppose we want to interpolate by a factor of two so the output sample rate is twice the input sample rate, $f_{s-out} = 2f_{s-in}$. In this case the process is: quadrature down-conversion by $f_{s-in}/4$, interpolation factor of two, quadrature up-conversion by $f_{s-out}/4$, and then take only the real part of the complex upconverted sequence. The implementation of this scheme is shown at the top of [Figure 13-36](#).

Figure 13-36 Bandpass signal interpolation scheme, and spectra.



The sequences applied to the first multiplier in the top signal path are the real $x(n)$ input and the repeating mixing sequence 1,0,-1,0. That mixing sequence is the real (or in-phase) part of the complex exponential

$$(13-71)$$

$$e^{-j2\pi(f_{s-in}/4)t_{s-in}} = e^{-j2\pi(f_{s-in}/4)(1/f_{s-in})} = e^{-j2\pi(1/4)}$$

needed for quadrature down-conversion by $f_s/4$. Likewise, the repeating mixing sequence 0,-1,0,1 applied to the first multiplier in the bottom path is the imaginary (or quadrature phase) part of the complex down-conversion exponential $e^{-j2\pi(f_{s-in}/4)t_{s-in}}$. The “ $\uparrow 2$ ” symbol means insert one zero-valued sample between each sample at the A nodes. The final subtraction to obtain $y(n)$ is how we extract the real part of the complex sequence at Node D. (That is, we’re extracting the real part of the product of the complex signal at Node C times $e^{j2\pi(1/4)}$.) The spectra at various nodes of this process are shown at the bottom of [Figure 13-35](#). The shaded spectra indicate true spectral components, while the white spectra represent spectral replications. Of course, the same lowpass filter must be used in both processing paths to maintain the proper time delay and orthogonal phase relationships.

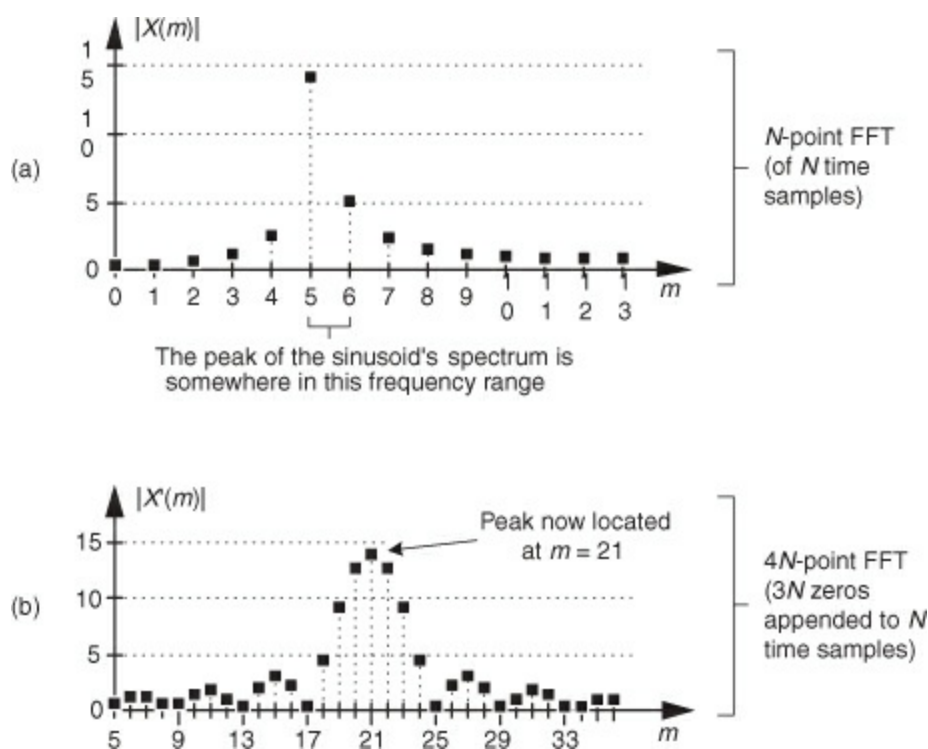
There are several additional issues worth considering regarding this interpolation process[\[38\]](#). If the amplitude loss, inherent in interpolation, of a factor of two is bothersome, we can make the final mixing sequences 2,0,-2,0 and 0,2,0,-2 to compensate for that loss. Because there are so many zeros in the sequences at Node B (three-fourths of the samples), we should consider those efficient polyphase filters for the lowpass filtering. Finally, if it’s sensible in your implementation, consider replacing the final adder with a multiplexer (because alternate samples of the sequences at Node D

are zeros). In this case, the mixing sequence in the bottom path would be changed to 0,−1,0,1.

13.15 Spectral Peak Location Algorithm

In the practical world of discrete spectrum analysis, we often want to estimate the frequency of a sinusoid (or the center frequency of a very narrowband signal of interest). Upon applying the radix-2 fast Fourier transform (FFT), our narrowband signals of interest rarely reside exactly on an FFT bin center whose frequency is exactly known. As such, due to the FFT's leakage properties, the discrete spectrum of a sinusoid having N time-domain samples may look like the magnitude samples shown in [Figure 13-37\(a\)](#). There we see the sinusoid's spectral peak residing between the FFT's $m = 5$ and $m = 6$ bin centers. (Variable m is an N -point FFT's frequency-domain index. The FFT bin spacing is f_s/N where, as always, f_s is the sample rate.) Close examination of [Figure 13-37\(a\)](#) allows us to say the sinusoid lies in the range of $m = 5$ and $m = 5.5$, because we see that the maximum spectral sample is closer to the $m = 5$ bin center than the $m = 6$ bin center. The real-valued sinusoidal time signal has, in this example, a frequency of $5.25f_s/N$ Hz. In this situation, our frequency estimation resolution is half the FFT bin spacing. We often need better frequency estimation resolution, and there are indeed several ways to improve that resolution.

Figure 13-37 Spectral magnitudes: (a) N -point FFT; (b) $4N$ -point FFT.



We could collect, say, $4N$ time-domain signal samples and perform a $4N$ -point FFT, yielding a reduced bin spacing of $f_s/4N$. Or we could pad (append to the end of the original time samples) the original N time samples with $3N$ zero-valued samples and perform a $4N$ -point FFT on the lengthened time sequence. That would also provide an improved spectral peak estimation granularity of $f_s/4N$, as shown in [Figure 13-37\(b\)](#). With the spectral peak located at bin $m_{\text{peak}} = 21$, we estimate the signal's center frequency, in Hz, using $f_{\text{peak}} = m_{\text{peak}}f_s/4N$.

Both schemes, *collect more data* and *zero padding*, are computationally expensive. Many other techniques for enhanced-precision tone frequency measurement have been described in the scientific literature—from the close-to-home field of geophysics to the lofty studies of astrophysics—but most

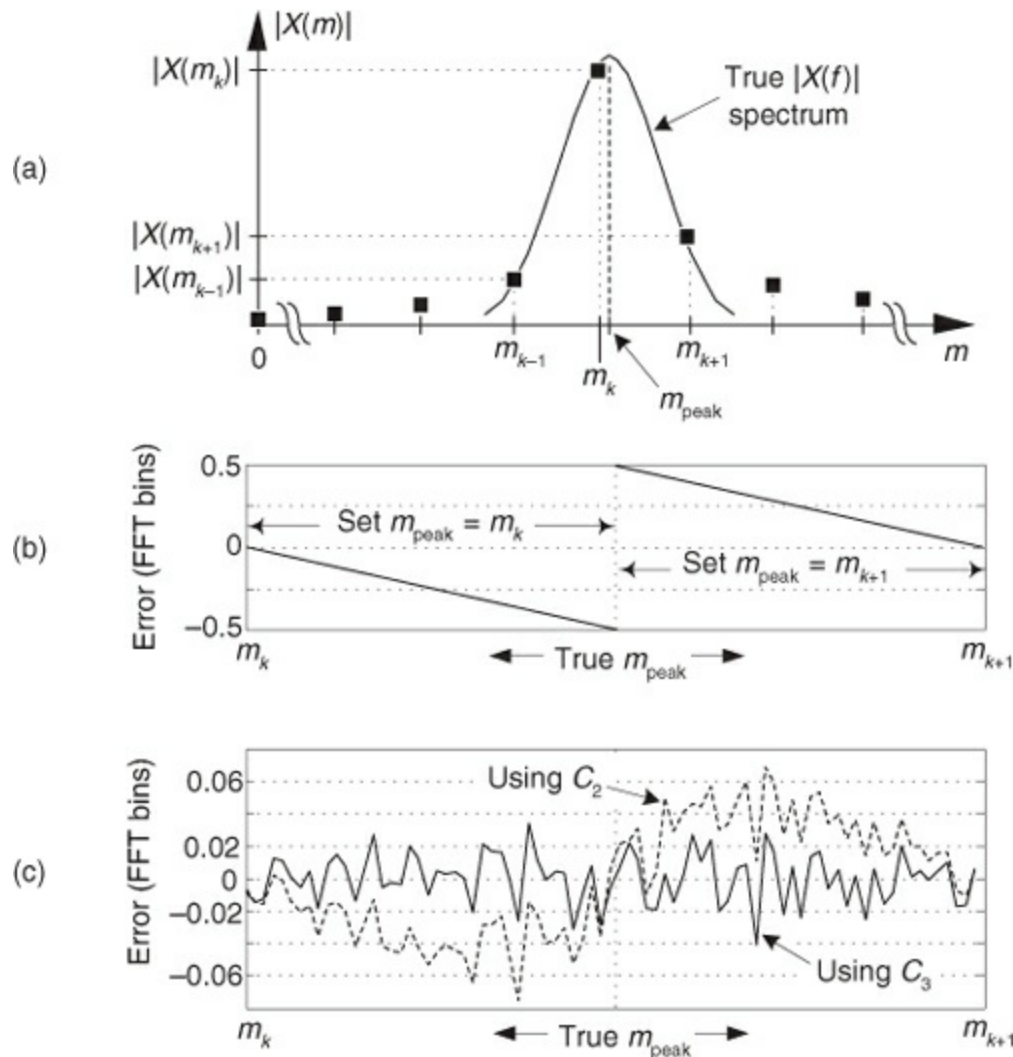
of those schemes seek precision without regard to computational complexity. Here we describe several computationally simple frequency estimation schemes.

Assume we have the $X(m)$ spectral samples from an N -point FFT of a sinusoidal time signal, whose magnitudes are shown in [Figure 13-38\(a\)](#). (The vertical magnitude axis is linear, not logarithmic.) The notation in the figure is that m_k is the integer index of the largest magnitude sample $|X(m_k)|$. The value m_{peak} , which in general will not be an integer, is the value we wish to estimate and use in

$$(13-72)$$

$$f_{\text{peak}} = m_{\text{peak}} \cdot \frac{f_s}{N}$$

Figure 13-38 Spectral peak detection: (a) FFT magnitudes; (b) m_{peak} error by naive assignment; (c) m_{peak} algorithm error performance.



to accurately estimate the sinusoid's center frequency in Hz.

Next, let's say the FFT's input sinusoid sweeps in frequency starting at the FFT's m_k bin center frequency to the center frequency of the m_{k+1} bin and we assign m_{peak} to be equal to the index value (either m_k or m_{k+1}) of the highest spectral magnitude sample. The error in that m_{peak} value will be that shown in [Figure 13-38\(b\)](#). The maximum error in that naive m_{peak} assignment scheme is 0.5 FFT bins (half the FFT bin spacing). Happily for us, there are more accurate methods for estimating m_{peak} .

As it turns out, we can estimate the signal's index-based center frequency, m_{peak} , using

(13-73)

$$m_{\text{peak}} = m_k + C_i$$

where C_i is a scalar correction factor in the range of $-0.5 \leq C_i \leq 0.5$. There are many algorithms, based on fitting a generic parabolic curve to the $|X(m)|$ samples, floating around in the literature of DSP for estimating C_i . Those algorithms have varying degrees of accuracy depending on the window function applied to the FFT's input samples.

A noteworthy correction factor expression is

(13-74)

$$C_1 = \text{real part of } \frac{X(m_{k-1}) - X(m_{k+1})}{2X(m_k) - X(m_{k-1}) - X(m_{k+1})}.$$

This complex-valued spectral peak location estimation algorithm is quite accurate for its simplicity[3]. Its maximum frequency estimation error is roughly 0.06, 0.04, and 0.03 bin widths for signal-to-noise ratios of 3, 6, and 9 dB respectively. Not bad at all! The nice features of the algorithm are that it does not require the original time samples to be windowed, as do some other spectral peak location algorithms; and it does not require computation of FFT magnitude samples.

If a time-domain window sequence has been applied to the FFT's input samples, then other C_i correction factor expressions should be used in place of [Eq. \(13-74\)](#). Three notable candidate expressions for C_i are

(13-75)

$$C_2 = \frac{|X(m_{k+1})| - |X(m_{k-1})|}{4|X(m_k)| - 2|X(m_{k-1})| - 2|X(m_{k+1})|}$$

(13-75')

$$C_3 = \frac{P[|X(m_{k+1})| - |X(m_{k-1})|]}{|X(m_{k-1})| + |X(m_k)| + |X(m_{k+1})|}$$

(13-75'')

$$C_4 = \text{real part of } \frac{Q[X(m_{k-1}) - X(m_{k+1})]}{2X(m_k) + X(m_{k-1}) + X(m_{k+1})},$$

where again we use subscripts on C merely to identify the different expressions for the correction factor C_i . The above window-dependent P and Q factors, determined empirically, are

- Hamming, $P = 1.22$, $Q = 0.60$;
- Hanning, $P = 1.36$, $Q = 0.55$;
- Blackman, $P = 1.75$, $Q = 0.55$; and
- Blackman-Harris (3-term), $P = 1.72$, $Q = 0.56$.

[Equation \(13-75\)](#) is the best known peak location algorithm and has been used in the DSP business for decades. The lesser-known [Eq. \(13-75'\)](#) provides a more accurate windowed-FFT peak location estimate than [Eq. \(13-75\)\[39\]](#). Inspired by [Eqs. \(13-74\)](#) and [\(13-75'\)](#), the author has developed [Eq. \(13-75''\)](#) which can be used in case the FFT magnitude samples are unavailable for use in [Eq. \(13-75'\)](#). [Equation \(13-75''\)](#) is also more accurate than the better-known [Eq. \(13-75\)](#).

The solid curve in [Figure 13-38\(c\)](#) shows the m_{peak} error in using [Eq. \(13-75'\)](#) with Blackman-

windowed time-domain samples whose signal-to-noise ratio is 9 dB. For comparison, the dashed curve is the m_{peak} error when using [Eq. \(13-75\)](#). [Equation \(13-75\)](#)'s accuracy is very similar to that of [Eq. \(13-75'\)](#).

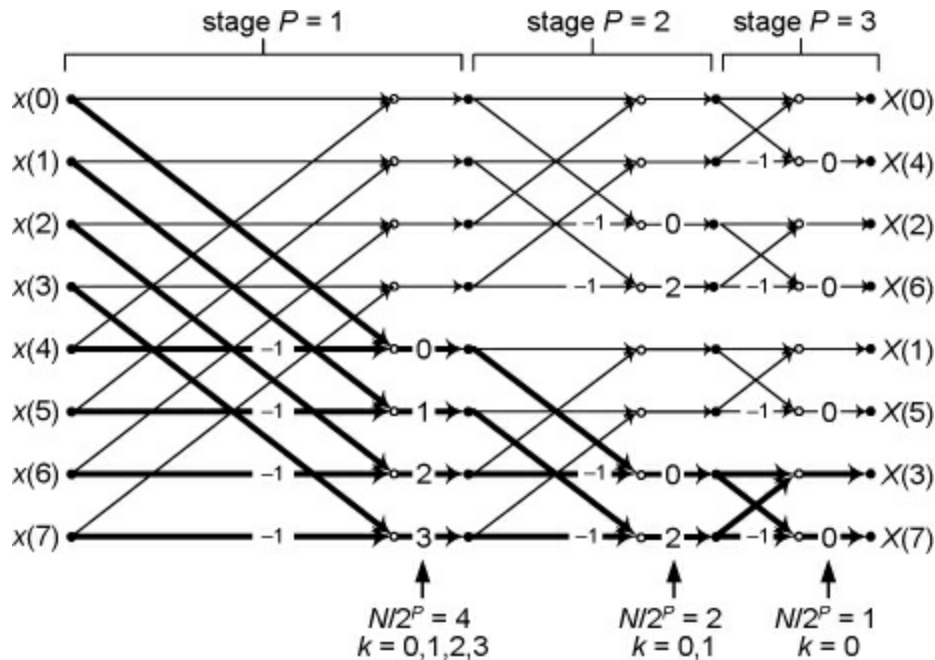
[Equations \(13-74\)](#) and [\(13-75'\)](#) have the advantage that FFT magnitude calculations, with their computationally costly square root operations, are not required as is necessary with other spectral peak location algorithms described above. However, the question naturally arises, "How do we determine the index m_k of the largest-magnitude FFT sample, $|X(m_k)|$, in [Figure 13-38\(a\)](#) without computing square roots to obtain FFT magnitudes?" The answer is that we can use the complex vector-magnitude approximations, requiring no square root computations, described in [Section 13.2](#).

Be aware that the above spectral peak location methods are only applicable when the majority of the signal's spectral energy lies within a single FFT bin width (f_s/N), and the FFT spectral samples are not substantially contaminated by *leakage* from another spectral component.

13.16 Computing FFT Twiddle Factors

Typical applications using an N -point radix-2 FFT accept $N x(n)$ input time samples and compute $N X(m)$ frequency-domain samples. However, there are non-standard FFT applications (for example, specialized harmonic analysis, or perhaps using an FFT to implement a bank of filters) where only a subset of the full $X(m)$ results is required. Consider [Figure 13-39](#) which shows the butterfly operations for an 8-point radix-2 decimation-in-frequency FFT. Notice that the FFT butterflies in [Figure 13-39](#) are the optimized butterflies introduced in [Figure 4-14](#). Assuming we are only interested in the $X(3)$ and $X(7)$ output samples, rather than compute the entire FFT we perform only the computations indicated by the bold lines in the figure.

Figure 13-39 Eight-point decimation-in-frequency FFT signal-flow diagram.



Reduced-computation FFTs are often called *pruned FFTs*[\[40-43\]](#). To implement pruned FFTs we need to know the twiddle phase angles associated with each necessary butterfly computation in the paths of any bold signal-flow line in [Figure 13-39](#). (As we did in [Chapter 4](#) for simplicity, the butterflies in [Figure 13-39](#) only show the twiddle phase-angle factors and not the entire complex-valued twiddle factors.) Here we show how to compute those individual twiddle phase angles.

13.16.1 Decimation-in-Frequency FFT Twiddle Factors

For the decimation-in-frequency (DIF) radix-2 FFT using the optimized butterflies:

- The N -point DIF FFT has $\log_2(N)$ stages, numbered $P = 1, 2, \dots, \log_2(N)$.
- Each stage comprises $N/2$ butterflies.
- Not counting the -1 twiddle factors, the P th stage has $N/2^P$ unique twiddle factors, numbered $k = 0, 1, 2, \dots, N/2^P - 1$ as indicated by the upward arrows at the bottom of [Figure 13-39](#).

Given those characteristics, the k th unique twiddle factor phase angle for the P th stage is computed using

$$(13-76)$$

$$k\text{th DIF twiddle factor angle} = k \cdot 2^P / 2$$

where $0 \leq k \leq N/2^P - 1$. For example, for the second stage ($P = 2$) of an $N = 8$ -point DIF FFT, the unique twiddle factor angles are

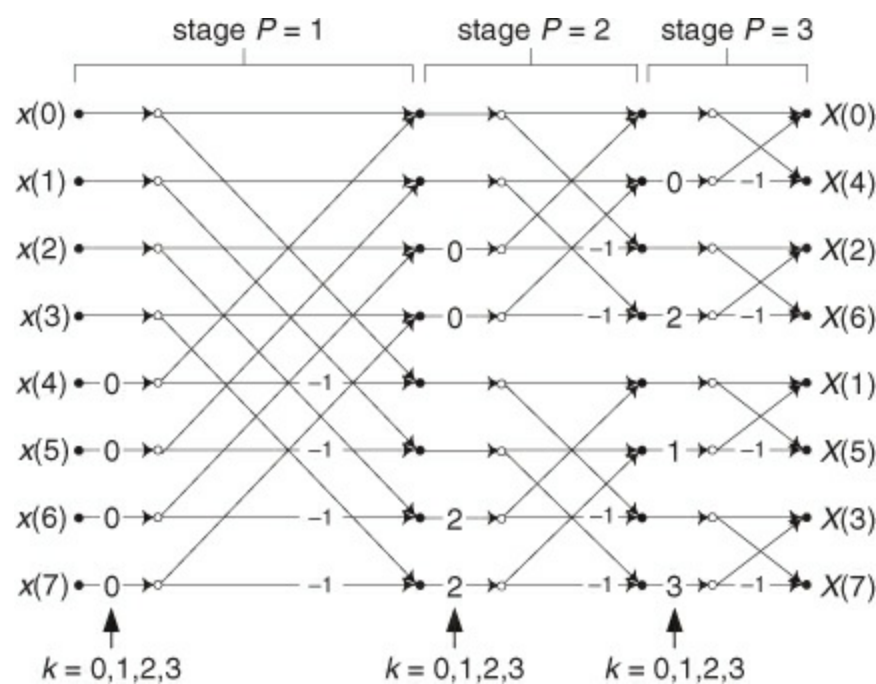
$$k = 0, \text{ angle} = 0 \cdot 2^2 / 2 = 0 \cdot 4 / 2 = 0$$

$$k = 1, \text{ angle} = 1 \cdot 2^2 / 2 = 1 \cdot 4 / 2 = 2.$$

13.16.2 Decimation-in-Time FFT Twiddle Factors

Here we present an interesting algorithm for computing the individual twiddle factor angles of a radix-2 decimation-in-time (DIT) FFT[44]. Consider [Figure 13-40](#) showing the butterfly signal flow of an 8-point DIT FFT.

Figure 13-40 Eight-point decimation-in-time FFT signal-flow diagram.



For the decimation-in-time (DIT) FFT using the optimized butterflies:

- The N -point DIT FFT has $\log_2(N)$ stages, numbered $P = 1, 2, \dots, \log_2(N)$.
- Each stage comprises $N/2$ butterflies.
- Not counting the -1 twiddle factors, the P th stage has $N/2$ twiddle factors, numbered $k = 0, 1, 2, \dots, N/2 - 1$ as indicated by the upward arrows at the bottom of [Figure 13-40](#).

Given those characteristics, the k th twiddle factor phase angle for the P th stage is computed using (13-76')

$$k\text{th DIT twiddle factor angle} = \llbracket k2^P/N \rrbracket_{\text{bit-rev}}$$

where $0 \leq k \leq N/2-1$. The $\llbracket q \rrbracket$ operation means the integer part of q . The $\llbracket z \rrbracket_{\text{bit-rev}}$ function represents the three-step operation of: convert decimal integer z to a binary number represented by $\log_2(N)-1$ binary bits, perform bit reversal on the binary number as discussed in [Section 4.5](#), and convert the bit-reversed number back to a decimal integer.

As an example of using [Eq. \(13-76'\)](#), for the second stage ($P = 2$) of an $N = 8$ -point DIT FFT, the $k = 3$ twiddle factor angle is

$$\text{3rd twiddle factor angle} = \llbracket 3 \cdot 2^2/8 \rrbracket_{\text{bit-rev}} = \llbracket 1.5 \rrbracket_{\text{bit-rev}} = \llbracket 1 \rrbracket_{\text{bit-rev}} = 2.$$

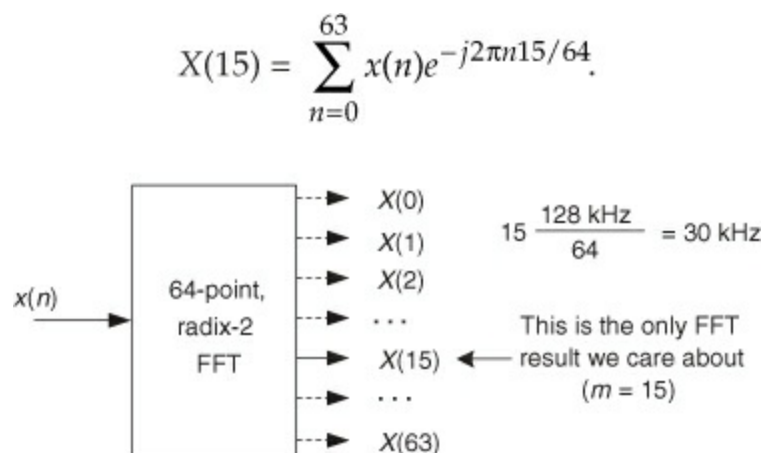
The above $\llbracket 1 \rrbracket_{\text{bit-rev}}$ operation is: Take the decimal number 1 and represent it with $\log_2(N)-1 = 2$ bits, i.e., as 01_2 . Next, reverse those bits to a binary 10_2 and convert that binary number to our desired decimal result of 2.

13.17 Single Tone Detection

In this section we present an IIR filter structure used to perform spectrum analysis in the detection and measurement of single sinusoidal tones. The standard method for spectral energy is the discrete Fourier transform (DFT), typically implemented using a fast Fourier transform (FFT) algorithm. However, there are applications that require spectrum analysis only over a subset of the N bin-center frequencies of an N -point DFT. A popular, as well as efficient, technique for computing sparse FFT results is the Goertzel algorithm, using an IIR filter implementation to compute a single complex DFT spectral bin value based upon N input time samples. The most common application of this process is to detect the presence of a single continuous-wave sinusoidal tone. With that in mind, let's look briefly at tone detection.

It's certainly possible to use the FFT to detect the presence of a single sinusoidal tone in a time-domain sequence $x(n)$. For example, if we wanted to detect a 30 kHz tone in a time-domain sequence whose sample rate was $f_s = 128$ kHz, we could start by performing a 64-point FFT as shown in [Figure 13-41](#). Then we would examine the magnitude of the $X(15)$ complex sample to see if it exceeds some predefined threshold.

Figure 13-41 DFT method, using an FFT algorithm, to detect a 30 kHz tone.



This FFT method is very inefficient. In our example, we'd be performing 192, $(64/2)(\log_2 64)$,

complex multiplies to obtain the 64-point complex $X(m)$ in order to compute the one $X(15)$ in which we're interested. We discarded 98 percent of our computation results! We could be more efficient and calculate our desired $X(15)$ using the single-point discrete Fourier transform (DFT) in [Eq. \(13-77\)](#), which requires $N = 64$ complex multiplies using

$$(13-77)$$

$$X(15) = \sum_{n=0}^{63} x(n)e^{-j2\pi n15/64}.$$

That would be an improvement but, happily, there's a better way. It's called the *Goertzel algorithm* (pronounced 'girt-zel).

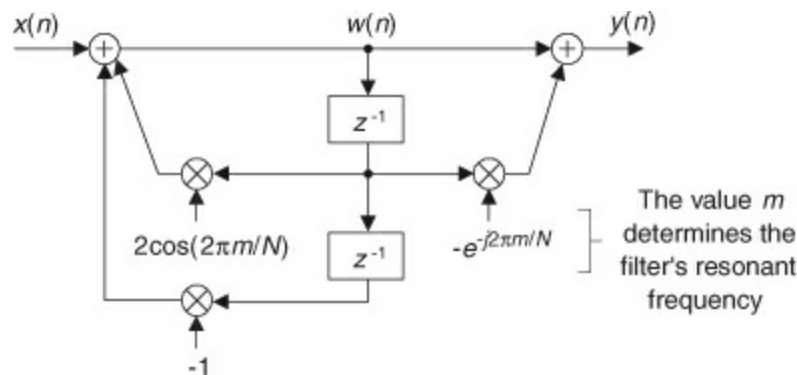
13.17.1 Goertzel Algorithm

The Goertzel algorithm is implemented in the form of a 2nd-order IIR filter, with two real feedback coefficients and a single complex feedforward coefficient, as shown in [Figure 13-42](#). (Although we don't use this process as a traditional filter, common terminology refers to the structure as a *filter*.) This filter computes a single-bin DFT output (the m th bin of an N -point DFT) defined by

$$(13-78)$$

$$X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N}.$$

Figure 13-42 IIR filter implementation of the Goertzel algorithm.



The filter's $y(n)$ output is equal to the DFT output frequency coefficient, $X(m)$, at the time index $n = N$, where the first time index value is $n = 0$. For emphasis, we remind the reader that the filter's $y(n)$ output is not equal to $X(m)$ at any time index when $n \neq N$. To be equivalent to the DFT, the frequency-domain index m must be an integer in the range $0 \leq m \leq N-1$. You're welcome to think of the Goertzel algorithm as a *single-bin DFT*. The derivation of this filter (this algorithm) structure is readily available in the literature[\[45–47\]](#).

The z -domain transfer function of the Goertzel filter is

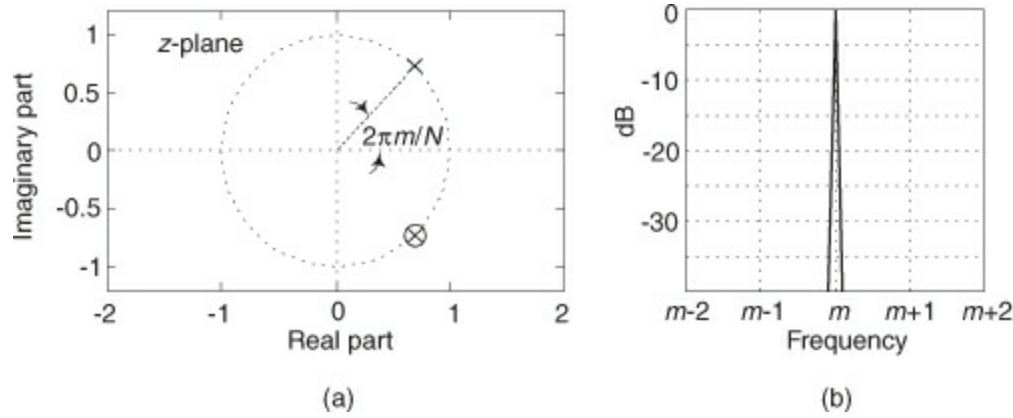
$$(13-79)$$

$$H_G(z) = \frac{Y(z)}{X(z)} = \frac{1 - e^{-j2\pi m/N} z^{-1}}{1 - 2\cos(2\pi m/N) z^{-1} + z^{-2}},$$

with a single z -domain zero located at $z = e^{-j2\pi m/N}$ and conjugate poles at $z = e^{\pm j2\pi m/N}$ as shown in [Figure 13-43\(a\)](#). The pole/zero pair at $z = e^{-j2\pi m/N}$ cancel each other. Having a filter pole on the unit circle is typically a risky thing to do for stability reasons, but not so with the Goertzel algorithm. Because it processes $N+1$ -length blocks of time samples (where N is usually in the hundreds), the

filter remains stable for such short time sequences because its internal data storage registers, $w(n-1)$ and $w(n-2)$, are reset to zero at the beginning of each new block of input data. The filter's frequency magnitude response, provided in [Figure 13-43\(b\)](#), shows resonance centered at a normalized frequency of $2\pi m/N$, corresponding to a cyclic frequency of mf_s/N Hz (where f_s is the signal sample rate).

Figure 13-43 Goertzel filter: (a) z -domain pole/zero locations; (b) frequency magnitude response.



The Goertzel algorithm is implemented with a complex resonator having an infinite-length unit impulse response, $h(n) = e^{j2\pi nm/N}$, and that's why its frequency magnitude response is so narrow. The time-domain difference equations for the Goertzel filter are

$$(13-80)$$

$$w(n) = 2\cos(2\pi m/N)w(n-1) - w(n-2) + x(n)$$

$$(13-81)$$

$$y(n) = w(n) - e^{-j2\pi m/N}w(n-1).$$

An advantage of the Goertzel filter in computing an N -point $X(m)$ DFT bin value is that [Eq. \(13-80\)](#) is implemented N times while [Eq. \(13-81\)](#), the feedforward path in [Figure 13-42](#), need only be computed once after the arrival of the N th input sample. Thus for real $x(n)$ inputs the filter requires $N+2$ real multiplies and $2N+1$ real adds to compute an N -point $X(m)$. However, when modeling the Goertzel filter, if the time index begins at $n = 0$, the filter must process $N+1$ time samples with $x(N) = 0$ to compute $X(m)$.

In typical applications, to minimize spectral leakage, we choose N so there's an integer number of cycles in our input sequence of the tone we're trying to detect. N can be any integer, and the larger N is, the better the frequency resolution and noise immunity. However, larger N means more computations.

It's worth noting that while the typical Goertzel algorithm description in the literature specifies the frequency resonance variable m to be an integer (making the Goertzel filter's output equivalent to an N -point DFT bin output), the m in [Figure 13-42](#) and [Eq. \(13-79\)](#) can in fact be any value between 0 and $N-1$, giving us full flexibility in specifying our filter's resonant frequency.

13.17.2 Goertzel Example

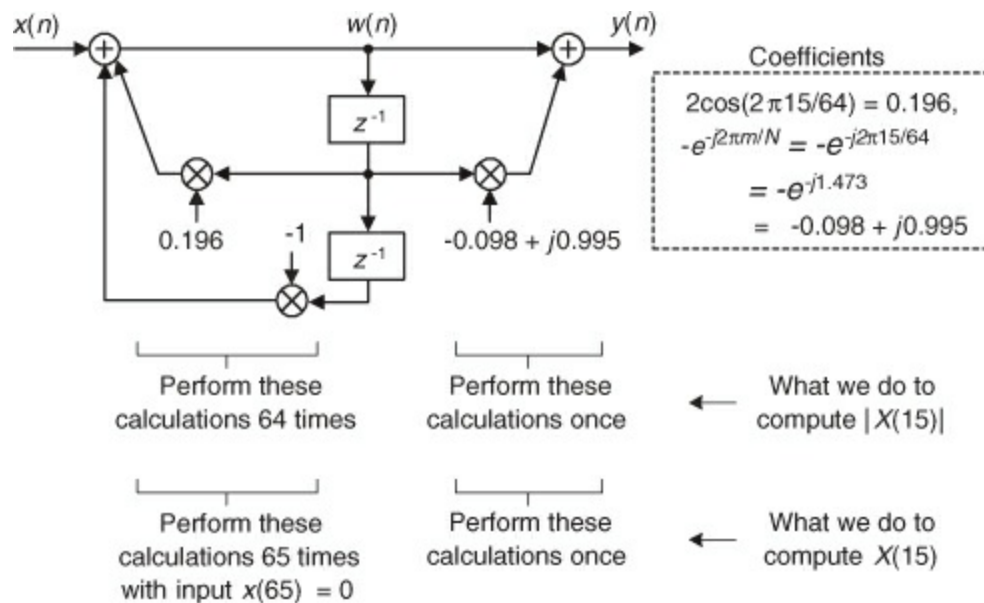
Let's use Goertzel to calculate the spectral magnitude of that $f_{\text{tone}} = 30$ kHz tone from the [Figure 13-41](#) example. When $f_s = 128$ kHz and $N = 64$, our resonant frequency integer m is

$$(13-82)$$

$$m = \frac{f_{\text{tone}}}{f_s / N} = \frac{(64)(30) \text{ kHz}}{128 \text{ kHz}} = 15.$$

The Goertzel filter and the necessary computations for our 30 kHz detection example are provided in [Figure 13-44](#).

Figure 13-44 Filter, coefficients, and computations to detect the 30 kHz tone.



It's useful to know that if we want to compute the power of $X(15)$, $|X(15)|^2$, the final feedforward complex calculations can be avoided by computing

$$(13-83)$$

$$\begin{aligned} |X(m)|^2 &= |y(N-1)|^2 \\ &= w(N-1)^2 + w(N-2)^2 - w(N-1)w(N-2)[2\cos(2\pi m/N)]. \end{aligned}$$

In our example, [Eq. \(13-83\)](#) becomes

$$(13-84)$$

$$|X(15)|^2 = |y(63)|^2 = w(63)^2 + w(62)^2 - w(63)w(62)[2\cos(2\pi 15/64)].$$

13.17.3 Goertzel Advantages over the FFT

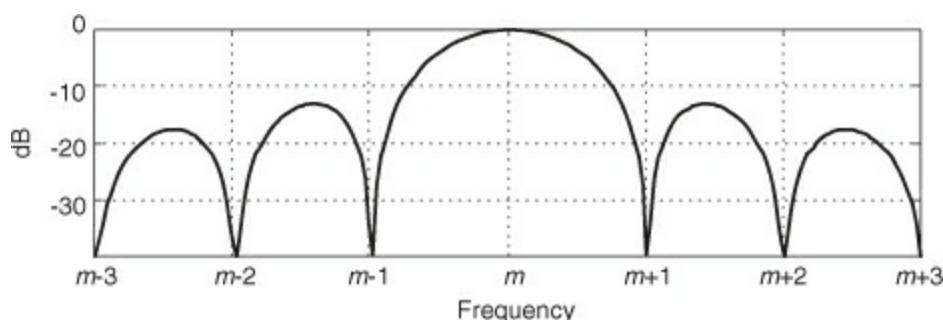
Here are some implementation advantages of the Goertzel algorithm over the standard radix-2 FFT for single tone detection:

- N does not need to be an integer power of two.
- The resonant frequency can be any value between zero and f_s Hz.
- The amount of filter coefficient (versus FFT twiddle factor) storage is reduced. If [Eq. \(13-83\)](#) is used, only one coefficient need be stored.
- No *storing a block of input data* is needed before processing can begin (as with the FFT). Processing can begin with the first input time sample.
- No data *bit reversal* is needed for Goertzel.
- If you implement the Goertzel algorithm M times to detect M different tones, Goertzel is more efficient (fewer multiplies) than the FFT when $M < \log_2 N$.
- Computational requirements to detect a single tone (assuming real-only $x(n)$ input) are given in

Table 13-4 Single-Bin DFT Computational Comparisons

Method	Real multiplies	Real additions
Single-bin DFT	$4N$	$2N$
FFT	$2N\log_2 N$	$N\log_2 N$
Goertzel	$N + 2$	$2N + 1$

As a final note, although the Goertzel algorithm is implemented with a complex resonating filter structure, it's not used as a typical filter where we retain each output sample. For the Goertzel algorithm we retain only every N th, or $(N+1)$ th, output sample. As such, the frequency magnitude response of the Goertzel algorithm when treated as a black-box process is equivalent to the $|\sin(x)/x|$ -like magnitude response of a single bin of an N -point DFT, a portion of which is shown in [Figure 13-45](#).

Figure 13-45 Goertzel algorithm frequency magnitude response.


13.18 The Sliding DFT

The above Goertzel algorithm computes a single complex DFT spectral bin value for every N input time samples. Here we describe a *sliding DFT* process whose spectral bin output rate is equal to the input data rate, on a sample-by-sample basis, with the advantage that it requires fewer computations than the Goertzel algorithm for real-time spectral analysis. In applications where a new DFT output spectrum is desired every sample, or every few samples, the sliding DFT is computationally simpler than the traditional radix-2 FFT.

13.18.1 The Sliding DFT Algorithm

The sliding DFT (SDFT) algorithm computes a single bin result of an N -point DFT on time samples within a sliding window. That is, for the m th bin of an N -point DFT, the SDFT computes

(13-85)

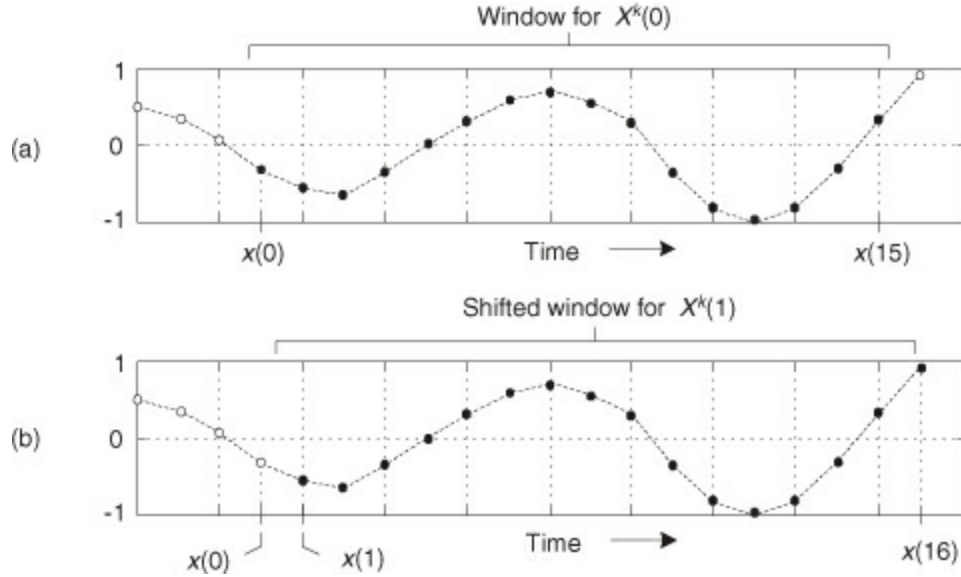
$$X^m(q) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N}.$$

Let's take care to understand the notation of $X^m(q)$. Typically, as in [Chapter 3](#), the index of a DFT result value was the frequency-domain index m . In [Eq. \(13-85\)](#) the index of the DFT result is a time-domain index $q = 0, 1, 2, 3, \dots$, such that our first m th-bin SDFT is $X^m(0)$, our second SDFT is $X^m(1)$, and so on.

An example SDFT analysis time window is shown in [Figure 13-46\(a\)](#) where $X^m(0)$ is computed for the $N = 16$ time samples $x(0)$ to $x(15)$. The time window is then advanced one sample, as in [Figure](#)

[13-46\(b\)](#), and the new $X^m(1)$ is calculated. The value of this process is that each new DFT result is efficiently computed directly from the result of the previous DFT. The incremental advance of the time window for each output computation leads to the name *sliding DFT* or *sliding-window DFT*.

Figure 13-46 Analysis window for two 16-point DFTs: (a) data samples in the first computation; (b) second computation samples.



We can develop the mathematical expression for the SDFT as follows: the standard N -point DFT equation, of the m th DFT bin, for the q th DFT of the time sequence $x(q), x(q+1), \dots, x(q+N-1)$ is

(13-86)

$$X^m(q) = \sum_{n=0}^{N-1} x(n+q)e^{-j2\pi nm/N}.$$

(Variable m is the frequency-domain index, where $m = 0, 1, 2, \dots, N-1$.) Likewise, the expression for the next DFT, the $(q+1)$ th DFT performed on time samples $x(q+1), x(q+2), \dots, x(q+N)$, is

(13-87)

$$X^m(q+1) = \sum_{n=0}^{N-1} x(n+q+1)e^{-j2\pi nm/N}.$$

Letting $p = n+1$ in [Eq. \(13-87\)](#), we can write

(13-88)

$$X^m(q+1) = \sum_{p=1}^N x(p+q)e^{-j2\pi(p-1)m/N}.$$

Shifting the limits of summation in [Eq. \(13-88\)](#), and including the appropriate terms (subtract the $p = 0$ term and add the $p = N$ term) to compensate for the shifted limits, we write

(13-89)

$$\begin{aligned} X^m(q+1) &= \left[\sum_{p=0}^{N-1} x(p+q)e^{-j2\pi(p-1)m/N} \right] - x(q)e^{-j2\pi(-1)m/N} + x(q+N)e^{-j2\pi(N-1)m/N}. \end{aligned}$$

Factoring the common exponential term ($e^{j2\pi m/N}$), we write

(13-90)

$$X^m(q+1) = e^{j2\pi m/N} \left[\sum_{p=0}^{N-1} x(p+q) e^{-j2\pi p m/N} \right] - x(q) + x(q+N) e^{-j2\pi N m/N}.$$

Recognizing the summation in the brackets being equal to the previous $X^m(q)$ in [Eq. \(13-86\)](#), and $e^{-j2\pi m} = 1$, we write the desired recursive expression for the sliding N -point DFT as

(13-91)

$$X^m(q+1) = e^{j2\pi m/N} [X^m(q) + x(q+N) - x(q)],$$

where $X^m(q+1)$ is the new single-bin DFT result and $X^m(q)$ is the previous single-bin DFT value. The superscript m reminds us that the $X^m(q)$ spectral samples are those associated with the m th DFT bin. Let's plug some numbers into [Eq. \(13-91\)](#) to reveal the nature of its time indexing. If $N = 20$, then 20 time samples ($x(0)$ to $x(19)$) are needed to compute the first result $X^m(0)$. The computation of $X^m(1)$ is then

$$(13-92)$$

$$X^m(1) = e^{j2\pi m/N} [X^m(0) + x(20) - x(0)].$$

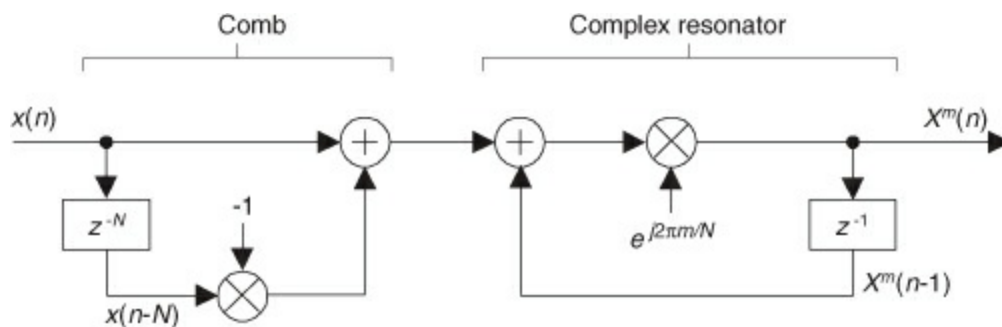
Due to our derivation method's time indexing, [Eq. \(13-92\)](#) appears compelled to look into the future for $x(20)$ to compute $X^m(1)$. With no loss in generality, we can modify [Eq. \(13-91\)](#)'s time indexing so that the $x(n)$ input samples and the $X^m(q)$ output samples use the same time index n . That modification yields our SDFT time-domain difference equation of

$$(13-93)$$

$$X^m(n) = e^{j2\pi m/N} [X^m(n-1) + x(n) - x(n-N)].$$

[Equation \(13-93\)](#) reveals the value of this process in computing real-time spectra. We compute $X^m(n)$ by subtracting the $x(n-N)$ sample and adding the current $x(n)$ sample to the previous $X^m(n-1)$, and phase shifting the result. Thus the SDFT requires only two real additions and one complex multiply per output sample. Not bad at all! [Equation \(13-93\)](#) leads to the single-bin SDFT filter implementation shown in [Figure 13-47](#).

Figure 13-47 Single-bin sliding DFT filter structure.



The single-bin SDFT algorithm is implemented as an IIR filter with a comb filter followed by a complex resonator. (If you need to compute all N DFT spectral components, N resonators with $m = 0$ to $N-1$ will be needed, all driven by a single comb filter.) The comb filter delay of N samples forces the SDFT filter's transient response to be N samples in length, so the output will not reach steady state until the $X^m(N-1)$ sample. The output will not be valid, or equivalent to [Eq. \(13-86\)](#)'s $X^m(q)$, until N input samples have been processed. The z -transform of [Eq. \(13-93\)](#) is

$$(13-94)$$

$$X^m(z) = e^{j2\pi m/N} [X^m(z)z^{-1} + X(z) - X(z)z^{-N}],$$

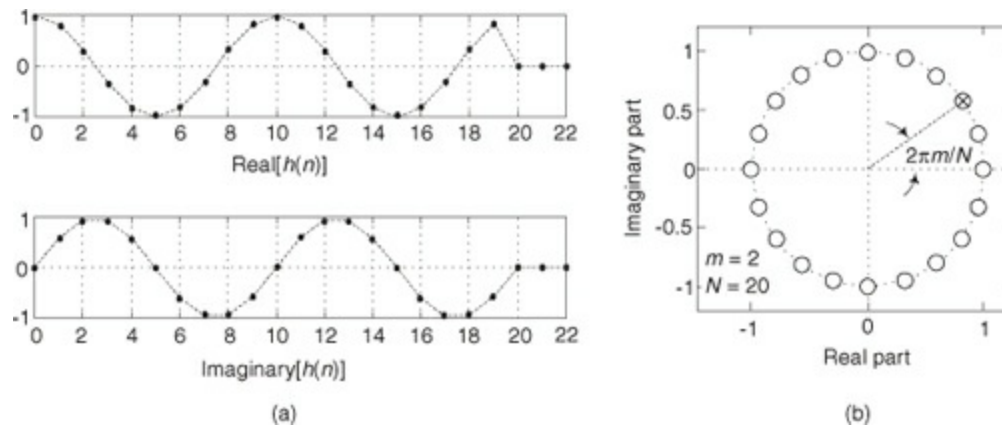
where factors of $X^m(z)$ and $X(z)$ are collected, yielding the z -domain transfer function for the m th bin of the SDFT filter as

$$(13-95)$$

$$H_{\text{SDFT}}(z) = \frac{X^m(z)}{X(z)} = \frac{(1 - z^{-N})e^{j2\pi m/N}}{1 - e^{j2\pi m/N}z^{-1}}.$$

This complex filter has N zeros equally spaced around the z -domain's unit circle, due to the N -delay comb filter, as well as a single pole canceling the zero at $z = e^{j2\pi m/N}$. The SDFT filter's complex unit impulse response $h(n)$ and pole/zero locations are shown in [Figure 13-48](#) for the example where $m = 2$ and $N = 20$.

Figure 13-48 Sliding DFT characteristics for $m = 2$ and $N = 20$: (a) complex impulse response; (b) pole/zero locations.



Because of the comb subfilter, the SDFT filter's complex sinusoidal unit impulse response is finite in length—truncated in time to N samples—and that property makes the frequency magnitude response of the SDFT filter identical to the $\sin(Nx)/\sin(x)$ response of a single DFT bin centered at a frequency of $2\pi m/N$.

One of the attributes of the SDFT is that once an $X^m(n)$ is obtained, the number of computations to compute $X^m(n+1)$ is fixed and independent of N . A computational workload comparison between the Goertzel and SDFT filters is provided later in this section. Unlike the radix-2 FFT, the SDFT's N can be any positive integer, giving us greater flexibility to *tune* the SDFT's center frequency by defining integer m such that $m = Nf_i/f_s$, when f_i is a frequency of interest in Hz and f_s is the signal sample rate in Hz. In addition, the SDFT requires no bit-reversal processing as does the FFT. Like the Goertzel algorithm, the SDFT is especially efficient for narrowband spectrum analysis.

For completeness, we mention that a radix-2 *sliding FFT* technique exists for computing all N bins of $X^m(q)$ in [Eq. \(13-85\)](#)[\[48,49\]](#). That technique is computationally attractive because it requires only N complex multiplies to update the N -point FFT for all N bins; however, it requires $3N$ memory locations ($2N$ for data and N for twiddle coefficients). Unlike the SDFT, the radix-2 sliding FFT scheme requires address bit-reversal processing and restricts N to be an integer power of two.

13.18.2 SDFT Stability

The SDFT filter is only marginally stable because its pole resides on the z -domain's unit circle. If filter coefficient numerical rounding error is not severe, the SDFT is bounded-input-bounded-output stable. Filter instability can be a problem, however, if numerical coefficient rounding causes the

filter's pole to move outside the unit circle. We can use a damping factor r to force the pole and zeros in [Figure 13-48\(b\)](#) to be at a radius of r just slightly inside the unit circle and guarantee stability using a transfer function of

$$(13-96)$$

$$H_{\text{SDFT,gs}}(z) = \frac{(1 - r^N z^{-N}) r e^{j2\pi m/N}}{1 - r e^{j2\pi m/N} z^{-1}},$$

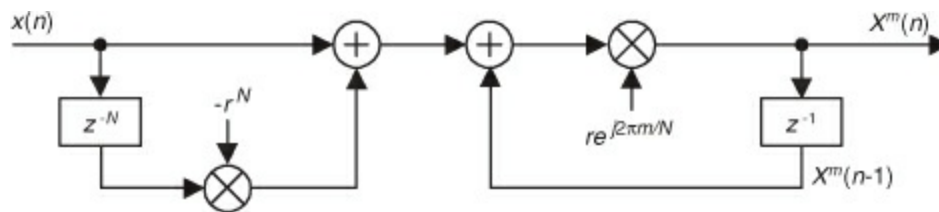
with the subscript “gs” meaning guaranteed-stable. ([Section 7.5.3](#) provides the mathematical details of moving a filter's poles and zeros inside the unit circle.) The stabilized feedforward and feedback coefficients become $-r^N$ and $r e^{j2\pi m/N}$, respectively. The difference equation for the stable SDFT filter becomes

$$(13-97)$$

$$X^m(n) = r e^{j2\pi m/N} [X^m(n-1) + x(n) - r^N x(n-N)]$$

with the stabilized-filter structure shown in [Figure 13-49](#). In this case, we perform five real multiplies and four real additions per output sample.

Figure 13-49 Guaranteed-stable sliding DFT filter structure.



Using a damping factor as in [Figure 13-49](#) guarantees stability, but the $X^m(q)$ output, defined by

$$(13-98)$$

$$X^m_{r<1}(q) = \sum_{n=0}^{N-1} x(n) r^{(N-n)} e^{-j2\pi n m/N},$$

is no longer exactly equal to the m th bin of an N -point DFT in [Eq. \(13-85\)](#). While the error is reduced by making r very close to (but less than) unity, a scheme does exist for eliminating that error completely once every N output samples at the expense of additional conditional logic operations[\[50\]](#). Determining if the damping factor r is necessary for a particular SDFT application requires careful empirical investigation. As is so often the case in the world of DSP, this means you have to test your SDFT implementation very thoroughly and carefully!

Another stabilization method worth consideration is decrementing the largest component (either real or imaginary) of the filter's $e^{j2\pi m/N}$ feedback coefficient by one least significant bit. This technique can be applied selectively to problematic output bins and is effective in combating instability due to rounding errors that result in finite-precision $e^{j2\pi m/N}$ coefficients having magnitudes greater than unity. Like the DFT, the SDFT's output is proportional to N , so in fixed-point binary implementations the designer must allocate sufficiently wide registers to hold the computed results.

13.18.3 SDFT Leakage Reduction

Being equivalent to the DFT, the SDFT also suffers from spectral leakage effects. As with the DFT, SDFT leakage can be reduced by the standard concept of windowing the $x(n)$ input time samples as discussed in [Section 3.9](#). However, windowing by time-domain multiplication would ruin the real-

time computational simplicity of the SDFT. Thanks to the convolution theorem properties of discrete systems, we can implement time-domain windowing by means of frequency-domain convolution, as discussed in [Section 13.3](#).

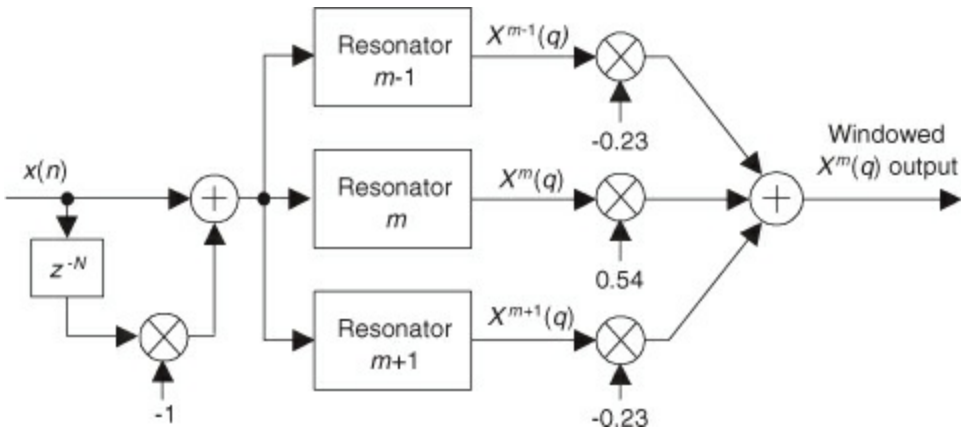
Spectral leakage reduction performed in the frequency domain is accomplished by convolving adjacent $X^m(q)$ values with the DFT of a window function. For example, the DFT of a Hamming window comprises only three nonzero values, -0.23 , 0.54 , and -0.23 . As such, we can compute a Hamming-windowed $X^m(q)$ with a three-point convolution using

(13-99)

$$\text{Hamming-windowed } X^m(q) = -0.23X^{m-1}(q) + 0.54X^m(q) - 0.23X^{m+1}(q).$$

[Figure 13-50](#) shows this process using three resonators, each tuned to adjacent DFT bins ($m-1$, m , and $m+1$). The comb filter stage need only be implemented once.

Figure 13-50 Three-resonator structure to compute a single Hamming-windowed $X^m(q)$.



[Table 13-5](#) provides a computational workload comparison of various spectrum analysis schemes in computing an initial $X^m(n)$ value and computing a subsequent $X^m(n+1)$ value.

Table 13-5 Single-Bin DFT Computation Comparison

Method	Compute initial $X^m(n)$		Compute $X^m(n+1)$	
	Real multiplies	Real adds	Real multiplies	Real adds
DFT	$4N$	$2N$	$4N$	$2N$
Goertzel algorithm	$N + 2$	$2N + 1$	$N + 2$	$2N + 1$
Sliding DFT (marginally stable)	$4N$	$4N$	4	4
Sliding DFT (guaranteed stable)	$5N$	$4N$	5	4
Three-term windowed sliding DFT (marginally stable)	$12N + 6$	$10N + 4$	18	14
Three-term windowed sliding DFT (guaranteed stable)	$13N + 6$	$10N + 4$	19	14

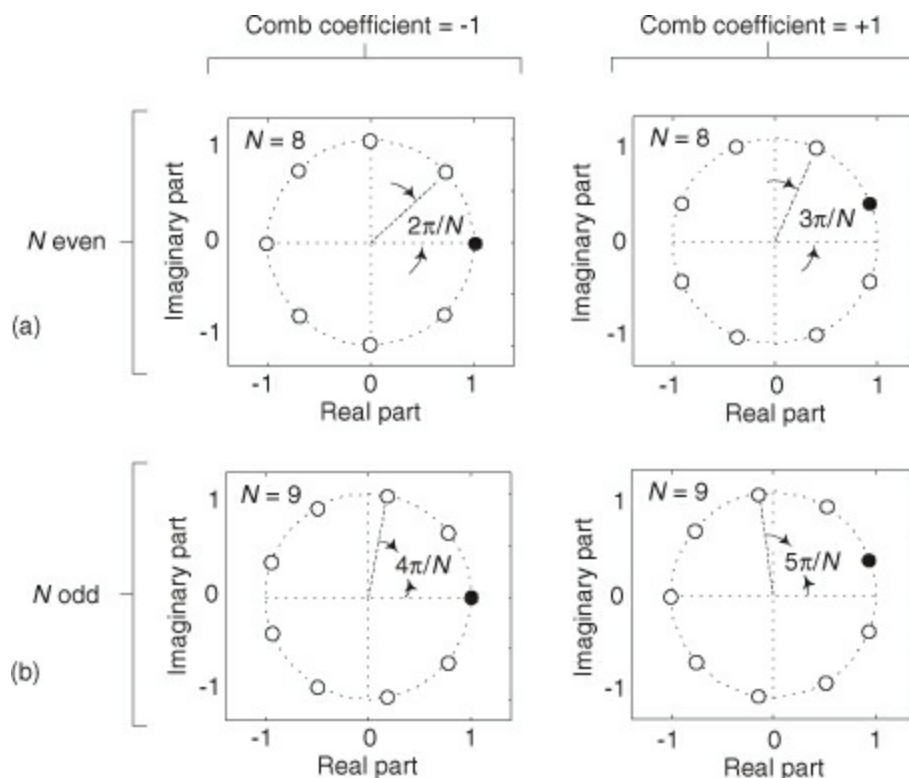
To compute the initial windowed $X^m(n)$ values in [Table 13-5](#), the three-term frequency-domain convolution need only be performed once, upon arrival of the N th time sample. However, the convolution needs to be performed for all subsequent computations

We remind the reader that [Section 13.3](#) discusses several implementation issues regarding Hanning windowing in the frequency domain, using binary shifts to eliminate the multiplications in [Eq. \(13-99\)](#), as well as the use of other window functions.

13.18.4 A Little-Known SDFT Property

The SDFT has a special property that's not widely known but is very important. If we change the SDFT's comb filter feedforward coefficient (in [Figure 13-47](#)) from -1 to $+1$, the comb's zeros will be rotated counterclockwise around the unit circle by an angle of π/N radians. This situation, for $N = 8$, is shown on the right side of [Figure 13-51\(a\)](#). The zeros are located at angles of $2\pi(m + 1/2)/N$ radians. The $m = 0$ zeros are shown as solid dots. [Figure 13-51\(b\)](#) shows the zeros locations for an $N = 9$ SDFT under the two conditions of the comb filter's feedforward coefficient being -1 and $+1$.

Figure 13-51 Four possible orientations of comb filter zeros on the unit circle.



This alternate situation is useful: we can now expand our set of spectrum analysis center frequencies to more than just N angular frequency points around the unit circle. The analysis frequencies can be either $2\pi m/N$ or $2\pi(m+1/2)/N$, where integer m is in the range $0 \leq m \leq N-1$. Thus we can build an SDFT analyzer that resonates at any one of $2N$ frequencies between 0 and f_s Hz. Of course, if the comb filter's feedforward coefficient is set to $+1$, the resonator's feedforward coefficient must be $e^{j2\pi(m+1/2)/N}$ to achieve pole/zero cancellation.

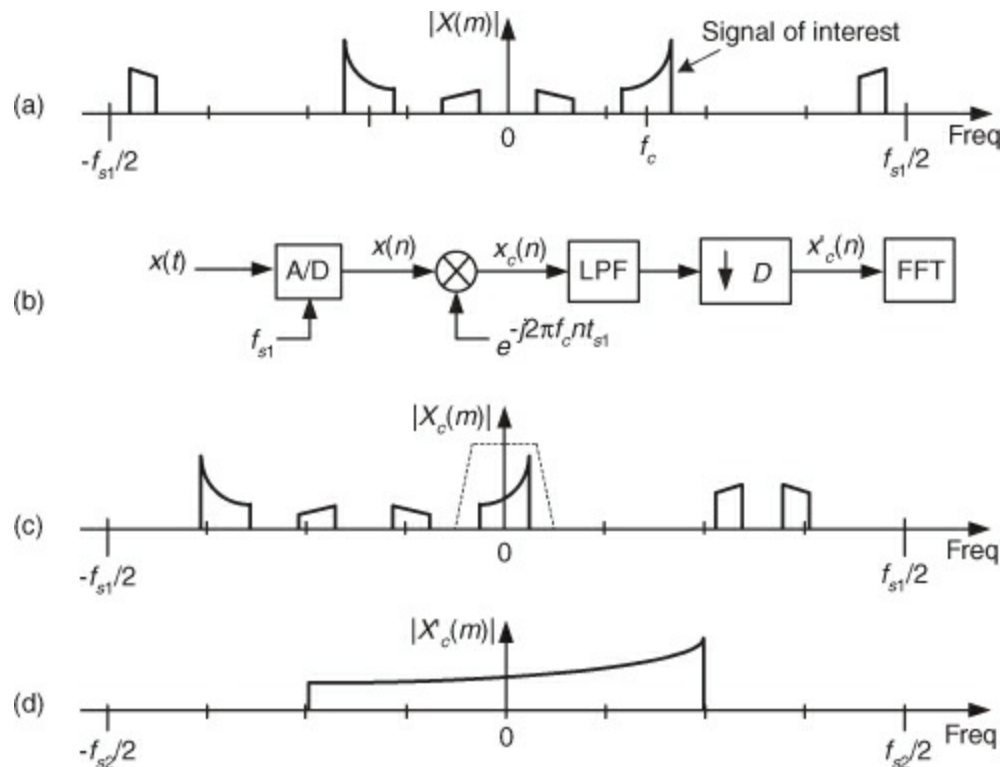
13.19 The Zoom FFT

The Zoom FFT is a spectrum analysis method that blends complex down-conversion, lowpass filtering, and sample rate change by way of decimation. The Zoom FFT scheme (also called the *zoom transform* or *spectral vernier*) is used when fine-grained spectral resolution is needed within a small portion of a signal's overall frequency bandwidth range. In some spectrum analysis situations, this technique can be more efficient than the traditional FFT. The Zoom FFT can also be useful if we're constrained, for some reason, to use software that performs N -point FFTs for spectrum analysis of signal sequences whose lengths are greater than N .

Think of the spectral analysis situation where we require *fine* frequency resolution, closely spaced FFT bins, over the frequency range occupied by the signal of interest shown in [Figure 13-52\(a\)](#). (The other signals are of no interest to us.) We could collect many time samples and perform a large-size

radix-2 FFT to satisfy our fine spectral resolution requirement. This solution is inefficient because we'd be discarding most of our FFT results. The Zoom FFT can help us improve our computational efficiency through

Figure 13-52 Zoom FFT spectra: (a) input spectrum; (b) processing scheme; (c) down-converted spectrum; (d) filtered and decimated spectrum.

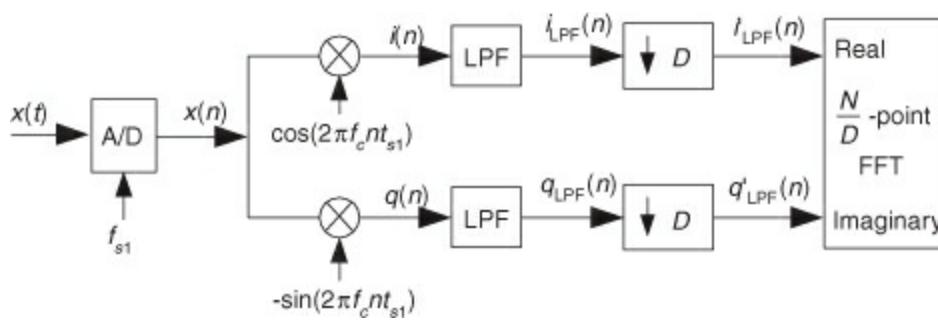


- frequency translation by means of complex down-conversion,
- lowpass filtering,
- decimation, and finally
- performing a smaller-size FFT.

The process begins with the continuous $x(t)$ signal being digitized at a sample rate of f_{s1} by an analog-to-digital (A/D) converter, yielding the N -point $x(n)$ time sequence whose spectral magnitude is $|X(m)|$ in [Figure 13-52\(a\)](#). The Zoom FFT technique requires narrowband filtering and decimation in order to reduce the number of time samples prior to the final FFT, as shown in [Figure 13-52\(b\)](#). The down-converted signal's spectrum, centered at zero Hz, is the $|X_c(m)|$ shown in [Figure 13-52\(c\)](#). (The lowpass filter's frequency response is the dashed curve.) After lowpass filtering $x_c(n)$, the filter's output is decimated by an integer factor D , yielding a time sequence $x'_c(n)$ whose sample rate is $f_{s2} = f_{s1}/D$ prior to the FFT operation. The key here is that the length of $x'_c(n)$ is N/D , allowing a reduced-size FFT. (N/D must be an integer power of two to enable the use of radix-2 FFTs.) We perform the FFT only over the decimated signal's bandwidth. It's of interest to note that, because its input is complex, the N/D -point FFT has a non-redundant frequency analysis range from $-f_{s2}/2$ to $+f_{s2}/2$ (unlike the case of real inputs, where the positive- and negative-frequency ranges are redundant).

The implementation of the Zoom FFT is given in [Figure 13-53](#), where all discrete sequences are real-valued.

Figure 13-53 Zoom FFT processing details.



Relating the discrete sequences in [Figure 13-52\(b\)](#) and [Figure 13-53](#), the complex time sequence $x_c(n)$ is represented mathematically as

$$(13-100)$$

$$x_c(n) = i(n) + jq(n),$$

while the complex decimated sequence $x'_c(n)$ is

$$(13-101)$$

$$x'_c(n) = i'_{LPF}(n) + jq'_{LPF}(n).$$

The complex mixing sequence $e^{-j2\pi f_c n t_{s1}}$, where $t_{s1} = 1/f_{s1}$, can be represented in the two forms of

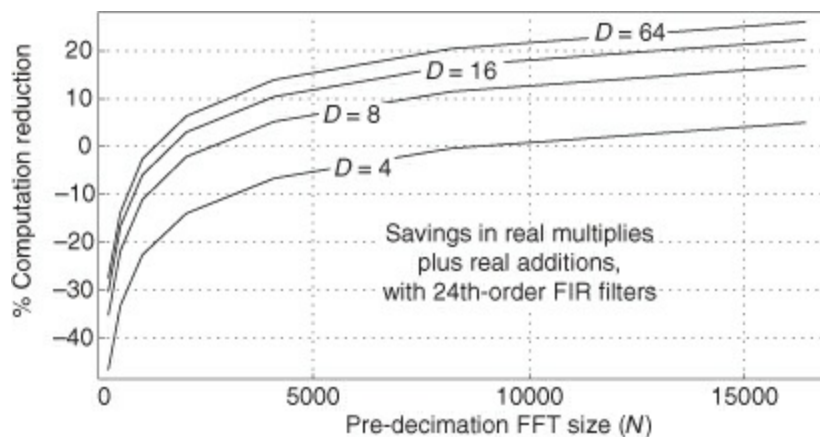
$$(13-102)$$

$$e^{-j2\pi f_c n t_{s1}} = \cos(2\pi f_c n t_{s1}) - j\sin(2\pi f_c n t_{s1}) = \cos(2\pi n f_c / f_{s1}) - j\sin(2\pi n f_c / f_{s1}).$$

Relative to FFT computations, we see that an N/D -point Zoom FFT yields a reduction in computations compared to a standard N -point FFT for spectrum analysis of a narrowband portion of some $X(m)$ spectrum—and the computational savings improve as the decimation factor D increases. Ah, but here's the rub. As D increases, the lowpass filters must become narrower, which increases their computational workload, and this is the trade-off we face. What we must ask ourselves is “Does the Zoom FFT's reduced FFT size compensate for the additional quadrature mixing and dual filtering computational workload?” (It certainly would if a large-size FFT is impossible with your available FFT hardware or software.)

To gain a rough appreciation for the computational savings gained by using an N/D -point Zoom FFT, compared to a standard N -point FFT, let's look at [Figure 13-54](#). That figure shows the percent computational savings of a Zoom FFT versus a standard N -point FFT for various decimation factors D .

Figure 13-54 Zoom FFT computation reduction.



The curves were computed using the following definition for percent computation reduction

% Computation reduction

$$= 100 \cdot \left(1 - \frac{N / D\text{-pt Zoom FFT computations}}{N\text{-pt FFT computations}} \right)$$

under the assumptions that the time sequences applied to the FFTs were windowed, and the Zoom FFT's lowpass filters were 24th-order (25 multiplications per output sample) tapped-delay line FIR filters using folded FIR structures. In [Eq. \(13-103\)](#) a single real multiply and a single real addition are both considered as a single computation.

The range where [Figure 13-54](#)'s curves have negative values means that the Zoom FFT is less efficient (more computations) than a standard N -point FFT. As it turns out, the curves in [Figure 13-54](#) quickly move downward in efficiency as the order of the lowpass filters increases. So it's in our best interest to make the lowpass filters as computationally efficient as possible. Some ways to do this are:

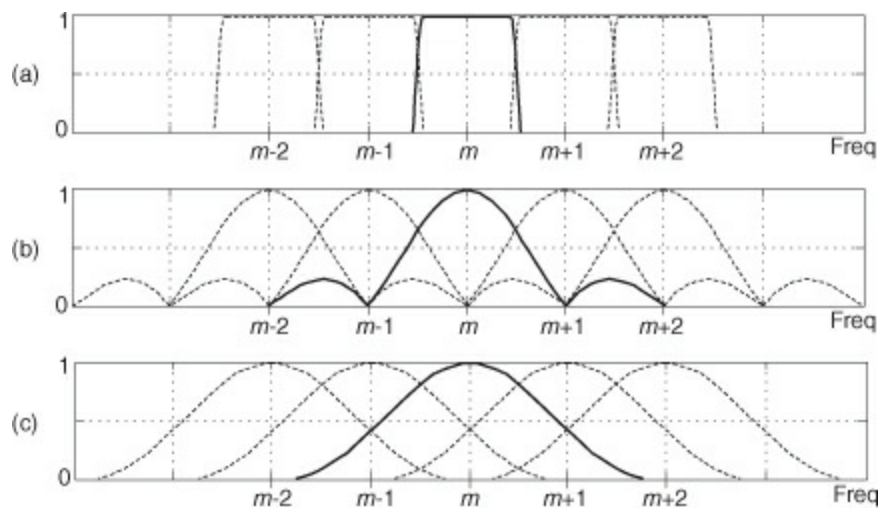
- Partition the lowpass filtering/decimation process into multiple stages (multistage decimation) as discussed in [Chapter 10](#).
- Incorporate cascaded integrator-comb (CIC) filters into the lowpass filtering if the spectrum of interest is very narrowband relative to the f_{s1} .
- Use interpolated FIR filters as discussed in [Chapter 7](#).
- Use polyphase filters as discussed in [Chapter 10](#).
- Restrict the decimation factor D to be an integer power of two such that efficient half-band filters can be used.
- Use IIR filters, if spectral phase distortion can be tolerated.

13.20 A Practical Spectrum Analyzer

Here's a clever trick for implementing a practical spectrum analyzer by modifying the time-domain data before applying a radix-2 FFT algorithm.

Let's say we need to build a spectrum analyzer to display, in some manner, the spectral magnitude of a time-domain sequence. We'd like our spectrum analyzer, a bank of bandpass filters, to have a frequency magnitude response something like that shown in [Figure 13-55\(a\)](#). For spectrum analysis, the radix-2 FFT algorithm comes to mind first, as it should. However, the frequency response of individual FFT bins is that shown in [Figure 13-55\(b\)](#), with their non-flat passbands, unpleasantly high sidelobes due to spectral leakage, and overlapped main lobes. We can reduce the leakage sidelobe levels by windowing the time-domain sequence, but that leads to the increased main lobe overlap shown in [Figure 13-55\(c\)](#) and degraded frequency resolution, and we still have considerable droop in the passband response.

Figure 13-55 Spectrum analyzer: (a) desired frequency response; (b) frequency response of standard FFT bins; (c) windowed-data FFT frequency response.



Here's how we can solve our problem. Consider an $x(n)$ sequence of time samples of length M whose M -point DFT is

$$(13-104)$$

$$X(k) = \sum_{n=0}^{M-1} x(n)e^{-j2\pi nk/M}.$$

Next, consider partitioning $x(n)$ into P subsequences, each of length N . Thus $PN = M$. If we add, element for element, the P subsequences, we'll obtain a new $y(n)$ sequence of length N whose N -point DFT is

$$(13-105)$$

$$Y(m) = \sum_{n=0}^{N-1} y(n)e^{-j2\pi nm/N}.$$

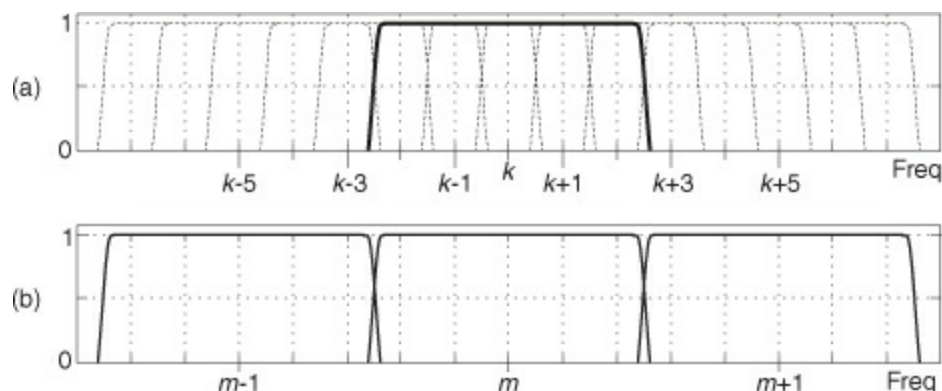
The good news is that

$$(13-106)$$

$$|Y(m)| = |X(Pm)|.$$

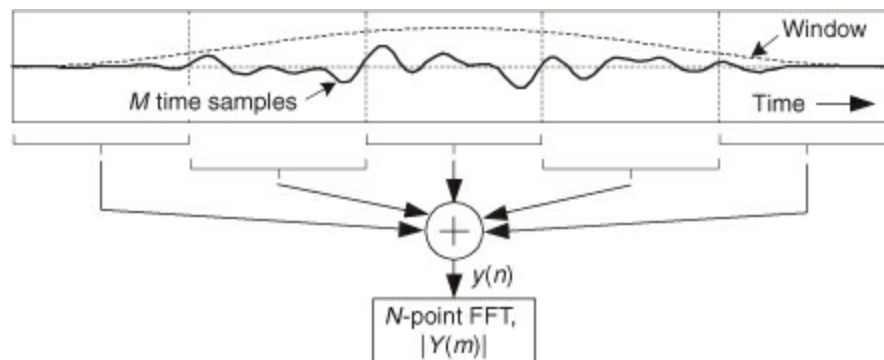
That is, the DFT magnitudes of sequence $y(n)$ are equal to a subset of the longer DFT magnitudes of $x(n)$. $Y(m)$ is equal to a decimated-by- P version of $X(k)$. The relationship between $|Y(m)|$ and $|X(Pm)|$ doesn't seem too important, but here's how we'll take advantage of that equality. We'll create an M -point window sequence whose single-bin frequency response, of an M -point FFT, is the bold curve in [Figure 13-56\(a\)](#). Instead of computing all M FFT outputs, we'll only compute every P th output of the M -point FFT, implementing [Eq. \(13-105\)](#), giving us the decimated FFT bins shown in [Figure 13-56\(b\)](#). In that figure $P = 5$.

Figure 13-56 FFT spectrum analyzer frequency responses.



That decimation of the frequency-domain $|X(k)|$ spectrum is accomplished in the time domain by a time-aliasing operation as shown in [Figure 13-57](#), where again, for example, $P = 5$. We partition the M -sample windowed- $x(n)$ time sequence into $P = 5$ subsequences and sum the subsequences element for element to obtain the time-aliased N -sample $y(n)$ sequence. Next, the $|Y(m)|$ spectral magnitudes are computed using the radix-2 FFT.

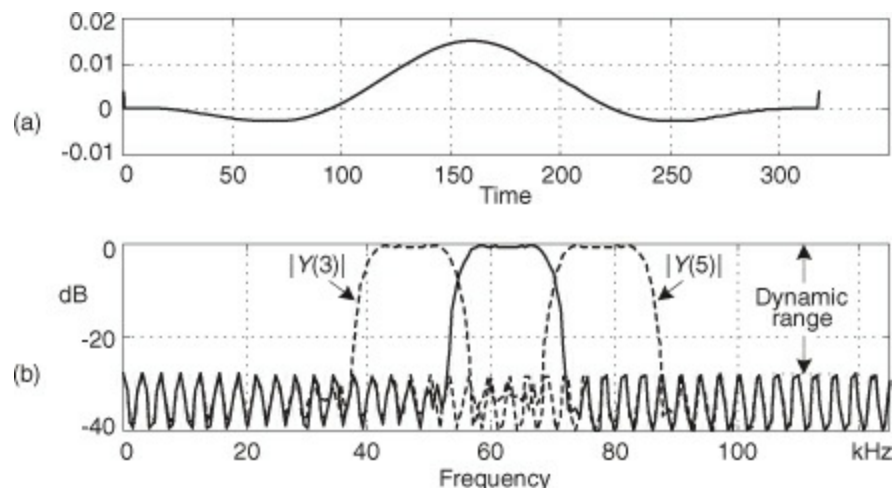
Figure 13-57 FFT spectrum analyzer process.



This process, sweet in its simplicity, is called the *weighted overlap-add* structure[\[51,52\]](#) and is alternatively referred to as the *window-presum FFT*[\[53\]](#). The most difficult part of building this analyzer is designing the M -point window sequence used to window the original $x(n)$ sequence. We do that by specifying the window's frequency-domain characteristics, just as if it were a digital filter frequency response, and using our favorite filter design software to compute the filter's time-domain impulse response. That impulse response is the window sequence. With the signal sample rate being f_s , the window's passband width will be just less than f_s/N . This makes the filter's one-sided passband width roughly $f_s/2N$.

[Figure 13-58](#) illustrates an example FFT analyzer with $f_s = 1$ MHz, $N = 64$, with $P = 5$ making $M = 320$. The FFT bin spacing is 15.63 kHz, so the window design was set for a passband width of 10 kHz (thus the filter's one-sided bandwidth was specified as 5 kHz in a Parks-McClellan design routine). [Figure 13-58\(a\)](#) is the 320-point window sequence, while [Figure 13-58\(b\)](#) shows the FFT analyzer's response for the $m = 3, 4$, and 5 bins, with the $|Y(4)|$ response being the solid curve.

Figure 13-58 FFT analyzer example: (a) window sequence; (b) analyzer response for 64-point FFT bins $|Y(3)|$, $|Y(4)|$, and $|Y(5)|$.



The width of the spectrum analyzer's passbands is primarily controlled by the window's passband width. The center frequencies of the analyzer's individual passbands are defined by f_s/N . What this means is that the amount of overlap in the analyzer's passbands depends on both the window's

passband width, f_s , and N . The dynamic range of the analyzer can be increased by increasing P , which increases M and lengthens the $x(n)$ sequence. As M is increased, the longer window sequence will yield analyzer passbands having a more rectangular shape, lower sidelobes, and reduced passband ripple.

Again, to implement this radix-2 FFT spectrum analyzer, the length of the time-domain sequence (M) must be an integer multiple (P) of an integer power of two (N).

13.21 An Efficient Arctangent Approximation

Fast and accurate methods for computing the arctangent of a complex number $x = I + jQ$ have been the subject of extensive study because estimating the angle θ of a complex value has so many applications in the field of signal processing. The angle of x is defined as $\theta = \tan^{-1}(Q/I)$.

Practitioners interested in computing high-speed (minimum computations) arctangents typically use look-up tables where the value Q/I specifies a memory address in read-only memory (ROM) containing an approximation of angle θ . For high accuracy, though, this method may require very large ROM tables. Those folk interested in enhanced accuracy implement compute-intensive high-order algebraic polynomials, where Chebyshev polynomials seem to be more popular than Taylor series, to approximate angle θ . But this polynomial method requires many computations. Unfortunately, because it is such a nonlinear function, the arctangent is resistant to accurate reasonable-length polynomial approximations. There is a processing method called “CORDIC” (an acronym for **CO**ordinate **R**otation **D**igital **C**omputer) that can compute accurate arctangents using only binary shifts and additions, but this technique can require long processing times. So, sadly, we end up choosing the *least undesirable* method for computing arctangents.

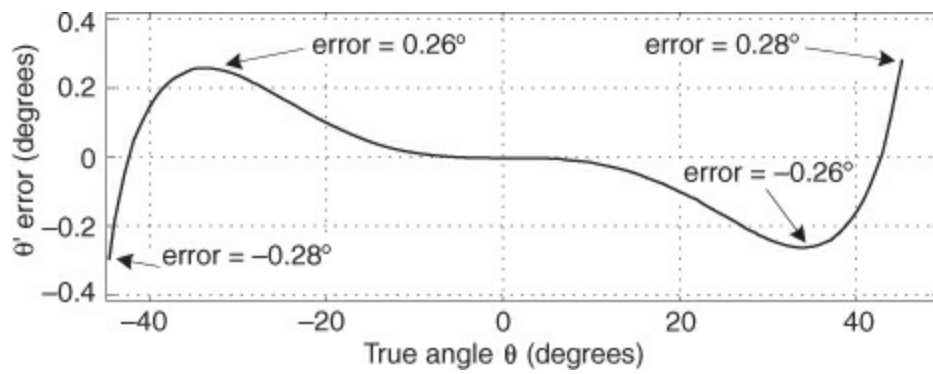
If you want to become famous in the field of signal processing, all you have to do is produce a very accurate arctangent algorithm that requires very few computations. (After solving that problem, you can then apply your skills to developing a perpetual-motion machine.)

Here’s another contender in the arctangent approximation race that uses neither look-up tables nor high-order polynomials. We can estimate the angle θ , in radians, of $x = I + jQ$ using the following approximation:

$$(13-107) \quad \tan^{-1}(Q/I) \approx \theta' = \frac{Q/I}{1 + 0.28125(Q/I)^2} \text{ radians,}$$

where $-1 \leq Q/I \leq 1$. That is, θ is in the range -45 to $+45$ degrees ($-\pi/4 \leq \theta \leq +\pi/4$ radians). [Equation \(13-107\)](#) has surprisingly good performance, particularly for a 90-degree ($\pi/2$ radians) angle range. [Figure 13-59](#) shows the maximum error is 0.28 degrees using [Eq. \(13-107\)](#) when the true angle θ is within the angular range of -45 to $+45$ degrees

Figure 13-59 Estimated angle θ' error in degrees.



A nice feature of this θ' computation is that it can be written as

$$(13-108)$$

$$\theta' = \frac{IQ}{I^2 + 0.28125Q^2},$$

eliminating [Eq. \(13-107\)](#)'s Q/I division operation, at the expense of two additional multiplies. Another attribute of [Eq. \(13-108\)](#) is that a single multiply can be eliminated with binary right shifts. The product $0.28125Q^2$ is equal to $(1/4+1/32)Q^2$, so we can implement the product by adding Q^2 shifted right by two bits to Q^2 shifted right by five bits. This arctangent scheme may be useful in a digital receiver application where I^2 and Q^2 have been previously computed in conjunction with an AM (amplitude modulation) demodulation process or envelope detection associated with automatic gain control (AGC).

We can extend the angle range over which our approximation operates. If we break up a circle into eight 45-degree octants, with the first octant being 0 to 45 degrees, we can compute the arctangent of a complex number residing in any octant. We do this by using the rotational symmetry properties of the arctangent:

$$(13-109)$$

$$\tan^{-1}(-Q/I) = -\tan^{-1}(Q/I)$$

$$(13-110)$$

$$\tan^{-1}(Q/I) = \pi/2 - \tan^{-1}(I/Q).$$

Those properties allow us to create [Table 13-6](#).

Table 13-6 Octant Location versus Arctangent Expressions

Octant	Arctan approximation
1st, or 8th	$\theta' = \frac{IQ}{I^2 + 0.28125Q^2}$
2nd, or 3rd	$\theta' = \pi/2 - \frac{IQ}{Q^2 + 0.28125I^2}$
4th, or 5th	$\theta' = \text{sign}(Q) \cdot \pi + \frac{IQ}{I^2 + 0.28125Q^2}$
6th, or 7th	$\theta' = -\pi/2 - \frac{IQ}{Q^2 + 0.28125I^2}$

So we have to check the signs of Q and I , and see if $|Q| > |I|$, to determine the octant location, and then use the appropriate approximation in [Table 13-6](#). [Section 13.38](#) gives a method for determining the

octant of the original θ . The maximum angle approximation error is 0.28 degrees for all octants.

13.22 Frequency Demodulation Algorithms

In [Section 9.2](#) we discussed the notion of measuring the instantaneous frequency of a complex sinusoidal signal by computing the derivative of the signal's instantaneous $\theta(n)$ phase as shown in [Figure 13-60](#). This is the traditional discrete-signal FM demodulation method, and it works fine. The demodulator's instantaneous output frequency is

$$(13-111)$$

$$f(n) = \frac{f_s [\Delta\theta_{\text{rad}}(n)]}{2\pi} \text{ Hz},$$

Figure 13-60 Frequency demodulator using an arctangent function.



where f_s is the sample rate in Hz.

Computing instantaneous phase $\theta(n)$ requires an arctangent operation, which is difficult to implement accurately without considerable computational resources. Here's a scheme for computing $\Delta\theta(n)$ for use in [Eq. \(13-111\)](#) without the intermediate $\theta(n)$ phase computation (and its pesky arctangent) [\[54,55\]](#). We derive the $\Delta\theta(n)$ computation algorithm as follows, initially using continuous-time variables based on the following definitions:

$$(13-112)$$

$$\begin{aligned} i(t) &= \text{in-phase signal,} \\ q(t) &= \text{quadrature phase signal,} \\ \theta(t) &= \tan^{-1}[q(t)/i(t)] = \text{instantaneous phase,} \\ \Delta\theta(t) &= \text{time derivative of } \theta(t). \end{aligned}$$

The following algorithm is based on the assumption that the spectrum of the $i(t) + jq(t)$ signal is centered at zero Hz. First, we let $r(t) = q(t)/i(t)$ be the signal for which we're trying to compute the derivative of its arctangent. The time derivative of $\tan^{-1}[r(t)]$, a calculus identity, is

$$(13-113)$$

$$\Delta\theta(t) = \frac{d\{\tan^{-1}[r(t)]\}}{dt} = \frac{1}{1+r^2(t)} \frac{d[r(t)]}{dt}.$$

Because $d[r(t)]/dt = d[q(t)/i(t)]/dt$, we use the calculus identity for the derivative of a ratio to write

$$(13-114)$$

$$\frac{d[r(t)]}{dt} = \frac{d[q(t)/i(t)]}{dt} = \frac{i(t) \frac{d[q(t)]}{dt} - q(t) \frac{d[i(t)]}{dt}}{i^2(t)}.$$

Plugging [Eq. \(13-114\)](#)'s result into [Eq. \(13-113\)](#), we have

$$(13-115)$$

$$\Delta\theta(t) = \frac{1}{1+r^2(t)} \frac{i(t) \frac{d[q(t)]}{dt} - q(t) \frac{d[i(t)]}{dt}}{i^2(t)}.$$

Replacing $r(t)$ in [Eq. \(13-115\)](#) with $q(t)/i(t)$ yields

(13-116)

$$\Delta\theta(t) = \frac{1}{1 + [q(t)/i(t)]^2} \frac{i(t) \frac{d[q(t)]}{dt} - q(t) \frac{d[i(t)]}{dt}}{i^2(t)}.$$

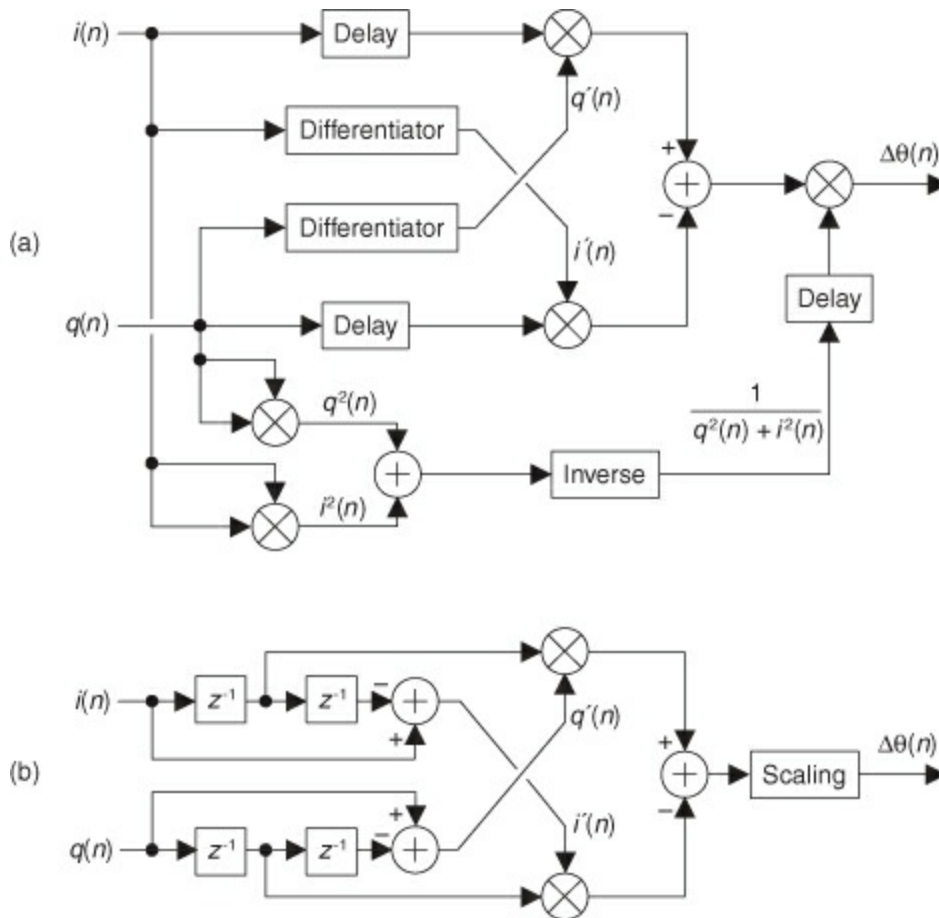
We're getting there. Next we multiply the numerator and denominator of the first ratio in [Eq. \(13-116\)](#) by $i^2(t)$ and replace t with our discrete time variable index n to arrive at our final result of

(13-117)

$$\Delta\theta(n) = \frac{i(n) \frac{d[q(n)]}{dn} - q(n) \frac{d[i(n)]}{dn}}{i^2(n) + q^2(n)}.$$

The implementation of this algorithm, where the derivatives of $i(n)$ and $q(n)$ are $i'(n)$ and $q'(n)$ respectively, is shown in [Figure 13-61\(a\)](#). The $\Delta\phi(n)$ output sequence is used in [Eq. \(13-111\)](#) to compute instantaneous frequency.

Figure 13-61 Frequency demodulator without arctangent: (a) standard process; (b) simplified process.



The Differentiators are tapped-delay line FIR differentiating filters with an odd number of taps. The z^{-D} delay elements in [Figure 13-61\(a\)](#) are used to time-align the input $i(n)$ and $q(n)$ sequences with the outputs of the differentiators. The delay is $D = (K-1)/2$ samples when a K -tap differentiator is used. In practice, those z^{-D} delays can be obtained by tapping off the center tap of the differentiating filter as shown in [Figure 13-61\(b\)](#), where the differentiator is an FIR filter having 1,0,-1 as coefficients, and $D = 1$ in this case[55]. Such a differentiator is the simple “central-difference differentiator” we discussed in [Chapter 7](#), and its optimum performance occurs when the input signal is low frequency relative to the demodulator’s input f_s sample rate. Reference [55] reports acceptable results using the

differentiator in [Figure 13-61\(b\)](#), but that's only true if the complex input signal has a bandwidth no greater than $f_s/10$.

If the $i(n)+jq(n)$ signal is purely FM and *hard limited* such that $i^2(n)+q^2(n) = \text{Constant}$, the denominator computations in [Eq. \(13-117\)](#) need not be performed. In this case, using the 1,0,-1 coefficient differentiators, the FM demodulator is simplified to that shown in [Figure 13-61\(b\)](#), where the Scaling operation is multiplication by the reciprocal of Constant.

Two final things to consider: First, in practice we may want to detect the unusual situation where both $i(n)$ and $q(n)$ are zero-valued, making the denominator of [Eq. \(13-117\)](#) equal to zero. We should set $\Delta\theta(n)$ to zero in that case. Second, for real-world noisy signals it may be prudent to apply the $\Delta\theta(n)$ output to a lowpass filter to reduce unwanted high-frequency noise.

13.23 DC Removal

When we digitize analog signals using an analog-to-digital (A/D) converter, the converter's output typically contains some small DC bias; that is, the average of the digitized time samples is not zero. That DC bias may have come from the original analog signal or from imperfections within the A/D converter. Another source of DC bias contamination in DSP is when we truncate a discrete sequence from a B -bit representation to word widths less than B bits. Whatever the source, unwanted DC bias on a signal can cause problems. When we're performing spectrum analysis, any DC bias on the signal shows up in the frequency domain as energy at zero Hz, the $X(0)$ spectral sample. For an N -point FFT the $X(0)$ spectral value is proportional to N and becomes inconveniently large for large-sized FFTs. When we plot our spectral magnitudes, the plotting software will accommodate any large $X(0)$ value and squash down the remainder of the spectrum in which we are more interested.

A nonzero DC bias level in audio signals is particularly troublesome because concatenating two audio signals, or switching between two audio signals, results in unpleasant audible clicks. In modern digital communications systems, a DC bias on quadrature signals degrades system performance and increases bit error rates. With that said, it's clear that methods for DC removal are of interest to many DSP practitioners.

13.23.1 Block-Data DC Removal

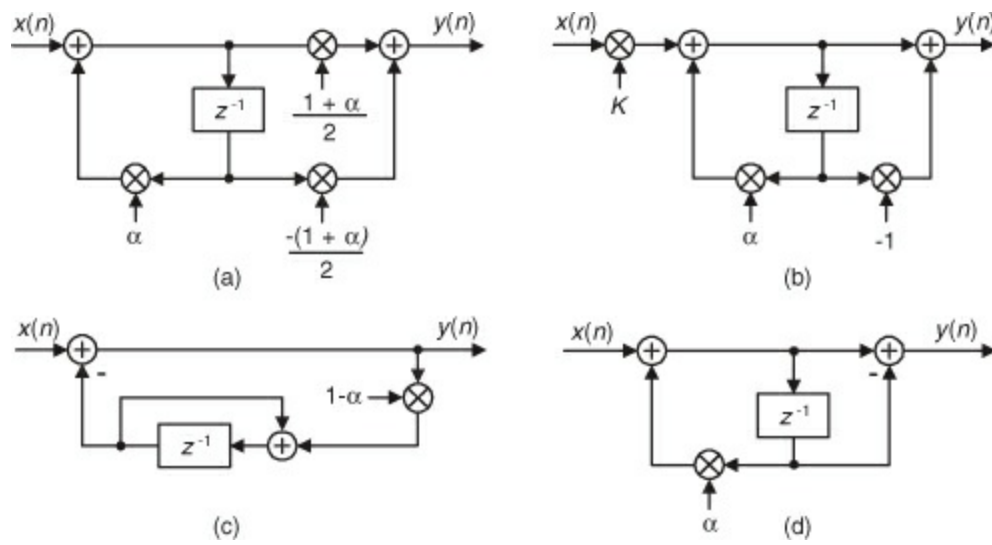
If you're processing in non-real time, and the signal data is acquired in blocks (fixed-length sequences) of block length N , DC removal is straightforward. We merely compute the average of our N time samples and subtract that average value from each original sample to yield a new time sequence whose DC bias will be extremely small.

This scheme, although very effective, is not compatible with continuous-throughput (real-time) systems. For real-time systems we're forced to use filters for DC removal.

13.23.2 Real-Time DC Removal

The author has encountered three proposed filters for DC removal[[56–58](#)]; their structures are shown in [Figures 13-62\(a\)](#), [13-62\(b\)](#), and [13-62\(c\)](#).

Figure 13-62 Filters used for DC bias removal.



Ignoring the constant gains of those DC-removal filters, all three filters have identical performance with the *general DC-removal filter* structure in [Figure 13-62\(d\)](#) having a z -domain transfer function of

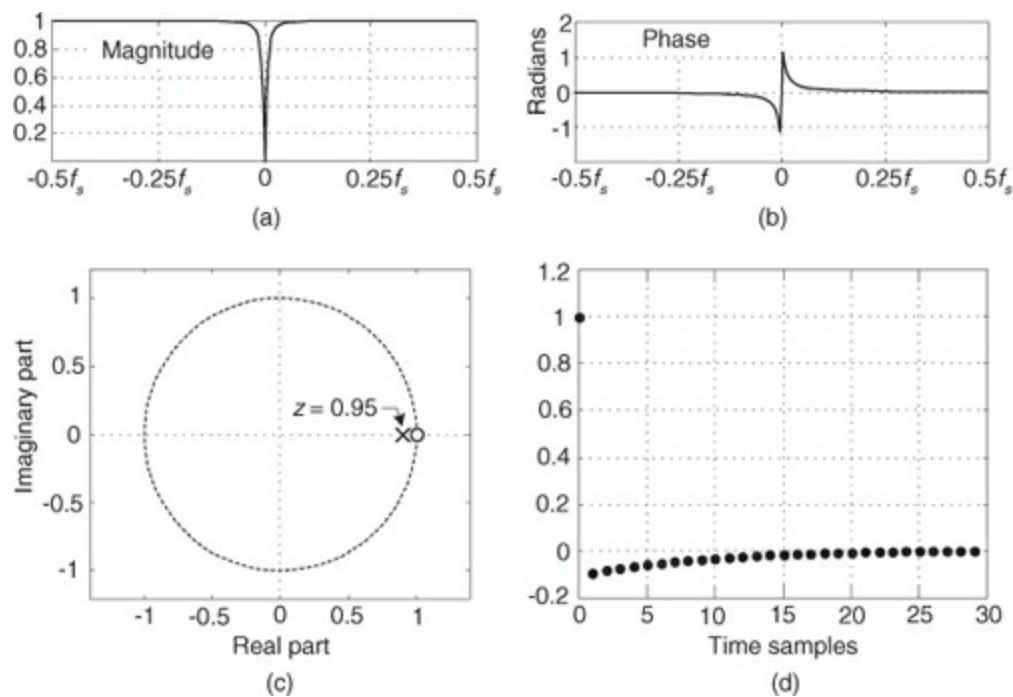
$$(13-118)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1 - z^{-1}}{1 - \alpha z^{-1}}.$$

(It's not immediately obvious that the filters in [Figures 13-62\(c\)](#) and [13-62\(d\)](#) are equivalent. You can verify that equivalency by writing the time-domain difference equations relating the various nodes in the feedback path of [Figure 13-62\(c\)](#)'s filter. Next, convert those equations to z -transform expressions and solve for $Y(z)/X(z)$ to yield [Eq. \(13-118\)](#)).

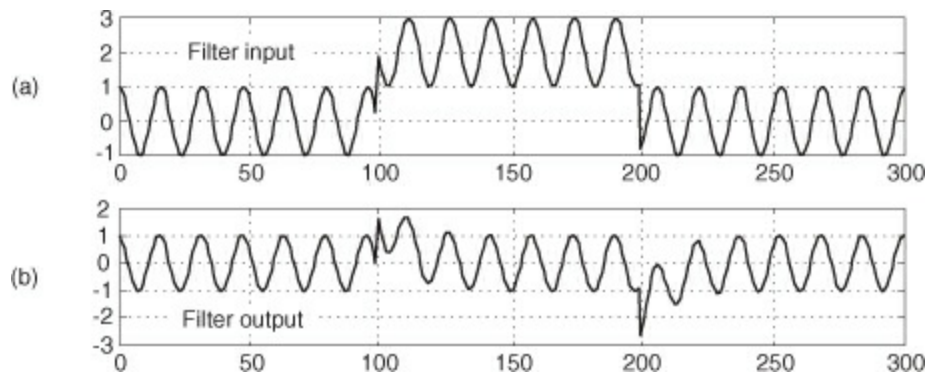
Because the DC-removal filters can be modeled with the general DC-removal filter in [Figure 13-62\(d\)](#), we provide the general filter's frequency magnitude and phase responses in [Figures 13-63\(a\)](#) and [13-63\(b\)](#) for $\alpha = 0.95$. The filter's pole/zero locations are given in [Figure 13-63\(c\)](#), where a zero resides at $z = 1$ providing infinite attenuation at DC (zero Hz) and a pole at $z = \alpha$ making the magnitude notch at DC very sharp. The closer α is to unity, the narrower the frequency magnitude notch centered at zero Hz. [Figure 13-63\(d\)](#) shows the general filter's unit-sample impulse response.

Figure 13-63 DC-removal filter, $\alpha = 0.95$: (a) magnitude response; (b) phase response; (c) pole/zero locations; (d) impulse response.



[Figure 13-64](#) shows the time-domain input/output performance of the general DC-removal filter (with $\alpha = 0.95$) when its input is a sinusoid suddenly contaminated with a DC bias of 2 beginning at the 100th time sample and disappearing at the 200th sample. The DC-removal filter works well.

Figure 13-64 DC-removal filter performance: (a) filter input with sudden DC bias; (b) filter output.



13.23.3 Real-Time DC Removal with Quantization

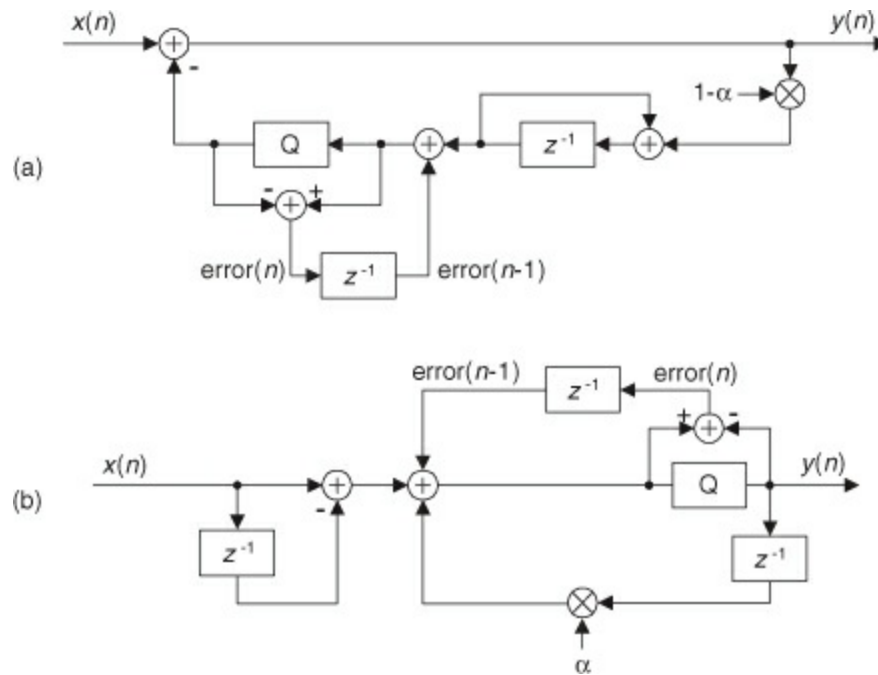
Because the general DC-removal filter has feedback, the $y(n)$ output samples may require wider binary word widths than those used for the $x(n)$ input samples. This could result in overflows in fixed-point binary implementations. The scaling factors of $(1+\alpha)/2$ and K , in [Figures 13-62\(a\)](#) and [13-62\(b\)](#), are less than one to minimize the chance of $y(n)$ binary overflow.

In fixed-point hardware the $y(n)$ samples are often truncated to the same word width as the input $x(n)$. This quantization (by means of truncation) will induce a negative DC bias onto the quantized output samples, degrading our desired DC removal. When we truncate a binary sample value, by discarding some of its least significant bits, we induce a negative error in the truncated sample. Fortunately, that error value is available for us to add to the next unquantized signal sample, increasing its positive DC bias. When that next sample is truncated, the positive error we've added minimizes the negative error induced by truncation of the next sample.

[Figure 13-65\(a\)](#) shows the addition of a quantizing sigma-delta modulator to the feedback path of the DC-removal filter given in [Figure 13-62\(c\)](#). The positive error induced by truncation quantization (the Q block) is delayed by one sample time and fed back to the quantizer input. Because the modulator has *an noise shaping* property where quantization error noise is shifted up in frequency,

away from zero Hz (DC), the overall DC bias at the output of the filter is minimized[57].

Figure 13-65 Two DC-removal filters using fixed-point quantization to avoid data overflow.



An equivalent quantization noise shaping process can be applied to a Direct Form I version of the [Figure 13-62\(d\)](#) general DC-removal filter as shown in [Figure 13-65\(b\)](#). Again, the positive quantization error is delayed by one sample time and added to the quantizer input[59–61]. To reiterate, the DC-removal filters in [Figure 13-65](#) are used to avoid binary data overflow, by means of quantization, without the use of scaling multipliers.

Later in this chapter we discuss a DC-removal filter whose frequency response exhibits linear phase.

13.24 Improving Traditional CIC Filters

A major design goal for cascaded integrator-comb (CIC) filters, as introduced in [Chapter 10](#) in conjunction with sample rate conversion, is to minimize their hardware power consumption by reducing data word width and reducing data clock rates wherever possible. Here we introduce a clever trick that reduces CIC filter power consumption using nonrecursive structures, by means of *polynomial factoring*, easing the word width growth problem. These nonrecursive structures require that the sample rate change R be an integer power of two, enhancing computational simplicity through *polyphase decomposition*, *transposed structures*, *simplified multiplication*, and *substructure sharing*[62–64]. (These processes are not complicated; they merely have fancy names.) Next, we'll review a nonrecursive scheme that enables sample rate changes other than powers of two. The following discussion assumes that the reader is familiar with the CIC filter material in [Chapter 10](#).

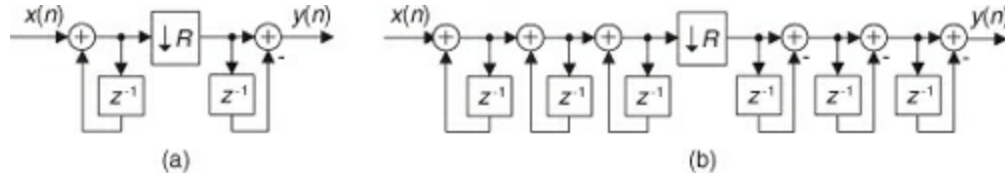
13.24.1 Nonrecursive CIC Filters

Recall that the structures of 1st-order ($M = 1$) and 3rd-order ($M = 3$) CIC decimation filters, having a comb delay equal to the sample rate change factor R , are those shown in [Figure 13-66](#). As presented in [Chapter 10](#), the transfer function of an M th-order decimating CIC filter can be expressed in either a recursive form or a nonrecursive form, as indicated in [Eq. \(13-119\)](#). (You could, if you wish, use the geometric series discussion in [Appendix B](#) to show the equality of the two forms of the filter's transfer function.)

$$H_{\text{cic}}(z) = \left[\frac{1 - z^{-R}}{1 - z^{-1}} \right]^M \quad \text{recursive form} \quad (13-119')$$

$$H_{\text{cic}}(z) = \left[\sum_{n=0}^{R-1} z^{-n} \right]^M = (1 + z^{-1} + z^{-2} + \dots + z^{-(R-1)})^M. \quad \text{nonrecursive form}$$

Figure 13-66 Recursive decimation CIC filters: (a) 1st-order filter; (b) 3rd-order filter.



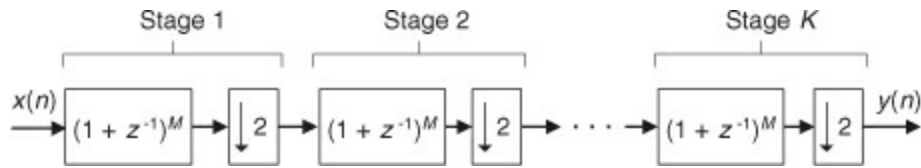
Now if the sample rate change factor R is an integer power of two, $R = 2^K$ where K is some positive integer, the [Eq. \(13-119'\)](#) M th-order nonrecursive polynomial form of $H_{\text{cic}}(z)$ can be factored as

$$(13-120)$$

$$H_{\text{cic}}(z) = (1 + z^{-1})^M (1 + z^{-2})^M (1 + z^{-4})^M \dots (1 + z^{-2^{K-1}})^M.$$

The reward for this factorization is that the CIC filter can then be implemented with K nonrecursive stages as shown in [Figure 13-67](#). This implementation eliminates filter feedback loops with their unpleasant binary word width growth. The data word width does increase in this nonrecursive structure by M bits for each stage, but the sampling rate is reduced by a factor of two for each stage. This nonrecursive structure has been shown to consume less power than the [Figure 13-66\(b\)](#) recursive implementation for filter orders greater than three and decimation/interpolation factors larger than eight[\[64\]](#). Thus the power savings from sample rate reduction are greater than the power consumption increase due to data word width growth.

Figure 13-67 Multistage M th-order nonrecursive CIC structure.



Happily, further improvements are possible with each stage of this nonrecursive structure[\[63\]](#). For example, assume we desire an $M = 5$ th-order decimating CIC for Stage 1 in [Figure 13-67](#). In that case, the stage's transfer function is

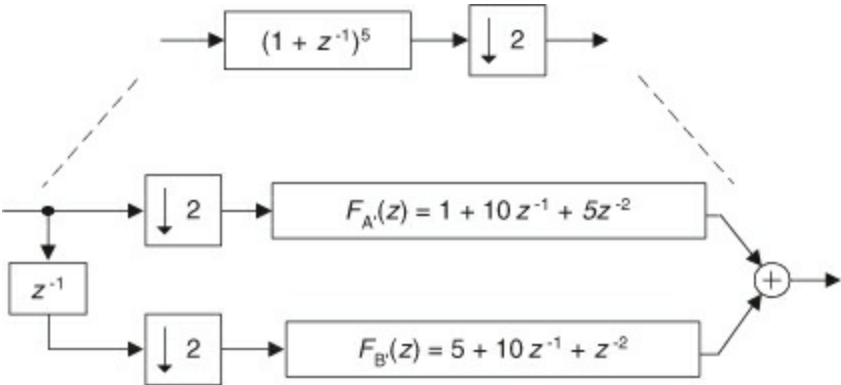
$$(13-121)$$

$$\begin{aligned} H_1(z) &= (1 + z^{-1})^5 = 1 + 5z^{-1} + 10z^{-2} + 10z^{-3} + 5z^{-4} + z^{-5} \\ &= 1 + 10z^{-2} + 5z^{-4} + (5 + 10z^{-2} + z^{-4})z^{-1} = F_A(z) + F_B(z)z^{-1}. \end{aligned}$$

The second step in [Eq. \(13-121\)](#), known as *polyphase decomposition*[\[65–69\]](#), enables a polyphase implementation having two parallel paths as shown in [Figure 13-68](#). The initial delay element and the dual decimation-by-two operations are implemented by routing the odd-indexed input samples to $F_A(z)$, and the even-indexed samples to $F_B(z)$. Because we implement decimation by two before the filtering, the new polyphase components are $F_A(z) = 1 + 10z^{-1} + 5z^{-2}$, and $F_B(z) = 5 + 10z^{-1} + z^{-2}$ implemented at half the data rate into the stage. (Reducing data rates as early as possible is a key

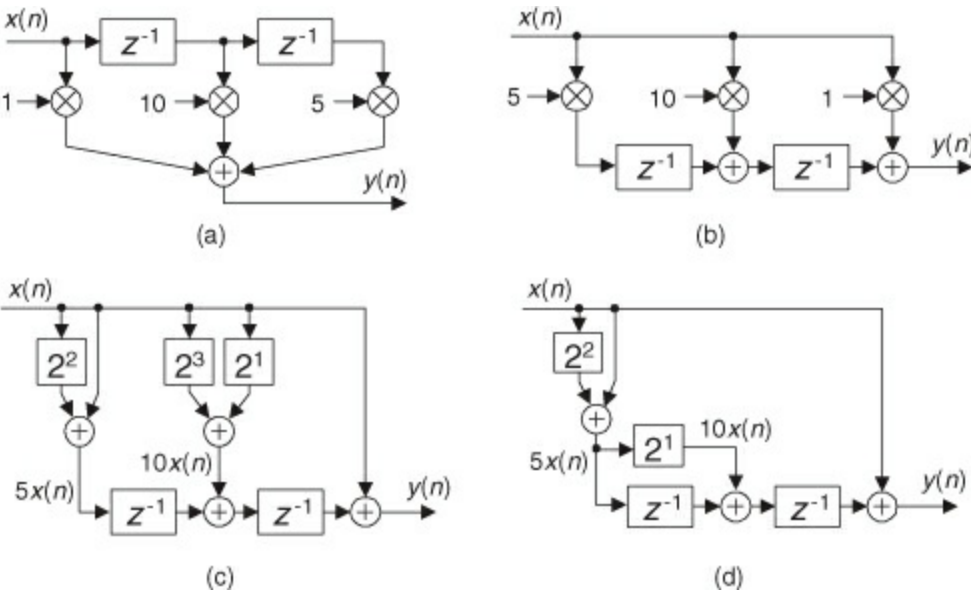
design goal in the implementation of CIC decimation filters.)

Figure 13-68 Polyphase structure of a single nonrecursive 5th-order CIC stage.



The $F_A(z)$ and $F_B(z)$ polyphase components are implemented in a tapped-delay line fashion and, fortunately, further simplifications are possible. Let's consider the $F_A(z)$ polyphase filter component, in a tapped-delay line configuration, shown in [Figure 13-69\(a\)](#). The transposed version of this filter is presented in [Figure 13-69\(b\)](#) with its flipped coefficient sequence. The adder in [Figure 13-69\(a\)](#) must perform two additions per input data sample, while in the transposed structure no adder need perform more than one add per data sample. Thus the transposed structure can operate at a higher speed.

Figure 13-69 Filter component $F_A(z)$: (a) delay line structure; (b) transposed structure; (c) simplified multiplication; (d) substructure sharing.



The next improvement uses simplified multiplication, as shown in [Figure 13-69\(c\)](#), by means of arithmetic shifts and adds. Thus a factor of five is implemented as $2^2 + 1$, eliminating all multiplications. Finally, because of the transposed structure, we can use the technique of *substructure sharing* in [Figure 13-69\(d\)](#) to reduce the hardware component count. Pretty slick! By the way, these nonrecursive filters are still called cascaded integrator-comb filters, even though they have no integrators. Go figure.

[Table 13-7](#) is provided to help the reader avoid computing the polynomial equivalent of several M th-order nonrecursive stages, as was performed in [Eq. \(13-121\)](#).

Table 13-7 Expansions of $(1 + z^{-1})^M$

M	$(1 + z^{-1})^M$
2	$(1 + z^{-1})^2 = 1 + 2z^{-1} + z^{-2}$
3	$(1 + z^{-1})^3 = 1 + 3z^{-1} + 3z^{-2} + z^{-3}$
4	$(1 + z^{-1})^4 = 1 + 4z^{-1} + 6z^{-2} + 4z^{-3} + z^{-4}$
5	$(1 + z^{-1})^5 = 1 + 5z^{-1} + 10z^{-2} + 10z^{-3} + 5z^{-4} + z^{-5}$
6	$(1 + z^{-1})^6 = 1 + 6z^{-1} + 15z^{-2} + 20z^{-3} + 15z^{-4} + 6z^{-5} + z^{-6}$
7	$(1 + z^{-1})^7 = 1 + 7z^{-1} + 21z^{-2} + 35z^{-3} + 35z^{-4} + 21z^{-5} + 7z^{-6} + z^{-7}$
8	$(1 + z^{-1})^8 = 1 + 8z^{-1} + 28z^{-2} + 56z^{-3} + 70z^{-4} + 56z^{-5} + 28z^{-6} + 8z^{-7} + z^{-8}$
9	$(1 + z^{-1})^9 = 1 + 9z^{-1} + 36z^{-2} + 84z^{-3} + 126z^{-4} + 126z^{-5} + 84z^{-6} + 36z^{-7} + 9z^{-8} + z^{-9}$

13.24.2 Nonrecursive Prime-Factor- R CIC Filters

The nonrecursive CIC decimation filters described above have the restriction that the R decimation factor must be an integer power of two. That constraint is loosened due to a clever scheme of factoring R into a product of prime numbers[70]. This *multiple prime-factor- R* technique is based on the process of factoring integer R into the form $R = 2^p 3^q 5^r 7^s 11^t \dots$, where 2, 3, 5, 7, 11 are the prime numbers. (This process is called *prime factorization*, or *prime decomposition*, and has been of interest since the days of Euclid.) Then the appropriate number of CIC subfilters are cascaded as shown in [Figure 13-70\(a\)](#). The fortunate condition is that those M th-order CIC filters are described by

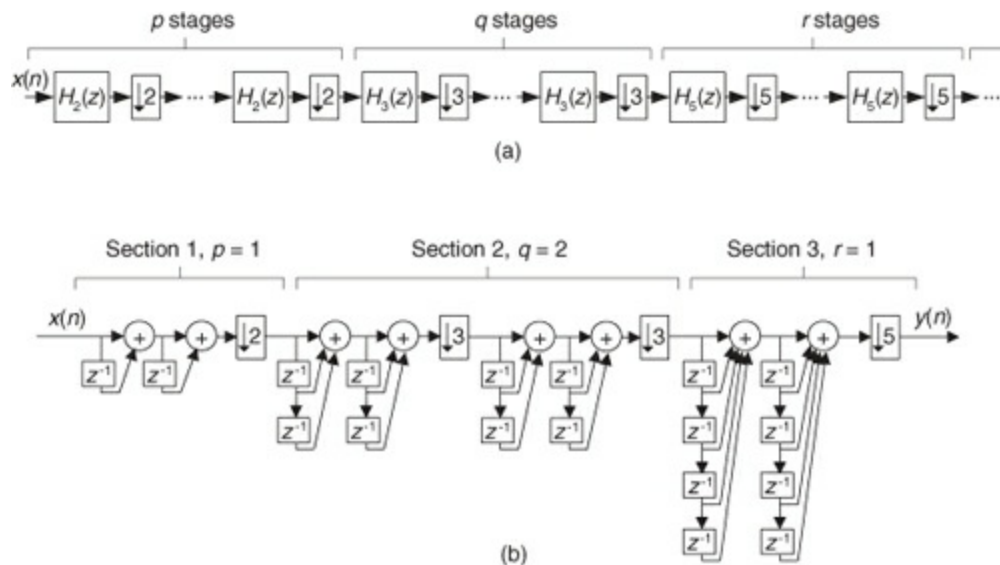
$$(13-122)$$

$$H_2(z) = \left[\frac{1 - z^{-2}}{1 - z^{-1}} \right]^M = (1 + z^{-1})^M$$

$$H_3(z) = \left[\frac{1 - z^{-3}}{1 - z^{-1}} \right]^M = (1 + z^{-1} + z^{-2})^M$$

$$H_5(z) = \left[\frac{1 - z^{-5}}{1 - z^{-1}} \right]^M = (1 + z^{-1} + z^{-2} + z^{-3} + z^{-4})^M$$

Figure 13-70 Multiple prime-factor nonrecursive CIC example: (a) cascaded-stage structure; (b) 2nd-order, $R = 90$, nonrecursive CIC example.



and so on, enabling nonrecursive implementations.

Due to space constraints, the elegant and arduous derivation of this technique is not given here; but this process can be illustrated with an example. Assume we desire a 2nd-order ($M = 2$) CIC filter with a decimation factor of $R = 90$. That decimation rate is factored as $90 = (2)(3)(3)(5)$, so $p = 1$, $q = 2$, and $r = 1$. Our composite CIC filter is implemented as $H_2(z)H_3(z)H_3(z)H_5(z)$ shown in [Figure 13-70\(b\)](#).

At first glance the many additions of the [Figure 13-70\(b\)](#) CIC filter appear to aggravate the power consumption of such a filter, but the reduced sample rates significantly reduce power requirements[\[70\]](#). If one addition in Section 1 of [Figure 13-70\(b\)](#) consumes P units of power, then Section 1 consumes $2P$ units of power, and each addition in the first portion of Section 2 consumes $P/2$ units of power. Each addition in the second portion of Section 2 consumes $P/6$ of units power, while each addition in Section 3 consumes $P/18$ units of power.

We have flexibility here because the subfilters in each section of [Figure 13-70\(b\)](#) can be implemented recursively or nonrecursively, as indicated in [Eq. \(13-122\)](#). In nonrecursive implementations the polyphase decomposition, transposed structures, simplified multiplication, and substructure sharing schemes can be applied. CIC filter design certainly has come a long way since its introduction in the early 1980s.

13.25 Smoothing Impulsive Noise

In practice we may be required to make precise measurements in the presence of high noise or interference. Without some sort of analog signal conditioning, or digital signal processing, it can be difficult to obtain stable and repeatable measurements. This impulsive-noise smoothing trick, originally developed to detect microampere changes in milliampere signals, describes a smoothing algorithm that improves the stability of precision measurements in the presence of impulsive noise[\[71\]](#).

Practical noise-reduction methods often involve multiple-sample averaging (*block averaging*) of a sequence of measured values, $x(n)$, to compute a sequence of N -sample arithmetic means, $M(q)$. As such, the block-averaged sequence $M(q)$ is defined by

(13-123)

$$M(q) = \frac{1}{N} \sum_{k=qN}^{(q+1)N-1} x(n)$$

where the time index of the averaging process is $q = 0, 1, 2, 3$, etc. When $N = 10$, for example, for the first block of data ($q = 0$), time samples $x(0)$ through $x(9)$ are averaged to compute $M(0)$. For the second block of data ($q = 1$), time samples $x(10)$ through $x(19)$ are averaged to compute $M(1)$, and so on[\[72\]](#).

The following impulsive-noise smoothing algorithm processes a block of time-domain samples, obtained through periodic sampling, and the number of samples, N , may be varied according to individual needs and processing resources. The processing of a single block of N time samples proceeds as follows: Collect $N+2$ samples of $x(n)$, discard the maximum (most positive) and minimum (most negative) samples to obtain an N -sample block of data, and compute the arithmetic mean, $M(q)$, of the N samples. Each sample in the block is then compared to the mean. The direction of each sample relative to the mean (greater than, or less than) is accumulated, as well as the cumulative magnitude of the deviation of the samples in one direction (which, by definition of the

mean, equals that of the other direction). This data is used to compute a correction term that is added to the mean according to the following formula,

(13-124)

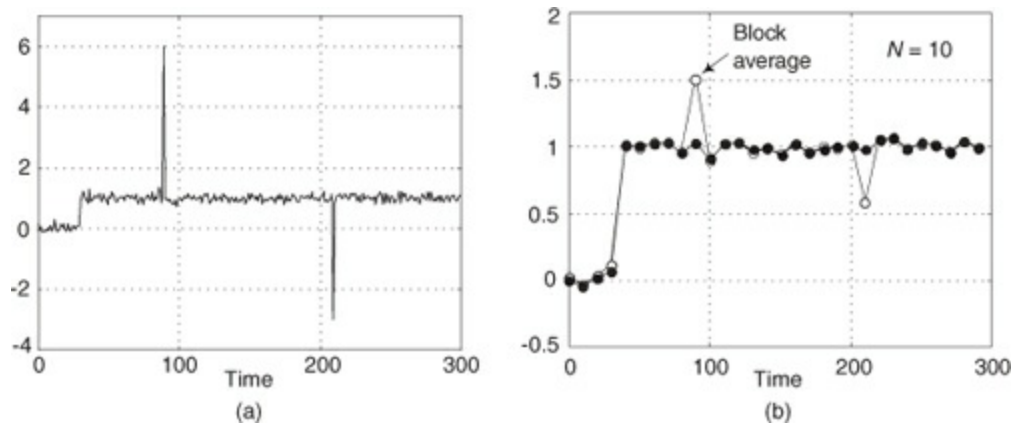
$$A(q) = M(q) + \frac{(P_{os} - N_{eg}) |D_{total}|}{N^2},$$

where $A(q)$ is the *corrected mean*, $M(q)$ is the arithmetic mean (average) from [Eq. \(13-123\)](#), P_{os} is the number of samples greater than $M(q)$, N_{eg} is the number of samples less than $M(q)$, and D_{total} is the sum of deviations from the mean (absolute values and one direction only). D_{total} , then, is the sum of the differences between the P_{os} samples and $M(q)$.

For an example, consider a system acquiring ten measured samples of 10, 10, 11, 9, 10, 10, 13, 10, 10, and 10. The mean is $M = 10.3$, the total number of samples positive is $P_{os} = 2$, and the total number of samples negative is $N_{eg} = 8$ (so $P_{os} - N_{eg} = -6$). The total deviation in either direction from the mean is 3.4 (using the eight samples less than the mean, $(10.3 - 10)$ times 7 plus $(10.3 - 9)$; or using the two samples greater than the mean, $(13 - 10.3)$ plus $(11 - 10.3)$). With $D_{total} = 3.4$, [Eq. \(13-124\)](#) yields an improved result of $A = 10.096$.

The smoothing algorithm's performance, relative to traditional block averaging, can be illustrated by example. [Figure 13-71\(a\)](#) shows a measured 300-sample $x(n)$ signal sequence comprising a step signal of amplitude one contaminated with random noise (with a variance of 0.1) and two large impulsive-noise spike samples.

Figure 13-71 Noise smoothing for $N = 10$: (a) input $x(n)$ signal; (b) block average output (white) and impulsive-noise smoothing algorithm output (solid).



A few meaningful issues regarding this noise smoothing process are:

- The block size (N) used in the smoothing algorithm can be any integer, but for real-time fixed binary-point implementations it's beneficial to set N equal to an integer power of two. In that case the compute-intensive division operations in [Eqs. \(13-123\)](#) and [\(13-124\)](#) can be accomplished by binary arithmetic right shifts to reduce the computational workload.
- If there's a possibility that more than two large noise spikes are contained in a block of input samples, then we collect more than $N+2$ samples of $x(n)$ and discard the appropriate number of maximum and minimum samples to eliminate the large impulsive noise samples.
- We could forgo the [Eq. \(13-124\)](#) processing and merely perform [Eq. \(13-123\)](#) to compute the mean $M(q)$. In that case, for a given N , the standard deviation of $M(q)$ would be roughly 15 to 20

percent greater than $A(q)$.

As pointed out by M. Givens, impulsive noise can also be reduced by a class of filters known as *median filters* [73]. Median filters, not covered in this text, are typically used in noise reduction of two-dimensional signals (images). However, median filters can also be implemented to process one-dimensional signals, such as our $x(n)$ signal here, and should be considered in any impulsive-noise reduction application.

13.26 Efficient Polynomial Evaluation

On the off chance that you didn't know, there are two popular tricks used to speed up polynomial evaluations (computations), known as *Horner's Rule* and *Estrin's Method*. We illustrate those two techniques below.

13.26.1 Floating-Point Horner's Rule

Horner's Rule uses nested operations to reduce the number of multiply operations needed to compute polynomials. An example of a polynomial computation is, for example, using the following expression to compute the arctangent of x :

(13-125)

$$\arctan(x) = 0.14007x^4 - 0.34241x^3 - 0.01522x^2 + 1.00308x - 0.00006.$$

To see how the computational workload of polynomial evaluations can be reduced, consider the following k th-order polynomial:

(13-126)

$$f_k(x) = c_k x^k + \dots + c_3 x^3 + c_2 x^2 + c_1 x + c_0.$$

It can be rewritten as

(13-127)

$$f_k(x) = f_{\text{H}k}(x) = x(x(x(\dots x(c_k x + c_{k-1}) + c_{k-2}) \dots + c_2) + c_1) + c_0$$

where the "H" subscript means Horner. Using this method to compute polynomials

- reduces the number of necessary multiply operations, and
- is straightforward to implement using programmable DSP chips with their *multiply and accumulate* (MAC) architectures.

For example, consider the 5th-order polynomial

(13-128)

$$f_5(x) = c_5 x^5 + c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0.$$

Evaluated in the standard way, [Eq. \(13-128\)](#) would require nine multiplies and five additions, whereas the Horner version

(13-128')

$$f_5(x) = f_{\text{H}5}(x) = x(x(x(x(c_5 x + c_4) + c_3) + c_2) + c_1) + c_0$$

requires only five multiplies and five adds when the computations begin with the innermost multiply and add operations ($c_5 x + c_4$).

Here are a few examples of polynomials in the Horner format:

(13-129)

$$c_2x^2 + c_1x + c_0 = x(c_2x + c_1) + c_0.$$

(13-130)

$$c_3x^3 + c_2x^2 + c_1x + c_0 = x(x(c_3x + c_2) + c_1) + c_0.$$

(13-131)

$$c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 = x(x(x(c_4x + c_3) + c_2) + c_1) + c_0.$$

By the way, the multiplications and additions cannot be performed in parallel. Because Horner's Rule is inherently serial, we need the result of the last multiplication before we can start the next addition, and that addition result is needed before the follow-on multiplication.

Horner's Rule is another of those handy computer techniques we use whose origins are very old. Chinese mathematicians described it in the 1200s. European mathematicians (including William Horner) rediscovered and publicized it in the early 1800s. However, it seems Sir Isaac Newton also invented and used it in the 1600s.

13.26.2 Horner's Rule in Binary Shift Multiplication/Division

The Horner's Rule method of nested multiplications has special significance for us when we evaluate polynomials using fixed-point number formats. Using Horner's Rule enables us to minimize the truncation quantization error when we use binary right shifts to implement fractional multiplications. For example, if we are using fractional fixed-point numbers in the 1.15 format, as described in [Section 12.1.6](#), and we want to multiply an $x(n)$ sample by 0.3125, we can perform that multiplication as

(13-132)

$$0.3125x(n) = 2^{-2}x(n) + 2^{-4}x(n).$$

Those scaling factors on the right side of [Eq. \(13-132\)](#) can be implemented using binary right shifts by two and four bits. The larger the right shifts, however, the greater the truncation quantization errors in this type of fractional multiplication. Using Horner's Rule, we can implement [Eq. \(13-132\)](#) as

(13-132')

$$0.3125x(n) = 2^{-2}[x(n) + 2^{-2}x(n)],$$

where the maximum binary right shift is by two bits, reducing the resultant truncation quantization error.

13.26.3 Estrin's Method

If your computing hardware is able to perform multiple parallel (simultaneous) *multiply and accumulate* (MAC) operations, we can increase the computational speed of Horner's Rule by using parallel processing in a technique called *Estrin's Method*.

Here's how Estrin's Method works: Various k th-order polynomials, such as that in [Eq. \(13-126\)](#), can be evaluated using

$$\begin{aligned}
f_1(x) &= (c_1x + c_0) \\
f_2(x) &= [c_2x^2 + (c_1x + c_0)] \\
f_3(x) &= [(c_3x + c_2)x^2 + (c_1x + c_0)] \\
f_4(x) &= \{c_4x^4 + [(c_3x + c_2)x^2 + (c_1x + c_0)]\} \\
f_5(x) &= \{(c_5x + c_4)x^4 + [(c_3x + c_2)x^2 + (c_1x + c_0)]\} \\
f_6(x) &= \{[c_6x^2 + (c_5x + c_4)]x^4 + [(c_3x + c_2)x^2 + (c_1x + c_0)]\} \\
f_7(x) &= \{[(c_7x + c_6)x^2 + (c_5x + c_4)]x^4 + [(c_3x + c_2)x^2 + (c_1x + c_0)]\} \\
f_8(x) &= [c_8x^8 + \{[(c_7x + c_6)x^2 + (c_5x + c_4)]x^4 + [(c_3x + c_2)x^2 + (c_1x + c_0)]\}] \\
f_9(x) &= [(c_9x + c_8)x^8 + \{[(c_7x + c_6)x^2 + (c_5x + c_4)]x^4 + [(c_3x + c_2)x^2 + (c_1x + c_0)]\}].
\end{aligned}$$

The above expressions look complicated, but they're really not. The terms inside parentheses, brackets, and curly brackets are nested sub-expressions of the form $ax^q + b$ —precisely what we need for MAC operations. For example, the sub-expressions within parentheses can be computed simultaneously with a DSP processor's parallel MAC operations.

To illustrate Estrin's Method, if your processing hardware can perform four simultaneous MAC operations, and assuming value x^2 has been previously computed, we can evaluate polynomial $f_7(x)$ in the following three steps:

1. $U = (c_7x + c_6)$, $V = (c_5x + c_4)$, $W = (c_3x + c_2)$, and $X = (c_1x + c_0)$
2. $Y = (Ux^2 + V)$, $Z = (Wx^2 + X)$, $x^4 = (x^2x^2 + 0)$
3. $f_7(x) = (Yx^4 + Z)$

The four computations in Step 1 are performed simultaneously. Likewise, the three computations in Step 2 are performed simultaneously. The final Step 3 is a single MAC operation.

Yes, Estrin's Method requires multiple processing steps, but this method is able to avoid much of the inherent (slow) serial processing dictated by Horner's Rule. The bottom line here is that while Estrin's Method does not reduce the computational workload (number of multiplies and additions) of Horner's Rule, it does increase the computational speed of polynomial evaluations by taking advantage of modern-day parallel processing hardware architectures.

13.27 Designing Very High-Order FIR Filters

There are linear-phase filtering applications wherein we're interested in designing very high-performance (very narrow passband widths, and/or very high attenuation) nonrecursive FIR filters. Consider the possibility that you've used [Eq. \(7-34\)](#), or some other algorithm, to determine that you need to implement a 2000-tap linear-phase FIR filter. Then when you try to design such a filter using your trusty Parks-McClellan Exchange-based (Remez) filter design software, you obtain unusable design results. It happens that some software incarnations of the Parks-McClellan Exchange algorithm have convergence problems (inaccurate results) when the number of filter taps, or filter order, exceeds 400 to 500. There's a slick way around this high-order FIR filter design problem using a frequency-domain zero-stuffing technique.[‡]

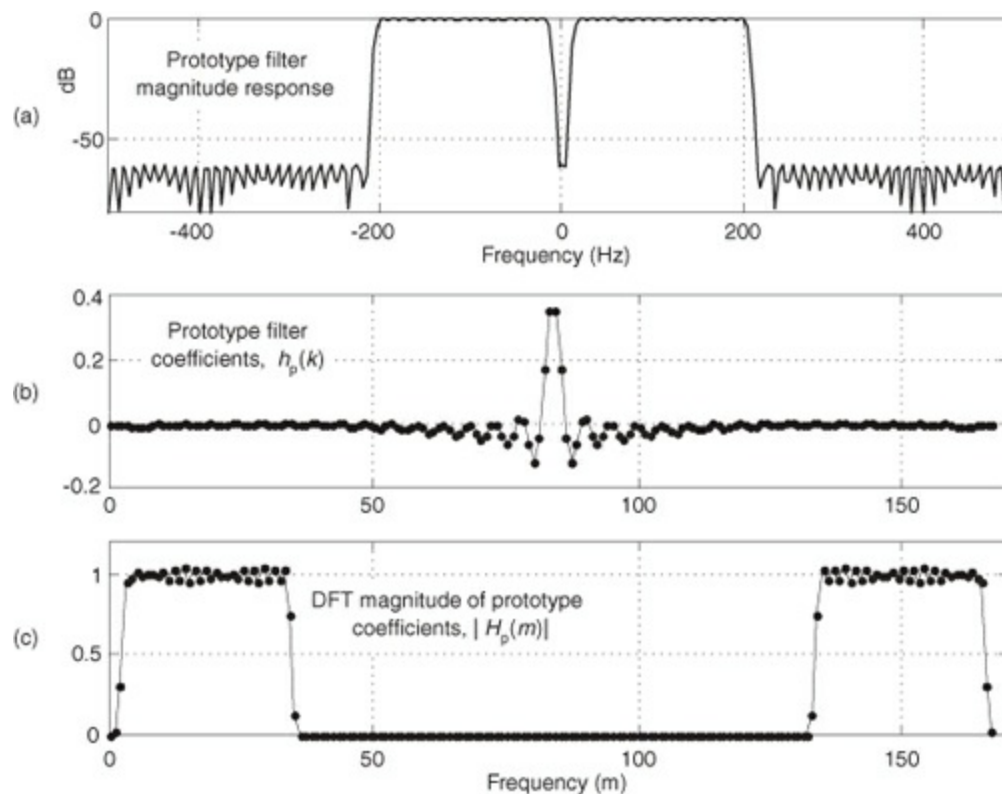
[‡] I thank my DSP pal Eric Jacobsen, Minister of Algorithms at Abineau Communications, for publicizing this technique.

If our FIR filter design software cannot generate FIR coefficient sets whose lengths are in the thousands, then we can design a shorter-length set of coefficients and interpolate those coefficients (time-domain impulse response) to whatever length we desire. Rather than use time-domain interpolation schemes and account for their inaccuracies, we can simplify the process by performing

time-domain interpolation by means of frequency-domain zero stuffing.

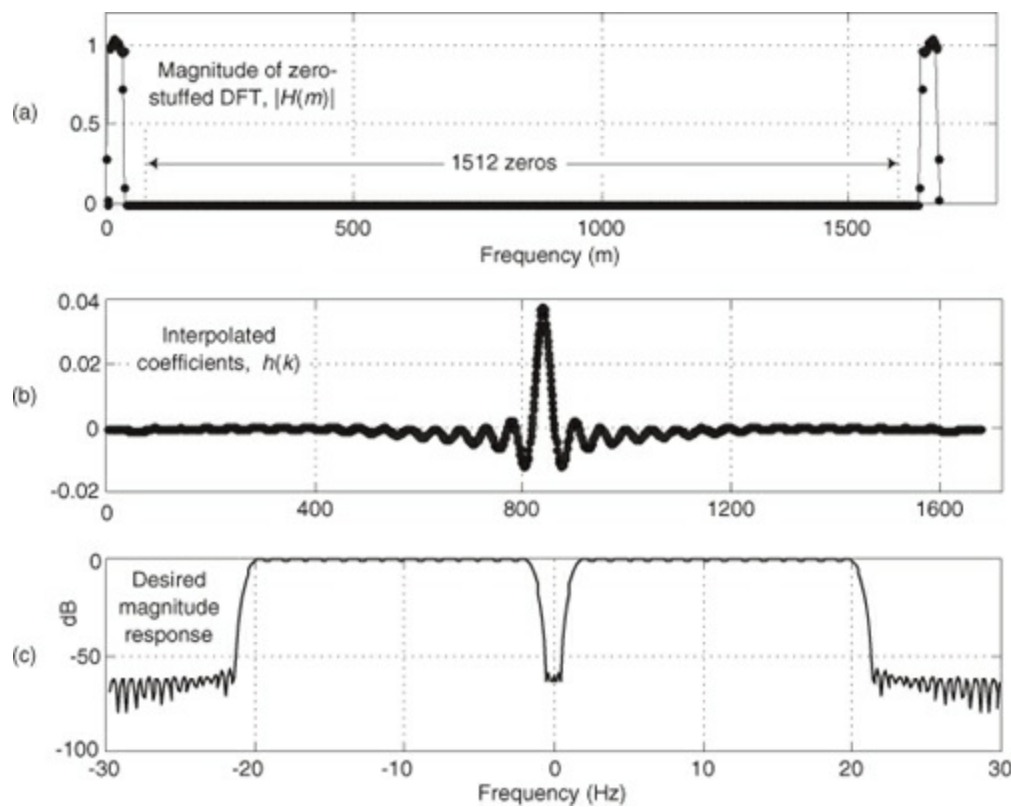
An example of the process is as follows: Assume that we have a signal sampled at a rate of $f_s = 1000$ Hz. We want a lowpass filter whose cutoff frequency is 20 Hz with 60 dB of stopband attenuation. Compounding the problem are the requirements for linear phase and removal of any DC (zero Hz) component from the signal. (Those last two requirements preclude using the DC-removal schemes in [Section 13.23](#).) First, we design a prototype nonrecursive FIR filter having, say, $N = 168$ coefficients whose desired frequency response magnitude is shown in [Figure 13-72\(a\)](#); its $h_p(k)$ coefficients are depicted in [Figure 13-72\(b\)](#). Next, we compute a 168-point DFT of the coefficients to obtain the frequency-domain samples $H_p(m)$ whose magnitudes are shown in [Figure 13-72\(c\)](#).

Figure 13-72 Prototype FIR filter: (a) magnitude response; (b) $h_p(k)$ coefficients; (c) $|H_p(m)|$ magnitudes of the 168-point DFT of $h_p(k)$.



Under the assumption that our final desired filter requires roughly 1600 taps, we'll interpolate the $h_p(k)$ prototype impulse response by a factor of $M = 10$. We perform the interpolation by inserting $(M - 1)N$ zeros in the center of the $H_p(m)$ frequency-domain samples, yielding a 1680-point $H(m)$ frequency-domain sequence whose magnitudes are shown in [Figure 13-73\(a\)](#). Finally, we perform a 1680-point inverse DFT on $H(m)$ to obtain the interpolated $h(k)$ impulse response (coefficients), shown in [Figure 13-73\(b\)](#), for our desired filter. (The ten-fold compression of the $H_p(m)$ passband samples results in a ten-fold expansion of the $h_p(k)$ impulse response samples.) The frequency magnitude response of our final very high-order FIR filter, over the frequency range of -30 to 30 Hz, is provided in [Figure 13-73\(c\)](#).

Figure 13-73 Desired FIR filter: (a) magnitude of zero-stuffed $H_p(m)$; (b) interpolated $h(k)$ coefficients; (c) magnitude of desired frequency response.



With this process, the prototype filter's $h_p(k)$ coefficients are preserved within the interpolated filter's coefficients if the $H_p(N/2)$ sample ($f_s/2$) is zero. That condition ensures that $H(m)$ exhibits conjugate symmetry and forces the $h(k)$ coefficients to be real-only.

The design steps for this high-order filter design scheme are:

- With the desired filter requiring MN taps, set the number of prototype filter coefficients, N , to an integer value small enough so your FIR filter design software provides usable results. The integer interpolation factor M equals the number of desired taps divided by N .
- Design the N -tap prototype FIR filter accounting for the M -fold frequency compression in the final filter. (That is, cutoff frequencies for the prototype filter are M times the desired final cutoff frequencies.)
- Perform an N -point DFT on the prototype filter's $h_p(k)$ coefficients to obtain $H_p(m)$.
- Insert $M-1$ zero-valued samples just before the $H_p(N/2)$ sample of $H_p(m)$ to obtain the new MN -point $H(m)$ frequency response.
- Compute the MN -point inverse DFT of $H(m)$, yielding an MN -length interpolated $h(k)$ coefficient set. (Due to computational errors, discard the imaginary part of $h(k)$, making it real-only.)
- Multiply $h(k)$ by M to compensate for the $1/M$ amplitude loss induced by interpolation.
- Test the $h(k)$ coefficient set to determine its actual frequency response using standard filter analysis methods. (One method: append thousands of zeros to $h(k)$ and perform a very large FFT on the expanded sequence.)

An example application of this filter design is when you're building a high-performance lowpass polyphase filter, as discussed in [Chapter 10](#). (The structures of the high-performance *interpolated FIR* and *frequency sampling* lowpass filters don't permit their decomposition into polyphase subfilters for such an application.)

13.28 Time-Domain Interpolation Using the FFT

The thoughtful reader may have looked at the above [Section 13.27](#) FIR filter impulse response interpolation scheme and wondered, “If we can interpolate time-domain impulse responses, we should be able to interpolate time-domain signals using the same frequency-domain zero-stuffing method.” To quote Rocky Balboa, “This is very true.” In fact, the [Section 13.27](#) interpolation-by- M process applied to time signals is sometimes called *exact interpolation* because its performance is equivalent to using an *ideal*, infinite-stopband attenuation, time-domain interpolation filter. Let’s see how this interpolation scheme works.

To establish our notation, let’s say we compute the FFT of an N -point $x(n)$ time sequence to produce its $X(m)$ frequency-domain samples. Next we stuff $(M-1)N$ zeros in the middle of $X(m)$ to yield the MN -length $X_{\text{int}}(m)$ frequency samples, where MN is an integer power of two. Then we perform an MN -point inverse FFT on $X_{\text{int}}(m)$ to obtain the interpolated-by- M $x_{\text{int}}(n)$ times samples. Using this frequency-domain zero stuffing to implement time-domain signal interpolation involves two important issues upon which we now focus.

13.28.1 Computing Interpolated Real Signals

The first issue: to ensure the interpolated $x_{\text{int}}(n)$ time sequence is real-only, conjugate symmetry must be maintained in the zero-stuffed $X_{\text{int}}(m)$ frequency samples. If the $X(m)$ sequence has a nonzero sample at $X_{\text{int}}(N/2)$, the $f_s/2$ frequency component, we must use the following steps in computing $X_{\text{int}}(m)$ to guarantee conjugate symmetry:

- Perform an N -point FFT on an N -point $x(n)$ time sequence, yielding N frequency samples, $X(m)$.
- Create an MN -point spectral sequence $X_{\text{int}}(m)$ initially set to all zeros.
- Assign $X_{\text{int}}(m) = X(m)$, for $0 \leq m \leq (N/2)-1$.
- Assign both $X_{\text{int}}(N/2)$ and $X_{\text{int}}(MN-N/2)$ equal to $X(N/2)/2$. (This step, to maintain conjugate symmetry and improve interpolation accuracy, is not so well known[\[74\]](#).)
- Assign $X_{\text{int}}(m) = X(q)$, where $MN-(N/2)+1 \leq m \leq MN-1$, and $(N/2)+1 \leq q \leq N-1$.
- Compute the real part of the MN -point inverse FFT of $X_{\text{int}}(m)$, yielding the desired MN -length interpolated $x_{\text{int}}(n)$ sequence.
- Finally, if desired, multiply $x_{\text{int}}(n)$ by M to compensate for the $1/M$ amplitude loss induced by interpolation.

Whew! Our mathematical notation makes this signal interpolation scheme look complicated, but it’s really not so bad. [Table 13-8](#) shows the frequency-domain $X_{\text{int}}(m)$ sample assignments, where $0 \leq m \leq 15$, to interpolate an $N = 8$ -point $x(n)$ sequence by a factor of $M = 2$.

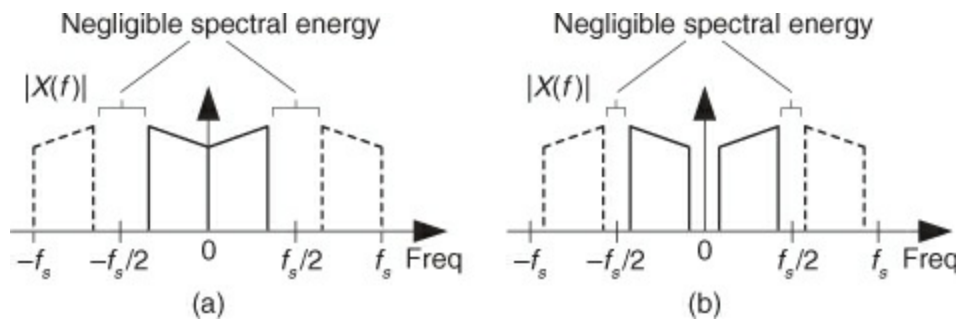
Table 13-8 $X_{\text{int}}(m)$ Assignments for Interpolation by Two

m	$X_{\text{int}}(m)$	m	$X_{\text{int}}(m)$
0	$X(0)$	8	0
1	$X(1)$	9	0
2	$X(2)$	10	0
3	$X(3)$	11	0
4	$X(4)/2$	12	$X(4)/2$
5	0	13	$X(5)$
6	0	14	$X(6)$
7	0	15	$X(7)$

One of the nice properties of the above algorithm is that every M th $x_{\text{int}}(n)$ sample coincides with the original $x(n)$ samples. In practice, due to our finite-precision computing, the imaginary parts of our final $x_{\text{int}}(n)$ may have small nonzero values. As such, we take $x_{\text{int}}(n)$ to be the real part of the inverse FFT of $X_{\text{int}}(m)$.

Here's the second issue regarding time-domain real signal interpolation. This technique of interpolation using FFT zero stuffing only provides acceptable results when the original $x(n)$ time sequence has a spectrum having negligible spectral energy in the vicinity of $\pm f_s/2$, as shown in [Figure 13-74](#) for lowpass and bandpass signals. By negligible we mean spectral magnitudes that are, say, below a discrete signal's quantization noise background spectral level.

Figure 13-74 Spectral restrictions for interpolation using the FFT: (a) lowpass signal case; (b) bandpass signal case.



An example of violating the above spectral restriction is when $x(n)$ is a sinusoidal sequence containing a noninteger number of cycles. That signal's positive-frequency spectrum will have nonzero spectral energy extending from zero Hz to $f_s/2$ Hz caused by spectral leakage. Trying to interpolate such an $x(n)$ using this FFT zero-stuffing scheme will yield an interpolated time sequence with unacceptably high amplitude errors at the beginning and end of the interpolated sequence.

With the advent of fast hardware DSP chips and pipelined FFT techniques, the above time-domain interpolation algorithm may be viable for a number of applications, such as computing selectable sample rate time sequences of a test signal that has a fixed spectral envelope shape; providing interpolation, by selectable factors, of signals that were filtered in the frequency domain using the fast convolution method ([Section 13.10](#)); or digital image resampling. One scenario to consider is using the efficient $2N$ -Point Real FFT technique, described in [Section 13.5.2](#), to compute the forward FFT of the real-valued $x(n)$. Of course, the prudent engineer would conduct a literature search to see what algorithms are available for efficiently performing inverse FFTs when many of the frequency-domain samples are zeros.

13.28.2 Computing Interpolated Analytic Signals

We can use the frequency-domain zero-stuffing scheme to generate an interpolated-by- M analytic (complex-valued) time signal based upon the real N -point time sequence $x(n)$, if N is even[75]. The process is as follows:

- Perform an N -point FFT on an N -point real $x_r(n)$ time sequence, yielding N frequency samples, $X_r(m)$.
- Create an MN -point spectral sequence $X_{\text{int}}(m)$ initially set to all zeros, where MN is an integer power of two.
- Assign $X_{\text{int}}(0) = X_r(0)$, and $X_{\text{int}}(N/2) = X_r(N/2)$.
- Assign $X_{\text{int}}(m) = 2X_r(m)$, for $1 \leq m \leq (N/2)-1$.
- Compute the MN -point inverse FFT of $X_{\text{int}}(m)$, yielding the desired MN -length interpolated analytic (complex) $x_{c,\text{int}}(n)$ sequence.
- Finally, if desired, multiply $x_{c,\text{int}}(n)$ by M to compensate for the $1/M$ amplitude loss induced by interpolation.

To minimize the interpolation error in the complex $x_{c,\text{int}}(n)$ sequence, the original $x_r(n)$ sequence must have negligible spectral energy in the vicinity of $\pm f_s/2$, as described earlier for real-valued interpolation.

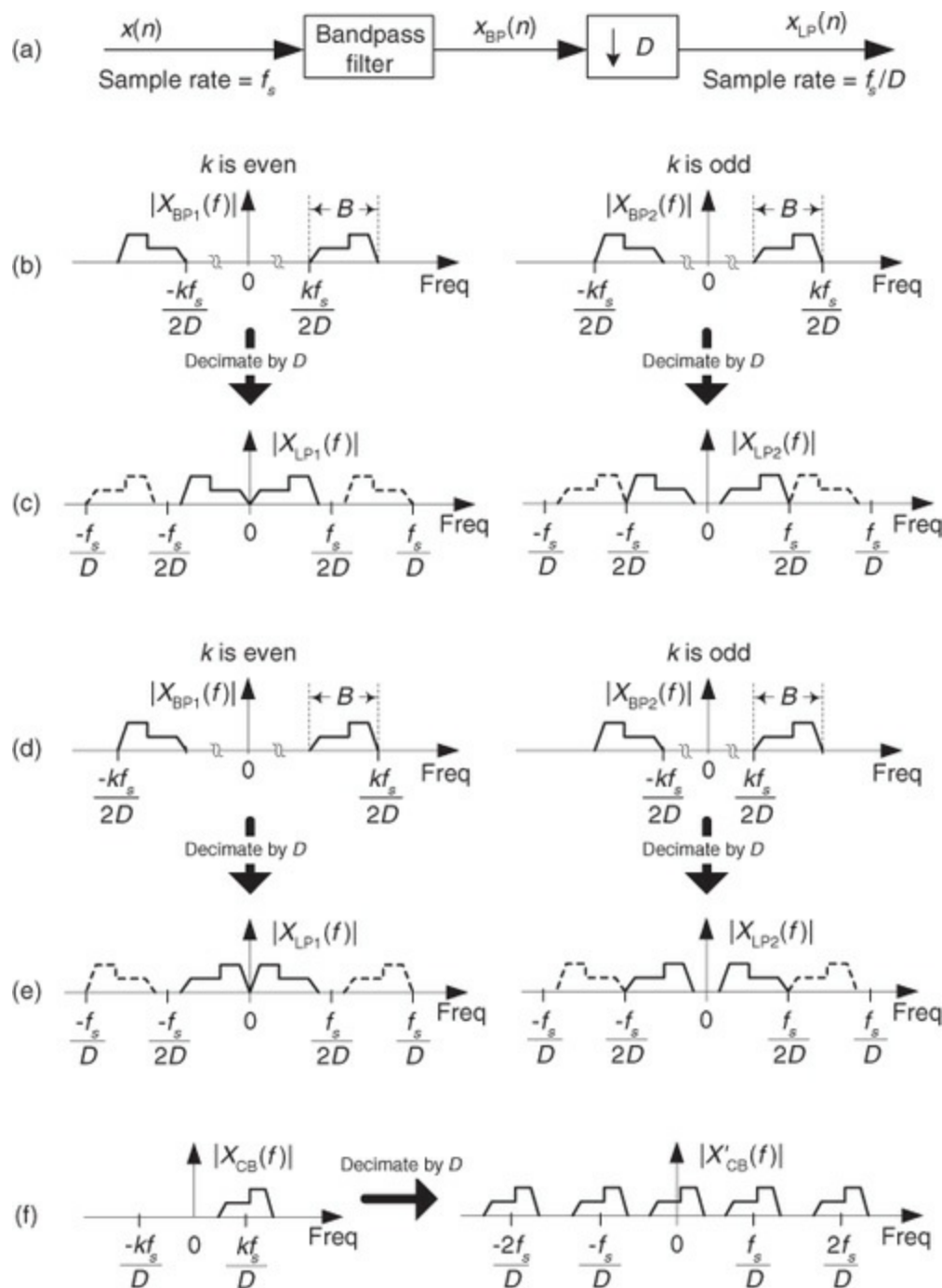
13.29 Frequency Translation Using Decimation

In this section we show tricks for implementing multiplierless frequency translation of both real and complex signals using simple decimation.

13.29.1 Translation of Real Signals Using Decimation

We can frequency translate a real bandpass signal toward zero Hz, converting it to a lowpass signal, without the need for mixing multipliers. We do this by performing decimation by an integer factor D as shown in [Figure 13-75\(a\)](#). If the bandpass filter provides an output signal of bandwidth B Hz, located as shown in [Figures 13-75\(b\)](#) and [13-75\(d\)](#) where k is a positive integer, decimation by D will yield lowpass signals whose spectra are shown in [Figures 13-75\(c\)](#) and [13-75\(e\)](#), depending on whether integer k is odd or even. Take care to notice the inverted spectra in [Figure 13-75\(e\)](#). To avoid decimated-output aliasing errors, we must satisfy the Nyquist criterion and ensure that $x_{\text{BP}}(n)$'s bandwidth B is not greater than $f_s/(2D)$.

Figure 13-75 Real and complex bandpass signal translation using decimation by D .



13.29.2 Translation of Complex Signals Using Decimation

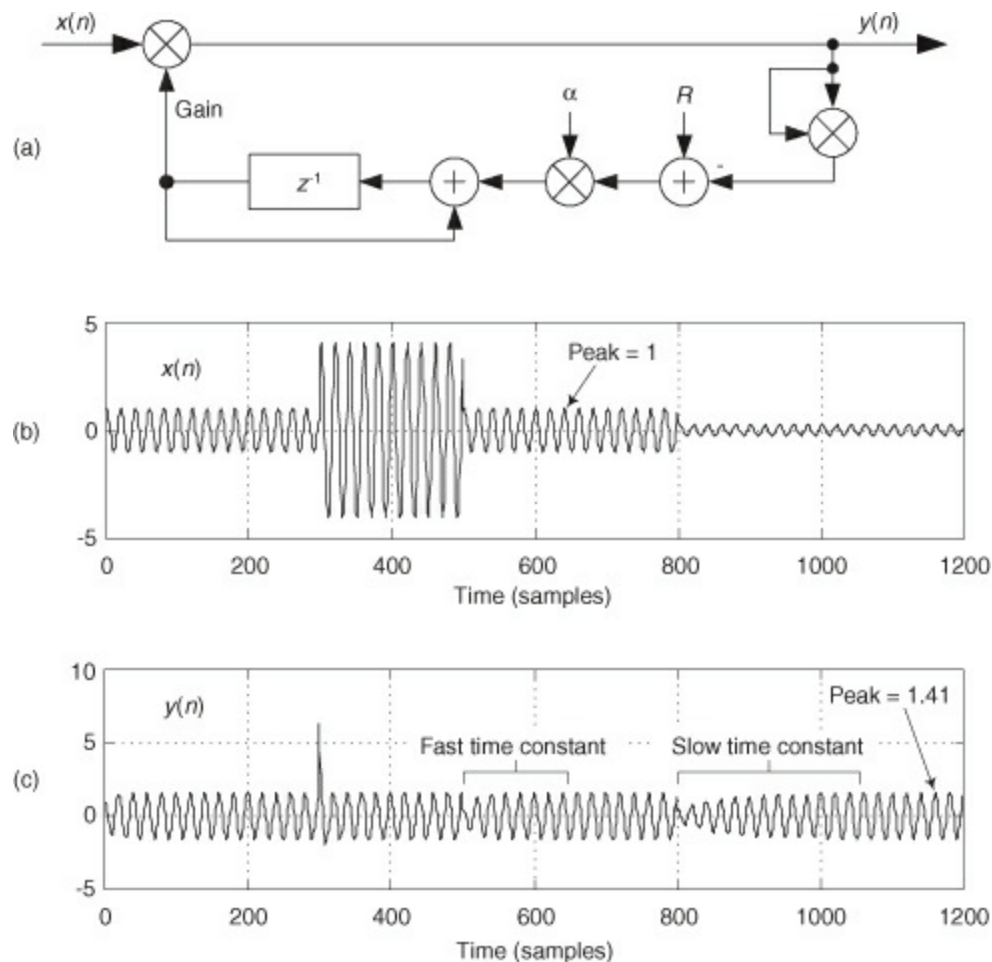
It's possible to frequency translate a complex bandpass signal, without the need for mixing multipliers, so that a spectral replication is centered at zero Hz. The process we're describing here is called *complex down-conversion*. The left side of [Figure 13-75\(f\)](#) shows the spectrum of a complex baseband signal whose $|X_{CB}(m)|$ spectral magnitude contains only positive-frequency spectral components.

If we individually decimate the real and imaginary parts of the complex time sequence $x_{CB}(n)$, whose spectrum is $X_{CB}(m)$, by D , the resulting complex sequence will have a spectral image centered exactly at zero Hz as shown by $|X'_{CB}(m)|$ in [Figure 13-75\(f\)](#). The key stipulation here, as you may have guessed, is that the original pre-decimated $|X_{CB}(m)|$ spectral energy must be centered at an integer multiple of f_s/D .

13.30 Automatic Gain Control (AGC)

Since the early days of vacuum tube radios, circuits were needed to automatically adjust a receiver's gain, as an input signal varied in amplitude, to maintain a (relatively) constant output signal level. These feedback mechanisms, called *automatic gain control* (AGC) circuits, are an important component of modern analog and digital communications receivers. [Figure 13-76\(a\)](#) illustrates a simple digital AGC process[76,77]. Its operation is straightforward: The output signal power is sampled and compared to a reference level R (the desired output amplitude rms level). If the output signal level is too high (low), a negative (positive) signal is fed back, reducing (increasing) the gain. The control parameter α regulates the amplitude of the feedback signal and is used to control the AGC's time constant (how rapidly gain changes take effect).

Figure 13-76 AGC process: (a) linear AGC circuit; (b) example input $x(n)$ with amplitude fluctuations; (c) $y(n)$ output for $\alpha = 0.01$ and $R = 1$.



Given an input signal $x(n)$ in [Figure 13-76\(b\)](#) whose amplitude envelope is fluctuating, the AGC structure provides the relatively constant amplitude $y(n)$ output shown in [Figure 13-76\(c\)](#).

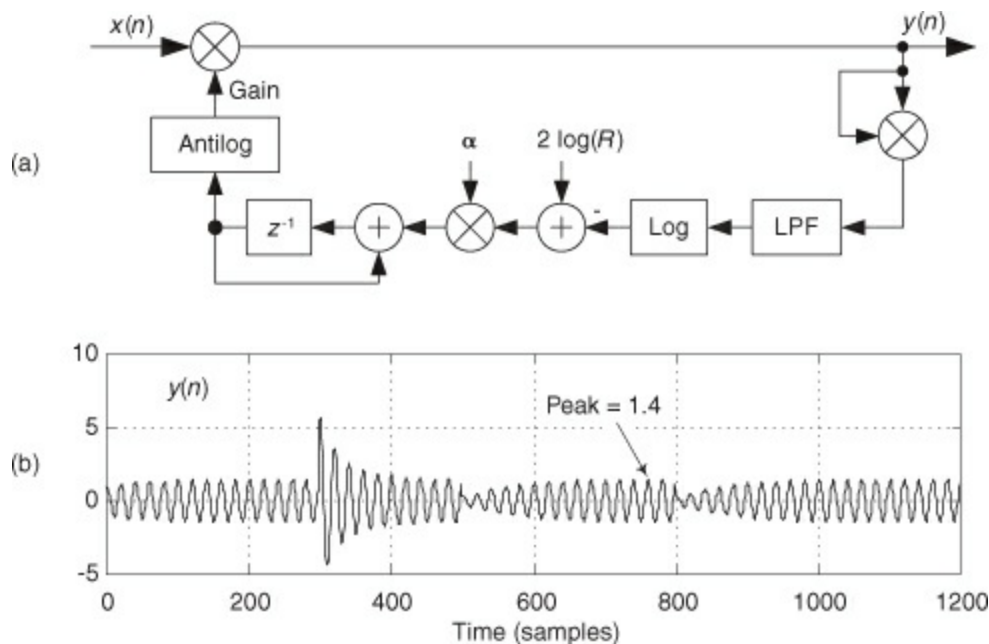
We called [Figure 13-76\(a\)](#) a "simple AGC process," but AGC is not all that simple. The process is a nonlinear, time-varying, signal-dependent feedback system. As such, it's highly resistant to normal time-domain or z -domain analysis. This is why AGC analysis is empirical rather than mathematical and explains why there's so little discussion of AGC in the DSP literature.

Depending on the nature of $x(n)$, the feedback signal may fluctuate rapidly and the feedback loop will attempt to adjust the system gain too often. This will cause a mild AM modulation effect, inducing low-level harmonics in the $y(n)$ output. That problem can be minimized by inserting a simple lowpass filter in the feedback loop just before, or just after, the R adder. But such filtering does not remedy the circuit's main drawback. The time constant (attack time) of this AGC scheme is input signal level dependent and is different depending on whether the $x(n)$ is increasing or decreasing. These

properties drastically reduce our desired control over the system's time constant. To solve this problem, we follow the lead of venerable radio AGC designs and enter the logarithmic domain.

We can obtain complete control of the AGC's time constant, and increase our AGC's dynamic range, by using logarithms as shown in [Figure 13-77\(a\)](#). As is typical in practice, this log AGC process has a lowpass filter (LPF) to eliminate too-rapid gain changes[\[78\]](#). That filter can be a simple moving average filter, a cascaded integrator-comb (CIC) filter, or a more traditional lowpass filter having a $\sin(x)/x$ impulse response.

Figure 13-77 AGC process: (a) logarithmic AGC circuit; (b) $y(n)$ output for $\alpha = 0.01$ and $R = 1$.



For the logarithmic AGC scheme, the feedback loop's time constant is dependent solely on α and independent of the input signal level, as can be seen in [Figure 13-77\(b\)](#) when the $x(n)$ input is that in [Figure 13-76\(b\)](#). The Log and Antilog operations can be implemented as $\log_2(x)$ and 2^x , respectively.

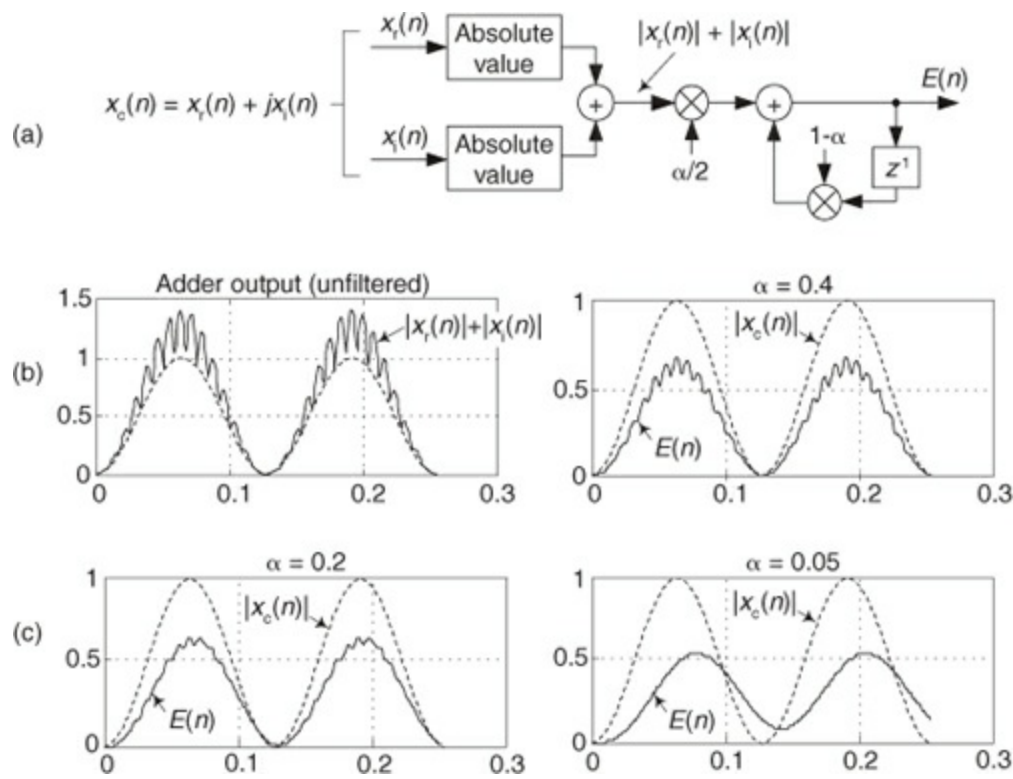
13.31 Approximate Envelope Detection

In this section, we present a crude (but simple to implement) complex signal envelope detection scheme. By “envelope detection” we mean estimating the instantaneous magnitude of a complex signal $x_c(n)$. The process is straightforward: we sum the absolute values of a complex signal's real and imaginary parts and apply that sum to a simple 1st-order lowpass IIR filter to obtain an envelope signal $E(n)$ as shown in [Figure 13-78\(a\)](#). The filter's feedback coefficient α is in the range of 0 to 1. (That lowpass filter is our exponential averager discussed in [Section 11.6](#), which some DSP folks call a *leaky integrator*.) The $E(n)$ sequence is proportional to the desired instantaneous magnitude of $x_c(n)$, or

$$(13-133)$$

$$E(n) \approx K |x_c(n)| = K \sqrt{x_r(n)^2 + x_i(n)^2}.$$

Figure 13-78 Envelope detection: (a) block diagram; (b) $|x_r(n)| + |x_i(n)|$ adder output, and $E(n)$ for $\alpha = 0.4$; (c) $E(n)$ for $\alpha = 0.2$ and $\alpha = 0.05$.



To gauge the envelope detector's performance, consider a sampled version of an amplitude-modulated sinusoid such as the $x_r(n)$ in [Figure 9-7\(a\)](#) from which a sampled analytic (complex) $x_c(n)$ can be generated. If $x_c(n)$ is applied to our envelope detection process, the processing results are shown in [Figures 13-78\(b\)](#) and [13-78\(c\)](#), where the solid curves represent $E(n)$ and the dashed curves are the true magnitude of $x_c(n)$. Notice how the amount of smoothing of the $E(n)$ fluctuations depends on the value of α .

If the scaling coefficient $\alpha/2$ can take the form

$$(13-133')$$

$$\frac{\alpha}{2} = \frac{2^K - 1}{2 \cdot 2^K} = \frac{1}{2} - \frac{1}{2^{K+1}}$$

where K is a positive integer, then we can eliminate the multipliers in [Figure 13-78\(a\)](#). If we satisfy [Eq. \(13-133'\)](#), the multiply by $\alpha/2$ can be replaced by two binary right shifts and a subtract operation, and the multiply by $(1-\alpha)$ can be replaced by a single binary right-shift operation. This situation gives us a multiplierless envelope detector.

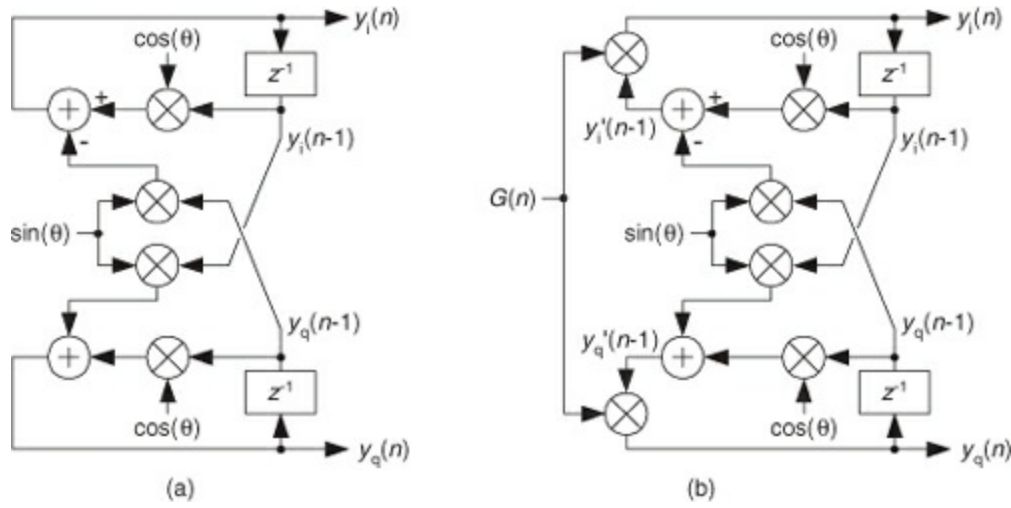
Sequence $x_r(n)$ must be used to generate a complex analytic $x_c(n)$ sequence (using one of the methods discussed in [Sections 9.4](#) and [9.5](#)) upon which this envelope detector scheme can be applied. The advantage of this envelope detection process is that, of course, no squaring or square root computations in [Eq. \(13-133\)](#), nor the $|x_r(n)|$ and $|x_i(n)|$ comparisons in the vector magnitude approximations in [Section 13.2](#), need be performed.

Whether this envelope approximation technique yields sufficiently accurate results is for the user to decide. Its accuracy may be below the requirements of most AM (amplitude modulation) detection requirements, but the process may well be useful for estimating signal magnitude in automatic gain control (AGC) or energy detection applications.

13.32 A Quadrature Oscillator

Here we present a well-behaved digital quadrature oscillator, whose output is $y_i(n) + jy_q(n)$, having the structure shown in [Figure 13-79\(a\)](#). If you're new to digital oscillators, that structure looks a little complicated but it's really not so bad. If you look carefully, you see the computations are

Figure 13-79 Quadrature oscillators: (a) standard structure; (b) structure with AGC.



$$(13-134)$$

$$y_i(n) = y_i(n-1)\cos(\theta) - y_q(n-1)\sin(\theta)$$

and

$$(13-134')$$

$$y_q(n) = y_i(n-1)\sin(\theta) + y_q(n-1)\cos(\theta).$$

Those computations are merely the rectangular form of multiplying the previous complex output by a complex exponential $e^{j\theta}$ as

$$(13-135)$$

$$\begin{aligned} y_i(n) + jy_q(n) &= [y_i(n-1) + jy_q(n-1)][\cos(\theta) + j\sin(\theta)] \\ &= [y_i(n-1) + jy_q(n-1)]e^{j\theta}. \end{aligned}$$

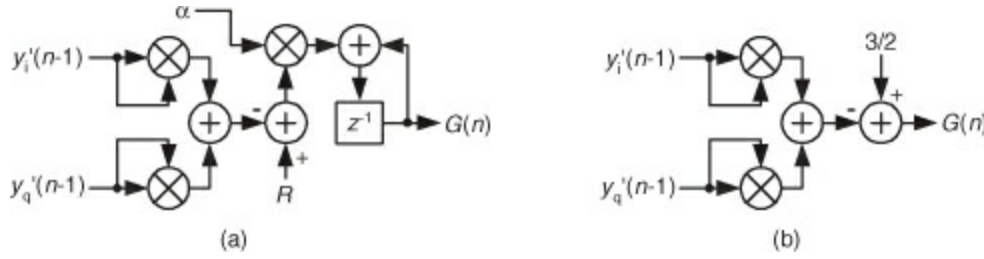
So the theory of operation is simple. Each new complex output sample is the previous output sample rotated by θ radians, where θ is $2\pi f_t/f_s$ with f_t and f_s being the oscillator tuning frequency and the sample rate, respectively, in Hz.

To start the oscillator, we set the initial conditions of $y_i(n-1) = 1$ and $y_q(n-1) = 0$ and repeatedly compute new outputs, as time index n advances, using [Eq. \(13-134\)](#). This oscillator is called a *coupled quadrature oscillator* because both of its previous outputs are used to compute each new in-phase and each new quadrature output. It's a useful oscillator because the full range of tuning frequencies is available (from nearly zero Hz up to roughly $f_s/2$), and its outputs are equal in amplitude, unlike some other quadrature oscillator structures[\[79\]](#). The tough part, however, is making this oscillator stable in fixed-point arithmetic implementations.

Depending on the binary word widths, and the value θ , the output amplitudes can either grow or decay as time increases because it's not possible to represent $e^{j\theta}$ having a magnitude of exactly one, over the full range of θ , using fixed-point number formats. The solution to amplitude variations is to compute $y_i'(n-1)$ and $y_q'(n-1)$ and multiply those samples by an instantaneous gain factor $G(n)$ as shown in [Figure 13-79\(b\)](#). The trick here is how to compute the gain samples $G(n)$.

We can use a linear automatic gain control (AGC) method, described in [Section 13.30](#), as shown in [Figure 13-80\(a\)](#) where α is a small value, say, $\alpha = 0.01$. The value R is the desired rms value of the oscillator outputs. This AGC method greatly enhances the stability of our oscillator. However, there's a computationally simpler AGC scheme for our oscillator that can be developed using the *Taylor series approximation* we learned in school. Here's how.

Figure 13-80 AGC schemes: (a) linear AGC; (b) simplified AGC.



Using an approach similar to reference [\[80\]](#), we can define the desired gain as

$$(13-136)$$

$$G(n) = \frac{M_{\text{des}}}{M_{\text{act}}}$$

This is the desired output signal magnitude M_{des} over the actual output magnitude M_{act} . We can also represent the gain using power as

$$(13-137)$$

$$G(n) = \frac{\sqrt{P_{\text{des}}}}{\sqrt{P_{\text{act}}}} = \frac{\sqrt{P_{\text{des}}}}{\sqrt{P_{\text{des}} + E}}$$

where the constant P_{des} is the desired output signal power and P_{act} is the actual output power. The right side of [Eq. \(13-137\)](#) shows P_{act} replaced by the desired power P_{des} plus an error component E , and that's the ratio we'll compute. To avoid square root computations and because the error E will be small, we'll approximate that ratio with a two-term Taylor series expansion about $E = 0$ using

$$(13-138)$$

$$G(n) \approx a_0 + a_1(E).$$

Computing the Taylor series' coefficients to be $a_0 = 1$ and $a_1 = -1/2P_{\text{des}}$, and recalling that $E = P_{\text{act}} - P_{\text{des}}$, we estimate the instantaneous gain as

$$(13-139)$$

$$G(n) \approx 1 - \frac{1}{2P_{\text{des}}} (P_{\text{act}} - P_{\text{des}}) = \frac{3}{2} - \frac{P_{\text{act}}}{2P_{\text{des}}} = \frac{3}{2} - \frac{y_i'(n-1)^2 + y_q'(n-1)^2}{2P_{\text{des}}}.$$

If we let the quadrature output peak amplitudes equal $1/\sqrt{2}$, P_{des} equals $1/2$ and we eliminate the division in [Eq. \(13-139\)](#), obtaining

$$(13-140)$$

$$G(n) \approx \frac{3}{2} - [y_i'(n-1)^2 + y_q'(n-1)^2].$$

The simplified structure of the $G(n)$ computation is shown in [Figure 13-80\(b\)](#).

As for practical issues, to avoid gain values greater than one (for those fixed-point fractional number systems that don't allow numbers ≥ 1), we use the clever recommendation from reference [\[79\]](#) of

multiplying by $G(n)/2$ and doubling the products in [Figure 13-79\(b\)](#). Reference [\[80\]](#) recommends using rounding, instead of truncation, for all intermediate computations to improve output spectral purity. Rounding also provides a slight improvement in tuning frequency control. Because this oscillator is guaranteed stable, and can be dynamically tuned, it's definitely worth considering for real-valued as well as quadrature oscillator applications[\[79\]](#).

13.33 Specialized Exponential Averaging

In [Chapter 11](#) we discussed the behavior and utility of using an exponential averaging lowpass filter, also called a *leaky integrator*, to reduce noise fluctuations that contaminate constant-amplitude signal measurements. In this section we present three specialized exponential averaging techniques in the form of

- single-multiply averaging,
- multiplier-free averaging, and
- dual-mode averaging.

13.33.1 Single-Multiply Exponential Averaging

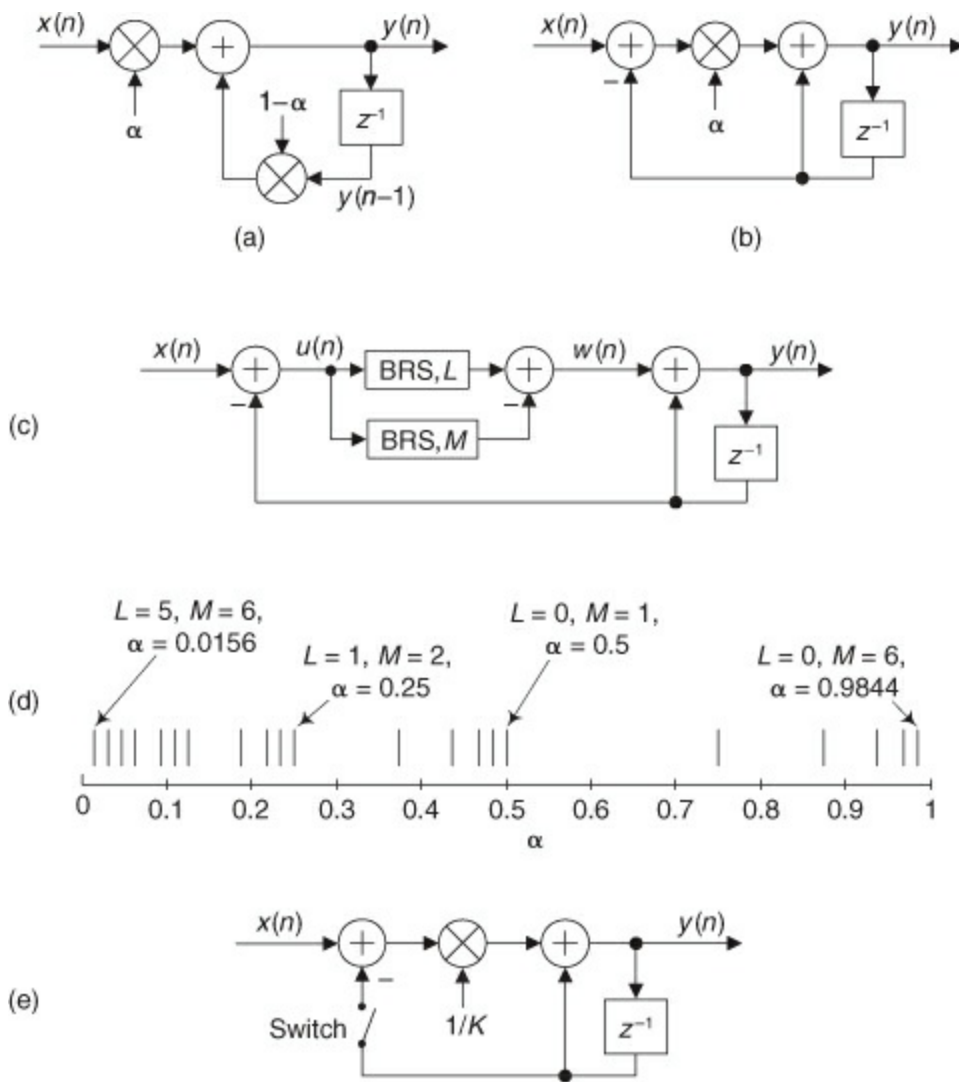
This DSP trick shows how to reduce the computational workload of the standard exponential averager[\[81\]](#). An exponential averager's difference equation is

$$(13-141)$$

$$y(n) = \alpha x(n) + (1 - \alpha)y(n-1)$$

where α is a constant called the averager's *weighting factor*, in the range $0 < \alpha < 1$. The process requires two multiplies per $y(n)$ output sample as shown in [Figure 13-81\(a\)](#).

Figure 13-81 Exponential averaging: (a) standard network; (b) single-multiply network; (c) multiplierless network; (d) possible values for α ; (e) dual-mode averaging.



We can rearrange [Eq. \(13-141\)](#) to the form

$$(13-141')$$

$$y(n) = y(n-1) + \alpha[x(n) - y(n-1)],$$

which eliminates one of the averager's multiplies, at the expense of an additional adder, giving us a single-multiply exponential averager shown in [Figure 13-81\(b\)](#). This neat single-multiply exponential averager maintains the DC (zero Hz) gain of unity exhibited by the traditional two-multiply exponential averager in [Figure 13-81\(a\)](#).

13.33.2 Multiplier-Free Exponential Averaging

It is possible to eliminate the multiplier in [Figure 13-81\(b\)](#) if we place restrictions on the permissible values of α . For example, if $\alpha = 0.125 = 1/8$, then the output of the multiplier is merely the multiplier's input sample shifted right by three bits.

On the other hand, if α takes the form

$$(13-142)$$

$$\alpha = \frac{1}{2^L} - \frac{1}{2^M}$$

where $L = 0, 1, 2, 3, \dots$, and $M = 1, 2, 3, \dots$, we can replace the multiplication by α in [Figure 13-81\(b\)](#) with two binary right shifts and a subtract operation as shown in [Figure 13-81\(c\)](#). In that figure the "BRS,L" block means an arithmetic, or hardwired, Binary Right Shift by L bits.

For example, if $L = 2$ and $M = 5$, then from [Eq. \(13-142\)](#), $\alpha = 0.2188$. The sequence $w(n) =$

$0.2188u(n) = (1/4 - 1/32)u(n)$ is computed by subtracting $u(n)$ shifted right by $M = 5$ bits from $u(n)$ shifted right by $L = 2$ bits.

The tick marks in [Figure 13-81\(d\)](#) show the possible values for the weighting factor α over the range of $0 \leq L \leq 5$, where for each L , M is in the range $L+1 \leq M \leq 6$ in [Eq. \(13-142\)](#). That figure tells us that we have a reasonable selection of α values for our noise-reduction filtering applications.

The point is, for fixed-point implementation of exponential averaging, check to see if your desired α weighting factor can be represented by the difference of various reciprocals of integer powers of two. If so, then binary word shifting enables us to implement a multiplierless exponential averager.

13.33.3 Dual-Mode Averaging

Here's a clever exponential averaging scheme that blends both the quick time response of a moving averager and the noise-reduction control of an exponential averager.[‡] The structure of this dual-mode averager is depicted in [Figure 13-81\(e\)](#). The averager operates as follows: The switch remains open for K input samples after which the $y(n)$ output is equal to the K -point average of the $x(n)$ input. Just prior to the arrival of the $K+1$ input sample the switch closes, converting the moving average filter to an exponential averager, giving us control over the filter's noise-reduction properties as described in [Section 11.6](#).

[‡] We thank DSP guru Fred Harris for recommending this dual-mode averager.

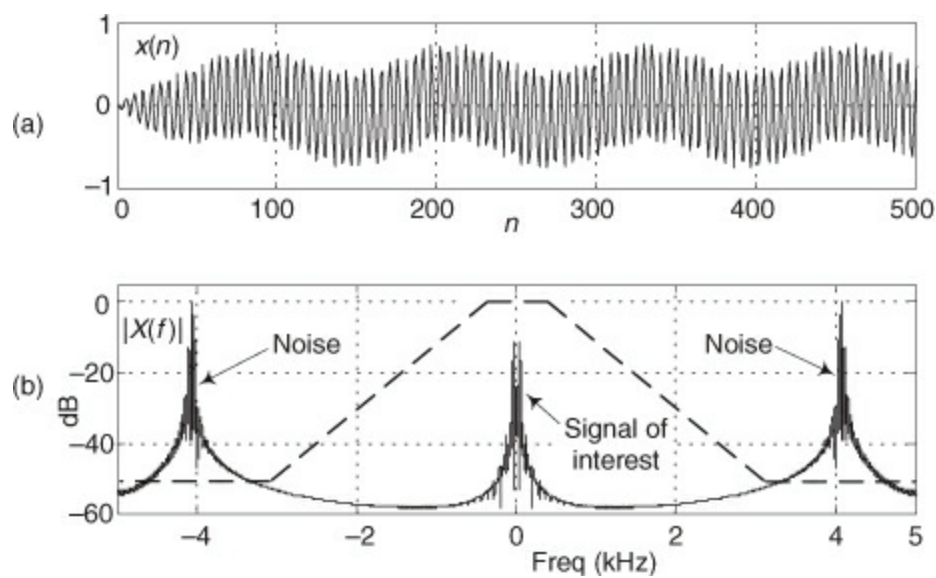
Of course, K does not have to be an integer. In this case we can still implement dual-mode averaging by closing the switch just prior to the arrival of the $x(\lfloor K \rfloor)$ input sample, where $\lfloor K \rfloor$ means the integer part of K . After the K th input sample has arrived, the averager's gain at zero Hz (DC gain) is unity. As discussed in the previous section, if the weighting factor $1/K$ can be represented by the difference of various reciprocals of integer powers of two, then we can implement a multiplierless dual-mode noise-reduction filter.

13.34 Filtering Narrowband Noise Using Filter Nulls

Here we present two filter design tricks that take advantage of the frequency-domain magnitude nulls of simple FIR filters. These schemes are particularly useful when used in AM and FM demodulation systems.

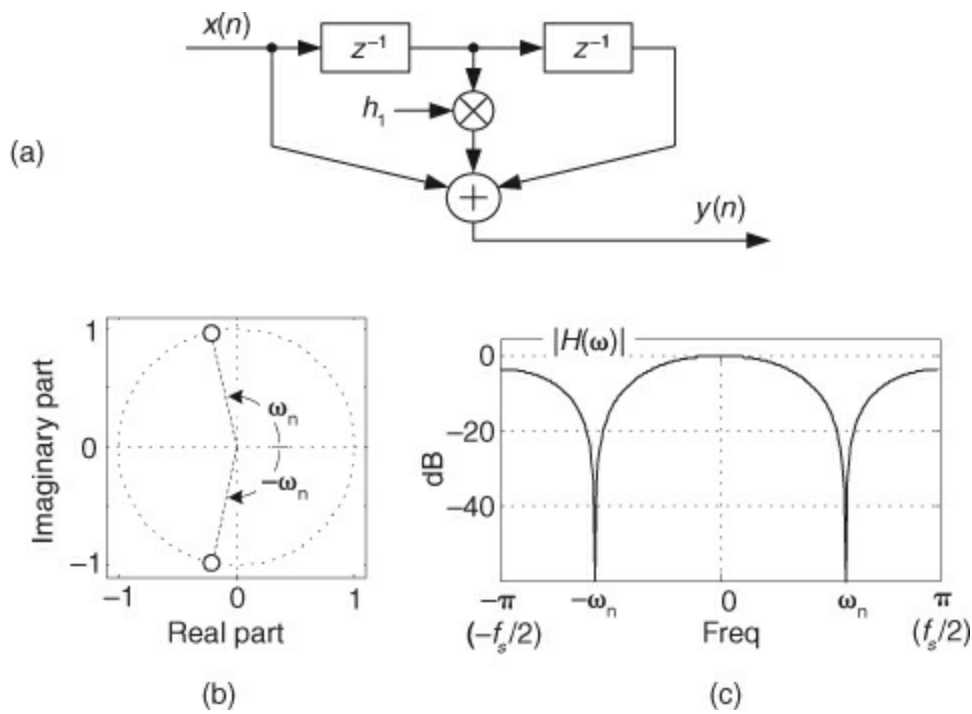
The first example uses a simple 3-tap nonrecursive FIR filter. Such a filter can be both computationally efficient, and useful, for narrowband-noise reduction. Here's how. Think about the $x(n)$ time-domain signal, contaminated with high-frequency noise, shown in [Figure 13-82\(a\)](#) with its spectrum provided in [Figure 13-82\(b\)](#). The sample rate of the signal is 10 kHz. Let's assume we want to recover the low-frequency signal of interest (centered at zero Hz) without inducing phase distortion, and we need to attenuate the narrowband high-frequency noise, centered at 4.1 kHz, by at least 50 dB. Our solution, of course, is to pass our noisy signal through a linear-phase lowpass FIR filter whose frequency magnitude response is indicated as the dashed curve in [Figure 13-82\(b\)](#).

Figure 13-82 A noisy $x(n)$: (a) time signal; (b) its $X(f)$ spectral magnitude.



Seeking the most computationally efficient filter possible, let's say we're clever and recall the special characteristic of a half-band FIR filter in which roughly half its coefficients are zero-valued. So we could design a 9-tap half-band FIR filter, having only five nonzero-valued coefficients, and that solution would be acceptable. Here's where our *trick* comes in; we decide to use the linear-phase 3-tap FIR filter shown in [Figure 13-83\(a\)](#) with its single non-unity coefficient h_1 .

Figure 13-83 A 3-tap FIR filter: (a) filter structure; (b) pole locations; (c) frequency magnitude response.



If $|h_1| \leq 2$, the 3-tap FIR filter's transfer function will have two zeros on the z -plane at angles $\pm\omega_n$ as shown in [Figure 13-83\(b\)](#). The frequency magnitude response of the filter is shown in [Figure 13-83\(c\)](#). (Here, the normalized frequency axis value of π corresponds to a continuous-time frequency of half the sample rate, $f_s/2$.) Our goal, then, is to choose the h_1 coefficient such that the filter's positive-frequency magnitude null lands right on the 4.1 kHz center frequency of the narrowband noise in [Figure 13-82\(b\)](#).

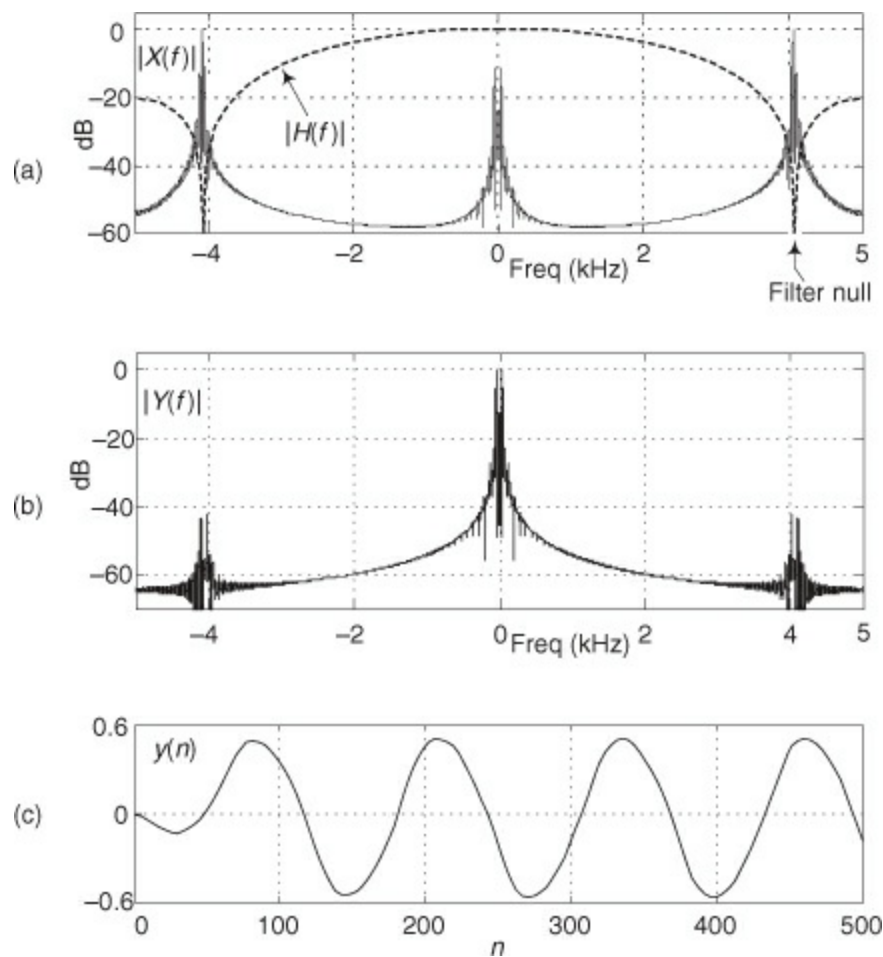
Our 3-tap filter design problem is easy because we have an expression for the h_1 coefficient as a function of the desired f_n null frequency in Hz. The h_1 coefficient value is

$$(13-143)$$

$$h_1 = -2\cos(2\pi f_n/f_s).$$

With $f_n = 4.1$ kHz and $f_s = 10$ kHz, our h_1 coefficient is 1.69. (The derivation of [Eq. \(13-143\)](#) was left as a homework problem in [Chapter 5](#).) The $H(f)$ frequency magnitude response of the $h_1 = 1.69$ filter is shown as the dotted curve in [Figure 13-84\(a\)](#). The $Y(f)$ spectrum of the filter's output is shown in [Figure 13-84\(b\)](#) where the narrowband noise has been attenuated by roughly 54 dB. (Recall that the noise magnitudes in the original $X(f)$ spectrum were approximately 12 dB above the signal's peak magnitude in [Figure 13-82\(b\)](#).) The filter's time-domain $y(n)$ output signal, our *signal of interest*, is shown in [Figure 13-84\(c\)](#). It's instructive to compare that output signal to the filter's $x(n)$ input signal in [Figure 13-82\(a\)](#).

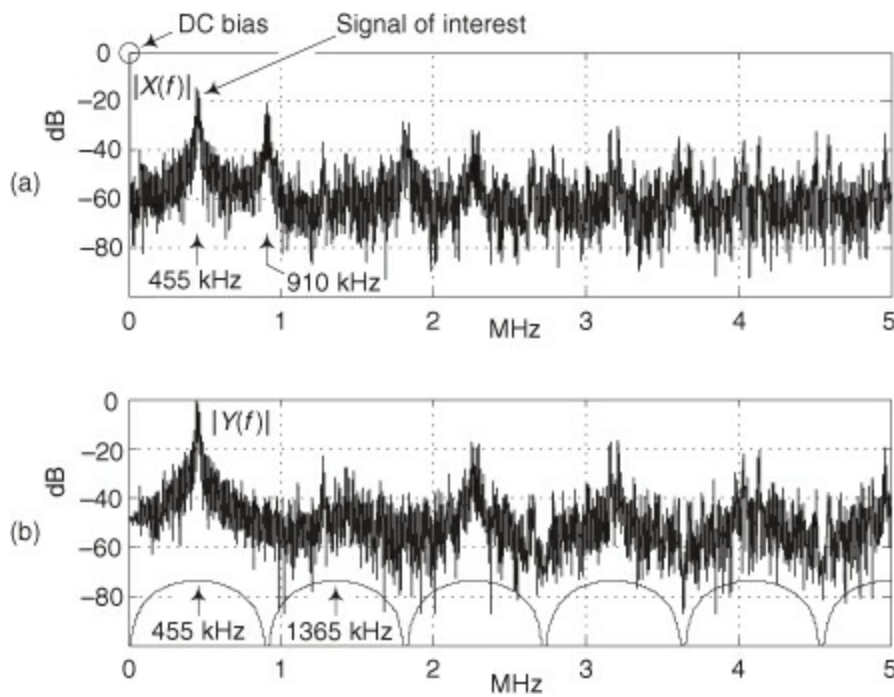
Figure 13-84 Three-tap filter performance: (a) $|H(f)|$ response; (b) filter output spectrum; (c) filter time-domain output signal.



So we solved our narrowband noise filtering problem with a linear-phase FIR filter requiring only two additions and one multiply per filter output sample. Neat, huh?

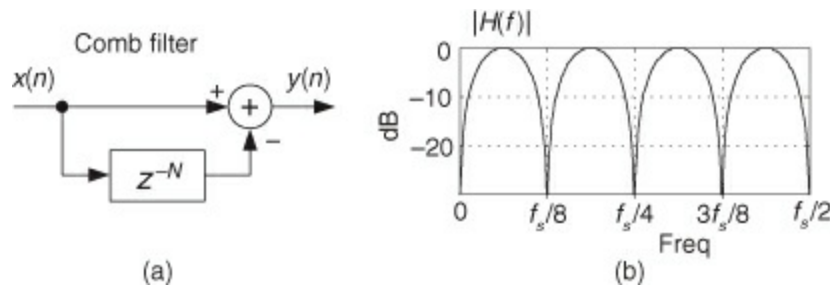
Our second example of this filter design approach that takes advantage of the frequency-domain magnitude nulls of simple FIR filters involves the attenuation of multiple narrowband noise spectral components whose center frequencies are harmonically related in a frequency shift keying (FSK) demodulation system[82]. Think about a signal of interest centered at 455 kHz as shown in [Figure 13-85\(a\)](#). That signal, sampled at $f_s = 10$ MHz, is contaminated with unwanted high-amplitude DC (zero Hz) bias noise and narrowband spectral noise components at multiples of 455 kHz. Removing the DC bias, whose magnitude is 0 dB in [Figure 13-85\(a\)](#), and extracting the signal of interest from the noise appears to require some sort of bandpass filter centered at 455 kHz.

Figure 13-85 Harmonic noise example: (a) $|X(f)|$ spectrum; (b) filter output spectrum.



However, the trick is to use a standard FIR comb filter to remove the unwanted DC bias and attenuate the harmonic noise components. A comb filter is shown in [Figure 13-86\(a\)](#) where the z^{-N} operation is merely a delay of N samples. The $|H(f)|$ frequency magnitude response of an $N = 8$, for example, comb filter is provided in [Figure 13-86\(b\)](#) where f_s is the sample rate.

Figure 13-86 Standard N -delay FIR comb filter: (a) filter structure; (b) frequency magnitude response when $N = 8$.



For N -delay comb filter design purposes, the following two equations give us the frequency locations of the magnitude nulls (f_{null}) and magnitude peaks (f_{peak}) in the filter's $|H(f)|$,

$$(13-144)$$

$$f_{\text{null}} = \frac{kf_s}{N}, \quad \text{where } k = 0, \pm 1, \pm 2, \dots, \pm \left\lfloor \frac{N}{2} \right\rfloor$$

$$(13-144')$$

$$f_{\text{peak}} = \frac{(2k+1)f_s}{2N}, \quad \text{where } k = 0, \pm 1, \pm 2, \dots, \pm \left\lfloor \frac{N-1}{2} \right\rfloor,$$

where $\lfloor X \rfloor$ means the integer part of X . These f_{null} and f_{peak} expressions are valid for both odd and even N so long as N is larger than one.

For this noise-reduction problem, we need a comb filter that provides a magnitude null at zero Hz and a magnitude peak at 455 kHz. Rearranging [Eq. \(13-144'\)](#) to find a candidate value for the comb delay N for $k = 0$, we have

$$(13-145)$$

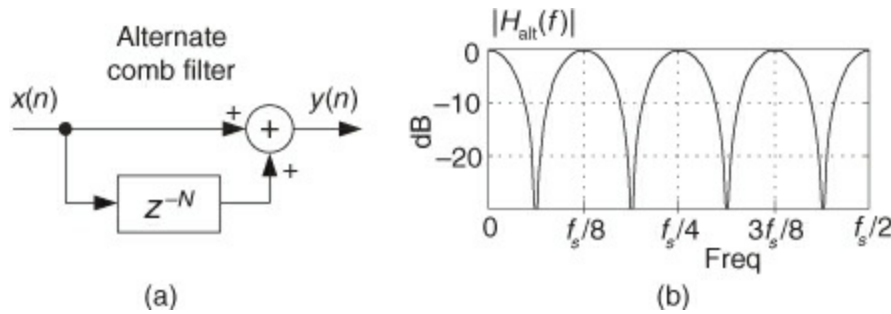
$$N = \frac{f_s}{2f_{\text{peak}}} = \frac{10^6}{2(455 \cdot 10^3)} = 10.99,$$

so we select N to be 11. The filter's output spectrum, when $N = 11$, is shown in [Figure 13-85\(b\)](#). There we see the dramatic reduction in the unwanted DC bias as well as the narrowband noise located at 910 kHz. (For reference purposes, we've included the $N = 11$ comb filter's magnitude response in [Figure 13-85\(b\)](#).)

So in this example we simplified our overall narrowband noise filtering problem using a linear-phase FIR comb filter requiring only one addition per filter output sample. In practice the comb filter is followed by a low-order lowpass filter, whose stopband would begin in the vicinity of 1365 kHz. That follow-on filter will have a significantly reduced computational workload compared to the case where the pre-filtering by the simple comb filter was not performed.

For completeness, we must mention here that an *alternate* comb filter can be built using the network in [Figure 13-87\(a\)](#) where addition is performed as opposed to the subtraction in [Figure 13-86\(a\)](#).

Figure 13-87 Alternate N -delay FIR comb filter: (a) filter structure; (b) $|H_{\text{alt}}(f)|$ frequency magnitude response when $N = 8$.



For the alternate comb filter in [Figure 13-87\(a\)](#) the following two equations give us the frequency locations of the magnitude nulls ($f_{\text{null,alt}}$) and magnitude peaks ($f_{\text{peak,alt}}$) in this N -delay comb filter's $|H_{\text{alt}}(f)|$,

$$(13-146)$$

$$f_{\text{null,alt}} = \frac{(2k+1)f_s}{2N}, \text{ where } k = 0, \pm 1, \pm 2, \dots, \pm \left\lfloor \frac{N-1}{2} \right\rfloor$$

$$(13-146')$$

$$f_{\text{peak,alt}} = \frac{kf_s}{N}, \text{ where } k = 0, \pm 1, \pm 2, \dots, \pm \left\lfloor \frac{N}{2} \right\rfloor$$

where $\lfloor X \rfloor$ means the integer part of X . This alternate comb filter gives us a bit of design flexibility because it passes low-frequency signals due to a frequency magnitude peak at zero Hz (DC).

13.35 Efficient Computation of Signal Variance

In this section we show how to reduce the computational workload, and required data storage, in computing the unbiased and biased variances of a signal sequence. (Definitions of *biased* and *unbiased variances* can be found in [Appendix D](#).)

We start our discussion with the traditional definition of the unbiased variance of $x(n)$, a sequence of N samples, written as

$$(13-147)$$

$$Var_{\text{unbiased}} = \frac{\sum_{n=1}^N [x(n) - x_{\text{ave}}]^2}{N-1}$$

where x_{ave} is the average of the N -length $x(n)$ sequence. Because N is a constant, we can treat the divide by N needed to compute x_{ave} , and the above divide by $(N-1)$, as multiplies by reciprocals, allowing us to say that [Eq. \(13-147\)](#) requires $3N-2$ addition and $N+2$ multiply operations. As it turns out, we can obtain an equivalent expression for Var_{unbiased} that has a reduced number of arithmetic operations[\[83\]](#). Here's how.

First, we square the bracketed term in the summation in [Eq. \(13-147\)](#) and write

(13-148)

$$\begin{aligned} Var_{\text{unbiased}} &= \frac{\sum_{n=1}^N [x(n)^2 - 2x(n)x_{\text{ave}} + x_{\text{ave}}^2]}{N-1} \\ &= \frac{\sum_{n=1}^N x(n)^2 - \sum_{n=1}^N 2x(n)x_{\text{ave}} + \sum_{n=1}^N x_{\text{ave}}^2}{N-1}. \end{aligned}$$

Because the center summation in [Eq. \(13-148\)](#) is

$$-\sum 2x(n)x_{\text{ave}} = -2x_{\text{ave}}\sum x(n) = -2x_{\text{ave}}Nx_{\text{ave}} = -2Nx_{\text{ave}}^2$$

we can rewrite Var_{unbiased} as

(13-148')

$$\begin{aligned} Var_{\text{unbiased}} &= \frac{\sum_{n=1}^N [x(n)^2] - 2Nx_{\text{ave}}^2 + \sum_{n=1}^N [x_{\text{ave}}^2]}{N-1} \\ &= \frac{\sum_{n=1}^N [x(n)^2] - 2Nx_{\text{ave}}^2 + Nx_{\text{ave}}^2}{N-1}. \end{aligned}$$

Next, we arrive at our desired expression by combining terms and write Var_{unbiased} as

(13-149)

$$Var_{\text{unbiased}} = \frac{\sum_{n=1}^N [x(n)^2] - Nx_{\text{ave}}^2}{N-1}.$$

The efficient [Eq. \(13-149\)](#) requires only $2N-1$ addition and $N+4$ multiply operations. So at the expense of two extra multiplies, we've reduced the number of additions needed to compute Var_{unbiased} by roughly N relative to [Eq. \(13-147\)](#).

There is a second advantage in using [Eq. \(13-149\)](#) instead of [Eq. \(13-147\)](#) in computing the variance of N incoming $x(n)$ samples. When using [Eq. \(13-147\)](#) to compute Var_{unbiased} , we first compute x_{ave} and must retain, in memory, the N -length $x(n)$ sequence in order to compute the $[x(n) - x_{\text{ave}}]^2$ sequence. When using [Eq. \(13-149\)](#) to compute Var_{unbiased} , we can simultaneously accumulate (sum)

the N incoming $x(n)$ samples and accumulate the N computed $x(n)^2$ samples without having to keep past $x(n)$ samples in memory. Thus [Eq. \(13-149\)](#) reduces the amount of data storage needed to compute $Var_{unbiased}$.

The traditional definition for the biased variance of N $x(n)$ samples is written as

(13-150)

$$Var_{biased} = \frac{\sum_{n=1}^N [x(n) - x_{ave}]^2}{N}.$$

Using a derivation similar to how we arrived at [Eq. \(13-149\)](#), we can write an efficient expression for computing a biased variance as

(13-150')

$$Var_{biased} = \frac{\sum_{n=1}^N [x(n)^2]}{N} - x_{ave}^2.$$

[Equation \(13-150'\)](#) requires $2N-1$ addition and $N+2$ multiply operations. Here again, we've reduced the number of additions needed to compute Var_{biased} by roughly N and reduced the necessary data storage, relative to [Eq. \(13-150\)](#). In the next section we discuss the hardware implementation of variance computations with a focus on real-time processing.

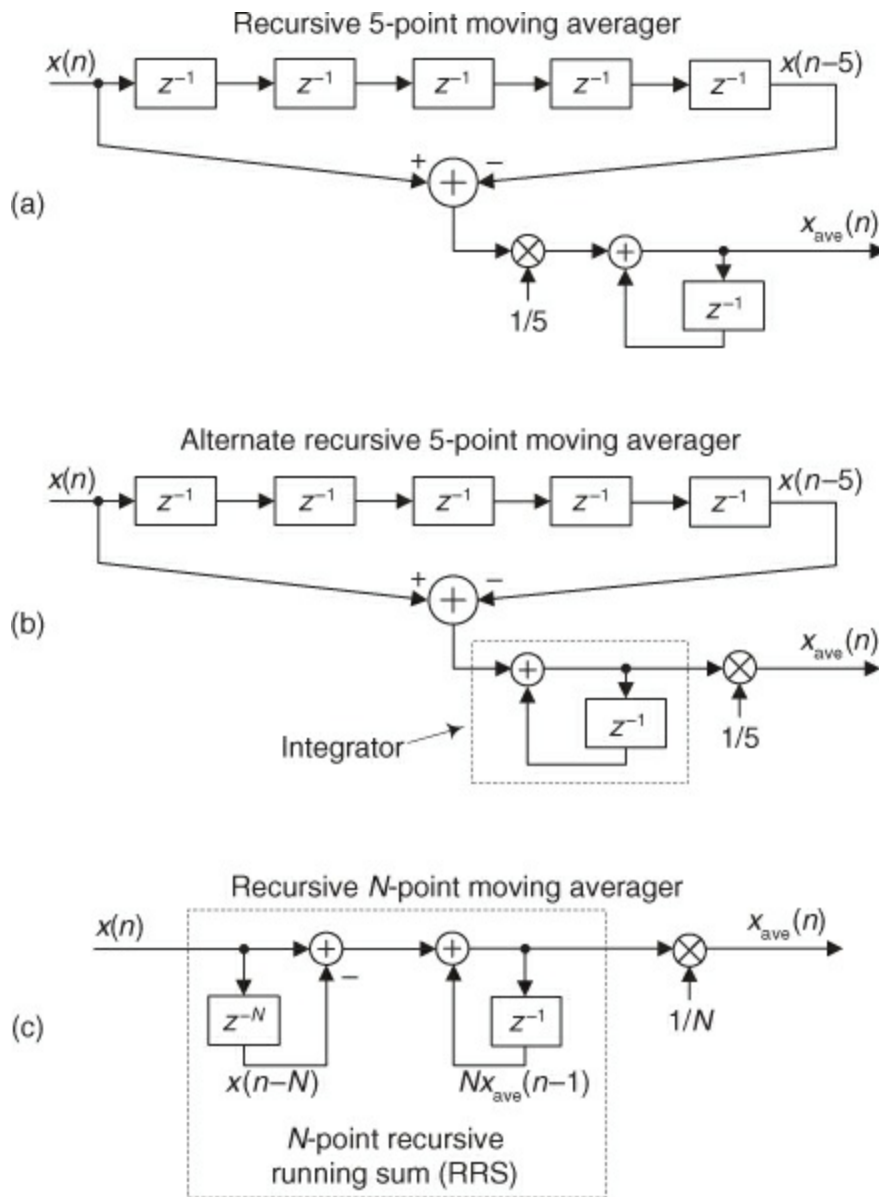
13.36 Real-time Computation of Signal Averages and Variances

In this section we present techniques for the efficient computation of real-time estimates of signal average and variance. By “real-time” we mean a continuing sequence of statistical estimates will be generated, in time, as a continuing sequence of input signal samples is applied to our processing networks.

13.36.1 Computing Moving Averages and Variances

[Figure 13-88\(a\)](#) shows a 5-point real-time recursive moving averager that we analyzed in [Section 11.5](#). For the reasons discussed in [Section 11.5](#) the recursive moving averager is the most computationally efficient method for computing moving averages.

Figure 13-88 Real-time recursive N -point moving averager: (a) standard $N = 5$ implementation; (b) alternate implementation; (c) general recursive depiction.



[Figure 13-88\(b\)](#) shows an alternate, but equivalent, recursive moving averager where the integrator now precedes the $1/5$ multiplication. In this alternate arrangement the binary register holding the integrator's accumulation results must be large enough to accommodate values in the range of five (number of unit-delay registers) times the true average of the most recent $N x(n)$ input samples.

In [Figure 13-88\(c\)](#) we redraw the alternate recursive moving averager in order to show the network of a general N -point recursive moving averager. There we use a single z^{-N} delay element symbol to represent an N -length delay line. In that figure we show a network inside the dashed-line box, which we'll use later for other statistical computations, called a *recursive running sum* (RRS).

Focusing now on the second topic of this section, there is a way to estimate the real-time N -point moving unbiased variance of a signal, $x_{\text{var,unbiased}}(n)$ [\[84\]](#). (A definition of *unbiased variance* is provided in [Appendix D](#).) To see how, we start with the expression for the unbiased variance of N time samples, [Eq. \(13-149\)](#) from the previous section, rewritten here as

$$(13-151)$$

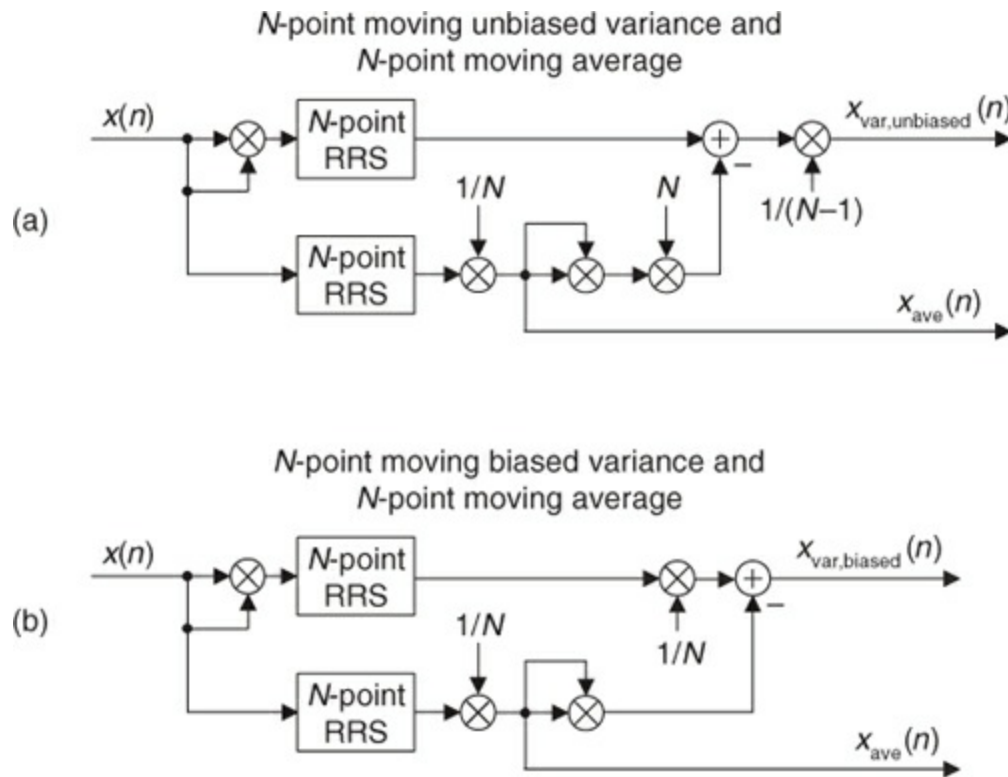
$$x_{\text{var,unbiased}}(n) = \frac{\sum_{n=N+1}^n [x(n)^2]}{N-1} - \frac{N[x_{\text{ave}}(n)]^2}{N-1},$$

where $x_{\text{ave}}(n)$ is the average of the most recent N input $x(n)$ samples. The limits on the summation in

Eq. (13-151) are such that we're summing a sliding-in-time block of N samples of $x(n)^2$.

The implementation of Eq. (13-151) is shown in Figure 13-89(a) where the process uses two N -point RRS networks from Figure 13-88(c) to compute the N -point moving unbiased variance $x_{\text{var,unbiased}}(n)$ and the $x_{\text{ave}}(n)$ N -point moving average of $x(n)$ [83,85]. Note that the $x_{\text{var,unbiased}}(n)$ and $x_{\text{ave}}(n)$ outputs are not valid until the N -stage delay lines are filled with input data.

Figure 13-89 Real-time N -point moving variance networks.



To estimate the real-time N -point moving biased variance of a signal, $x_{\text{var,based}}(n)$, we compute

(13-152)

$$x_{\text{var,based}}(n) = \frac{\sum_{n=N+1}^n [x(n)^2]}{N} - [x_{\text{ave}}(n)]^2$$

using the network shown in Figure 13-89(b).

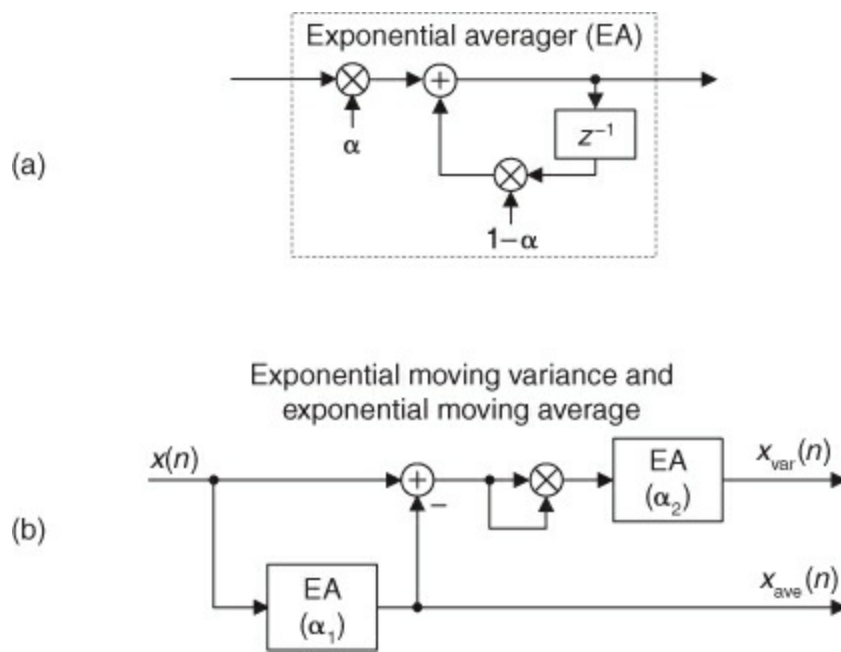
From a practical standpoint, in fixed-point systems, note that the binary word width of the upper RRS delay lines, in Figure 13-89, must be twice as wide as the lower RRS delay lines.

The above real-time fixed-length moving average and moving variance networks require data memory to implement their N -point delay lines. The following section describes techniques for estimating cumulative averages and variances with reduced data memory requirements.

13.36.2 Computing Exponential Moving Average and Variance

An alternate method to generate estimates of both the real-time moving average and real-time moving variance of a signal is to use the exponential averager that we discussed in Section 11.6, shown in Figure 13-90(a). The coefficient α is the exponential averager's *weighting factor* that controls the amount of averaging that takes place at the output of the network.

Figure 13-90 Exponential moving average and exponential moving variance: (a) standard exponential averaging network; (b) full structure.



To generate our desired exponential moving average and exponential moving variance, we use two independent exponential averaging (EA) networks as shown in [Figure 13-90\(b\)](#). The two weighting factors, α_1 and α_2 , are constants in the range of zero to one.

The process in [Figure 13-90\(b\)](#) has several attractive properties. The α_1 and α_2 coefficients permit control over the averaging behavior of the process; and the [Figure 13-90\(b\)](#) process requires fewer computations per output sample, and reduced delay-line element (data memory) requirements, relative to the networks in [Figure 13-89](#).

13.37 Building Hilbert Transformers from Half-band Filters

This section discusses two techniques for obtaining the coefficients of a Hilbert transformer from the coefficients of an N -point nonrecursive FIR half-band filter[[86,87](#)]. The first scheme is useful for someone who needs to design a Hilbert transformer when only generic lowpass FIR filter design software is available. The second scheme is useful for those unfortunate folks who have no FIR filter design software at hand but have available the coefficients of a half-band filter.

13.37.1 Half-band Filter Frequency Translation

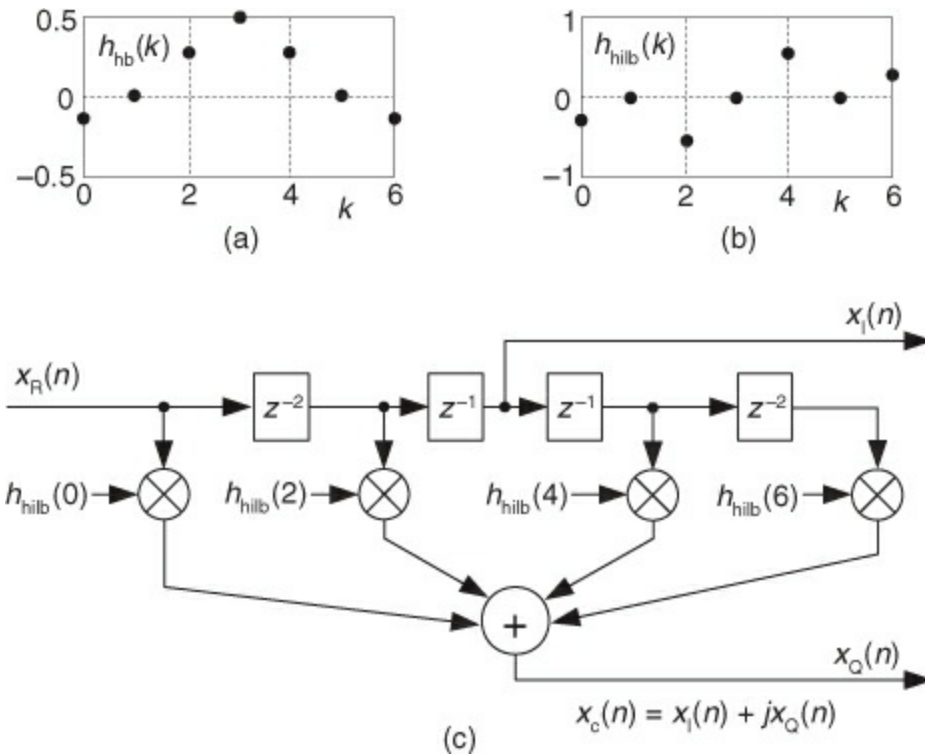
We can design a Hilbert transformer by first designing an N -tap half-band filter using our favorite FIR filter design software, with the restriction that $N+1$ is an integer multiple of four. Let's call the half-band filter's coefficients $h_{\text{hb}}(k)$, where the coefficients' index variable k is 0, 1, 2, ..., $N-1$. Next we obtain the Hilbert transformer's $h_{\text{hilb}}(k)$ coefficients using

$$(13-153)$$

$$h_{\text{hilb}}(k) = 2\sin\left[\frac{\pi}{2}\left(k - \frac{N-1}{2}\right)\right]h_{\text{hb}}(k).$$

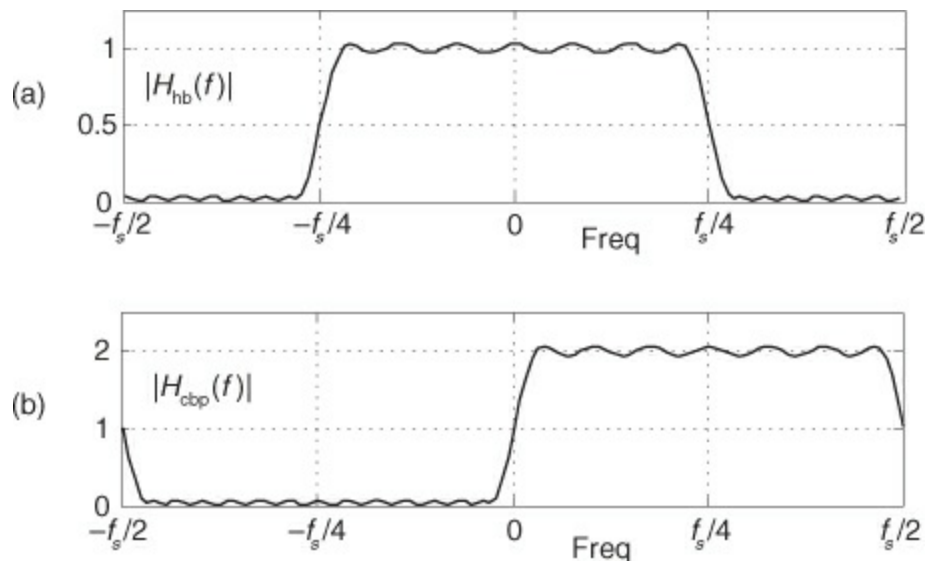
[Figure 13-91\(a\)](#) shows the coefficients of a simple 7-tap half-band filter whose DC gain is unity. [Figure 13-91\(b\)](#) shows the Hilbert transformer's $h_{\text{hilb}}(k)$ coefficients obtained from [Eq. \(13-153\)](#). The network using those $h_{\text{hilb}}(k)$ coefficients to generate a complex (analytic) $x_c(n) = x_I(n) + jx_Q(n)$ sequence from an original real-valued $x_R(n)$ sequence is shown in [Figure 13-91\(c\)](#). (Notice the z^{-2} delay blocks comprising two unit-delay elements.)

Figure 13-91 Seven-tap half-band FIR transformation: (a) $h_{hb}(k)$; (b) $h_{hilb}(k)$; (c) complex bandpass filter structure.



Let's call the network in [Figure 13-91\(c\)](#) a complex bandpass filter and describe its characteristics a bit further. [Figure 13-92\(a\)](#) shows the $|H_{hb}(f)|$ frequency magnitude response of a half-band filter, and [Figure 13-92\(b\)](#) shows us that the complex bandpass filter's $|H_{cbp}(f)|$ frequency magnitude response is $|H_{hb}(f)|$ translated up in frequency by $f_s/4$. However, notice that $|H_{cbp}(f)|$'s passband gain and ripple, as well as its stopband ripple, are twice that of $|H_{hb}(f)|$. To make the complex bandpass filter's gain unity, rather than two, we decrease its coefficients by a factor of two and multiply the $x_I(n)$ sequence in [Figure 13-91\(c\)](#) by 0.5. That 0.5 multiply could, of course, be implemented with an arithmetic right shift of the $x_I(n)$ samples.

Figure 13-92 Frequency magnitude responses: (a) half-band filter; (b) complex bandpass filter.



The nifty part of this complex bandpass filter is as follows: To build a complex nonrecursive FIR filter having the performance (transition region width, stopband attenuation, etc.) of a real N -tap lowpass FIR filter, we typically must implement two real N -tap FIR filters having an overall

computational workload of $2(N-1)$ additions and $2N$ multiplications per complex output sample, as well as provide $2N$ memory locations to store the complex coefficients. The complex bandpass filter in [Figure 13-91\(c\)](#) reduces those computations and the memory requirement by a factor of two.

Here's another attribute: because the complex filter's coefficients are antisymmetrical, we can use the *folded* FIR filter scheme described in [Section 13.7](#) to reduce the number of multipliers by another factor of two!

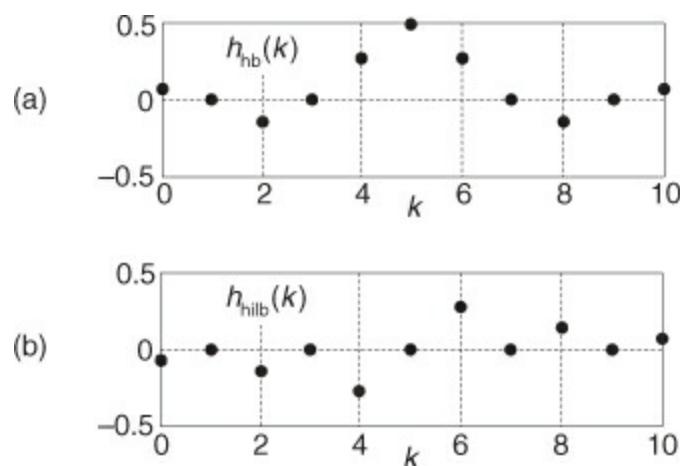
13.37.2 Half-band Filter Coefficient Modification

This second *half-band filter* to *Hilbert transformer* conversion scheme is useful for those unfortunate folks who have no nonrecursive FIR filter design software at hand but do happen to have the coefficients of a half-band filter. We can obtain the $h_{\text{hilb}}(k)$ coefficients of a Hilbert transformer with a straightforward modification of the half-band FIR filter's $h_{\text{hb}}(k)$ coefficients. The modification steps are as follows:

1. Identify the center coefficient of $h_{\text{hb}}(k)$; call it h_{center} .
2. Make the signs (polarity) of all nonzero coefficients before h_{center} negative.
3. Make the signs (polarity) of all nonzero coefficients after h_{center} positive.
4. Set the h_{center} coefficient equal to zero.

An example of this *half-band filter coefficient modification* process is shown for an 11-tap FIR half-band filter's $h_{\text{hb}}(k)$ in [Figure 13-93](#). In order to use the [Figure 13-93\(b\)](#) $h_{\text{hilb}}(k)$ coefficients in the complex bandpass filter in [Figure 13-92\(c\)](#), those $h_{\text{hilb}}(k)$ coefficients will need to be multiplied by a factor of two, or the $x_I(n)$ sequence in [Figure 13-91\(c\)](#) must be multiplied by 0.5. The 0.5 multiply can be implemented with an arithmetic right shift of the $x'_r(n)$ samples if desired.

Figure 13-93 Half-band filter coefficient modification: (a) original $h_{\text{hb}}(k)$ coefficients; (b) $h_{\text{hilb}}(k)$ coefficients.



13.38 Complex Vector Rotation with Arctangents

It's often the case in quadrature (I/Q) processing systems that we want to compute the angle of a complex time-domain sample. That angle computation for a complex sample $C = I + jQ$ is, of course,

$$(13-154)$$

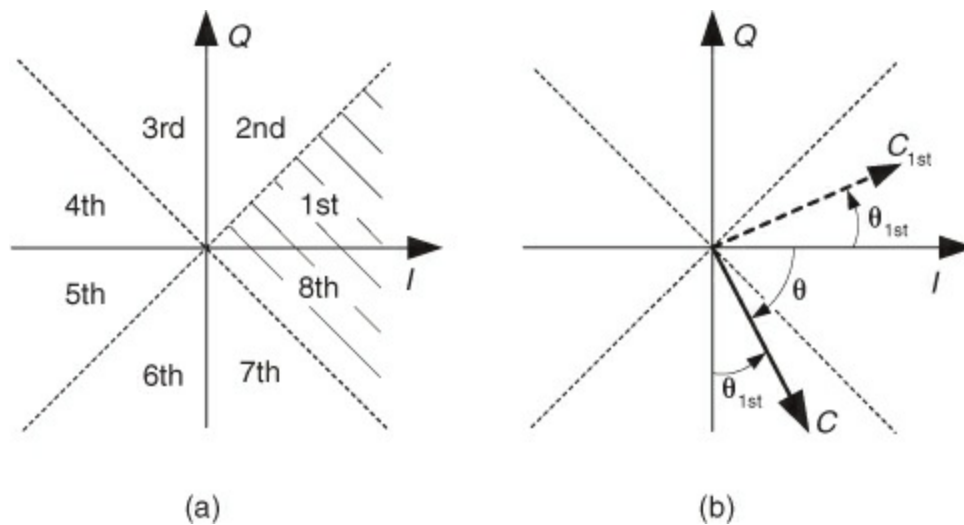
$$\theta = \tan^{-1}\left(\frac{Q}{I}\right).$$

As it turns out, the arctangent computation in [Eq. \(13-154\)](#) can be performed in many ways—anywhere from slow but accurate computationally intensive high-order polynomial evaluation, to high-speed crude-accuracy table look-up methods. However, regardless of the method used, we can improve the accuracy and speed of an arctangent computation if we limit the angular range over which it is performed. And that's where the vector rotation tricks presented here come into play.

13.38.1 Vector Rotation to the 1st Octant

Plotted on the complex plane, a complex sample $C = I + jQ$ can reside in any of the eight octants shown in [Figure 13-94\(a\)](#). When performing arctangents, please know that arctangent algorithms, be they high precision and computationally expensive or be they computationally simple and lower precision, are more accurate for small angles. (That is because the problematic arctangent function is only approximately linear for small angles.) So what does this mean to us? It means that if we can *effectively* rotate the angle of complex sample C into [Figure 13-94\(a\)](#)'s 1st or 8th octant, a smaller angle, arctangent algorithms will provide more accurate results.

Figure 13-94 Complex vector rotation: (a) octant definitions; (b) vector rotation from the 7th octant to the 1st octant.



For example, consider the complex number represented by vector C in [Figure 13-94\(b\)](#) residing in the 7th octant. The angle θ we want to compute is more negative than $-\pi/4$ radians (-45 degrees). Our trick is to rotate C to a new (and smaller) angle θ_{1st} , compute θ_{1st} with an arctangent algorithm, and add θ_{1st} to $-\pi/2$ to obtain the desired value for θ .

Rotating vector C can be implemented as follows:

- If vector C 's Q component is negative (C is in the 5th through the 8th octant), we can rotate C by 180 degrees by negating both the I and Q components.
- If vector C is in the 3rd or 4th octant, we can rotate C clockwise by 90 degrees by setting the new I equal to the old Q value, and setting the new Q equal to the negative of the old I value. (Note that the negative of the old I value is equal to the absolute value of the old I value.)
- If vector C is in the 2nd octant, we can rotate C clockwise by 45 degrees by swapping the I and Q components.

Using the above rotation operations for our [Figure 13-94\(b\)](#) example, we can rotate the original 7th-

octant $C = I + jQ$ to the 3rd octant by creating vector $C_{3rd} = -I - jQ$. Next we rotate C_{3rd} to the 1st octant by creating vector $C_{1st} = -Q + jI$. We compute θ_{1st} as

$$(13-155)$$

$$\theta_{1st} = \tan^{-1}\left(\frac{I}{-Q}\right) = -\tan^{-1}\left(\frac{I}{Q}\right)$$

using an arctangent algorithm and finally add θ_{1st} to $-\pi/2$ to obtain our desired value for θ .

OK, here’s the neat part of this trick. We don’t actually have to perform any of the above vector rotations to obtain angle θ_{1st} . We merely need to find the signs of the original I and Q components and determine which component has the larger magnitude. With those three pieces of information we determine in which octant vector C is located by using [Table 13-9](#).

Table 13-9 Octant Identification

Sign(I)	Sign(Q)	$ Q - I $	Octant
+	+	$ Q - I < 0$	1st
+	+	$ Q - I \geq 0$	2nd
-	+	$ Q - I \geq 0$	3rd
-	+	$ Q - I < 0$	4th
-	-	$ Q - I < 0$	5th
-	-	$ Q - I \geq 0$	6th
+	-	$ Q - I \geq 0$	7th
+	-	$ Q - I < 0$	8th

Once we know vector C ’s octant, we take advantage of the following *rotational symmetries* of arctangents

$$(13-156)$$

$$\tan^{-1}\left(-\frac{Q}{I}\right) = -\tan^{-1}\left(\frac{Q}{I}\right)$$

$$(13-156')$$

$$\tan^{-1}\left(\frac{Q}{I}\right) = \frac{\pi}{2} - \tan^{-1}\left(\frac{I}{Q}\right)$$

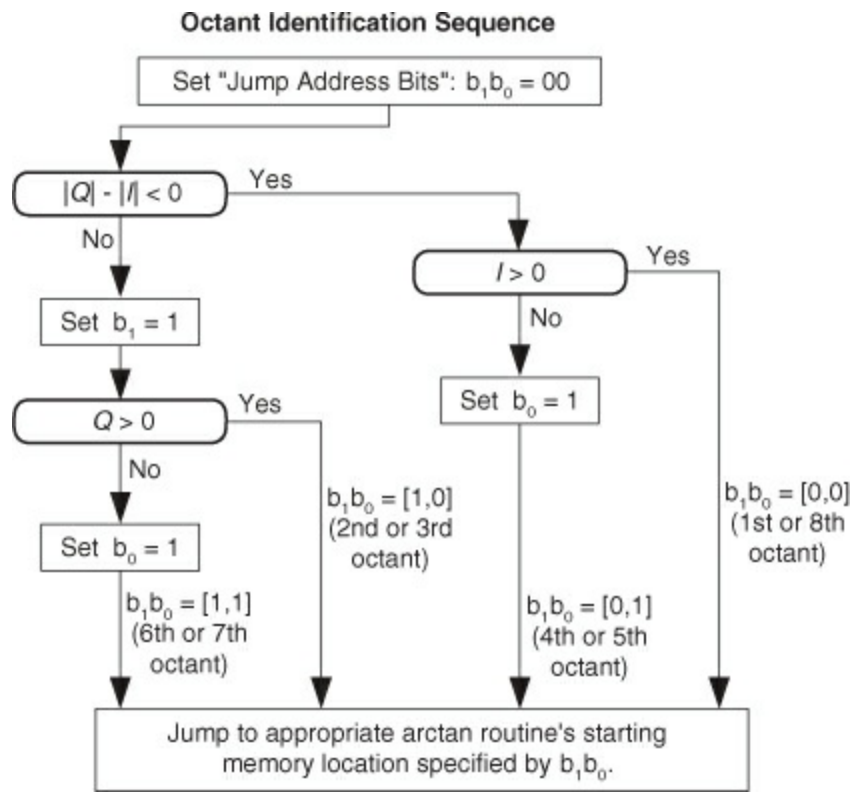
and compute our desired θ value using the appropriate expression in [Table 13-10](#).

Table 13-10 Arctan Computation

Quadrant	Angle θ
1st, or 8th	$\theta = \tan^{-1}(Q/I)$
2nd, or 3rd	$\theta = \pi/2 - \tan^{-1}(I/Q)$
4th, or 5th	$\theta = \text{sign}(Q)\pi + \tan^{-1}(Q/I)$
6th, or 7th	$\theta = -\pi/2 - \tan^{-1}(I/Q)$

Given that this arctangent process is implemented with programmable hardware, we’ll have [Table 13-10](#)’s four different arctangent approximation routines located at four different memory locations to which we’ll *jump*. The process to determine the necessary two *jump address* index bits (b_1, b_0) based on vector C ’s octant is shown in [Figure 13-95](#).

Figure 13-95 Octant and jump address identification flow.



To avoid division by zero when using the algorithms in [Table 13-10](#), it's prudent to precede the [Figure 13-95](#) processing with *checking* to see if I or Q is zero:

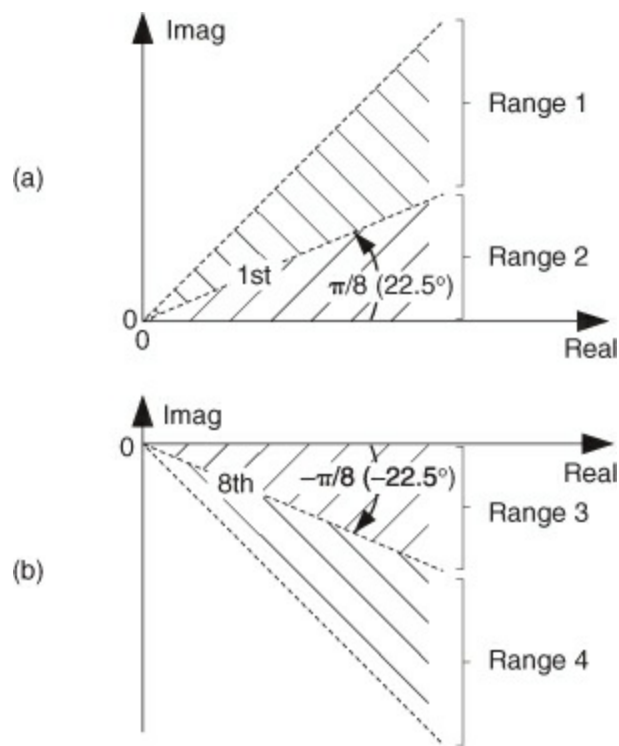
- If $I = 0$, θ is set to $\pi/2$ or $-\pi/2$ depending on the sign of Q .
- If $Q = 0$, θ is set to 0 or π depending on the sign of I .
- If I and Q are both zero, set θ to 0 .

Again, this section does not present any specific arctangent algorithms. This material shows how to make a given arctangent algorithm more accurate.

13.38.2 Vector Rotation by $\pm\pi/8$

While we're on the subject of vector rotation, if a 1st-octant vector C_{1st} resides in the angle range of $\pi/8 \leq \theta_{1st} \leq \pi/4$ radians (Range 1 in [Figure 13-96\(a\)](#), $22.5^\circ \leq \theta_{1st} \leq 45^\circ$), we can rotate that vector by $-\pi/8$ radians (-22.5°), forcing the new vector into Region 2. We may want to perform this rotation because arctangent algorithms have improved accuracy in Region 2.

Figure 13-96 Angle ranges of the 1st and 8th octants.



We rotate a vector $C_{1st} = I_{1st} + jQ_{1st}$ residing in Range 1 to Range 2 by multiplying C_{1st} by the complex number $e^{-j\pi/8} = (A - jB)$, where

$$(13-157)$$

$$A = 0.923879, \text{ and } B = 0.382683.$$

We can simplify the complex multiply by dividing A and B by 0.923879, yielding

$$(13-158)$$

$$A' = 1, \text{ and } B' = 0.414213.$$

This gives us a new $(A' - jB') = (1 - j0.414213)$ multiplier, reducing the number of necessary real multiplies in this $-\pi/8$ rotation process[88]. However, be aware that this $(A' - jB')$ rotation induces a vector magnitude gain of 1.0824 (0.69 dB) in the rotated vector.

Here's how we decide if the 1st-octant vector C_{1st} lies in the Range 1 of $\pi/8 \leq \theta_{1st} \leq \pi/4$ radians. If the minimum of I_{1st} or Q_{1st} is less than 0.414213 times the maximum of I_{1st} or Q_{1st} , then C_{1st} lies in Region 1, in which case vector rotation by $(A' - jB')$ multiplication is performed. Otherwise the 1st-octant vector is in Range 2, requiring no rotation.

In a similar manner, if an 8th-octant vector C_{8th} resides in the angle range of $-\pi/4 \leq \theta_{8th} \leq -\pi/8$ radians (Range 4 in [Figure 13-96\(b\)](#)), we can rotate that vector by $\pi/8$ radians (22.5°), forcing the new vector into Region 3 by multiplying C_{8th} by $(A' + jB')$.

Again, the angle range reduction schemes in this section allow us to use arctangent algorithms that are computationally simpler (and thus faster) for a given accuracy. Of course, this technique forces us to perform additional angle range checking and to compute products such as $(I_{1st} + jQ_{1st})(A' - jB')$. Perhaps this scheme is most useful when used with an arctangent look-up table method. You make the call.

13.39 An Efficient Differentiating Network

This section presents a computationally efficient differentiating network that approximates the

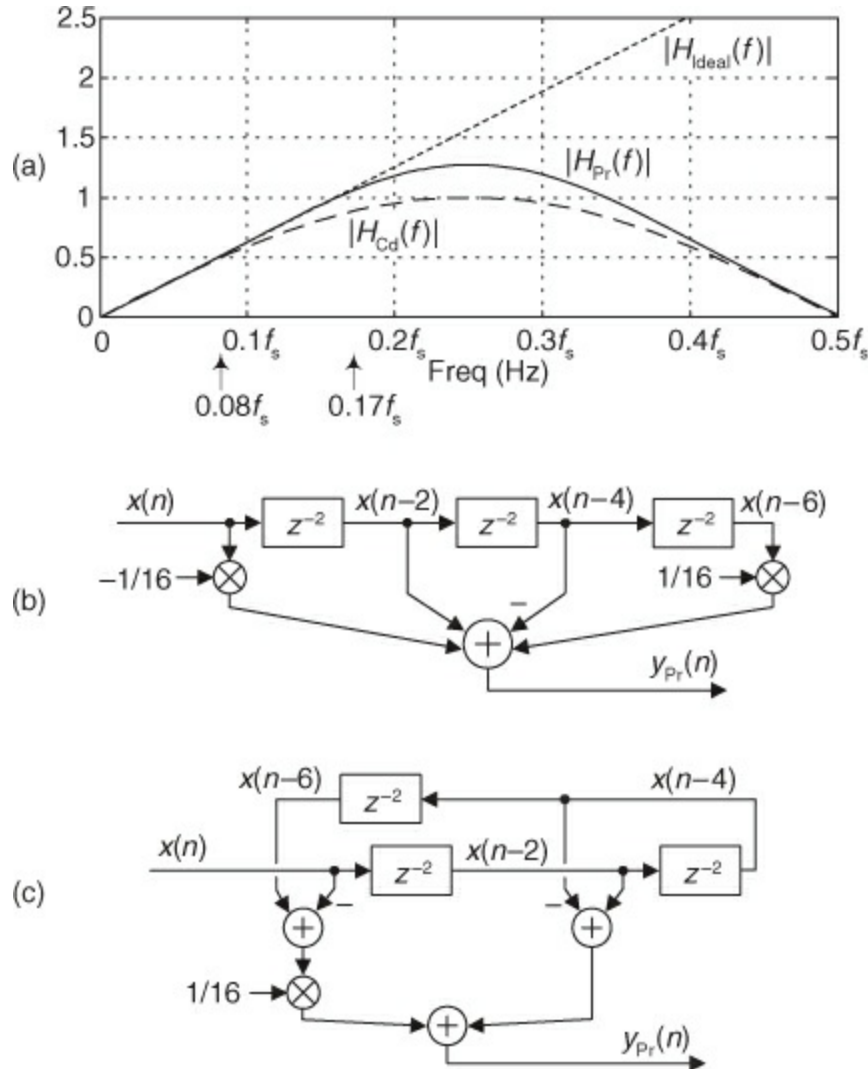
process of taking the derivative of a discrete time-domain sequence. In [Chapter 7](#) we introduced the *central-difference* differentiator, defined by

$$(13-159)$$

$$y_{Cd}(n) = [x(n) - x(n-2)]/2,$$

as a simple differentiating network that has desirable high-frequency (noise) attenuation behavior. The frequency magnitude response of that differentiator is the dashed $|H_{Cd}(f)|$ curve in [Figure 13-97\(a\)](#). (For comparison, we show an ideal differentiator's straight-line $|H_{Ideal}(f)|$ magnitude response in [Figure 13-97\(a\)](#). The frequency axis in that figure covers the positive-frequency range $0 \leq \omega \leq \pi$ samples/radian, corresponding to a continuous-time frequency range of 0 to $f_s/2$, where f_s is the input sample rate in Hz.) The central-difference differentiator's frequency range of linear operation is from zero to roughly $0.08f_s$ Hz.

Figure 13-97 Proposed differentiator: (a) performance; (b) standard structure; (c) folded structure.



Here we recommend a computationally efficient differentiator that maintains the central-difference differentiator's beneficial high-frequency attenuation behavior but extends its frequency range of linear operation. The proposed differentiator is defined by

$$(13-160)$$

$$y_{Pr}(n) = \frac{-x(n)}{16} + x(n-2) - x(n-4) + \frac{x(n-6)}{16}.$$

The [Eq. \(13-160\)](#) differentiator's frequency magnitude response is the solid $|H_{Pr}(f)|$ curve in [Figure](#)

[13-97\(a\)](#), where its frequency range of linear operation extends from zero to approximately $0.17f_s$ Hz, roughly twice the usable frequency range of the central-difference differentiator. The differentiator in [Eq. \(13-160\)](#) has a gain greater than that of the central-difference differentiator, so the solid curve in [Figure 13-97\(a\)](#) was scaled for easy comparison of $|H_{cd}(f)|$ and $|H_{dif}(f)|$. The $|H_{dif}(f)|$ curve is the DFT of $0.6 \cdot y_{dif}(n)$.

The structure of the proposed differentiator is shown in [Figure 13-97\(b\)](#) where a delay block comprises two unit-delay elements. The *folded*-FIR structure for this differentiator is presented in [Figure 13-97\(c\)](#) where only a single multiply need be performed per $y_{pr}(n)$ output sample. The really slick aspect of the $y_{pr}(n)$ differentiator is that its non-unity coefficients ($\pm 1/16$) are integer powers of two. This means that a multiplication in [Figure 13-97](#) can be implemented with an arithmetic right shift by four bits. Happily, such a binary right-shift implementation is a linear-phase multiplierless differentiator.

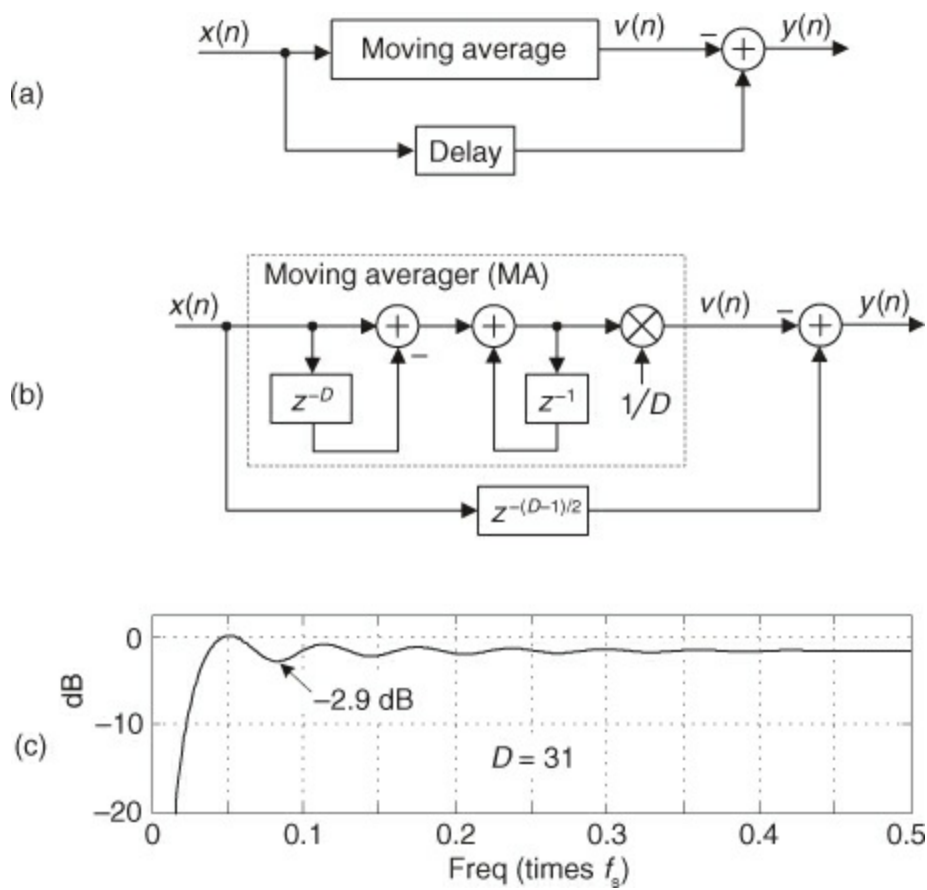
Another valuable feature of the $y_{pr}(n)$ differentiator is that its time delay (group delay) is exactly three samples. Such an integer delay makes this differentiator convenient for use when the $y_{pr}(n)$ output must be time-synchronized with other signals within a system. For fairness, we point out that the disadvantage of this very efficient differentiator is that for proper operation its $x(n)$ input signals must be low frequency, less than one-fifth the input sample rate.

In terms of performance and computational efficiency, the only contender to the proposed differentiator is the first narrowband “super Lanczos low-noise differentiator” discussed in [Chapter 7](#). However, the $y_{pr}(n)$ differentiator proposed here has better high-frequency noise attenuation than the Lanczos differentiator.

13.40 Linear-Phase DC-Removal Filter

In this section we introduce a linear-phase DC-removal filter useful for removing the DC bias from a time-domain signal. The filter is based on the notion of subtracting an input signal’s moving average (DC bias) from that signal, as shown in [Figure 13-98\(a\)](#).

Figure 13-98 DC-removal filter: (a) filter concept; (b) filter structure; (c) filter frequency response.



In order to reduce the delay line length of a standard tapped-delay line moving average network, we use the D -point recursive moving averager (MA), shown in [Figure 13-98\(b\)](#). The bottom path, in [Figure 13-98\(b\)](#), is a simple delay line having a length equal to the averager's group delay, $(D-1)/2$ samples. This enables us to time-synchronize the averager's $v(n)$ output with the $x(n)$ input in preparation for the subtraction operation. There are two delay lines in [Figure 13-98\(b\)](#): the D -length z^{-D} delay line in the top path and the bottom path's $(D-1)/2$ -length delay line.

The D -point recursive moving averager (MA) in [Figure 13-98\(b\)](#) has a transfer function defined by

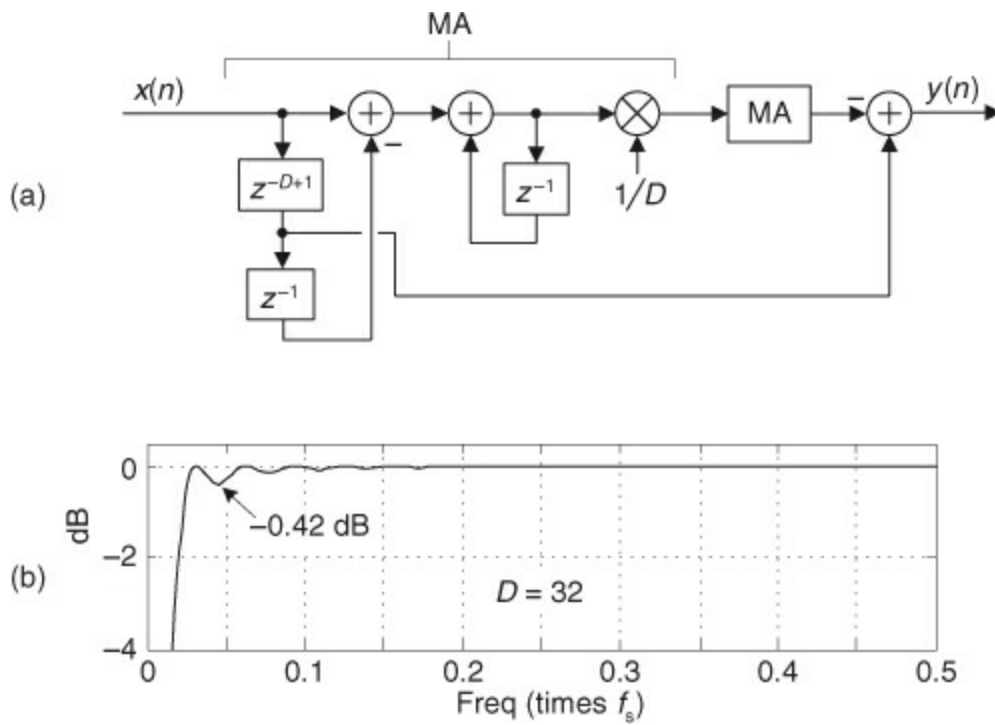
$$(13-161)$$

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

This DC-removal network's passband performance, when $D = 31$, is shown in [Figure 13-98\(c\)](#). (The frequency axis value of 0.5 corresponds to a cyclic frequency of half the input signal's f_s sample rate.) While the network has the desired infinite attenuation at zero Hz, its passband peak-peak ripple is unpleasantly large at 2.9 dB. We can do better, as we shall see.

If D is an integer power of two, the $1/D$ scaling in (1) can be performed using a binary right shift by $\log_2(D)$ bits, making [Figure 13-98\(b\)](#) a multiplier-free network. However, in that scenario the MA's group delay is not an integer number of samples, making it difficult to synchronize the delayed $x(n)$ and the $v(n)$ sequences. To solve this problem we can use two cascaded D -point MAs as shown in [Figure 13-99\(a\)](#). Because the cascaded MAs have an integer group delay of $D-1$ samples, we can be clever and tap off the first moving averager's comb delay line, eliminating the bottom-path delay line in [13-98\(b\)](#). This way we still only need implement two delay lines in [Figure 13-99\(a\)](#), one z^{-D} delay line in each MA.

Figure 13-99 Dual-MA filter: (a) filter structure; (b) filter frequency response.



The magnitude response of the [Figure 13-99\(a\)](#) dual-MA DC-removal network, for $D = 32$, is shown in [Figure 13-99\(b\)](#). In that figure we show the DC-removal filter's passband with its narrower transition region width and a much improved peak-peak ripple of 0.42 dB. What we've created, then, is a linear-phase, multiplierless, DC-removal network having a narrow transition region near zero Hz.

Happily, it's worth noting that standard tapped-delay line, linear-phase, highpass FIR filter designs using least-squares error minimization, or the Parks-McClellan method, require more than 100 taps to approximate our $D = 32$ DC-removal filter's performance.

On a practical note, the MAs in [Figure 13-99\(a\)](#) contain integrators that can experience data overflow. (An integrator's gain is infinite at DC.) Using two's complement fixed-point arithmetic avoids integrator overflow errors if we ensure that the integrator (accumulator) bit width is at least

$$(13-162)$$

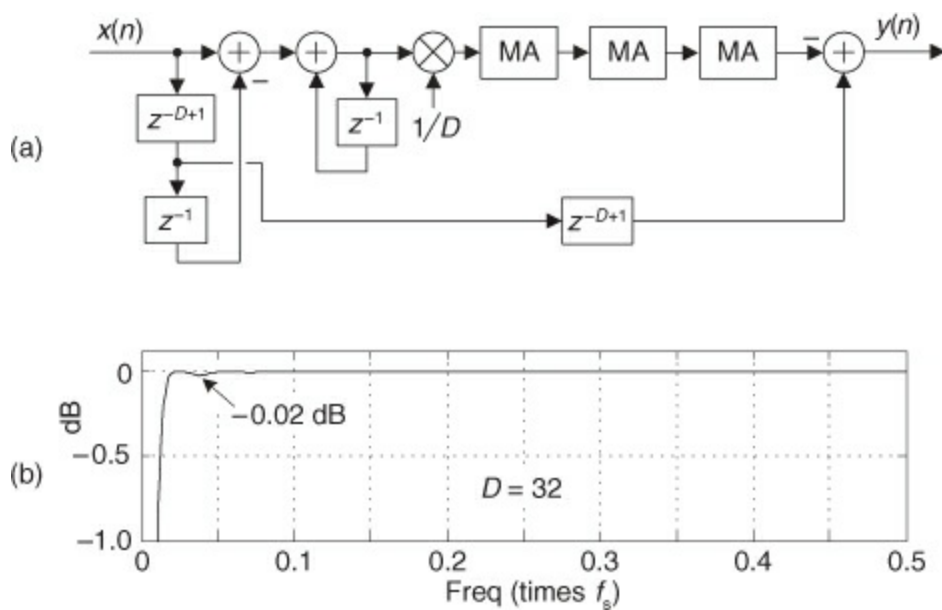
$$\text{accumulator bits} = \text{number of bits in } q(n) + \lceil \log_2(D) \rceil$$

where $q(n)$ is the input sequence to an accumulator, and $\lceil k \rceil$ means that if k is not an integer, round it up to the next larger integer.

For an even narrower filter transition region width, in the vicinity of zero Hz, than that shown in [Figure 13-99\(b\)](#), we can set D to a larger integer power of two; however, this will not reduce the DC-removal filter's passband ripple.

At the expense of three additional delay lines, and four new addition operations per output sample, we can implement the linear-phase DC-removal filter shown in [Figure 13-100\(a\)](#). That quad-MA implementation, having a group delay of $2D-2$ samples, yields an improved passband peak-peak ripple of only 0.02 dB, as shown in [Figure 13-100\(b\)](#), as well as a reduced-width transition region relative to the dual-MA implementation.

Figure 13-100 Quad-MA filter: (a) filter structure; (b) filter frequency response.



The DC removal network in [Figure 13-100\(a\)](#) contains four $1/D$ scaling operations which, of course, can be combined and implemented as a single binary right shift by $4\log_2(D)$ bits. So the bottom line here is that at the expense of multiple delay lines, it is possible to efficiently perform linear-phase DC removal.

13.41 Avoiding Overflow in Magnitude Computations

Here we present a little trick to help avoid a common problem when computing the magnitude of a complex number using fixed-point binary number formats. Let's say we have a complex number c represented by $c = R + jI$, and we want to compute the magnitude c using the familiar expression

$$(13-163)$$

$$|c| = \sqrt{R^2 + I^2}.$$

[Equation \(13-163\)](#) is troublesome because the R^2 and I^2 terms will cause data word overflow errors if either R or I is greater than the square root of your fixed-point number format's largest positive number. For example, in a signed 16-bit number format, $|R|$ and $|I|$ must be less than 181 to avoid overflow errors. At the expense of absolute value comparison, branch, and divide operations, [Eq. \(13-164\)](#) alleviates overflow problems[\[89\]](#):

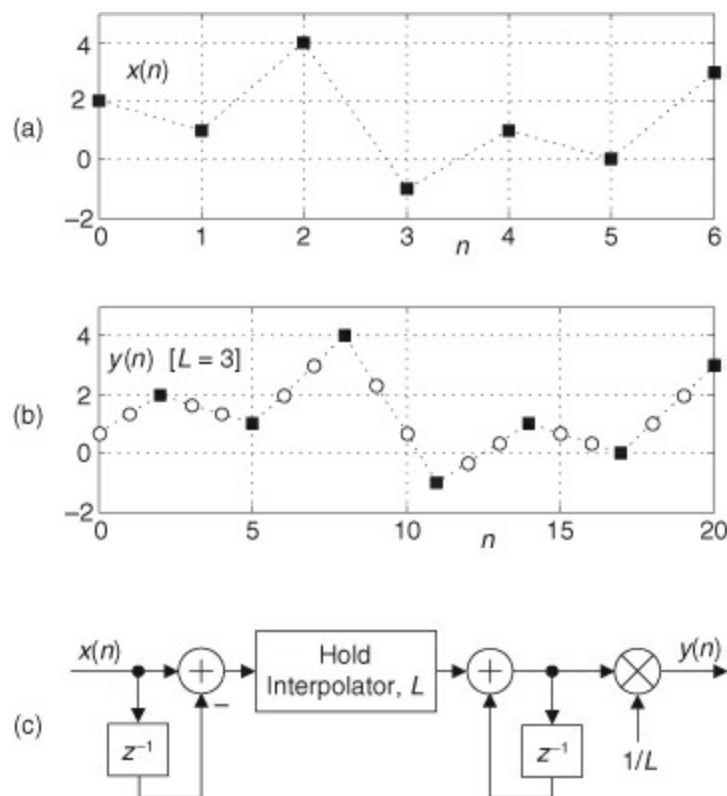
$$(13-164)$$

$$|c| = \begin{cases} |R| \sqrt{1 + (I/R)^2} & \text{if } |R| \geq |I| \\ |I| \sqrt{1 + (R/I)^2} & \text{if } |R| < |I|. \end{cases}$$

13.42 Efficient Linear Interpolation

In this section we present a computationally efficient linear interpolation trick that's useful because it performs linear interpolation requiring at most one multiply per output sample[\[90\]](#). For example, given the $x(n)$ time sequence in [Figure 13-101\(a\)](#), this linear interpolator will generate the $y(n)$ sequence shown in [Figure 13-101\(b\)](#) when the interpolation factor is $L = 3$. Notice how the original $x(n)$ samples are preserved in the $y(n)$ output sequence.

Figure 13-101 Linear interpolation: (a) input sequence; (b) $L = 3$ interpolated sequence; (c) interpolator structure.



The block diagram of this efficient linear interpolator is that in [Figure 13-101\(c\)](#). That mysterious block labeled “Hold Interpolator, L ” is merely the operation where each input sample to the block is repeated $L-1$ times. For example, if the input to the Hold Interpolator operation is $\{1, 4, 3\}$, and $L = 3$, the output of the Hold Interpolator is $\{1, 1, 1, 4, 4, 4, 3, 3, 3\}$.

In fixed-point binary implementations if we’re able to select L to be an integer power of two, then, happily, the final $1/L$ multiplication can be implemented with a binary arithmetic right shift by $\log_2 L$ bits, yielding a multiplierless linear interpolator. Of course, if a gain of L is acceptable, no $1/L$ scaling need be performed at all.

The neat part of this interpolator is that the computational workload, the number of additions and multiplies per output sample, remains fixed regardless of the value of interpolation factor L .

The experienced reader might now say, “Ah, while this network is computationally simple, linear interpolation is certainly not the most accurate method of interpolation, particularly for large interpolation factors of L .” That is true, but if interpolation is being done in multiple sections, using this efficient linear interpolation as the final section at the highest data rate (when the signal samples are already very close together) will introduce only a small interpolation error.

13.43 Alternate Complex Down-conversion Schemes

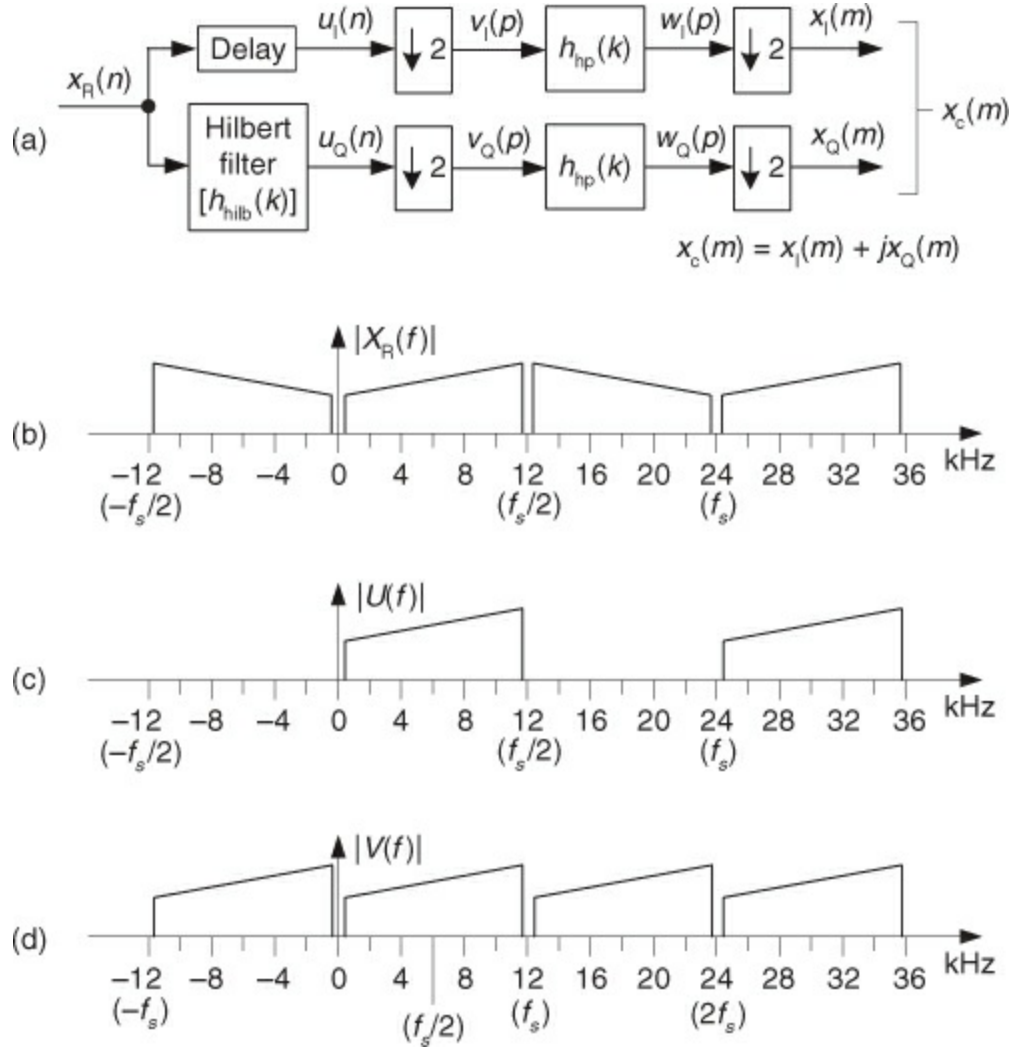
Here we present two interesting complex down-conversion and decimation techniques used to generate an analytic (complex) version, centered at zero Hz, of a real bandpass signal that was originally centered at $\pm f_s/4$ (one-fourth the sample rate). Both methods perform signal frequency translation by way of decimation.

13.43.1 Half-band Filter Down-conversion

The first complex down-conversion method makes use of computationally efficient half-band filters[\[91\]](#). The process is shown in [Figure 13-102\(a\)](#), where we use indices n , p , and m to clarify the multirate nature of this process. The real $x_R(n)$ input signal has the spectrum shown in [Figure 13-](#)

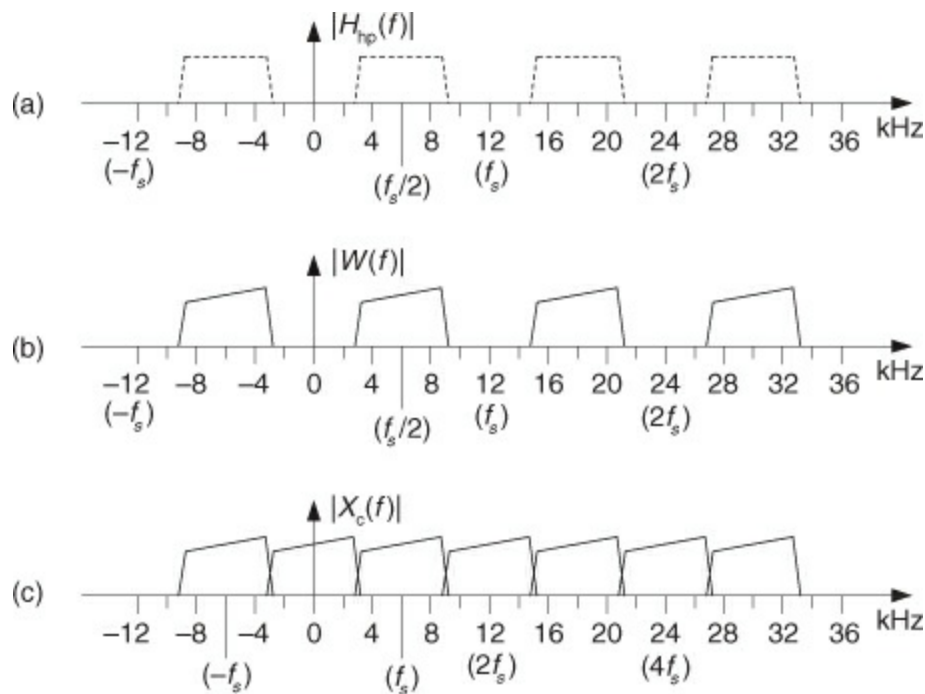
102(b), and for our example the sample rate is $f_s = 24$ kHz. The Delay/Hilbert transform filter combination attenuates the negative-frequency spectral components of $X_R(f)$ to produce the complex $u_I(n) + ju_Q(n)$ signal whose spectrum is provided in [Figure 13-102\(c\)](#). (The Delay function is a cascade of unit-delay elements, whose length is the group delay of the Hilbert filter, needed to time-synchronize the $u_I(n)$ and $u_Q(n)$ sequences.) The follow-on downsample by two, discard every other sample, produces the complex $v(p)$ sequence having the spectrum shown in [Figure 13-102\(d\)](#) where the new sample rate is 12 kHz.

Figure 13-102 Analytic signal generation and decimation by two.



Next, sequences $v_I(p)$ and $v_Q(p)$ are applied to two identical real-valued highpass half-band filters, each having the frequency magnitude response shown in [Figure 13-103\(a\)](#), yielding the complex $w(p) = w_I(p) + jw_Q(p)$ whose spectrum is that in [Figure 13-103\(b\)](#). The final step in this down-conversion process is another decimation by two, producing the desired $x_c(m)$ sequence having the spectrum given in [Figure 13-103\(c\)](#) where the output sample rate is 6 kHz. Due to the nature of half-band filters there will be some amount of spectral overlap in $X_c(f)$ as shown in [Figure 13-103\(c\)](#). The amount of spectral overlap is proportional to the transition region width of an $h_{\text{hp}}(k)$ filter (inversely proportional to the number of filter taps).

Figure 13-103 Highpass filtering, down-conversion, and decimation by two.



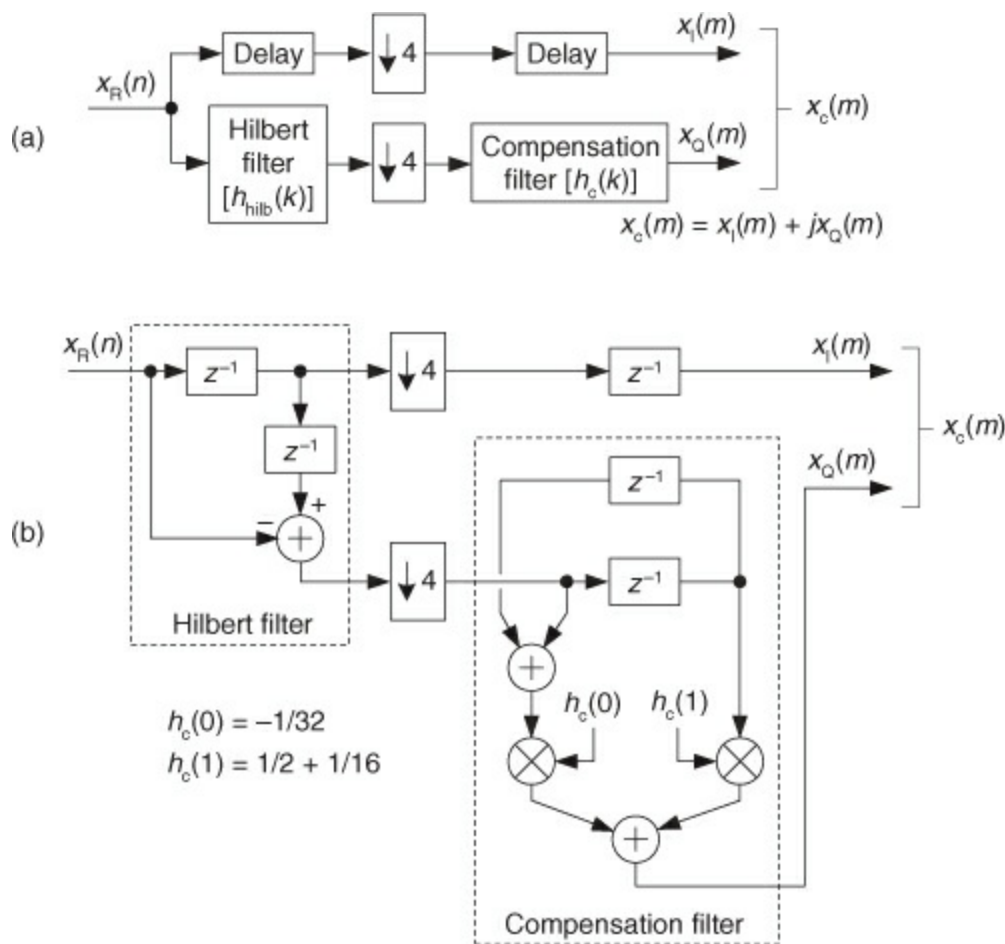
There are three useful aspects to this first complex down-conversion scheme that enhance its computational efficiency:

- If the Hilbert transform filter has an odd number of taps, roughly half of its coefficients will be zero-valued, and the Delay function is an integer number of unit-delay elements.
- Roughly half of the coefficients of the highpass half-band filters, with their transition regions centered at $f_s/4$ and $3f_s/4$, will be zero-valued.
- Because the coefficients of the filters in [Figure 13-102\(a\)](#) are either symmetrical or antisymmetrical, we can use the *folded* FIR filter scheme described in [Section 13.7](#) to reduce the number of multipliers by another factor of two.

13.43.2 Efficient Single-Decimation Down-conversion

Our second complex down-conversion trick is a very computationally efficient scheme, shown in [Figure 13-104\(a\)](#), that operates on real $x_R(n)$ signals centered at $\pm f_s/4$. Just as in [Figure 13-102\(a\)](#), the Delay/Hilbert transform filter combination attenuates the negative-frequency spectral components of $x_R(n)$ to produce a complex analytic signal whose spectrum is centered at $f_s/4$ (6 kHz). The downsample-by-four, retain every fourth sample, operation down-converts (frequency translates) the desired complex signal originally centered at $f_s/4$ Hz down to a center frequency of zero Hz. The compensation filter is used to compensate for the non-flat frequency magnitude response of the simple 2-tap Hilbert filter in order to widen the down-converter's usable passband width. (The Delay function after the downsampling in the top path is needed to time-synchronize the $x_I(m)$ and $x_Q(m)$ sequences.) The detailed block diagram of the down-converter is shown in [Figure 13-104\(b\)](#), where the compensation filter's coefficients are $h_c(0) = -1/32$, and $h_c(1) = 1/2 + 1/16$.

Figure 13-104 High-efficiency complex down-conversion: (a) process; (b) detailed structure.



If the $x_R(n)$ input signal's bandwidth is no greater than $f_s/6$, then the Hilbert filter attenuates $x_R(n)$'s undesired negative-frequency spectral components, at the $x_c(n)$ output, by approximately 35 dB. That much attenuation may not be something to write home about, but keep in mind that this down-converter requires no multipliers because the multiplies by the $h_c(0)$ and $h_c(1)$ coefficients can be implemented with binary shifts and adds. At the expense of two multiplies per output sample, the compensation filter coefficients can be set to $h_c(0) = -0.02148$ and $h_c(1) = 0.54128$ to attenuate $x_R(n)$'s undesired negative-frequency spectral components by roughly 45 dB.

13.44 Signal Transition Detection

When we are tasked to build a system that must detect transitions in a pulsed signal, we generally look to a digital differentiator as the solution to our problem. However, when a pulsed signal's transition spans many samples, and particularly if the signal is noisy, digital differentiators do not provide reliable signal transition detection. One compelling solution to this problem uses a standard tapped-delay line (time-domain convolution) filtering scheme developed by C. Turner^[92]. Called *time-domain slope filtering*, the transition detection tapped-delay line filter uses N coefficients defined by

$$(13-165)$$

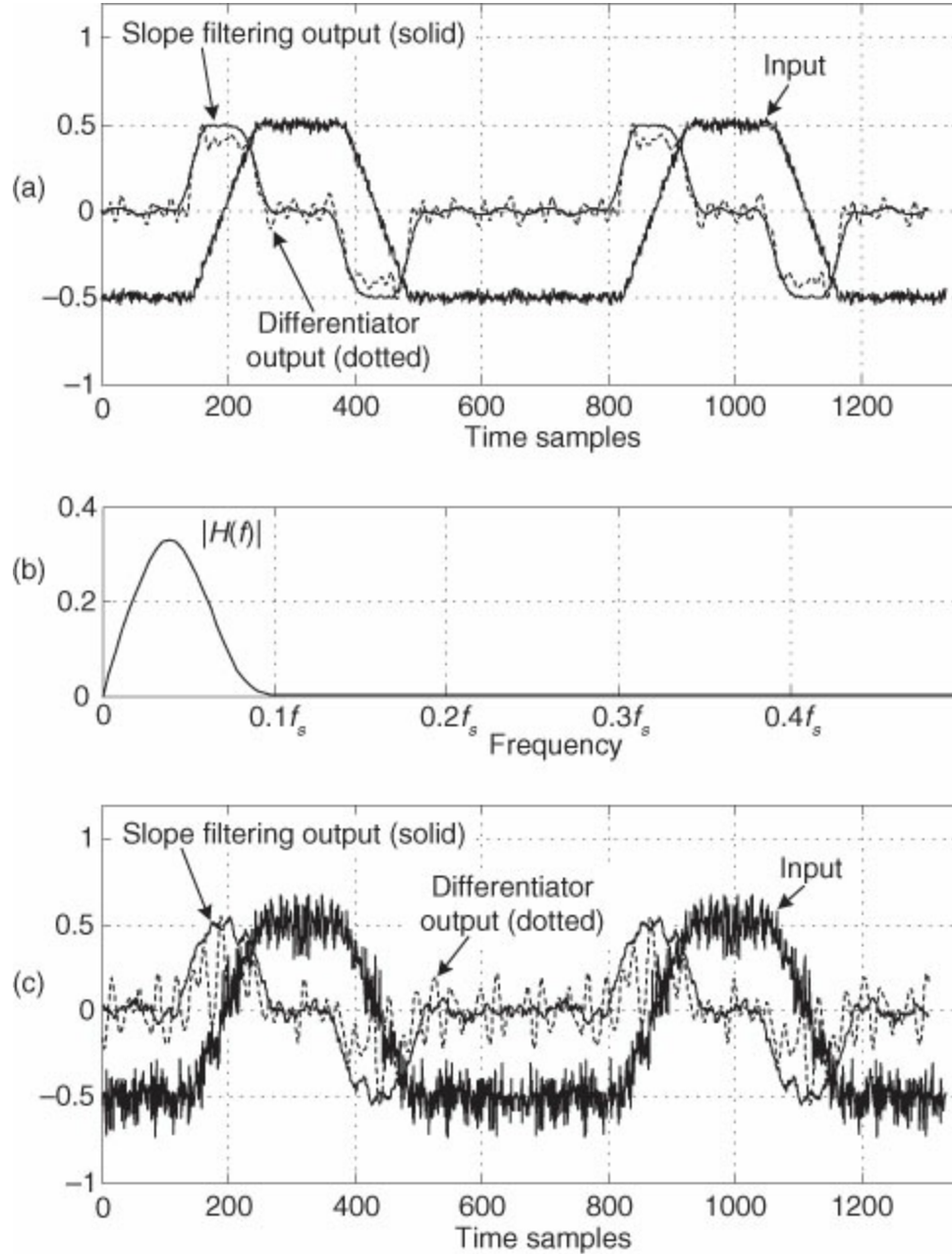
$$C_k = -\frac{12k - 6(N-1)}{N^3 - N}$$

where the coefficient index k covers the range $0 \leq k \leq N-1$.

For any integer N , the slope filtering C_k coefficients comprise a linear ramp, making that sequence quite useful for detecting linear transitions in an input signal. [Figure 13-105\(a\)](#) shows the output of the

time-domain slope filtering process, when $N = 53$. In that figure we see that the slope filter performs well in detecting the transitions of the Input signal. The dotted curve in [Figure 13-105\(a\)](#) is the output of a traditional tapped-delay line digital differentiator having 53 taps. (The frequency magnitude of the traditional digital differentiator, specifically designed to attenuate high-frequency noise, is provided in [Figure 13-105\(b\)](#).)

Figure 13-105 Time-domain slope filtering: (a) pulsed input performance; (b) digital differentiator magnitude response; (c) high-noise input performance.



The superiority of the time-domain slope filtering scheme over traditional differentiation is further illustrated in [Figure 13-105\(c\)](#) where the pulsed Input signal is contaminated with high-level noise.

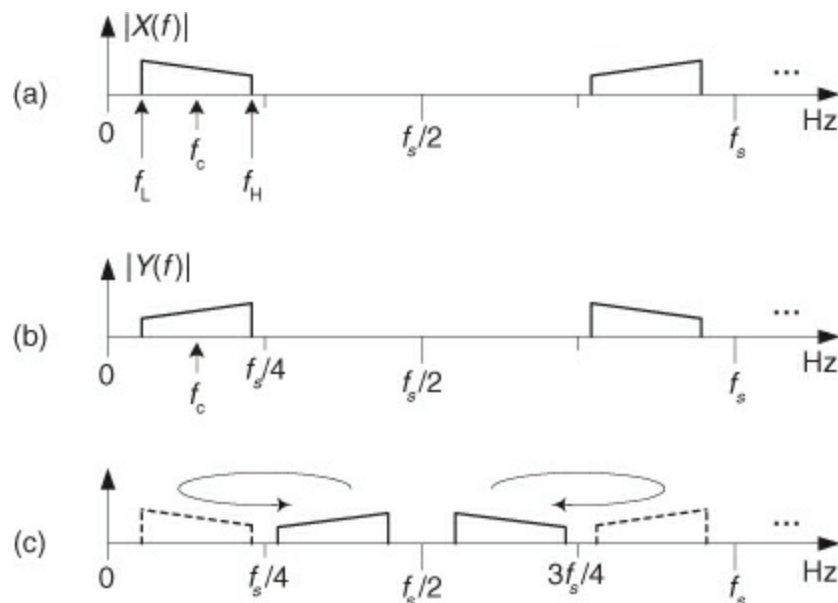
Concerning two practical issues, if the number of samples in a pulsed input signal's transition is L , the value for N , found empirically, is generally in the range of $L/4$ to L . It's convenient to set N to be an odd integer, forcing the filter's delay to be an integer number, $(N-1)/2$, of samples. This facilitates the time synchronization of the filter's output to other sequences in a system. Also, if the C_k coefficients are to be used in correlation processing (as opposed to the convolution processing discussed above), the correlation's C_k coefficients should be the coefficients from [Eq. \(13-165\)](#) reversed in time order.

13.45 Spectral Flipping around Signal Center Frequency

In [Section 2.4](#), we discussed a super-simple method of spectral flipping (spectral inversion) of a real signal where the center of spectral rotation was $f_s/4$. In this section we discuss a different kind of spectral flipping process.

Consider the situation where we need to process a real-valued $x(n)$ time signal, whose $X(f)$ spectrum is shown in [Figure 13-106\(a\)](#), to obtain a real-valued $y(n)$ time signal whose spectrum is the flipped $Y(f)$ spectrum shown in [Figure 13-106\(b\)](#). Notice that the center of rotation of the desired spectral flipping is not $f_s/4$ Hz but is instead the $x(n)$ signal's f_c center frequency. The spectral flipping process described in [Section 2.4](#) does not solve our problem because that process would result in the undesirable spectrum shown in [Figure 13-106\(c\)](#), where the original $X(f)$ spectrum is the dashed curve.

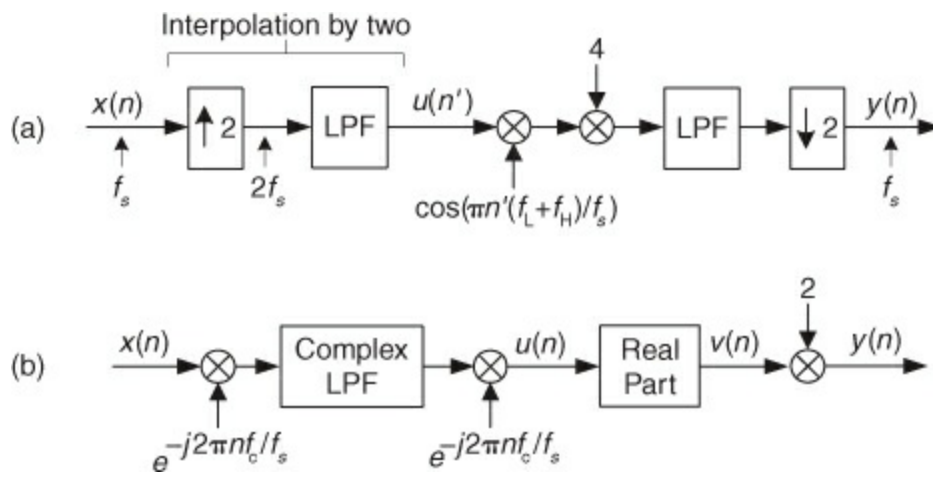
Figure 13-106 Spectral flipping, centered at f_c : (a) original spectrum; (b) desired spectrum; (c) incorrect spectrum.



There are two methods to solve our f_c -centered spectral flipping problem. [Figure 13-107\(a\)](#) shows the first method, comprising a multirate processing technique. In considering this spectral flipping method, the user should keep in mind that:

- The two lowpass filters (LFPs) have passbands that extend from zero Hz to f_H Hz. (Note that the sample rate for both filters is $2f_s$ Hz.) The second LFP's transition region width is less than $2f_L$.
- The cosine mixing sequence uses the upsampled-by-two time index variable n' .
- The multiply-by-four operation compensates for the sequence $u(n')$ amplitude loss by a factor of two caused by interpolation, and the amplitude loss by another factor of two due to the cosine mixing.

Figure 13-107 Spectral flipping techniques: (a) first method; (b) second method.



Of course, a smart engineer will eliminate the multiply-by-four operation altogether by increasing the DC (zero Hz) gain of one of the lowpass filters by four.

The second method we could use to obtain a signal having the desired [Figure 13-106\(b\)](#) spectrum, promoted by D. Bell, is the process shown in [Figure 13-107\(b\)\[93\]](#). While somewhat more computationally intensive than the above multirate method, this technique works well and deserves mention here. The first complex multiplication and the Complex LPF are identical to the quadrature sampling operations we discussed in [Figure 8-18\(a\)](#). The two identical lowpass filters, comprising the Complex LPF, have passbands that extend from zero Hz to $(f_H - f_L)/2$ Hz, and transition region widths of less than $2f_L$. The Real Part operation merely means take the real part of sequence $v(n)$.

We can eliminate the multiply-by-two operation by increasing the DC (zero Hz) gain of the complex filter by two. In this method, as Bell recommends, we can combine the second complex multiply and Real Part extraction stages by computing only the real part of sequence $u(n)$, yielding sequence $v(n)$. The multiply-by-two operation compensates for the amplitude loss by a factor of two caused by the Real Part operation.

13.46 Computing Missing Signal Samples

Consider the situation where we need to process a time-domain signal that has been corrupted such that every Q th sample is missing from the desired signal sequence. This section provides a trick for how to recover periodically spaced missing samples of a corrupted time sequence[\[94\]](#).

To explain our problem, assume we want to process an $x_0(n)$ time sequence, whose sample rate is f_s Hz, but all we have available to us is a corrupted $x_q(n)$ sequence where:

- $x_q(n)$ is equal to the desired $x_0(n)$ with every Q th sample of $x_0(n)$ missing. The missing samples in $x_q(n)$, $x_q(pQ)$ where $p = 0, 1, 2, \dots$ are represented by zero-valued samples.
- $x_0(n)$ is band-limited with negligible energy above B Hz where

$$(13-166)$$

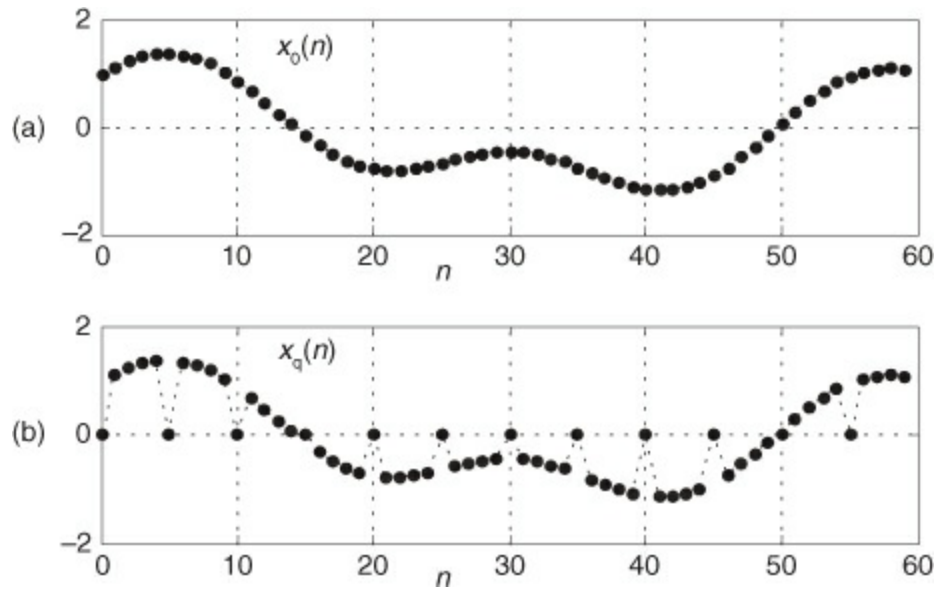
$$B < \frac{Q-1}{Q} \cdot \frac{f_s}{2}$$

for some integer $Q \geq 2$ where f_s is the data sample rate in Hz.

As an example, when $Q = 5$, if the desired $x_0(n)$ is the sequence in [Figure 13-108\(a\)](#), then $x_q(n)$ is the corrupted sequence shown in [Figure 13-108\(b\)](#). Our job, then, is to recover (interpolate) the missing

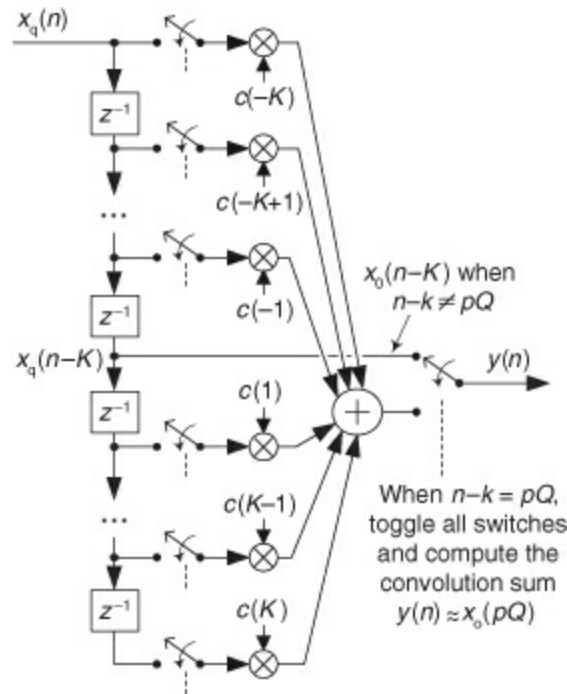
samples in $x_q(n)$, $x_q(0)$, $x_q(5)$, $x_q(10)$, ... etc., to reconstruct the desired $x_0(n)$ sequence.

Figure 13-108 Time sequences: (a) original $x_0(n)$; (b) corrupted $x_q(n)$ when $Q = 5$.



The solution to our problem is to apply the $x_q(n)$ sequence to the tapped-delay line reconstruction filter shown in [Figure 13-109](#). Describing [Figure 13-109](#)'s operation in words: our desired $x_0(n-K)$ samples are the $x_q(n-K)$ samples at the center tap of the filter unless that $x_q(n-K)$ sample is a zero-valued missing sample, in which case the switches toggle and we compute the estimated $x_0(n-K) = x_0(pQ)$.

Figure 13-109 Reconstruction filter implementation.



The filter's $c(k)$ coefficients are determined by first evaluating the following expression:

$$(13-167)$$

$$h(k) = \frac{Q-1}{Q} \cdot \text{sinc}\left(\frac{Q-1}{Q}k\right) \cdot w(k)$$

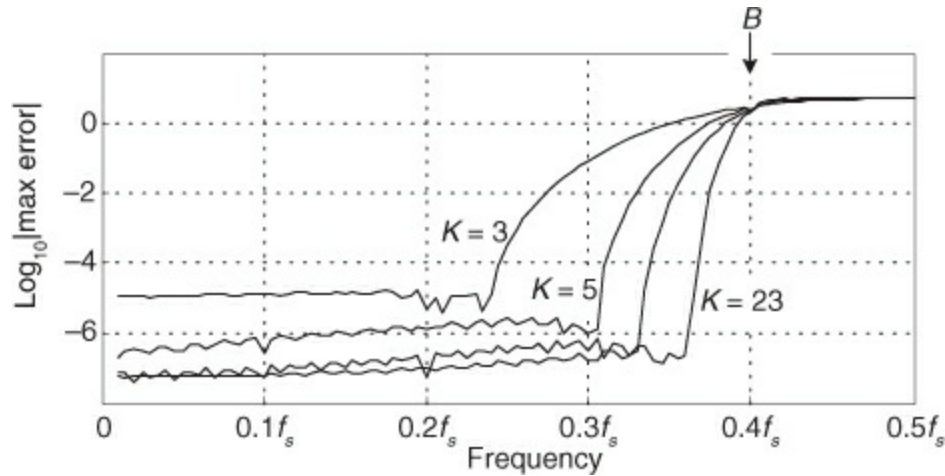
where integer index k is in the range $-K \leq k \leq K$, $\text{sinc}(x) = \sin(\pi x)/\pi x$, and $w(k)$ is a time-symmetric window sequence of length $2K+1$. Next, we use $h(k)$ to compute our desired filter coefficients as

$$c(k) = \begin{cases} 0, & k = 0 \\ h(k) / [1 - h(0)], & k \neq 0. \end{cases}$$

This missing sample recovery process can also be applied to complex $x_q(n)$ signals, in which case the real and imaginary parts of a complex $x_q(n)$ must be filtered separately.

There are two practical considerations to keep in mind when using this missing sample recovery process. The first consideration is to be aware that the maximum bandwidth B given in [Eq. \(13-166\)](#) is based on the assumption that the reconstruction filter has an infinite number of taps. As such, for practical-length filters the B bandwidth requirement must be reduced. To show this, [Figure 13-110](#) illustrates the missing sample recovery error when $Q = 5$, $B = 0.4f_s$, using a Chebyshev window with -100 dB sidelobes, for various values of K . The input signal is a noiseless sinusoid, with unity peak amplitude, swept in frequency from a very low frequency up to $f_s/2$ (half the sample rate).

Figure 13-110 Recovery error curves, for various K , versus input tone frequency.



In that figure we see that a $K = 3$ filter (7 taps) exhibits low missing sample recovery error until the input signal's frequency approaches roughly $0.25f_s$, where the recovery error starts to become large. When $K = 5$, the recovery error doesn't become large until the input signal's frequency approaches roughly $0.3f_s$. (The unlabeled curve in [Figure 13-110](#) is a $K = 7$ curve.) So what we see is that to minimize our missing sample recovery error for short-length filters, the maximum input signal bandwidth must be kept substantially lower than the B Hz specified in [Eq. \(13-166\)](#).

The second practical consideration to consider when using this missing sample recovery process is the $w(k)$ window sequence in [Eq. \(13-167\)](#). There seems to be no "best" window sequence that minimizes the recovery error for all real-world signals that we might encounter. So experimentation, using various window functions, becomes necessary. A good place to start is to use either Kaiser or Chebyshev window sequences whose control parameters are set such that the windows' frequency-domain sidelobes are very low relative to their main lobe levels.

We conclude this section by mentioning that reference [\[95\]](#) describes a missing sample recovery technique that is applicable when the pattern of missing samples is more complicated than the simple every Q th sample described here.

13.47 Computing Large DFTs Using Small FFTs

It is possible to compute N -point discrete Fourier transforms (DFTs) using radix-2 fast Fourier

transforms (FFTs) whose sizes are less than N . For example, let's say the largest size FFT software routine we have available is a 1024-point FFT. With the following trick we can combine the results of multiple 1024-point FFTs to compute DFTs whose sizes are greater than 1024.

The simplest form of this idea is computing an N -point DFT using two $N/2$ -point FFT operations. Here's how the trick works for computing a 16-point DFT, of a 16-sample $x(n)$ input sequence, using two 8-point FFTs. First we perform an 8-point FFT on the $x(n)$ samples where $n = 0, 2, 4, \dots, 14$. We'll call those FFT results $X_0(k)$. Then we store two copies of $X_0(k)$ in Memory Array 1 as shown in [Figure 13-111](#). Next we compute an 8-point FFT on the $x(n)$ samples where $n = 1, 3, 5, \dots, 15$. We call those FFT results $X_1(k)$. We store two copies of $X_1(k)$ in Memory Array 3 in [Figure 13-111](#).

Figure 13-111 A 16-point DFT using two 8-point FFTs.

Memory Array 4		Memory Array 1		Memory Array 2		Memory Array 3
$X(0)$		$X_0(0)$		$e^{-j2\pi 0/16}$		$X_1(0)$
$X(1)$		$X_0(1)$		$e^{-j2\pi 1/16}$		$X_1(1)$
$X(2)$		$X_0(2)$		$e^{-j2\pi 2/16}$		$X_1(2)$
...	
$X(6)$		$X_0(6)$		$e^{-j2\pi 6/16}$		$X_1(6)$
$X(7)$		$X_0(7)$		$e^{-j2\pi 7/16}$		$X_1(7)$
$X(8)$	=	$X_0(0)$	+	$e^{-j2\pi 8/16}$	•	$X_1(0)$
$X(9)$		$X_0(1)$		$e^{-j2\pi 9/16}$		$X_1(1)$
$X(10)$		$X_0(2)$		$e^{-j2\pi 10/16}$		$X_1(2)$
...	
$X(14)$		$X_0(6)$		$e^{-j2\pi 14/16}$		$X_1(6)$
$X(15)$		$X_0(7)$		$e^{-j2\pi 15/16}$		$X_1(7)$

In Memory Array 2 we have stored 16 samples of one cycle of the complex exponential $e^{-j2\pi m/N}$, where $N = 16$, and $0 \leq m \leq 15$. Finally we compute our desired 16-point $X(m)$ samples by performing the arithmetic shown in [Figure 13-111](#) on the horizontal rows of the memory arrays. That is,

$$X(0) = X_0(0) + e^{-j2\pi 0/16} X_1(0),$$

$$X(1) = X_0(1) + e^{-j2\pi 1/16} X_1(1),$$

...

$$X(15) = X_0(7) + e^{-j2\pi 15/16} X_1(7).$$

The desired $X(m)$ DFT results are stored in Memory Array 4.

We describe the above process, algebraically, as

$$(13-169)$$

$$X(k) = X_0(k) + e^{-j2\pi k/16} X_1(k)$$

and

(13-169')

$$X(k+8) = X_0(k) + e^{-j2\pi(k+8)/16}X_1(k)$$

for k in the range $0 \leq k \leq 7$.

Notice that we did nothing to reduce the size of Memory Array 2 due to redundancies in the complex exponential sequence $e^{-j2\pi m/N}$. As it turns out, for an N -point DFT, only $N/4$ complex values need be stored in Memory Array 2. The reason for this is that

(13-170)

$$e^{-j2\pi(m+N/2)/N} = -e^{-j2\pi m/N},$$

which involves a simple sign change on $e^{-j2\pi m/N}$. In addition,

(13-170')

$$e^{-j2\pi(m+N/4)/N} = -je^{-j2\pi m/N},$$

which is merely swapping the real and imaginary parts of $e^{-j2\pi m/N}$ plus a sign change of the resulting imaginary part. So [Eqs. \(13-170\)](#) and [\(13-170'\)](#) tell us that only the values $e^{-j2\pi m/N}$ for $0 \leq m \leq N/4-1$ need be stored in Memory Array 2. With that reduced storage idea aside, to be clear regarding exactly what computations are needed for our “multiple-FFTs” technique, we leave Memory Array 2 unchanged from that in [Figure 13-111](#).

The neat part of this “multiple-FFTs” scheme is that our DFT length, N , is not restricted to be an integer power of two. We can use computationally efficient radix-2 FFTs to compute DFTs whose lengths are any integer multiple of an integer power of two. For example, we can compute an $N = 24$ -point DFT using three 8-point FFTs. To do so, we perform an 8-point FFT on the $x(n)$ samples, where $n = 0, 3, 6, \dots, 21$, to obtain $X_0(k)$. Next we compute an 8-point FFT on the $x(n)$ samples, where $n = 1, 4, 7, \dots, 22$, to yield $X_1(k)$. And then we perform an 8-point FFT on the $x(n)$ samples, where $n = 2, 5, 8, \dots, 23$, to obtain an $X_2(k)$ sequence. Finally, we compute our desired 24-point DFT results using

(13-171)

$$X(k) = X_0(k) + e^{-j2\pi k/24}X_1(k) + e^{-j2\pi(2k)/24}X_2(k)$$

(13-171')

$$X(k+8) = X_0(k) + e^{-j2\pi(k+8)/24}X_1(k) + e^{-j2\pi[2(k+8)]/24}X_2(k)$$

and

(13-171'')

$$X(k+16) = X_0(k) + e^{-j2\pi(k+16)/24} \cdot X_1(k) + e^{-j2\pi[2(k+16)]/24} \cdot X_2(k)$$

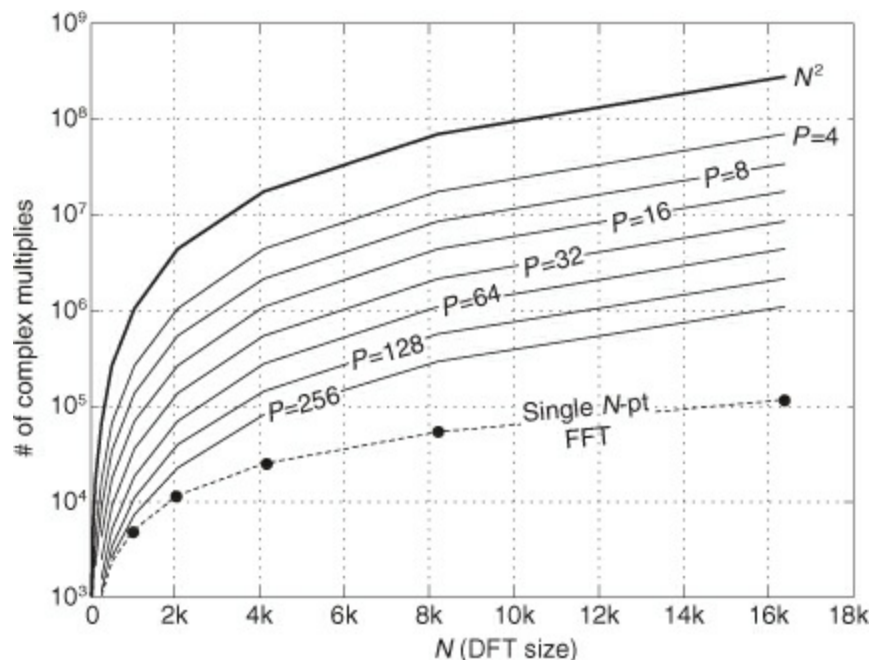
for k in the range $0 \leq k \leq 7$. The memory-array depiction of this process is shown in [Figure 13-112](#), with our final 24-point DFT results residing in Memory Array 6. Memory Array 2 contains $N = 24$ samples of one cycle of the complex exponential $e^{-j2\pi m/24}$, where $0 \leq m \leq 23$. Memory Array 4 contains 24 samples of two cycles of the complex exponential $e^{-j2\pi(2m)/24}$.

Figure 13-112 A 24-point DFT using three 8-point FFTs.

Memory Array 6		Memory Array 1		Memory Array 2		Memory Array 3		Memory Array 4		Memory Array 5
$X(0)$		$X_0(0)$		$e^{-j2\pi 0/24}$		$X_1(0)$		$e^{-j2\pi 0/24}$		$X_2(0)$
$X(1)$		$X_0(1)$		$e^{-j2\pi 1/24}$		$X_1(1)$		$e^{-j2\pi 2/24}$		$X_2(1)$
...	
$X(7)$		$X_0(7)$		$e^{-j2\pi 7/24}$		$X_1(7)$		$e^{-j2\pi 14/24}$		$X_2(7)$
$X(8)$		$X_0(0)$		$e^{-j2\pi 8/24}$		$X_1(0)$		$e^{-j2\pi 16/24}$		$X_2(0)$
$X(9)$		$X_0(1)$		$e^{-j2\pi 9/24}$		$X_1(1)$		$e^{-j2\pi 18/24}$		$X_2(1)$
...	
$X(15)$		$X_0(7)$		$e^{-j2\pi 15/24}$		$X_1(7)$		$e^{-j2\pi 30/24}$		$X_2(7)$
$X(16)$		$X_0(0)$		$e^{-j2\pi 16/24}$		$X_1(0)$		$e^{-j2\pi 32/24}$		$X_2(0)$
$X(17)$		$X_0(1)$		$e^{-j2\pi 17/24}$		$X_1(1)$		$e^{-j2\pi 34/24}$		$X_2(1)$
...	
$X(23)$		$X_0(7)$		$e^{-j2\pi 23/24}$		$X_1(7)$		$e^{-j2\pi 46/24}$		$X_2(7)$

To conclude this section, we state that the larger the size of the FFTs, the more computationally efficient is this “multiple-FFTs” spectrum analysis technique. This behavior is illustrated in [Figure 13-113](#) where we show the number of complex multiplies required by the “multiple-FFTs” algorithm versus the desired DFT size (N). The top bold curve is the number of complex multiplies required by the standard (inefficient) DFT algorithm, and the bottom dashed curve is the number of complex multiplies required by a single N -point radix-2 FFT. The curves in the center of the figure show the number of complex multiplies required by the “multiple-FFTs” algorithm when various FFT sizes (P) are used to compute an N -point DFT. For example, if we must perform a 4096-point DFT using this “multiple-FFTs” algorithm, it’s better for us to perform sixteen 256-point FFTs rather than one hundred twenty-eight 32-point FFTs.

Figure 13-113 Number of complex multiplies versus N .



13.48 Computing Filter Group Delay without Arctangents

Here we present an interesting scheme used to compute the group delay of digital filters that does not require the phase unwrapping process needed when computing arctangents in traditional group delay measurement algorithms. The technique is based on the following: Assume we have the N -sample $h(k)$ impulse response of a digital filter, with k ($0 \leq k \leq N-1$) being our time-domain index, and that we represent the filter's discrete-time Fourier transform (DTFT), $H(\omega)$, in polar form as

$$(13-172)$$

$$H(\omega) = M(\omega)e^{j\phi(\omega)}.$$

In [Eq. \(13-172\)](#), $M(\omega)$ is the frequency magnitude response of the filter, $\phi(\omega)$ is the filter's phase response, and ω is continuous frequency measured in radians/second. Taking the derivative of $H(\omega)$ with respect to ω , and performing a variety of algebraic acrobatics, we can write

$$(13-173)$$

$$\frac{j d[H(\omega)] / d\omega}{M(\omega) \cdot e^{j\phi(\omega)}} = -\frac{d[\phi(\omega)]}{d\omega} + j \frac{d[M(\omega)] / d\omega}{M(\omega)}.$$

So what does that puzzling gibberish in [Eq. \(13-173\)](#) tell us? As it turns out, it tells us a lot if we recall the following items:

- $j d[H(\omega)] / d\omega$ = the DTFT of $k \cdot h(k)$
- $M(\omega) \cdot e^{j\phi(\omega)} = H(\omega)$ = the DTFT of $h(k)$
- $-d[\phi(\omega)] / d\omega$ = group delay of the filter

Now we are able to translate [Eq. \(13-173\)](#) into the meaningful expression

$$(13-173')$$

$$\frac{\text{DTFT}[k \cdot h(k)]}{\text{DTFT}[h(k)]} = \text{group delay} + j \frac{d[M(\omega)] / d\omega}{M(\omega)}.$$

Discretizing expression [\(13-173'\)](#) by replacing the DTFT with the discrete Fourier transform (DFT), we arrive at our scheme for computing the group delay of a digital filter, measured in samples:

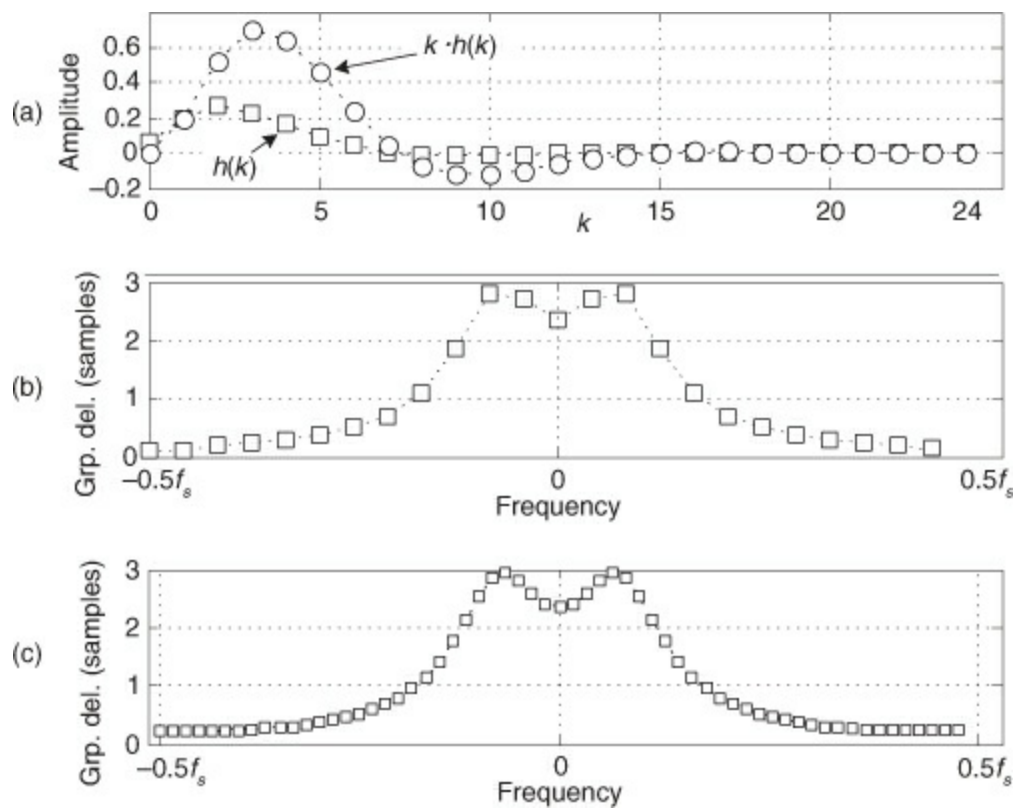
$$(13-174)$$

$$\text{Filter group delay} = \text{real} \left[\frac{\text{DFT}[k \cdot h(k)]}{\text{DFT}[h(k)]} \right].$$

So, starting with a filter's N -sample $h(k)$ impulse response, performing two N -point DFTs and an N -sample complex division, we can compute the filter's passband group delay. (Of course, to improve our group delay granularity we can zero-pad our original $h(k)$ before computing the DFTs). Again, the advantage of the process in expression [\(13-174\)](#) is that the phase unwrapping process needed in traditional group delay algorithms is not needed here. Note that in implementing the process in expression [\(13-174\)](#), we must be prepared to accommodate the situation where a frequency-domain $\text{DFT}[h(k)]$ sample is zero-valued, which will make a group delay sample unrealistically large.

As an example, the square dots in [Figure 13-114\(a\)](#) show the $N = 25$ -sample $h(k)$ impulse response of a 2nd-order IIR lowpass filter. A 25-sample filter group delay estimation, using expression [\(13-174\)](#), is shown in [Figure 13-114\(b\)](#). When we zero-pad the $h(k)$ and $k \cdot h(k)$ sequences to a length of 64 samples ($0 \leq k \leq 63$), expression [\(13-174\)](#) yields the group delay estimate in [Figure 13-114\(c\)](#).

Figure 13-114 Group delay computation: (a) 25-sample $h(k)$ and $k \cdot h(k)$; (b) 25-point group delay; (c) 64-point group delay.

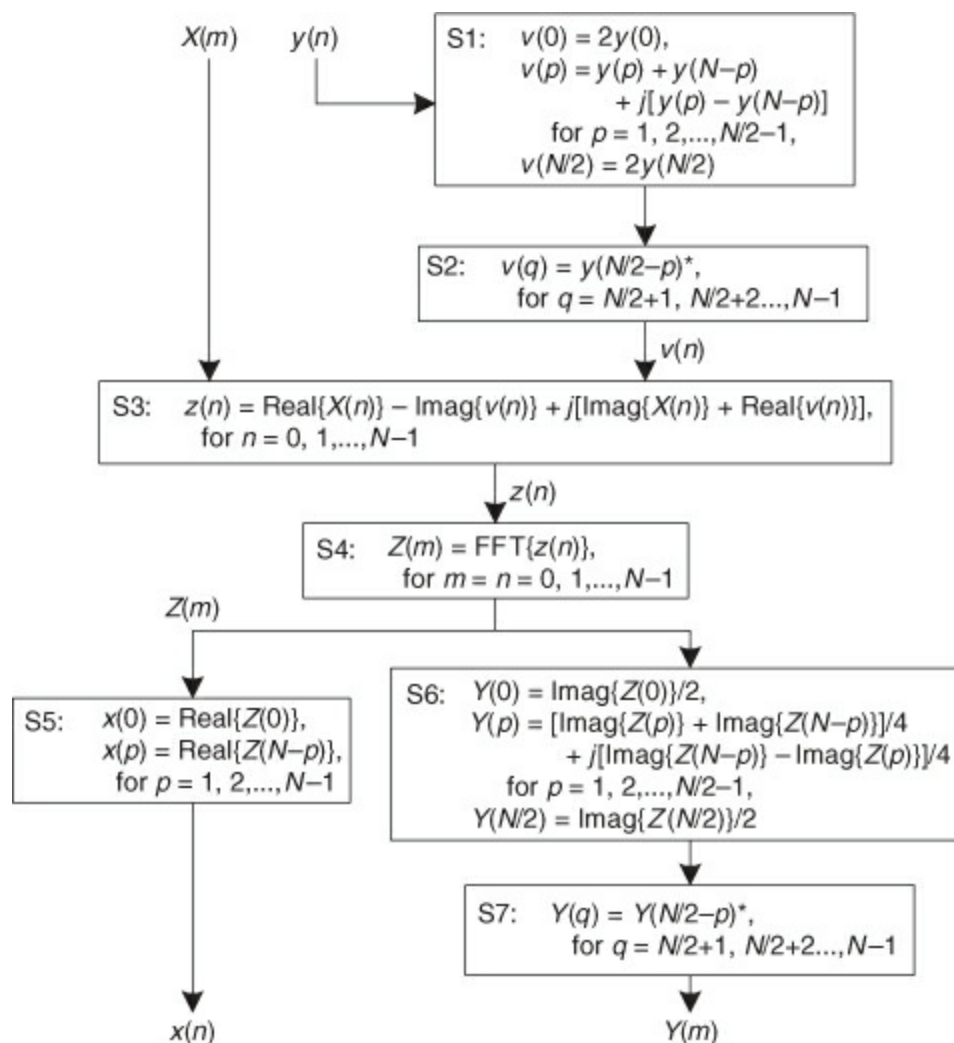


13.49 Computing a Forward and Inverse FFT Using a Single FFT

In [Section 13.5](#) we described the processes of using a single N -point complex FFT to perform both a $2N$ -Point Real FFT and two independent N -Point Real FFTs. This section presents the algorithm for simultaneously computing a forward FFT and an inverse FFT using a single radix-2 FFT[\[96\]](#).

Our algorithm is depicted by the seven steps, S1 through S7, shown in [Figure 13-115](#). In that figure, we compute the $x(n)$ inverse FFT of the N -point frequency-domain conjugate-symmetric input sequence $X(m)$, as well as compute the $Y(m)$ forward FFT of the N -point time-domain real-valued input sequence $y(n)$ using the single complex FFT in Step S4. Sample indices n and m both range from 0 to $N-1$ where N is an integer power of two.

Figure 13-115 Simultaneous FFT and inverse FFT algorithm.



At first glance [Figure 13-115](#) looks more complicated than it actually is, and here's why:

- Steps S1 and S2 create a complex sequence that we call $v(n)$.
- Step S1 generates the first $N/2+1$ samples of $v(n)$ based on the real-valued input sequence $y(n)$.
- Step S2 extends $v(n)$ to a length of N samples and forces $v(n)$ to be conjugate symmetric. The “*” symbol in Step S2 means conjugation.
- Step S3 combines the conjugate-symmetric sequences $X(m)$ and $v(n)$ to create a sequence we call $z(n)$. (Sequence $z(n)$ is not conjugate symmetric.)
- Step S4 is the algorithm's single radix-2 FFT operation, generating complex sequence $Z(m)$.
- Step S5 generates the desired real-valued $x(n)$ time sequence by performing a *circular reversal* of the real part of $Z(m)$. (That is, other than the first sample, the real parts of $Z(m)$ samples are reversed in order to produce $x(n)$. This type of sequence reversal is discussed in [Appendix C](#).)
- Steps S6 and S7 generate the desired frequency-domain $Y(m)$ sequence.
- Step S6 generates the first $N/2+1$ samples of $Y(m)$.
- Step S7 extends the sequence from Step S6 to a length of N samples and forces conjugate symmetry, to produce $Y(m)$. The “*” symbol in Step S7 means conjugation.

The [Figure 13-115](#) algorithm's computational workload is one complex N -point FFT and roughly $2N$ additions/subtractions.

13.50 Improved Narrowband Lowpass IIR Filters

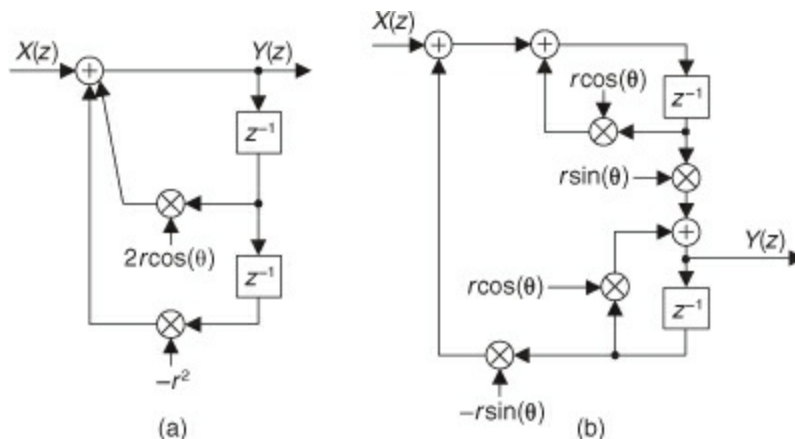
Due to their resistance to quantized-coefficient errors, traditional 2nd-order infinite impulse response (IIR) filters are the fundamental building blocks in computationally efficient high-order IIR digital filter implementations. However, when used in fixed-point number systems, the inherent properties of quantized-coefficient 2nd-order IIR filters do not readily permit their use in narrowband lowpass filtering applications. Narrowband lowpass IIR filters have traditionally had a bad reputation—for example, MATLAB’s Signal Processing Toolbox documentation warns: “All classical IIR lowpass filters are ill-conditioned for extremely low cutoff frequencies.”

This section presents a neat trick to overcome the shortcomings of narrowband 2nd-order lowpass IIR filters, with no increase in filter coefficient bit widths and no increase in the number of filter multiplies per output sample.

13.50.1 The Problem with Narrowband Lowpass IIR Filters

Narrowband lowpass IIR filters are difficult to implement because of intrinsic limitations on their z -plane pole locations. Let’s examine the restrictions on the z -plane pole locations of a standard 2nd-order IIR filter whose structure is shown in [Figure 13-116\(a\)](#).

Figure 13-116 Second-order IIR filters: (a) standard form; (b) coupled form.



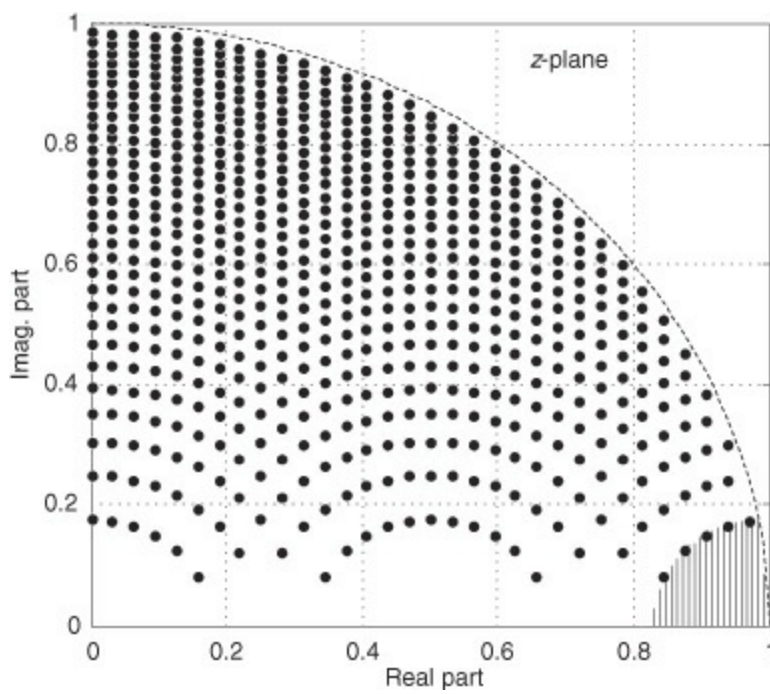
Such an IIR filter, having a transfer function given by

(13-175)

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

has a pair of conjugate poles located at radii of r , at angles of $\pm\theta$ radians. (For filter stability reasons, we always ensure that $r < 1$.) In fixed-point implementations, quantizing the $2r \cos(\theta)$ and $-r^2$ coefficients restricts the possible pole locations[97,98]. On the z -plane, a pole can only reside at the intersection of a vertical line defined by the quantized value of $2r \cos(\theta)$ and a concentric circle whose radius is defined by the square root of the quantized value of r^2 . For example, [Figure 13-117](#) shows the first quadrant of possible z -plane pole locations using five magnitude bits to represent the filter’s two coefficients. Notice the irregular spacing of those permissible pole locations. (Due to trigonometric symmetry, the pole locations in the other three quadrants of the z -plane are mirror images of those shown in [Figure 13-117](#).)

Figure 13-117 Possible pole locations for five magnitude bit coefficient quantization.



So here's the problem we have with standard 2nd-order IIR filters: If we use floating-point software to design a very narrowband (high-order) lowpass IIR filter (implemented as cascaded 2nd-order filters) having poles residing in the shaded area near $z = 1$, subsequent quantizing of the designed filter coefficients to five magnitude bits will make the poles shift to one of the locations shown by the dots on the border of the shaded region in [Figure 13-117](#). Unfortunately that pole shifting, inherent in the [Figure 13-116\(a\)](#) IIR filter implementation due to coefficient quantization in fixed-point systems, prevents us from realizing the desired narrowband lowpass filter. We can always reduce the size of the shaded forbidden zone near $z = 1$ in [Figure 13-117](#) by increasing the number of bits used to represent the 2nd-order filters' coefficients. However, in some filter implementation scenarios increasing coefficient binary-word bit widths may not be a viable option.

One solution to the above problem is to use the so-called *coupled-form* IIR filter (also called the Gold-Rader filter[\[99\]](#)) structure, shown in [Figure 13-116\(b\)](#), having a transfer function given by

(13-176)

$$H_{cf}(z) = \frac{r \sin(\theta) z^{-1}}{1 - 2r \cos(\theta) z^{-1} + r^2 z^{-2}}.$$

Because the coupled-form filter's quantized coefficients in [Figure 13-116\(b\)](#) are linear in $r \cos(\theta)$ and $r \sin(\theta)$, its possible pole locations are on a regularly spaced grid on the z -plane defined by $z = r \cos(\theta) + j r \sin(\theta)$. This enables us to build 2nd-order narrowband lowpass IIR filters with poles in the desired shaded region of [Figure 13-117](#).

This pole placement behavior is a beautiful example of the difference between apparently equivalent filter implementations. With infinite-precision coefficients the standard and coupled-form IIR filters, having identical denominators in their transfer functions, will have identical z -plane pole locations. But with quantized coefficients the two filters will have different pole locations.

Back to our problem. While the coupled-form IIR filter gives us increased flexibility in placing z -plane poles for lowpass filtering, unfortunately, this coupled-form implementation requires twice the number of multiplies needed by the standard 2nd-order IIR filter in [Figure 13-116\(a\)](#).

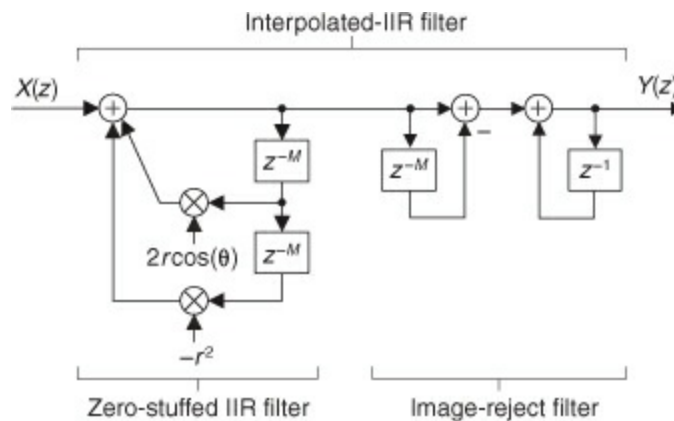
In the following material we describe a slick narrowband lowpass IIR filter structure, proposed by Harris and Loudermilk, having poles residing in the shaded region of [Figure 13-117](#) with no increase

in coefficient bit width and no additional multiplication operations beyond those needed for a standard 2nd-order IIR filter[100].

13.50.2 An Improved Narrowband Lowpass IIR Filter

The improved lowpass IIR filter is created by replacing each unit-delay element in a standard 2nd-order IIR filter with multiple unit-delay elements as shown in the left portion of [Figure 13-118](#). This *zero-stuffed IIR filter* will retain its original lowpass passband and have multiple passband images, exactly as did the interpolated finite impulse response (IFIR) filters that we studied in [Chapter 7](#). The zero-stuffed IIR filter is followed by a lowpass image-reject filter that attenuates those unwanted passband images. Given this cascaded structure, which we'll demonstrate shortly, we call the filter combination in [Figure 13-118](#) an *interpolated infinite impulse response* (interpolated-IIR) filter.

Figure 13-118 Interpolated-IIR filter.



The M -length delay lines, where M is a positive integer, in the zero-stuffed IIR filter shift a standard IIR filter's conjugate poles, originally located at $z = re^{\pm j\theta}$, to the new locations of

$$(13-177)$$

$$z_{\text{new}} = \sqrt[M]{r} \cdot e^{\pm j\theta/M}.$$

That is, the new conjugate pole locations are at radii of the M th root of r , at angles of $\pm\theta/M$ radians. Happily, those interpolated-IIR filter pole locations can now reside in the desired shaded region of [Figure 13-117](#) without using more bits to represent the zero-stuffed IIR filter's coefficients.

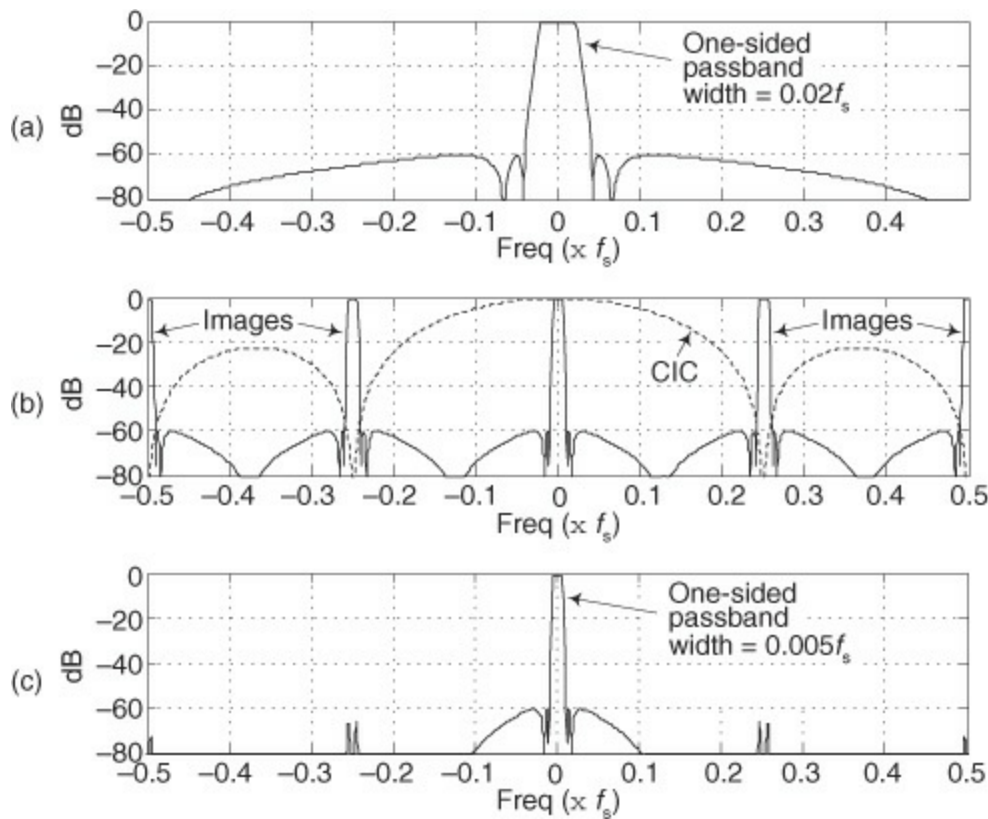
If the original [Figure 13-116\(a\)](#) 2nd-order IIR filter contains feedforward coefficients, those coefficients are also delayed by M -length delay lines.

13.50.3 Interpolated-IIR Filter Example

Let's show an example of an interpolated-IIR filter in action. With f_s representing a filter's input signal sample rate in Hz, assume we want to implement a recursive lowpass filter whose one-sided passband width is $0.005f_s$ with a stopband attenuation greater than 60 dB. If we choose to set $M = 4$, then we start our interpolated-IIR filter design process by designing a standard IIR filter having a one-sided passband width of $M \cdot 0.005f_s = 0.02f_s$. Using our favorite IIR filter design software (for an elliptic IIR filter in this example), we obtain a 5th-order prototype IIR filter. Partitioning that 5th-order prototype IIR filter into two 2nd-order and one single-order IIR filter sections, all in cascade and having coefficients represented by 12-bit words, yields the frequency magnitude response shown in [Figure 13-119\(a\)](#).

Figure 13-119 Frequency magnitude responses: (a) original IIR prototype filter; (b) zero-stuffed

interpolated-IIR filter and CIC filters (dashed); (c) final narrowband 12-bit coefficient filter.



Next, replacing the unit-delay elements in the filter sections with $M = 4$ unit-delay elements results in the frequency magnitude response shown in [Figure 13-119\(b\)](#). There we see the multiple narrowband passband images induced by the $M = 4$ -length delay lines of the interpolated-IIR filter. Our final job is to attenuate those unwanted passband images. We can do so by following the cascaded increased-delay IIR filter sections with a cascaded integrator-comb (CIC) filter, whose structure is shown on the right side of [Figure 13-118](#). (The CIC filter is computationally advantageous because it requires no multiplications.) To satisfy our desired 60 dB stopband attenuation requirement, we use a 2nd-order CIC filter—two 1st-order CIC filters in cascade—to attenuate the passband images in [Figure 13-119\(b\)](#). The result of our design is the interpolated-IIR and CIC filter combination whose composite frequency magnitude response meets our filter requirements as shown [Figure 13-119\(c\)](#).

In practice, 2nd-order subfilters may have large gains requiring unpleasantly large bit-width multipliers and large bit-width registers to store intermediate results. For this reason it may be necessary to scale the IIR subfilters' coefficients as discussed in [Chapter 6](#), or truncate the subfilters' output samples, in order to avoid undesirably large bit-width processing.

To recap this material, we discussed the limitations encountered when using traditional 2nd-order quantized-coefficient IIR filters to perform narrowband lowpass filtering and mentioned the coupled-form IIR filter that reduced those limitations albeit with an increased computational cost of doubling the number of multiplies per filter output sample. Next we described, and then demonstrated, an interpolated-IIR filter that overcomes the shortcomings of traditional lowpass IIR filters. The interpolated-IIR filter provides improved lowpass IIR filter performance while requiring no increase in filter coefficient bit widths and no additional multiply operations beyond a traditional IIR filter. When it comes to narrowband lowpass IIR filters, there's a new sheriff in town.

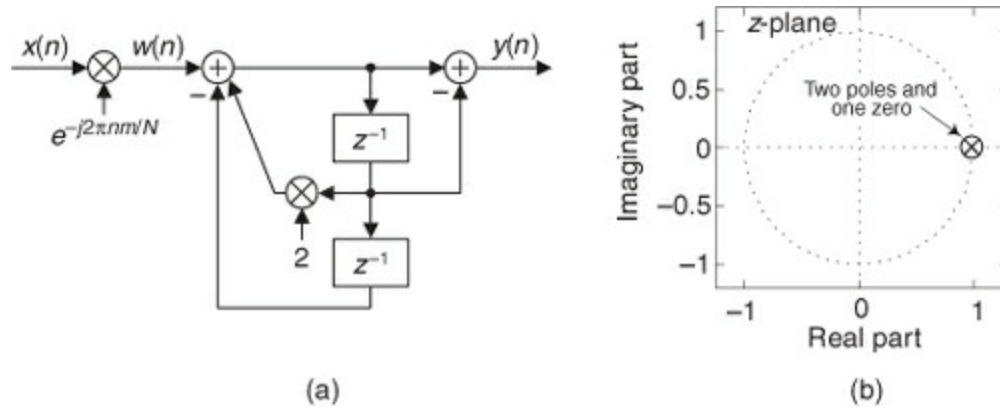
13.51 A Stable Goertzel Algorithm

In [Section 13.17.1](#) we discussed the computational value of the Goertzel algorithm for computing

discrete Fourier transform (DFT) spectral components. However, we also mentioned that the [Figure 13-42](#) complex resonator implementation of the Goertzel algorithm places resonator z -domain poles on the z -plane's unit circle. Having a resonator pole on the unit circle leads to potential instability problems because we cannot represent the resonator's coefficients with infinite precision. We're forced to represent the coefficients as accurately as a fixed number of binary bits allows. This means the resonator's poles will not lie exactly on the unit circle. If an imprecise binary representation of the coefficient $2\cos(2\pi m/N)$ places the poles slightly inside the z -plane's unit circle, then the computed $X(m)$ spectral sample will contain a small error. Even worse, if an imprecise binary representation of $2\cos(2\pi m/N)$ places the poles slightly outside the unit circle, then the resonator is unstable. For this reason, typical applications of the Goertzel algorithm restrict the transform length N to be in the hundreds.

One way to avoid those potential stability problems, and let N be any value we wish, is by way of a heterodyning scheme. That is, instead of building an imperfect resonator centered at our frequency of interest, $2\pi m/N$ radians/sample, we frequency translate our signal of interest down to zero frequency where we can build a perfect resonator as shown in [Figure 13-120\(a\)](#). We say "perfect resonator" because that resonator, centered at zero frequency (frequency index $m = 0$), has coefficients of two and one, which can be represented by binary words with perfect precision.

Figure 13-120 Stable Goertzel algorithm: (a) resonator implementation; (b) z -plane poles and zero.



Such a resonator has a z -domain transfer function of

$$(13-178)$$

$$H_{G,\text{stable}}(z) = \frac{Y(z)}{W(z)} = \frac{1 - z^{-1}}{1 - 2z^{-1} + z^{-2}}$$

with a single z -domain zero located at $z = 1$ and two poles at $z = 1$ as shown in [Figure 13-120\(b\)](#). One of the poles cancels the zero at $z = 1$. The advantages of the network in [Figure 13-120\(a\)](#) are that it is guaranteed stable, and it exhibits no output error due to a pole or zero being slightly inside or outside the z -plane unit circle.

Now the perceptive reader would factor [Eq. \(13-178\)](#) as

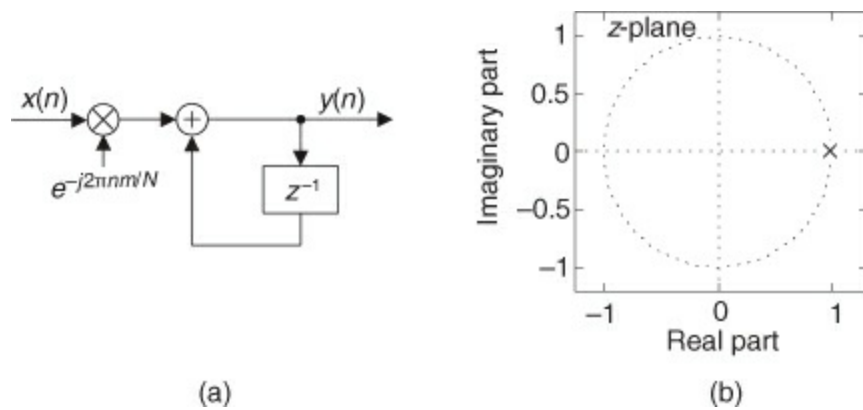
$$(13-179)$$

$$H_{G,\text{stable}}(z) = \frac{1 - z^{-1}}{(1 - z^{-1})(1 - z^{-1})} = \frac{1}{1 - z^{-1}}$$

and redraw [Figure 13-120\(a\)](#) as shown in [Figure 13-121\(a\)](#).

Figure 13-121 Simplified stable Goertzel algorithm: (a) simplified resonator implementation; (b) z -

plane pole.



[Figure 13-121\(a\)](#) tells us that our desired $X(m) = y(n)$ spectral sample is equal to the sum of the N samples output by the multiplier in [Figure 13-121\(a\)](#). (This makes perfect sense because the zero-frequency spectral sample of an N -point DFT, $X(0)$, is computed by merely summing a DFT's N input samples.) So our “stable Goertzel algorithm” now becomes quite simple.

Ah, but there's trouble in paradise. The “weak link in the chain” of the [Figure 13-121\(a\)](#) network is that we're assuming the heterodyning sequence $e^{-j2\pi m/N}$ is ideal in its precision. If you've ever tried to generate a complex $e^{-j2\pi m/N}$ sequence using binary arithmetic, you know that your sequence must be quantized to some fixed number of bits, and thus have imperfect precision. That means the output of your $e^{-j2\pi m/N}$ oscillator will either increase in magnitude, or decrease in magnitude, as time index n increases. However, we solve that problem by using the guaranteed-stable quadrature oscillator described in [Section 13.32](#).

It's fair to copy a slogan from the Aston Martin automobile company and say that the [Figure 13-121\(a\)](#) Goertzel algorithm, using the stable quadrature oscillator, is “engineered to exceed all expectations.”

References

- [1] Powell, S. “Design and Implementation Issues of All Digital Broadband Modems,” *DSP World Workshop Proceedings*, Toronto, Canada, September 13–16, 1998, pp. 127–142.
- [2] Frerking, M. *Digital Signal Processing in Communications Systems*, Chapman & Hall, New York, 1994, p. 330.
- [3] Jacobsen, E., Minister of Algorithms, Abineau Communications, private communication, September 11, 2003.
- [4] Palacherls, A. “DSP-mP Routine Computes Magnitude,” *EDN*, October 26, 1989.
- [5] Mikami, N., Kobayashi, M., and Yokoyama, Y. “A New DSP-Oriented Algorithm for Calculation of the Square Root Using a Nonlinear Digital Filter,” *IEEE Trans. on Signal Processing*, Vol. 40, No. 7, July 1992.
- [6] Lyons, R. “Turbocharge Your Graphics Algorithm,” *ESD: The Electronic System Design Magazine*, October 1988.
- [7] Adams W., and Brady, J. “Magnitude Approximations for Microprocessor Implementation,” *IEEE Micro*, Vol. 3, No. 5, October 1983.
- [8] Harris Semiconductor Corp. HSP50110 Digital Quadrature Tuner Data Sheet, File Number 3651,

February 1994.

- [9] Sabin, W., and Schoenike, E., eds., *Single Sideband Systems and Circuits*, McGraw-Hill, New York, 1987.
- [10] Schreiner, C. "Subject: Re: Approximation for Sum of Two Squares," Usenet group *comp.dsp* post, October 4, 1999.
- [11] Filip, A. "Linear Approximations to $\sqrt{x^2+y^2}$ Having Equiripple Error Characteristics," *IEEE Trans. on Audio and Electroacoustics*, December 1973, pp. 554–556.
- [12] Bingham, C., Godfrey, M., and Tukey, J. "Modern Techniques for Power Spectrum Estimation," *IEEE Trans. on Audio and Electroacoust.*, Vol. AU-15, No. 2, June 1967.
- [13] Bergland, G. "A Guided Tour of the Fast Fourier Transform," *IEEE Spectrum Magazine*, July 1969, p. 47.
- [14] Harris, F. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proceedings of the IEEE*, Vol. 66, No. 1, January 1978.
- [15] Nuttall, A. "Some Windows with Very Good Sidelobe Behavior," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-29, No. 1, February 1981.
- [16] Cox, R. "Complex-Multiply Code Saves Clocks Cycles," *EDN*, June 25, 1987.
- [17] Rabiner, L., and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1975.
- [18] Sorenson, H., Jones, D., Heideman, M., and Burrus, C. "Real-Valued Fast Fourier Transform Algorithms," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-35, No. 6, June 1987.
- [19] Cooley, J., Lewis, P., and Welch, P. "The Fast Fourier Transform Algorithm: Programming Considerations in the Calculation of Sine, Cosine and Laplace Transforms," *Journal Sound Vib.*, Vol. 12, July 1970.
- [20] Brigham, E. *The Fast Fourier Transform and Its Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.
- [21] Burrus, C., et al. *Computer-Based Exercises for Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1994, p. 53.
- [22] Hewlett-Packard, "The Dynamic Range Benefits of Large-Scale Dithered Analog-to-Digital Conversion," *HP Product Note*: 89400-7.
- [23] Blesser, B., and Locanthi, B. "The Application of Narrow-Band Dither Operating at the Nyquist Frequency in Digital Systems to Provide Improved Signal-to-Noise Ratio over Conventional Dithering," *J. Audio Eng. Soc.*, Vol. 35, June 1987.
- [24] Coleman, B., et al. "Coherent Sampling Helps When Specifying DSP A/D Converters," *EDN*, October 1987.
- [25] Ushani, R. "Classical Tests Are Inadequate for Modern High-Speed Converters," *EDN Magazine*, May 9, 1991.
- [26] Meehan, P., and Reidy, J. "FFT Techniques Give Birth to Digital Spectrum Analyzer," *Electronic Design*, August 11, 1988, p. 120.
- [27] Beadle, E. "Algorithm Converts Random Variables to Normal," *EDN Magazine*, May 11, 1995.

- [28] Spiegel, M. *Theory and Problems of Statistics*, Shaum's Outline Series, McGraw-Hill, New York, 1961, p. 142.
- [29] Davenport, W., Jr., and Root, W. *Random Signals and Noise*, McGraw-Hill, New York, 1958.
- [30] Salibrici, B. "Fixed-Point DSP Chip Can Generate Real-Time Random Noise," *EDN Magazine*, April 29, 1993.
- [31] Marsaglia, G., and Tsang, W. "The Ziggurat Method for Generating Random Variables," *Journal of Statistical Software*, Vol. 5, No. 8, 2000.
- [32] http://finmath.uchicago.edu/~wilder/Code/random/Papers/Marsaglia_00_ZMGRV.pdf.
- [33] <http://www.jstatsoft.org/v05/i08/ziggurat.pdf>.
- [34] Donadio, M. "Lost Knowledge Refound: Sharpened FIR Filters," *IEEE Signal Processing Magazine*, Vol. 20, No. 5, September 2003, pp. 61–63.
- [35] Kwentus, A., et al. "Application of Filter Sharpening to Cascaded Integrator-Comb Decimation Filters," *IEEE Transactions on Signal Processing*, Vol. 45, February 1997, pp. 457–467.
- [36] Gentili, P., et al. "Improved Power-of-Two Sharpening Filter Design by Genetic Algorithm," 1996 IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP '96), Atlanta, Georgia, Vol. 3, 1996, p. 1375.
- [37] Graychip Inc. "Upconverting Signals with the GC2011 for Easier Digital to Analog Conversion," Application Note: GC2011-AN9804, December 20, 1998.
- [38] Donadio, M., private communication, September 11, 2003.
- [39] Jacobsen, E., and Kootsookos, P. "Fast, Accurate Frequency Estimators," *IEEE Signal Processing Magazine*, "DSP Tips & Tricks" column, Vol. 24, No. 3, May 2007.
- [40] Nagai, K. "Pruning the Decimation-in-Time FFT Algorithm with Frequency Shift," *IEEE Trans. on ASSP*, Vol. ASSP-34, August 1986, pp. 1008–1010.
- [41] Skinner, D. "Pruning the Decimation-in-Time FFT Algorithm," *IEEE Trans. on ASSP*, Vol. ASSP-24, April 1976, pp. 193–194.
- [42] Markel, J. D. "FFT Pruning," *IEEE Trans on Audio Electroacoust.*, Vol. AU-19, December 1971, pp. 305–311.
- [43] Sreenivas, T., and Rao, P. "FFT Algorithm for Both Input and Output Pruning," *IEEE Trans. on ASSP*, Vol. ASSP-27, June 1979, pp. 291–292.
- [44] Lyons, R. "Program Aids Analysis of FFT Algorithms," *EDN Magazine*, August 6, 1987.
- [45] Goertzel, G. "An Algorithm for the Evaluation of Finite Trigonometric Series," *American Math. Monthly*, Vol. 65, 1958, pp. 34–35.
- [46] Proakis, J., and Manolakis, D. *Digital Signal Processing: Principles, Algorithms, and Applications*, 3rd ed., Prentice Hall, Upper Saddle River, New Jersey, 1996, pp. 480–481.
- [47] Oppenheim, A., Schaffer, R., and Buck, J. *Discrete-Time Signal Processing*, 2nd ed., Prentice Hall, Upper Saddle River, New Jersey, 1999, pp. 633–634.
- [48] Farhang-Boroujeny, B., and Lim, Y. "A Comment on the Computational Complexity of Sliding FFT," *IEEE Trans. Circuits and Syst. II*, Vol. 39, No. 12, December 1992, pp. 875–876.
- [49] Farhang-Boroujeny, B., and Gazor, S. "Generalized Sliding FFT and Its Application to

Implementation of Block LMS Adaptive Filters,” *IEEE Trans. Sig. Proc.*, Vol. 42, No. 3, March 1994, pp. 532–538.

- [50] Douglas, S., and Soh, J. “A Numerically-Stable Sliding-Window Estimator and Its Application to Adaptive Filters,” *Proc. 31st Annual Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, California, Vol. 1, November 1997, pp. 111–115.
- [51] Crochiere, R., and Rabiner, L. *Multirate Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1983, pp. 315–319.
- [52] Zoran Corp. “Vernier Spectral Analysis with the ZR34161 Vector Signal Processor,” *Tech. Note ZAN34003*, Santa Clara, California, 1989.
- [53] Gumas, C. “Window-Presum FFT Achieves High-Dynamic Range, Resolution,” *Personal Engineering and Instrumentation News*, July 1997, pp. 58–64.
- [54] Hack, T. “IQ Sampling Yields Flexible Demodulators,” *RF Design*, April 1991.
- [55] Bateman, A. “Quadrature Frequency Discriminator,” *GlobalDSP Magazine*, October 2002.
- [56] <http://aulos.calarts.edu/pipermail/test/1998-March/001028.html>.
- [57] Dick, C., and Harris, F. “FPGA Signal Processing Using Sigma-Delta Modulation,” *IEEE Signal Proc. Magazine*, Vol. 17, No. 1, January 2000.
- [58] Bateman, A. “Implementing a Digital AC Coupling Filter,” *GlobalDSP Magazine*, February 2003.
- [59] Shenoi, K. *Digital Signal Processing in Communications Systems*, Chapman & Hall, New York, 1994, p. 275.
- [60] Bristow-Johnson, R. “Subject: Fixed-PointDC Blocking Filter with Noise Shaping,” Usenet group *comp.dsp* post, June 22, 2000.
- [61] Bristow-Johnson, R. “Subject: Virtues of Noise Shaping,” Usenet group *comp.dsp* post, August 21, 2001.
- [62] Ascari, L., et al. “Low Power Implementation of a Sigma Delta Decimation Filter for Cardiac Applications,” *IEEE Instrumentation and Measurement Technology Conference*, Budapest, Hungary, May 21–23, 2001, pp. 750–755.
- [63] Gao, Y., et al. “Low-Power Implementation of a Fifth-Order Comb Decimation Filter for Multi-Standard Transceiver Applications,” *Int. Conf. on Signal Proc. Applications and Technology (ICSPAT)*, Orlando, Florida, 1999.
- [64] Gao, Y., et al. “A Comparison Design of Comb Decimators for Sigma-Delta Analog-to-Digital Converters,” *Int. Journal: Analog Integrated Circuits and Signal Processing*, Kluwer Academic Publishers, ISSN: 0925–1030, 1999.
- [65] Ballanger, M., et al. “Digital Filtering by Polyphase Network: Application to Sample-Rate Alteration and Filter Banks,” *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-24, No. 2, April 1976, pp. 109–114.
- [66] Brandt, B., and Wooley, B. “A Low-Power Area-Efficient Digital Filter for Decimation and Interpolation,” *IEEE Journ. of Solid-State Circuits*, Vol. 29, June 1994, pp. 679–687.
- [67] Willson, A., Jr. “A Programmable Interpolation Filter for Digital Communications Applications,” Final report for MICRO Project 96–149, UCLA, 1996–1997.

- [68] Dumonteix, Y., et al. "Low Power Comb Decimation Filter Using Polyphase Decomposition for Mono-Bit $\Sigma\Delta$ Analog-to-Digital Converters," Int. Conf. on Signal Processing Applications and Technology (ICSPAT), San Jose, California, 2000.
- [69] Yang, H., and Snelgrove, W. "High Speed Polyphase CIC Decimation Filters," *IEEE Int. Symposium on Circuits and Systems*, Vol. 2, 1996, pp. 229–232.
- [70] Jang, Y., and Yang, S. "Non-Recursive Cascaded Integrator-Comb Decimation Filters with Integer Multiple Factors," 44th IEEE Midwest Symposium on Circuits and Systems (MWSCAS), Dayton, Ohio, August 2001.
- [71] Dvorak, R. "Software Filter Boosts Signal-Measurement Stability, Precision," *Electronic Design*, February 3, 2003.
- [72] Lynn, P., and Fuerst, W. *Introductory Digital Signal Processing, with Computer Applications*, John Wiley and Sons, New York, 1997, pp. 285–297.
- [73] Givens, M., private communication, October 12, 2004.
- [74] Fraser, D. "Interpolation by the FFT Revisited—An Experimental Investigation," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-37, No. 5, May 1989, pp. 665–676.
- [75] Marple, S., Jr. "Computing the Discrete-Time 'Analytic' Signal via FFT," *IEEE Trans. on Signal Proc.*, Vol. 47, No. 9, September 1999, pp. 2600–2603.
- [76] Harris, F. "T102: Digital Signal Processing for Digital Modems," DSP World Spring Design Conf., Santa Clara, California, April 1999.
- [77] Harris, F. "On the Design, Implementation, and Performance of a Microprocessor-Controlled AGC System for a Digital Receiver," IEEE Military Communications Conf., San Diego, California, October 1988.
- [78] Analog Devices, Inc. "80 MSPS, Dual-Channel WCDMA Receive Signal Processor (RSP) AD6634," Data Sheet Rev. 0, 2002, pp. 28–34.
- [79] Turner, C. "Recursive Discrete-Time Sinusoidal Oscillators," *IEEE Signal Processing Magazine*, Vol. 20, No. 3, May 2003, pp. 103–111.
- [80] Paillard, B., and Boudreau, A. "Fast, Continuous, Sinewave Generator," *GlobalDSP On-line Magazine*, December 2003.
- [81] Vassilevsky, V. "Efficient Multi-tone Detection," *IEEE Signal Processing Magazine*, "DSP Tips & Tricks" column, Vol. 24, No. 2, March 2007.
- [82] Shiung, D., Ferng, H., and Lyons, R. "Filtering Tricks for FSK Demodulation," *IEEE Signal Processing Magazine*, "DSP Tips & Tricks" column, Vol. 22, No. 3, May 2005.
- [83] Spiegel, M. *Statistics*, Shaum's Outline Series, McGraw-Hill, New York, 1961, p. 77.
- [84] Hadstate, J. "Subject: Re: Question about Computing a "Moving Variance," Usenet group *comp.dsp* post, March 1, 2005.
- [85] Turner, C. "Subject: Re: Question About Computing a "Moving Variance," Usenet group *comp.dsp* post, February 27, 2005.
- [86] Jackson, L. "On the Relationship Between Digital Hilbert Transformers and Certain Low-Pass Filters," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-23, No. 4, August 1975.

- [87] Harris, F. *Multirate Signal Processing for Communication Systems*, Prentice Hall, Upper Saddle River, New Jersey, 2004, pp. 210–212.
- [88] Turner, C. “Subject: How Do You Rotate a Phasor by $\pi/8$ Radians,” Usenet group *comp.dsp* post, May 29, 2002.
- [89] Press, W., et al., *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, New York, 1992, p. 177.
- [90] Losada, R., and Lyons, R. “Reducing CIC Filter Complexity,” *IEEE Signal Processing Magazine*, “DSP Tips and Tricks” column, Vol. 23, No. 4, July 2006.
- [91] Ohlsson, H., et al. “Design of a Digital Down Converter Using High Speed Digital Filters,” in *Proc. Symp. on Gigahertz Electronics*, Gothenburg, Sweden, March 13–14, 2000, pp. 309–312.
- [92] Turner, C. “Slope Filtering: An FIR Approach to Linear Regression,” *IEEE Signal Processing Magazine*, “DSP Tips & Tricks” column, Vol. 25, No. 6, November 2008.
- [93] Bell, D. “Subject: Re: Frequency Inversion,” Usenet group *comp.dsp* post, August 30, 2006.
- [94] Adams, R. “Nonuniform Sampling of Audio Signals,” *J. Audio Eng. Soc.*, Vol. 40, No. 11, November 1992, pp. 886–894.
- [95] Bariska, A. “Recovering Periodically-Spaced Missing Samples,” *IEEE Signal Processing Magazine*, “DSP Tips and Tricks” column, Vol. 24, No. 6, November 2007.
- [96] Moshe, S., and Hertz, D. “On Computing DFT of Real N-Point Vector and IDFT of DFT-Transformed Real N-Point Vector via Single DFT,” *IEEE Signal Processing Letters*, IEEE, Vol. 6, No. 6, June 1999, p. 141.
- [97] Proakis, J., and Manolakis, D. *Digital Signal Processing: Principles, Algorithms, and Applications*, 3rd ed., Prentice Hall, Upper Saddle River, New Jersey, 1996, pp. 572–576.
- [98] Oppenheim, A., and Schaffer, R. *Discrete-Time Signal Processing*, 2nd ed., Prentice Hall, Englewood Cliffs, New Jersey, 1989, pp. 382–386.
- [99] Gold, B., and Rader, C. “Effects of Parameter Quantization on the Poles of a Digital Filter,” *Proceedings of the IEEE*, Vol. 55, May 1967, pp. 688–689.
- [100] Harris, F., and Lowdermilk, W. “Implementing Recursive Filters with Large Ratio of Sample Rate to Bandwidth,” in *Conference Record of the Forty-first Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, California, November 4–7, 2007, pp. 1149–1153.

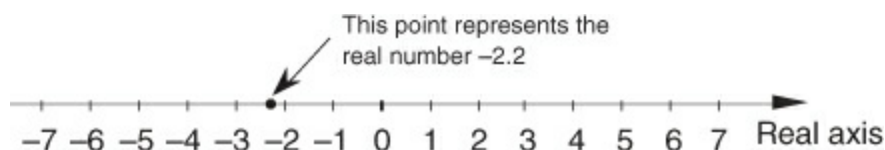
Appendix A. The Arithmetic of Complex Numbers

To understand digital signal processing, we have to get comfortable using complex numbers. The first step toward this goal is learning to manipulate complex numbers arithmetically. Fortunately, we can take advantage of our knowledge of real numbers to make this job easier. Although the physical significance of complex numbers is discussed in [Chapter 8](#), the following discussion provides the arithmetic rules governing complex numbers.

A.1 Graphical Representation of Real and Complex Numbers

To get started, real numbers are those positive or negative numbers we're used to thinking about in our daily lives. Examples of real numbers are 0.3, -2.2, 5.1, etc. Keeping this in mind, we see how a real number can be represented by a point on a one-dimensional axis, called the *real axis*, as shown in [Figure A-1](#).

Figure A-1 The representation of a real number as a point on the one-dimensional real axis.

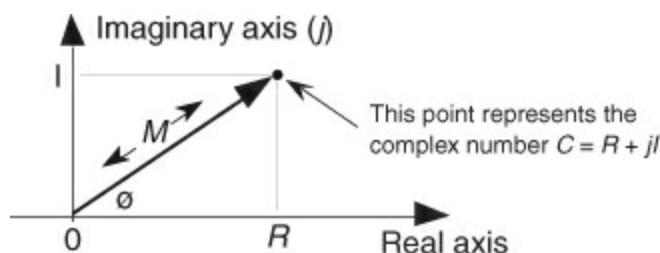


We can, in fact, consider that all real numbers correspond to all of the points on the real axis line on a one-to-one basis.

A complex number, unlike a real number, has two parts: a real part and an imaginary part. Just as a real number can be considered to be a point on the one-dimensional real axis, a complex number can be treated as a point on a complex plane as shown in [Figure A-2](#). We'll use this geometrical concept to help us understand the arithmetic of complex numbers.[‡]

[‡] The complex plane representation of a complex number is sometimes called an *Argand diagram*—named after the French mathematician Jean Robert Argand (1768–1825).

Figure A-2 The phasor representation of the complex number $C = R + jI$ on the complex plane.



A.2 Arithmetic Representation of Complex Numbers

A complex number C is represented in a number of different ways in the literature, such as

(A-1)

Rectangular form: →

$$C = R + jI,$$

(A-1')

Trigonometric form: →

$$C = M[\cos(\theta) + j\sin(\theta)],$$

(A-1'')

Exponential form: →

$$C = M e^{j\theta},$$

$$(A-1''')$$

Magnitude and angle form: \rightarrow

$$C = M \angle \phi.$$

[Equations \(A-1''\)](#) and [\(A-1'''\)](#) remind us that the complex number C can also be considered the tip of a phasor on the complex plane, with magnitude M , in the direction of ϕ degrees relative to the positive real axis as shown in [Figure A-2](#). (We'll avoid calling phasor M a vector because the term *vector* means different things in different contexts. In linear algebra, *vector* is the term used to signify a one-dimensional matrix. On the other hand, in mechanical engineering and field theory, vectors are used to signify magnitudes and directions, but there are vector operations (*scalar* or *dot product*, and *vector* or *cross-product*) that don't apply to our definition of a phasor. The relationships between the variables in this figure follow the standard trigonometry of right triangles. Keep in mind that C is a complex number, and the variables R , I , M , and ϕ are all real numbers. The magnitude of C , sometimes called the *modulus* of C , is

$$(A-2)$$

$$M = |C| = \sqrt{R^2 + I^2},$$

and, by definition, the phase angle, or *argument*, of C is the arctangent of I/R , or

$$(A-3)$$

$$\phi = \tan^{-1}\left(\frac{I}{R}\right).$$

The variable ϕ in [Eq. \(A-3\)](#) is a general angle term. It can have dimensions of degrees or radians. Of course, we can convert back and forth between degrees and radians using π radians = 180° . So, if ϕ_r is in radians and ϕ_d is in degrees, then we can convert ϕ_r to degrees by the expression

$$(A-4)$$

$$\phi_d = \frac{180\phi_r}{\pi}.$$

Likewise, we can convert ϕ_d to radians by the expression

$$(A-5)$$

$$\phi_r = \frac{\pi\phi_d}{180}.$$

The exponential form of a complex number has an interesting characteristic that we need to keep in mind. Whereas only a single expression in rectangular form can describe a single complex number, an infinite number of exponential expressions can describe a single complex number; that is, while, in the exponential form, a complex number C can be represented by $C = Me^{j\phi}$, it can also be represented by

$$(A-6)$$

$$C = Me^{j\phi} = Me^{j(\phi + 2\pi n)},$$

where $n = \pm 1, \pm 2, \pm 3, \dots$ and ϕ is in radians. When ϕ is in degrees, [Eq. \(A-6\)](#) is in the form

$$(A-7)$$

$$C = Me^{j\phi} = Me^{j(\phi + n360^\circ)}.$$

[Equations \(A-6\)](#) and [\(A-7\)](#) are *almost* self-explanatory. They indicate that the point on the complex plane represented by the tip of the phasor C remains unchanged if we rotate the phasor some integral

multiple of 2π radians or an integral multiple of 360° . So, for example, if $C = Me^{j(20^\circ)}$, then

$$(A-8)$$

$$C = Me^{j(20^\circ)} = Me^{j(380^\circ)} = Me^{j(740^\circ)}.$$

The variable ϕ , the angle of the phasor in [Figure A-2](#), need not be constant. We'll often encounter expressions containing a complex sinusoid that takes the form

$$(A-9)$$

$$C = Me^{j\omega t}.$$

[Equation \(A-9\)](#) represents a phasor of magnitude M whose angle in [Figure A-2](#) is increasing linearly with time at a rate of ω radians each second. If $\omega = 2\pi$, the phasor described by [Eq. \(A-9\)](#) is rotating counterclockwise at a rate of 2π radians per second—one revolution per second—and that's why ω is called the radian frequency. In terms of frequency, [Eq. \(A-9\)](#)'s phasor is rotating counterclockwise at $\omega = 2\pi f$ radians per second, where f is the cyclic frequency in cycles per second (Hz). If the cyclic frequency is $f = 10$ Hz, the phasor is rotating at 20π radians per second. Likewise, the expression

$$(A-9')$$

$$C = Me^{-j\omega t}$$

represents a phasor of magnitude M that rotates in a clockwise direction about the origin of the complex plane at a negative radian frequency of $-\omega$ radians per second.

A.3 Arithmetic Operations of Complex Numbers

A.3.1 Addition and Subtraction of Complex Numbers

Which of the above forms for C in [Eq. \(A-1\)](#) is the best to use? It depends on the arithmetic operation we want to perform. For example, if we're adding two complex numbers, the rectangular form in [Eq. \(A-1\)](#) is the easiest to use. The addition of two complex numbers, $C_1 = R_1 + jI_1$ and $C_2 = R_2 + jI_2$, is merely the sum of the real parts plus j times the sum of the imaginary parts as

$$(A-10)$$

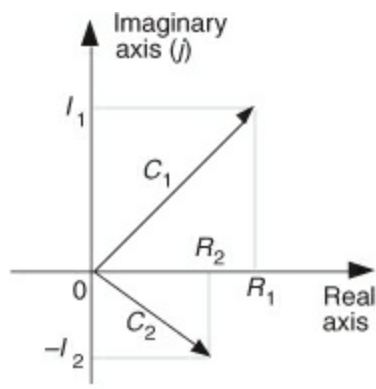
$$C_1 + C_2 = R_1 + jI_1 + R_2 + jI_2 = R_1 + R_2 + j(I_1 + I_2).$$

[Figure A-3](#) is a graphical depiction of the sum of two complex numbers using the concept of phasors. Here the sum phasor $C_1 + C_2$ in [Figure A-3\(a\)](#) is the new phasor from the beginning of phasor C_1 to the end of phasor C_2 in [Figure A-3\(b\)](#). Remember, the R s and the I s can be either positive or negative numbers. Subtracting one complex number from the other is straightforward as long as we find the differences between the two real parts and the two imaginary parts separately. Thus

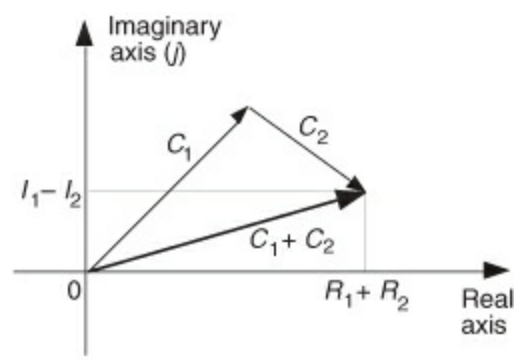
$$(A-11)$$

$$C_1 - C_2 = (R_1 + jI_1) - (R_2 + jI_2) = R_1 - R_2 + j(I_1 - I_2).$$

Figure A-3 Geometrical representation of the sum of two complex numbers.



(a)



(b)

An example of complex number addition is discussed in [Section 11.3](#), where we covered the topic of averaging fast Fourier transform outputs.

A.3.2 Multiplication of Complex Numbers

We can use the rectangular form to multiply two complex numbers as

(A-12)

$$C_1 C_2 = (R_1 + jI_1)(R_2 + jI_2) = (R_1 R_2 - I_1 I_2) + j(R_1 I_2 + R_2 I_1).$$

However, if we represent the two complex numbers in exponential form, their product takes the simpler form

(A-13)

$$C_1 C_2 = M_1 e^{j\theta_1} M_2 e^{j\theta_2} = M_1 M_2 e^{j(\theta_1 + \theta_2)}$$

because multiplication results in the addition of the exponents. Of some interest is the fact that the product of the magnitudes of two complex numbers is equal to the magnitude of their product. That is,

(A-13')

$$|C_1| \cdot |C_2| = |C_1 C_2|.$$

As a special case of multiplication of two complex numbers, *scaling* is multiplying a complex number by another complex number whose imaginary part is zero. We can use the rectangular or exponential forms with equal ease as follows:

(A-14)

$$kC = k(R + jI) = kR + jkI,$$

or in exponential form,

(A-15)

$$kC = k(Me^{j\theta}) = kMe^{j\theta}.$$

A.3.3 Conjugation of a Complex Number

The complex conjugate of a complex number is obtained merely by changing the sign of the number's imaginary part. So, if we denote C^* as the complex conjugate of the number $C = R + jI = Me^{j\theta}$, then C^* is expressed as

(A-16)

$$C^* = R - jI = Me^{-j\theta}.$$

There are three characteristics of conjugates that occasionally come in handy. First, the conjugate of a

product is equal to the product of the conjugates. That is, if $C = C_1 C_2$, then from [Eq. \(A-13\)](#)

(A-17)

$$\begin{aligned} C^* &= (C_1 C_2)^* = (M_1 M_2 e^{j(\theta_1 + \theta_2)})^* = M_1 M_2 e^{-j(\theta_1 + \theta_2)} \\ &= M_1 e^{-j\theta_1} M_2 e^{-j\theta_2} = C_1^* C_2^*. \end{aligned}$$

Second, the sum of conjugates of two complex numbers is equal to the conjugate of the sum. We can show this in rectangular form as

(A-17')

$$\begin{aligned} (R_1 + jI_1)^* + (R_2 + jI_2)^* &= (R_1 - jI_1) + (R_2 - jI_2) \\ &= R_1 + R_2 - j(I_1 + I_2) = [R_1 + R_2 + j(I_1 + I_2)]^*. \end{aligned}$$

Third, the product of a complex number and its conjugate is the complex number's magnitude squared. It's easy to prove this in exponential form as

(A-18)

$$CC^* = Me^{j\theta} Me^{-j\theta} = M^2 e^{j0} = M^2.$$

(This property is often used in digital signal processing to determine the relative power of a complex sinusoidal phasor represented by $Me^{j\omega t}$.)

A.3.4 Division of Complex Numbers

The division of two complex numbers is also convenient using the exponential and magnitude and angle forms, such as

(A-19)

$$\frac{C_1}{C_2} = \frac{M_1 e^{j\theta_1}}{M_2 e^{j\theta_2}} = \frac{M_1}{M_2} e^{j(\theta_1 - \theta_2)}$$

and

(A-19')

$$\frac{C_1}{C_2} = \frac{M_1}{M_2} \angle (\theta_1 - \theta_2).$$

Although not nearly so handy, we can perform complex division in rectangular notation by multiplying the numerator and the denominator by the complex conjugate of the denominator as

(A-20)

$$\begin{aligned} \frac{C_1}{C_2} &= \frac{R_1 + jI_1}{R_2 + jI_2} \\ &= \frac{R_1 + jI_1}{R_2 + jI_2} \cdot \frac{R_2 - jI_2}{R_2 - jI_2} \\ &= \frac{(R_1 R_2 + I_1 I_2) + j(R_2 I_1 - R_1 I_2)}{R_2^2 + I_2^2}. \end{aligned}$$

A.3.5 Inverse of a Complex Number

A special form of division is the inverse, or reciprocal, of a complex number. If $C = Me^{j\theta}$, its inverse is given by

(A-21)

$$\frac{1}{C} = \frac{1}{Me^{j\theta}} = \frac{1}{M} e^{-j\theta}.$$

In rectangular form, the inverse of $C = R + jI$ is given by

(A-22)

$$\frac{1}{C} = \frac{1}{R + jI} \cdot \frac{R - jI}{R - jI} = \frac{R - jI}{R^2 + I^2} = \frac{C^*}{M^2}.$$

We obtain [Eq. \(A-22\)](#) by substituting $R_1 = 1$, $I_1 = 0$, $R_2 = R$, and $I_2 = I$ in [Eq. \(A-20\)](#).

A.3.6 Complex Numbers Raised to a Power

Raising a complex number to some power is easily done in the exponential form. If $C = Me^{j\theta}$, then

(A-23)

$$C^k = M^k (e^{j\theta})^k = M^k e^{jk\theta}.$$

For example, if $C = 3e^{j125^\circ}$, then C cubed is

(A-24)

$$(C)^3 = 3^3 (e^{j3 \cdot 125^\circ}) = 27e^{j375^\circ} = 27e^{j15^\circ}.$$

We conclude this appendix with four complex arithmetic operations that are not very common in digital signal processing—but you may need them sometime.

A.3.7 Roots of a Complex Number

The k th root of a complex number C is the number that, multiplied by itself k times, results in C . The exponential form of C is the best way to explore this process. When a complex number is represented by $C = Me^{j\theta}$, remember that it can also be represented by

(A-25)

$$C = Me^{j(\theta + n360^\circ)}.$$

In this case, the variable θ in [Eq. \(A-25\)](#) is in degrees. There are k distinct roots when we're finding the k th root of C . By “distinct,” we mean roots whose exponents are less than 360° . We find those roots by using the following:

(A-26)

$$\sqrt[k]{C} = \sqrt[k]{Me^{j(\theta + n360^\circ)}} = \sqrt[k]{M} e^{j(\theta + n360^\circ)/k}.$$

Next, we assign the values 0, 1, 2, 3, . . . , $k-1$ to n in [Eq. \(A-26\)](#) to get the k roots of C . OK, we need an example here! Let's say we're looking for the cube (third) root of $C = 125e^{j(75^\circ)}$. We proceed as follows:

(A-27)

$$\sqrt[3]{C} = \sqrt[3]{125e^{j(75^\circ)}} = \sqrt[3]{125e^{j(75^\circ + n360^\circ)}} = \sqrt[3]{125e^{j(75^\circ + n360^\circ)/3}}.$$

Next we assign the values $n = 0$, $n = 1$, and $n = 2$ to [Eq. \(A-27\)](#) to get the three roots of C . So the three distinct roots are

$$\text{1st root: } \rightarrow \sqrt[3]{C} = 5e^{j(75^\circ + 0 \cdot 360^\circ)/3} = 5e^{j(25^\circ)};$$

$$\text{2nd root: } \rightarrow \sqrt[3]{C} = 5e^{j(75^\circ + 1 \cdot 360^\circ)/3} = 5e^{j(435^\circ)/3} = 5e^{j(145^\circ)};$$

and

$$\text{3rd root: } \rightarrow \sqrt[3]{C} = 5e^{j(75^\circ + 2 \cdot 360^\circ)/3} = 5e^{j(795^\circ)/3} = 5e^{j(265^\circ)}.$$

A.3.8 Natural Logarithms of a Complex Number

Taking the natural logarithm of a complex number $C = Me^{j\varnothing}$ is straightforward using exponential notation; that is,

$$(A-28)$$

$$\ln C = \ln(Me^{j\varnothing}) = \ln M + \ln(e^{j\varnothing}) = \ln M + j\varnothing,$$

where $0 \leq \varnothing < 2\pi$. By way of example, if $C = 12e^{j\pi/4}$, the natural logarithm of C is

$$(A-29)$$

$$\ln C = \ln(12e^{j\pi/4}) = \ln(12) + j\pi/4 = 2.485 + j0.785.$$

This means that $e^{(2.485 + j0.785)} = e^{2.485} \cdot e^{j0.785} = 12e^{j\pi/4}$.

Before leaving this topic of the natural logarithm of complex numbers, we remind the reader that $e^{j\pi} = -1$, which allows us to write

$$(A-30)$$

$$\ln(-1) = j\pi,$$

showing how the natural logarithm of a negative real number is defined.

As an interesting aside, rearranging the $e^{j\pi} = -1$ expression enables us to write what many mathematicians call “the most beautiful formula in mathematics.” That equation is

$$(A-31)$$

$$e^{j\pi} + 1 = 0.$$

[Equation \(A-31\)](#) is famous because the natural constants e , π , 0, and 1, along with the fundamental operations of addition, multiplication, exponentiation, the “ j ” operator, and equality, all appear exactly once!

A.3.9 Logarithm to the Base 10 of a Complex Number

We can calculate the base 10 logarithm of the complex number $C = Me^{j\varnothing}$ using

$$(A-32)$$

$$\log_{10} C = \log_{10}(Me^{j\varnothing}) = \log_{10} M + \log_{10}(e^{j\varnothing}) = \log_{10} M + j\varnothing \cdot \log_{10}(e).^\dagger$$

[†] For the second term of the result in [Eq. \(A-32\)](#) we used $\log_a(x^n) = n \cdot \log_a x$ according to the law of logarithms.

Of course e is the irrational number, approximately equal to 2.71828, whose log to the base 10 is approximately 0.43429. Keeping this in mind, we can simplify [Eq. \(A-32\)](#) as

$$(A-32')$$

$$\log_{10} C \approx \log_{10} M + j(0.43429 \cdot \varnothing).$$

Repeating the above example with $C = 12e^{j\pi/4}$ and using the [Eq. \(A-32'\)](#) approximation, the base 10 logarithm of C is

(A-33)

$$\begin{aligned}\log_{10} C &= \log_{10}(12e^{j\pi/4}) = \log_{10}(12) + j(0.43429 \cdot \pi/4) \\ &= 1.079 + j(0.43429 \cdot 0.785) = 1.079 + j0.341.\end{aligned}$$

The result from [Eq. \(A-33\)](#) means that

(A-33')

$$\begin{aligned}10^{(1.079 + j0.341)} &= 10^{1.079} \cdot 10^{j0.341} = 12 \cdot (e^{2.302})^{j0.341} \\ &= 12e^{j(2.302 \cdot 0.341)} = 12e^{j0.785} = 12e^{j\pi/4}.\end{aligned}$$

A.3.10 Log to the Base 10 of a Complex Number Using Natural Logarithms

Unfortunately, some software mathematics packages have no base 10 logarithmic function and can calculate only natural logarithms. In this situation, we just use

(A-34)

$$\log_{10}(x) = \frac{\ln(x)}{\ln(10)}$$

to calculate the base 10 logarithm of x . Using this *change of base* formula, we can find the base 10 logarithm of a complex number $C = Me^{j\theta}$; that is,

(A-35)

$$\log_{10} C = \frac{\ln C}{\ln 10} = (\log_{10} e)(\ln C).$$

Because $\log_{10}(e)$ is approximately equal to 0.43429, we use [Eq. \(A-35\)](#) to state that

(A-36)

$$\log_{10} C \approx 0.43429 \cdot (\ln C) = 0.43429 \cdot (\ln M + j\theta).$$

Repeating, again, the example above of $C = 12e^{j\pi/4}$, the [Eq. \(A-36\)](#) approximation allows us to take the base 10 logarithm of C using natural logs as

(A-37)

$$\begin{aligned}\log_{10} C &= 0.43429 \cdot (\ln(12) + j\pi/4) \\ &= 0.43429 \cdot (2.485 + j0.785) = 1.079 + j0.341,\end{aligned}$$

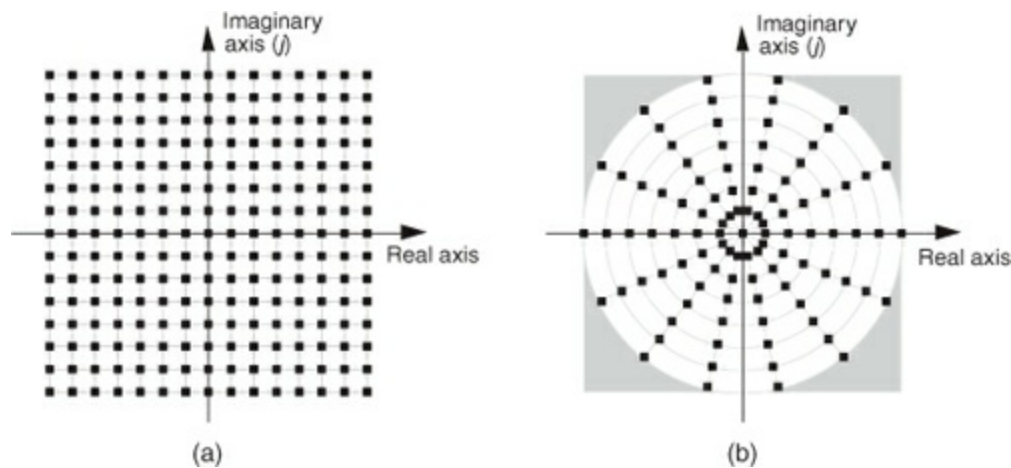
giving us the same result as [Eq. \(A-32\)](#).

A.4 Some Practical Implications of Using Complex Numbers

At the beginning of [Section A.3](#), we said that the choice of using the rectangular versus the polar form of representing complex numbers depends on the type of arithmetic operations we intend to perform. It's interesting to note that the rectangular form has a practical advantage over the polar form when we consider how numbers are represented in a computer. For example, let's say we must represent our complex numbers using a four-bit sign-magnitude binary number format. This means that we can have integral numbers ranging from -7 to $+7$, and our range of complex numbers covers a square on the complex plane as shown in [Figure A-4\(a\)](#) when we use the rectangular form. On the other hand, if we used four-bit numbers to represent the magnitude of a complex number in polar form, those numbers must reside on or within a circle whose radius is 7 as shown in [Figure A-4\(b\)](#). Notice how the four shaded corners in [Figure A-4\(b\)](#) represent locations of valid complex values using the

rectangular form but are *out of bounds* if we use the polar form. Put another way, a complex number calculation, yielding an acceptable result in rectangular form, could result in an overflow error if we use polar notation in our computer. We could accommodate the complex value $7 + j7$ in rectangular form but not its polar equivalent, because the magnitude of that polar number is greater than 7.

Figure A-4 Complex integral numbers represented as points on the complex plane using a four-bit sign-magnitude data format: (a) using rectangular notation; (b) using polar notation.



Although we avoid any further discussion here of the practical implications of performing complex arithmetic using standard digital data formats, it is an intricate and interesting subject. To explore this topic further, the inquisitive reader is encouraged to start with the references.

References

- [1] Plauger, P. J. "Complex Math Functions," *Embedded Systems Programming*, August 1994.
- [2] Kahan, W. "Branch Cuts for Complex Elementary Functions, or Much Ado About Nothing's Sign Bit," *Proceedings of the Joint IMA/SIAM Conference on the State of the Art in Numerical Analysis*, Clarendon Press, 1987.
- [3] Plauger, P. J. "Complex Made Simple," *Embedded Systems Programming*, July 1994.

Appendix B. Closed Form of a Geometric Series

In the literature of digital signal processing, we often encounter geometric series expressions like

(B-1)

$$\sum_{n=p}^{N-1} r^n = \frac{r^p - r^N}{1 - r},$$

or

(B-2)

$$\sum_{n=0}^{N-1} e^{-j2\pi mn/N} = \frac{1 - e^{-j2\pi m}}{1 - e^{-j2\pi m/N}}.$$

Unfortunately, many authors make a statement like “and we know that” and drop [Eqs. \(B-1\)](#) or [\(B-2\)](#) on the unsuspecting reader who’s expected to accept these expressions on faith. Assuming that you don’t have a Ph.D. in mathematics, you may wonder exactly what arithmetic sleight of hand allows us to arrive at [Eqs. \(B-1\)](#) or [\(B-2\)](#)? To answer this question, let’s consider a general expression for a geometric series such as

(B-3)

$$S = \sum_{n=p}^{N-1} ar^n = ar^p + ar^{p+1} + ar^{p+2} + \dots + ar^{N-1},$$

where n , N , and p are integers and a and r are any constants. Multiplying [Eq. \(B-3\)](#) by r gives us

(B-4)

$$Sr = \sum_{n=p}^{N-1} ar^{n+1} = ar^{p+1} + ar^{p+2} + \dots + ar^{N-1} + ar^N.$$

Subtracting [Eq. \(B-4\)](#) from [Eq. \(B-3\)](#) gives the expression

$$S - Sr = S(1 - r) = ar^p - ar^N,$$

or

(B-5)

$$S = a \cdot \frac{r^p - r^N}{1 - r}.$$

So here’s what we’re after. The *closed form* of the series is

(B-6)

Closed form of a general
geometric series: \rightarrow

$$\sum_{n=p}^{N-1} ar^n = a \cdot \frac{r^p - r^N}{1 - r}.$$

(By “closed form,” we mean taking an infinite series and converting it to a simpler mathematical form without the summation.) When $a = 1$, [Eq. \(B-6\)](#) validates [Eq. \(B-1\)](#). We can quickly verify [Eq. \(B-6\)](#) with an example. Letting $N = 5$, $p = 0$, $a = 2$, and $r = 3$, for example, we can create the following list:

n	$ar^n = 2 \cdot 3^n$
0	$2 \cdot 3^0 = 2$
1	$2 \cdot 3^1 = 6$
2	$2 \cdot 3^2 = 18$
3	$2 \cdot 3^3 = 54$
4	$2 \cdot 3^4 = 162$
	The sum of this column is
	$\sum_{n=0}^4 2 \cdot 3^n = 242.$

Plugging our example N, p, a , and r values into [Eq. \(B-6\)](#),

(B-7)

$$\sum_{n=p}^{N-1} ar^n = a \cdot \frac{r^p - r^N}{1-r} = 2 \cdot \frac{3^0 - 3^5}{1-3} = 2 \cdot \frac{1-243}{-2} = 242,$$

which equals the sum of the rightmost column in the list above.

As a final step, the terms of our earlier [Eq. \(B-2\)](#) are in the form of [Eq. \(B-6\)](#) as $p = 0$, $a = 1$, and $r = e^{-j2\pi m/N}$.[‡] So plugging those terms from [Eq. \(B-2\)](#) into [Eq. \(B-6\)](#) gives us

[‡] From the math identity $a^{xy} = (a^x)^y$, we can say $e^{-j2\pi nm/N} = (e^{-j2\pi m/N})^n$, so $r = e^{-j2\pi m/N}$.

(B-8)

$$\sum_{n=0}^{N-1} e^{-j2\pi nm/N} = 1 \cdot \frac{e^{-j2\pi n0/N} - e^{-j2\pi nN/N}}{1 - e^{-j2\pi m/N}} = \frac{1 - e^{-j2\pi m}}{1 - e^{-j2\pi m/N}},$$

confirming [Eq. \(B-2\)](#).

Appendix C. Time Reversal and the DFT

The notion of time reversal in discrete systems occasionally arises in the study of the discrete Fourier transform (DFT), the mathematical analysis of digital filters, and even in practice (straight time reversal is used in a digital filtering scheme described in [Section 13.12](#)). We give the topic of time reversal some deserved attention here because it illustrates one of the truly profound differences between the worlds of continuous and discrete systems. In addition, the spectral effects of reversing a time sequence are (in my opinion) not obvious and warrant investigation.

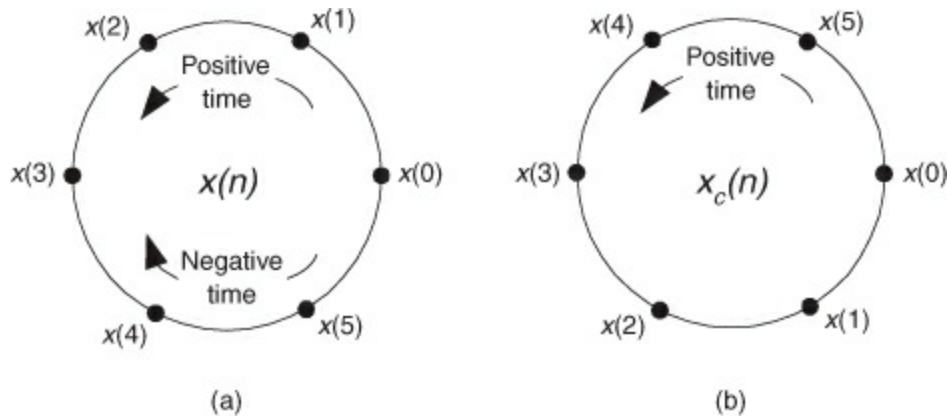
Actually, in discrete-time systems there are two forms of time reversal we need to think about. Consider the 6-point $x(n)$ time-domain sequence

$$(C-1)$$

$$x(n) = x(0), x(1), x(2), x(3), x(4), x(5).$$

Due to the periodicity properties of discrete sampled representations (discussed in Section 3.17), we can depict the $x(n)$ time sequence as samples on a circle as shown in [Figure C-1\(a\)](#). There we arbitrarily assign positive time flow as counterclockwise rotation. (For our UK friends, counterclockwise means your *anticlockwise*.)

Figure C-1 Circular representations of periodic sequences: (a) original $x(n)$ sequence; (b) circular time reversal of $x(n)$.



Time reversal, as defined here for sequences that are treated as periodic, means traveling clockwise around the circle (in the negative time direction), creating a new time sequence

$$(C-2)$$

$$x_c(n) = x(0), x(5), x(4), x(3), x(2), x(1).$$

We call $x_c(n)$ the circular time reversal of $x(n)$, where the subscript "c" means *circular* reversal, and depict $x_c(n)$ as in [Figure C-1\(b\)](#).

The interesting issue here is that for real N -point time sequences, the DFT of $x_c(n)$ is the complex conjugate of the DFT of $x(n)$. That is,

$$(C-3)$$

$$X_c(m) = X^*(m)$$

where the DFT index is $0 \leq m \leq N-1$. Due to the conjugate symmetry of DFTs of real sequences, we should realize that $X^*(m)$ is a straight reversal of the $X(m)$ samples.

Let's illustrate [Eq. \(C-3\)](#) with an example. With $X(m)$ representing the DFT of $x(n)$, we can write down $X(m)$'s $m = 4$ sample $X(4)$ as

(C-4)

$$X(4) = x(0)e^{-j2\pi 0/6} + x(1)e^{-j2\pi 4/6} + x(2)e^{-j2\pi 8/6} \\ + x(3)e^{-j2\pi 12/6} + x(4)e^{-j2\pi 16/6} + x(5)e^{-j2\pi 20/6}.$$

Because $e^{-j2\pi k/6}$ has a period of 6, we can write [Eq. \(C-4\)](#) as

(C-5)

$$X(4) = x(0)e^{-j2\pi 0/6} + x(1)e^{-j2\pi 4/6} + x(2)e^{-j2\pi 2/6} \\ + x(3)e^{-j2\pi 0/6} + x(4)e^{-j2\pi 4/6} + x(5)e^{-j2\pi 2/6}.$$

Next, let's write down the (circular-reversed) $X_c(m)$'s $m = 4$ -sample $X_c(4)$ as

(C-6)

$$X_c(4) = x(0)e^{-j2\pi 0/6} + x(5)e^{-j2\pi 4/6} + x(4)e^{-j2\pi 8/6} \\ + x(3)e^{-j2\pi 12/6} + x(2)e^{-j2\pi 16/6} + x(1)e^{-j2\pi 20/6}$$

or

(C-7)

$$X_c(4) = x(0)e^{-j2\pi 0/6} + x(5)e^{-j2\pi 4/6} + x(4)e^{-j2\pi 2/6} \\ + x(3)e^{-j2\pi 0/6} + x(2)e^{-j2\pi 4/6} + x(1)e^{-j2\pi 2/6}.$$

Replacing $X_c(4)$'s negative angles with their positive-angle equivalents yields

(C-8)

$$X_c(4) = x(0)e^{j2\pi 0/6} + x(5)e^{j2\pi 2/6} + x(4)e^{j2\pi 4/6} \\ + x(3)e^{j2\pi 0/6} + x(2)e^{j2\pi 2/6} + x(1)e^{j2\pi 4/6},$$

which is the conjugate of [Eq. \(C-5\)](#), demonstrating that $X(m)$ and $X_c(m)$ are complex conjugates.

An alternate time reversal concept, which we'll call *straight* time reversal, is the simple reversal of [Eq. \(C-1\)](#)'s $x(n)$, yielding an $x_s(n)$ sequence

(C-9)

$$x_s(n) = x(5), x(4), x(3), x(2), x(1), x(0),$$

where the subscript "s" means *straight* reversal. For real N -point time sequences, the DFT of $x_s(n)$ is

(C-10)

$$X_s(m) = X^*(m)e^{-j2\pi m(N-1)/N}.$$

We can demonstrate [Eq. \(C-10\)](#) the same way we did [Eq. \(C-3\)](#), but consider [Figure C-2](#). There we show the samples of repeated revolutions around the $x_c(n)$ circle in [Figure C-1\(b\)](#), indicating both the 6-point $x_s(n)$ and the 6-point $x_c(n)$ sequences. Notice how $x_s(n)$ is shifted backward in time by five samples from $x_c(n)$.

Figure C-2 Periodic sequences $x_s(n)$ and $x_c(n)$.

$$\begin{array}{c}
 \leftarrow x_c(n) \rightarrow \\
 \dots x(1), x(0), x(5), x(4), x(3), x(2), x(1), x(0), x(5), x(4), x(3), x(2), x(1), x(0), \dots \\
 \leftarrow x_s(n) \rightarrow
 \end{array}$$

Using the principle of the DFT's shifting theorem from [Section 3.6](#), we know that $X_s(m)$ is equal to $X_c(m)$ times a linear phase shift of $e^{-j2\pi m(5)/6}$ for our $N = 6$ example. So, in the general N -point sequence case,

(C-11)

$$X_s(m) = X_c(m)e^{-j2\pi m(N-1)/N} = X^*(m)e^{-j2\pi m(N-1)/N},$$

which validates [Eq. \(C-10\)](#).

Appendix D. Mean, Variance, and Standard Deviation

In our studies, we're often forced to consider noise functions. These are descriptions of noise signals that we cannot explicitly describe with a time-domain equation. Noise functions can be quantified, however, in a worthwhile way using the statistical measures of mean, variance, and standard deviation. Although here we only touch on the very broad and important field of statistics, we will describe why, how, and when to use these statistical indicators, so that we can add them to our collection of signal analysis tools. First we'll determine how to calculate these statistical values for a series of discrete data samples, cover an example using a continuous analytical function, and conclude this appendix with a discussion of the probability density functions of several random variables that are common in the field of digital signal processing. So let's proceed by sticking our toes in the chilly waters of the mathematics of statistics to obtain a few definitions.

D.1 Statistical Measures

Consider a continuous sinusoid having a frequency of f_0 Hz with a peak amplitude of A_p expressed by the equation

(D-1)

$$x(t) = A_p \sin(2\pi f_0 t).$$

[Equation \(D-1\)](#) completely specifies $x(t)$ —that is, we can determine $x(t)$'s exact value at any given instant in time. For example, when time $t = 1/4f_0$, we know that $x(t)$'s amplitude will be A_p , and at the later time $t = 1/2f_0$, $x(t)$'s amplitude will be zero. On the other hand, we have no definite way to express the successive values of a random function or of random noise.[‡] There's no equation like [Eq. \(D-1\)](#) available to predict future noise-amplitude values, for example. (That's why they call it random noise.) Statisticians have, however, developed powerful mathematical tools to characterize several properties of random functions. The most important of these properties have been given the names *mean*, *variance*, and *standard deviation*.

[‡] We define *random noise* to be unwanted, unpredictable disturbances contaminating a signal or a data sequence of interest.

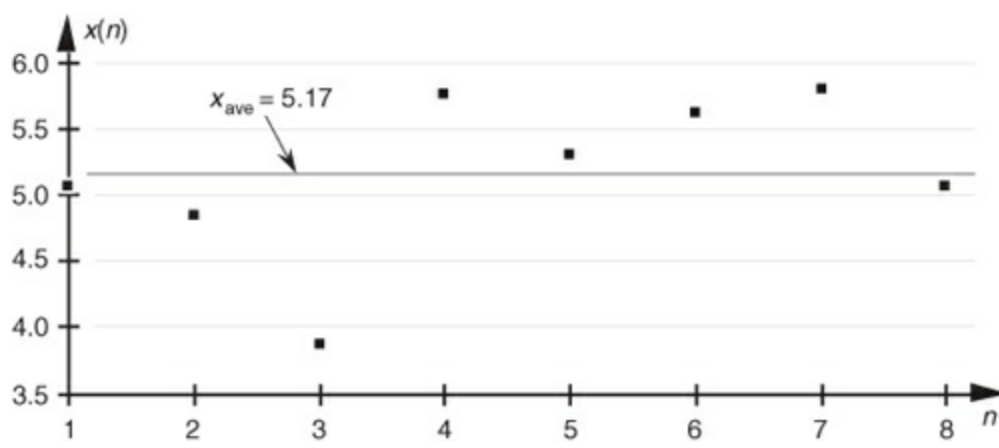
Mathematically, the *sample mean*, or *average*, of N separate values of a sequence x , denoted x_{ave} , is defined as[\[1\]](#)

(D-2)

$$x_{\text{ave}} = \frac{1}{N} \sum_{n=1}^N x(n) = \frac{x(1) + x(2) + x(3) + \dots + x(N)}{N}.$$

[Equation \(D-2\)](#), already familiar to most people, merely states that the average of a sequence of N numbers is the sum of those numbers divided by N . Graphically, the average can be depicted as that value around which a series of sample values cluster, or congregate, as shown in [Figure D-1](#). If the eight values depicted by the dots in [Figure D-1](#) represent some measured quantity and we applied those values to [Eq. \(D-2\)](#), the average of the series is 5.17, as shown by the dotted line.

Figure D-1 Average of a sequence of eight values.



An interesting property of the average (mean value) of an $x(n)$ sequence is that x_{ave} is the value that makes the sum of the differences between $x(n)$ and x_{ave} equal to zero. That is, the sum of the sequence $d_{\text{iff}}(n) = x(n) - x_{\text{ave}}$ is zero.

Now that we've defined *average*, another key definition is the variance of a sequence, σ^2 , defined as

(D-3)

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^N [x(n) - x_{\text{ave}}]^2$$

$$= \frac{[x(1) - x_{\text{ave}}]^2 + [x(2) - x_{\text{ave}}]^2 + [x(3) - x_{\text{ave}}]^2 + \dots + [x(N) - x_{\text{ave}}]^2}{N}.$$

Sometimes in the literature we'll see σ^2 defined with a $1/(N-1)$ factor before the summation instead of the $1/N$ factor in [Eq. \(D-3\)](#). In a moment we'll explain why this is so.

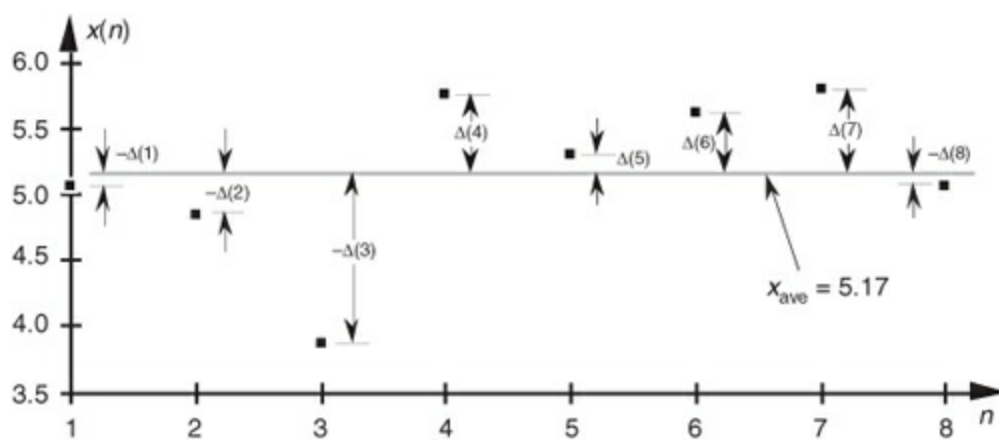
Variance is a very important concept because it's the yardstick with which we measure, for example, the effect of quantization errors and the usefulness of signal-averaging algorithms. It gives us an idea how the aggregate values in a sequence fluctuate around the sequence's average and provides us with a well-defined quantitative measure of those fluctuations. Mathematicians call those fluctuations the *dispersion* of the sequence. (Because the positive square root of the variance, the standard deviation, is typically denoted as σ in the literature, we'll use the conventional notation of σ^2 for the variance.)

[Equation \(D-3\)](#) looks a bit perplexing if you haven't seen it before. Its meaning becomes clear if we examine it carefully. The $x(1) - x_{\text{ave}}$ value in the bracket, for example, is the difference between the $x(1)$ value and the sequence average x_{ave} . For any sequence value $x(n)$, the $x(n) - x_{\text{ave}}$ difference, which we denote as $\Delta(n)$, can be either positive or negative, as shown in [Figure D-2](#). Specifically, the differences $\Delta(1)$, $\Delta(2)$, $\Delta(3)$, and $\Delta(8)$ are negative because their corresponding sequence values are below the sequence average shown by the dotted line. If we replace the $x(n) - x_{\text{ave}}$ difference terms in [Eq. \(D-3\)](#) with $\Delta(n)$ terms, the variance can be expressed as

(D-4)

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^N [\Delta(n)]^2, \text{ where } \Delta(n) = x(n) - x_{\text{ave}}.$$

Figure D-2 Difference values $\Delta(n)$ of the sequence in [Figure D-1](#).



The reader might wonder why the squares of the differences are summed, instead of just the differences themselves. This is because, by the very nature of the definition of x_{ave} , the sum of the $\Delta(n)$ difference samples will always be zero. Because we need an unsigned measure of each difference, we use the difference-squared terms as indicated by [Eq. \(D-4\)](#). In that way, individual $\Delta(n)$ difference terms will contribute to the overall variance regardless of whether the difference is positive or negative. Plugging the $\Delta(n)$ values from the example sequence in [Figure D-2](#) into [Eq. \(D-4\)](#), we get a variance value of 0.34. Another useful measure of a signal sequence is the square root of the variance known as the *standard deviation*. Taking the square root of [Eq. \(D-3\)](#) to get the standard deviation σ ,

(D-5)

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{1}{N} \sum_{n=1}^N [x(n) - x_{\text{ave}}]^2}.$$

So far, we have three measurements to use in evaluating a sequence of values: the average x_{ave} , the variance σ^2 , and the standard deviation σ . Where x_{ave} indicates around what constant level the individual sequence values vary, σ^2 is a measure of the magnitude of the noise fluctuations around the average x_{ave} . If the sequence represents a series of random signal samples, we can say that x_{ave} specifies the average, or constant, value of the signal. The variance σ^2 is the magnitude squared, or power, of the fluctuating component of the signal. The standard deviation, then, is an indication of the magnitude of the fluctuating component of the signal.

D.2 Statistics of Short Sequences

In this section we discuss a subtle issue regarding the variance of a discrete sequence. The variance [Eq. \(D-3\)](#) is only *exactly* correct if N is infinitely large. When N is a small number and we're computing an $[x(4) - x_{\text{ave}}]$ term, for example, that $[x(4) - x_{\text{ave}}]$ value is too highly influenced (biased) by the single $x(4)$ sample. This results in an $[x(4) - x_{\text{ave}}]$ value that's slightly smaller than it should be[\[2\]](#). As such, [Eq. \(D-3\)](#) is often called a *biased estimate* of the true variance of $x(n)$. Mathematicians have determined that using a $1/(N-1)$ factor, called *Bessel's correction*, before the summation in [Eq. \(D-3\)](#) yields a more accurate estimation of the true variance of the infinite-length sequence $x(n)$, when we use only N samples of $x(n)$ to estimate the true variance. That is,

(D-6)

$$\sigma_{\text{unbiased}}^2 = \frac{1}{N-1} \sum_{n=1}^N [x(n) - x_{\text{ave}}]^2.$$

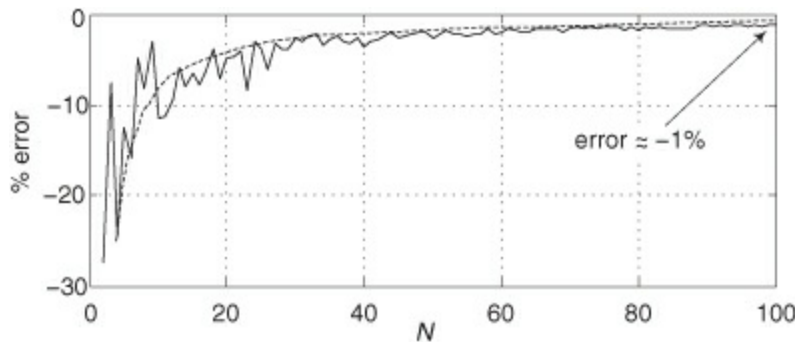
[Equation \(D-6\)](#) is called an *unbiased estimate* of the variance of $x(n)$. However, when N is greater than, say, 100, as it often is in real-world applications, the difference between [Eqs. \(D-3\)](#) and [\(D-6\)](#) will have little practical significance.

We can justify that claim by showing an example of the percent difference in using [Eqs. \(D-3\)](#) and [\(D-6\)](#), as a function of the $x(n)$ sequence length N , as the solid curve in [Figure D-3](#). Considering the unbiased variance to be correct (zero error), the solid error curve in [Figure D-3](#) shows how much smaller (negative percent error) the biased variance will be compared to the unbiased variance when $x(n)$ is Gaussian (to be described later) distributed random noise of unity variance. For instance, the percent error between the biased and the unbiased variance estimates is roughly -1 percent when $N = 100$. The dashed curve in [Figure D-3](#) is equal to -100 percent times the true $x(n)$ variance divided by N , so we can say that the percent error in using [Eq. \(D-3\)](#) compared to [Eq. \(D-6\)](#) is roughly

(D-7)

$$\text{Biased variance percent error} = -100 \times \frac{\sigma_{\text{true}}^2}{N}.$$

Figure D-3 Percent error in [Eq. \(D-3\)](#) relative to [Eq. \(D-6\)](#).



The bottom line here is that [Eq. \(D-6\)](#) should be considered for use in computing the variances of discrete sequences when N is small. [Section 13.35](#) discusses a computationally efficient, and memory-saving, way to compute variances.

D.3 Statistics of Summed Sequences

Here we discuss the statistical effects of adding two sequences. This material has great utility in noise-reduction operations. If we add two equal-length independent (uncorrelated) sequences $q(n)$ and $r(n)$, such that

(D-8)

$$p(n) = q(n) + r(n),$$

thanks to the good work of dead mathematicians we can say[\[3\]](#):

- The average (mean) of the $p(n)$ sequence is equal to the sum of the individual averages of the $q(n)$ and $r(n)$ sequences.
- The variance of the $p(n)$ sequence is equal to the sum of the individual variances of the $q(n)$ and $r(n)$ sequences. That is,

$$\sigma_p^2 = \sigma_q^2 + \sigma_r^2.$$

This means that if we consider the variances of two signals as being measures of their noise powers, then when two noisy signals are added, the resultant signal's noise power is the sum of the two individual noise powers.

- The variance of $C \cdot p(n) = C \cdot q(n) + C \cdot r(n)$, where C is a constant, is C^2 times the variance of the $p(n)$ sequence, or

$$\sigma_{C \cdot p}^2 = C^2 \sigma_p^2.$$

The above properties are related to a key characteristic of sampled signals that we can use for noise reduction by way of averaging. Assume we have an infinitely long $x(n)$ sequence contaminated with uncorrelated noise, and the variance of $x(n)$ is K . If we extract N blocks of samples from $x(n)$, with each block sequence being M samples in length, and average those N sequences, the variance of the resultant single M -sample average sequence is

(D-9)

$$\begin{aligned} \text{Variance of } \left(\frac{x_0(n) + x_1(n) + \dots + x_{N-2}(n) + x_{N-1}(n)}{N} \right) \\ = \frac{1}{N^2} \cdot (K + K + \dots + K + K) = \frac{NK}{N^2} = \frac{K}{N}. \end{aligned}$$

The square root of [Eq. \(D-9\)](#) yields the standard deviation of the single M -sample average sequence as

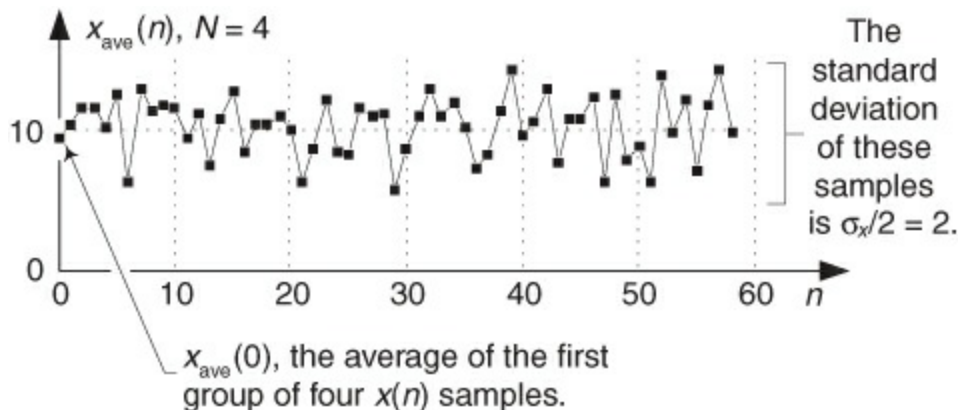
(D-10)

$$\sigma_{\text{ave}} = \frac{\sigma_x}{\sqrt{N}}$$

where σ_x is the standard deviation of the original $x(n)$ sequence.

As an example of [Eq. \(D-10\)](#), say that we have an $x(n)$ sequence and compute the average of the first N samples of $x(n)$, $x(0)$ through $x(N-1)$, to produce an $x_{\text{ave}}(0)$ sample. Next we compute the average of the second set of N samples of $x(n)$, $x(N)$ through $x(2N-1)$, to produce an $x_{\text{ave}}(1)$ sample, and so on. If the standard deviation of an $x(n)$ sequence, having an average value of 10 and standard deviation $\sigma_x = 4$, [Figure D-4](#) shows the $N = 4$ -point averaged $x_{\text{ave}}(n)$ sequence having an average value of 10 and a reduced standard deviation of $\sigma_x/N = 4/2 = 2$. [Chapter 11](#) gives practical examples of using [Eq. \(D-10\)](#) in real-world situations.

Figure D-4 $x_{\text{ave}}(n)$ sequence when $N = 4$.



On a practical note, if $x_s(n)$ are signal samples and $x_n(n)$ are noise samples, we can think of the $x(n)$ samples in [Eqs. \(D-9\)](#) and [\(D-10\)](#) as being represented by $x(n) = x_s(n) + x_n(n)$. The notion of contaminating noise being uncorrelated means that all the $x_n(n)$ noise samples are independent from each other, which implies that no information about any one noise sample can be determined from knowledge of any of the other noise samples. This assumption is not always valid if a noisy $x(n)$ signal has been filtered. With lowpass filtering, adjacent noise samples will be correlated (their amplitudes will be similar); the narrower the lowpass filter's passband, the more adjacent noise samples tend to be correlated. If the lowpass filter's passband is wide relative to half the sample rate ($f_s/2$), then the correlation among noise samples will be low and the noise samples can be considered uncorrelated. If the lowpass filter's passband is very narrow relative to $f_s/2$, then averaging is not as effective as we might expect from [Eqs. \(D-9\)](#) and [\(D-10\)](#).

We have discussed many statistical measures of real-valued discrete sequences, so [Table D-1](#) compiles what we've learned so far. The $x(n)$ sequence in the table can be an information-carrying signal, a noise-only signal, or a combination of the two.

Table D-1 Statistical Measures of Real-Valued Sequences

Signal Statistical Measure	Interpretation
$\text{Power} = \frac{1}{N} \sum_{n=0}^{N-1} [x(n)]^2$	Mean of the $x(n)$ signal samples squared. Interpreted as the average signal power.
$x_{\text{rms}} = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} [x(n)]^2}$	Root mean square (rms) of $x(n)$ signal. An equivalent signal amplitude whose square is the average signal power. rms has the same dimensions (units) as $x(n)$.
$x_{\text{ave}} = \frac{1}{N} \sum_{n=0}^{N-1} x(n)$	The average (mean) value of $x(n)$ signal. Interpreted as the DC (zero Hz) bias, or DC component, of a signal. x_{ave} has the same dimensions as $x(n)$.
$(x_{\text{ave}})^2$	DC power.
$\sigma^2 = \frac{1}{N} \sum_{n=1}^N [x(n) - x_{\text{ave}}]^2$	The variance of $x(n)$ signal. Interpreted as the power of the fluctuating (alternating, AC) portion of a signal. When $x_{\text{ave}} = 0$, $\sigma^2 = \text{average signal power}$.
$\sigma = \sqrt{\frac{1}{N} \sum_{n=1}^N [x(n) - x_{\text{ave}}]^2}$	The standard deviation of $x(n)$ signal. An equivalent signal amplitude of the fluctuating (alternating, AC) portion of a signal, whose square is the signal variance. σ has the same dimensions as $x(n)$. When $x_{\text{ave}} = 0$, $\sigma = x_{\text{rms}}$.

D.4 Standard Deviation (RMS) of a Continuous Sinewave

In computing the average power in electric circuits, for sinewave signals engineers often use a parameter called the *rms* value of the sinewave. That parameter, x_{rms} , for discrete samples is defined as

$$(D-11)$$

$$x_{\text{rms}} = \sqrt{\frac{1}{N} \sum_{n=1}^N x(n)^2}.$$

The $x(n)_{\text{rms}}$ in [Eq. \(D-11\)](#) is the square root of the mean (average) of the squares of the sequence $x(n)$. For a continuous sinusoid $x(t) = A_p \sin(2\pi ft) = A_p \sin(\omega t)$ whose average value is zero, x_{rms} is $x_{\text{rms-sine}}$ defined as

(D-12)

$$\begin{aligned} x_{\text{rms-sine}} &= \sqrt{\frac{1}{2\pi} \cdot \int_0^{2\pi} [A_p \sin(\omega t)]^2 d(\omega t)} \\ &= \sqrt{\frac{A_p^2}{2\pi} \cdot \int_0^{2\pi} \frac{1}{2} [1 - \cos(\omega t)] d(\omega t)} = \sqrt{\frac{A_p^2}{2\pi} \cdot \left[\frac{\omega t}{2} - \frac{1}{2} \sin(\omega t) \right]_0^{2\pi}} \\ &= \sqrt{\frac{A_p^2}{2\pi} \cdot \left[\frac{2\pi}{2} \right]} = \frac{A_p}{\sqrt{2}}. \end{aligned}$$

This $x_{\text{rms-sine}}$ expression is a lot easier to use for calculating average power dissipation in circuit elements than performing the integral of more complicated expressions. When a signal's average value is zero, then its rms value is equal to the signal's standard deviation. The variance of a sinewave is, of course, the square of [Eq. \(D-12\)](#).

We've provided the equations for the mean (average) and variance of a sequence of discrete values, introduced an expression for the standard deviation or rms of a sequence, and given an expression for the rms value of a continuous sinewave. The next question is “How can we characterize random functions for which there are no equations to predict their values and we have no discrete sample values with which to work?” The answer is that we must use probability density functions. Before we do that, in [Section D.6](#), let's first show how to use our statistical measures to estimate the signal-to-noise ratio of a discrete signal.

D.5 Estimating Signal-to-Noise Ratios

Given the above statistics of sampled signals, we now discuss a widely used way to quantify the quality of a noise-contaminated signal. By “quality” we mean the difference between listening to a recording of the Beatles' song “Hey Jude” on your iPod in a library and listening to the song while standing next to a running jet engine. We quantify the quality of a noise-contaminated signal by measuring, or estimating, its signal-power-to-noise-power ratio (SNR). The SNR of a signal is the ratio of the power of the noise-free signal over the power of the noise, or

(D-13)

$$\text{SNR} = \frac{\text{Signal power}}{\text{Noise power}}.$$

To illustrate the notion of SNR, the following list shows the SNRs (in dB) of a few common signal processing devices:

Signal type	SNR
Intelligible human speech	10 dB
Home video	30 dB
Analog telephone line	32 dB
Studio-quality video	40 dB
Maximum possible for 8-bit data	49.7 dB
AM analog radio	50 dB
Analog cassette tape player	60 dB
FM analog radio	65 dB
Modern long-play (LP) vinyl records	70 dB
Portable CD player	80 dB
Top-quality consumer CD player	95 dB
Maximum possible for 16-bit data	97.7 dB
Highest-quality studio audio	120 dB

The SNR of a signal can be estimated in either the time domain or the frequency domain. We discuss those operations next.

D.5.1 Estimating SNR in the Time Domain

We can estimate, by way of time-domain measurement, the SNR of a signal based on time-domain sample values. If $x_s(n)$ are real-valued signal samples and $x_n(n)$ are real-valued noise samples, the SNR of a signal $x(n) = x_s(n) + x_n(n)$ is

(D-14)

$$\text{SNR} = \frac{\text{Signal power}}{\text{Noise power}} = \frac{\frac{1}{N} \sum_{n=0}^{N-1} [x_s(n)]^2}{\frac{1}{N} \sum_{n=0}^{N-1} [x_n(n)]^2}$$

where the divide-by- N operations are shown for clarity but need not be performed because they cancel in the numerator and denominator. If we know the variances of $x_s(n)$ and $x_n(n)$, we can express the SNR of the fluctuating (AC) portion of a signal as

(D-15)

$$\text{SNR} = \frac{\text{Signal variance}}{\text{Noise variance}} = \frac{\sigma_s^2}{\sigma_n^2}.$$

In practice signal powers can vary over many orders of magnitude. For example, military radar systems transmit signals whose power is measured in megawatts, while the signal received by your cell phone antenna is measured in microwatts. That's 12 orders of magnitude! As such, it's both convenient and common to describe signal power and noise power logarithmically using decibels. (Decibels are discussed in [Appendix E](#).) We express signal-to-noise ratios measured in decibels (dB) as

(D-16)

$$\text{SNR}_{\text{dB}} = 10 \cdot \log_{10}(\text{SNR}) \text{ dB}$$

where the SNR term in [Eq. \(D-16\)](#) is the SNR value from [Eqs. \(D-14\)](#) or [\(D-15\)](#). If we know the rms values of $x_s(n)$ and $x_n(n)$, then we can express a signal's SNR in dB as

(D-17)

$$SNR_{dB} = 20 \cdot \log_{10} \left(\frac{\text{Signal rms}}{\text{Noise rms}} \right) \text{dB}.$$

Because the ratio in [Eq. \(D-17\)](#) is in terms of amplitudes (voltages or currents), rather than powers, we're forced to use the factor of 20 in computing SNR_{dB} based on rms values. If we know the standard deviations of $x_s(n)$ and $x_n(n)$, we can express the SNR of the fluctuating (AC) portion of a signal in dB as

(D-18)

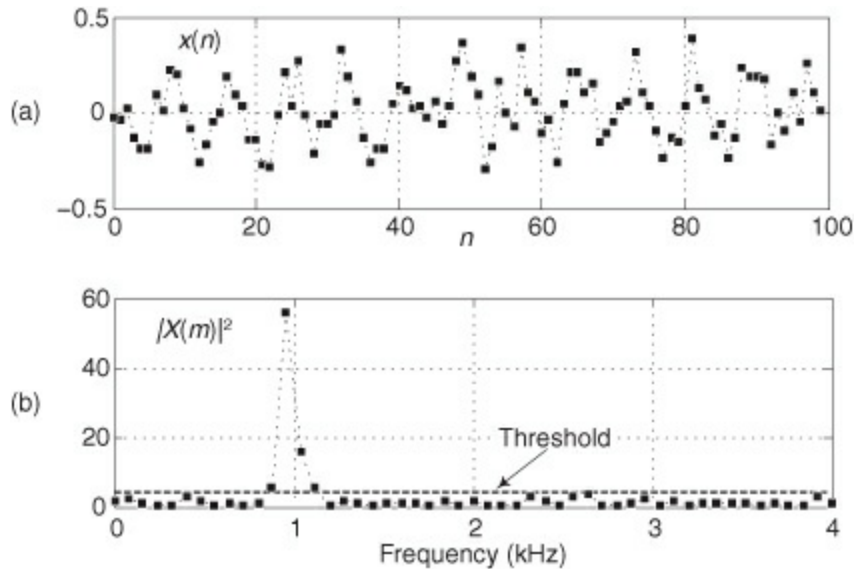
$$SNR_{dB} = 20 \cdot \log_{10} \left(\frac{\sigma_s}{\sigma_n} \right) \text{dB}.$$

The values for linear SNR, [Eq. \(D-14\)](#), are always positive, but values for SNR_{dB} can be positive or negative. For example, if a signal's linear SNR is 4, then its SNR_{dB} is $10 \cdot \log_{10}(4) = 6$ dB. If a signal's linear SNR is $1/4$, then its SNR_{dB} is $10 \cdot \log_{10}(1/4) = -6$ dB.

D.5.2 Estimating SNR in the Frequency Domain

We can obtain a rough estimate of the SNR of a signal based on its frequency-domain characteristics. The standard procedure for doing so is as follows: Assume we have $N = 100$ samples of the noisy 986 Hz real-valued $x(n)$ sinusoid, where the sample rate is $f_s = 8$ kHz, as shown in [Figure D-5\(a\)](#). After performing a 100-point DFT, and computing the spectral magnitude-squared sample values, we obtain the positive-frequency $|X(m)|^2$ power spectral samples depicted in [Figure D-5\(b\)](#).

Figure D-5 SNR estimation example: (a) noisy time-domain sinusoid; (b) 100-point DFT power samples.



Next we determine a Threshold power value, the dashed line in [Figure D-5\(b\)](#), above which only signal-related power samples exist and below which are the noise-only power samples. The estimated SNR of $x(n)$ is then

(D-19)

$$SNR = \frac{\text{Sum of the } |X(m)|^2 \text{ samples above Threshold}}{\text{Sum of the } |X(m)|^2 \text{ samples below Threshold}}.$$

The SNR measured in dB is found using

$$(D-20)$$

$$SNR_{dB} = 10 \cdot \log_{10}(SNR) \text{ dB.}$$

There are several practical topics to keep in mind when estimating SNR by way of frequency-domain samples:

- For computational-efficiency reasons, the length of $x(n)$ should be an integer power of two so that fast Fourier transforms (FFTs) can be used to obtain an $|X(m)|^2$ sequence.
- Due to the spectral symmetry of real-only time samples, we need only examine the $|X(m)|^2$ power samples in the range $0 \leq m \leq N/2$, i.e., positive frequency.
- The Threshold value should be set such that as many of the signal power samples as possible, including any harmonics of the fundamental signal, are above that Threshold value.
- If we repeat our SNR estimation computation on multiple non-overlapping N -sample $x(n)$ sequences, we'll see a noticeable variation (variance) in the various SNR estimation results. To improve the accuracy, and repeatability, of our SNR estimation it's prudent to collect many blocks of N -sample $x(n)$ sequences and perform many FFTs to compute multiple $|X(m)|$ magnitude sequences. Then those multiple $|X(m)|$ sequences are averaged before computing a single $|X(m)|^2$ power sequence for use in [Eq. \(D-19\)](#). The idea is to improve the accuracy (reduce the variance) of our SNR estimations by way of averaging as indicated by [Eq. \(D-2\)](#).

D.5.3 Controlling Test Signal SNR in Software

For completeness, below are methods for adjusting the SNR of a real-valued discrete test signal generated in software. Here's what we mean. Assume we have generated a noise-contaminated zero-mean signal sequence $x(n) = x_s(n) + x_n(n)$, where $x_s(n)$ are noise-free signal samples and $x_n(n)$ are noise-only samples. We can adjust the SNR of $x(n)$ to a desired value of SNR_{new} , measured in dB, by scaling the $x_n(n)$ noise samples as

$$(D-21)$$

$$x_{n,new}(n) = \sqrt{K} \cdot x_n(n)$$

where

$$(D-22)$$

$$K = \frac{\sum_{n=0}^{N-1} [x_s(n)]^2}{\sum_{n=0}^{N-1} [x_n(n)]^2} \cdot 10^{-SNR_{new}/10}.$$

So the SNR of the new $x_{new}(n) = x_s(n) + x_{n,new}(n)$ sequence will be SNR_{new} dB where the original $x_s(n)$ noise-free samples remain unchanged. Notice that the ratio in [Eq. \(D-22\)](#) is the linear (not dB) SNR of the original $x(n)$ sequence.

In a similar manner, we scale the original $x_s(n)$ noise-free samples as

$$(D-23)$$

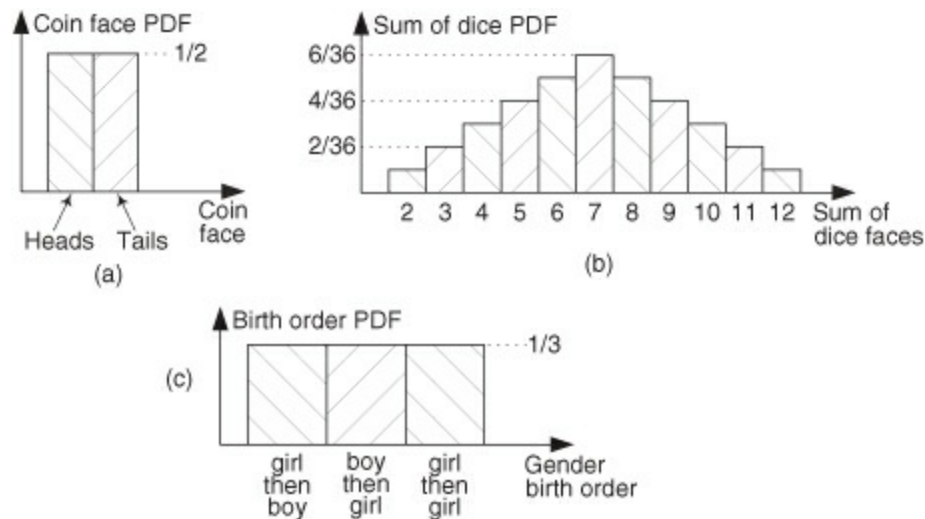
$$x_{s,\text{new}}(n) = x_s(n) / \sqrt{K}$$

so that the SNR of the new $x_{\text{new}}(n) = x_{s,\text{new}}(n) + x_n(n)$ sequence will be the desired SNR_{new} dB. In this case the original $x_n(n)$ noise samples remain unchanged.

D.6 The Mean and Variance of Random Functions

To determine the mean or variance of a random function, we use what's called the *probability density function*. The probability density function (PDF) is a measure of the likelihood of a particular value occurring in some function. We can explain this concept with simple examples of flipping a coin or throwing dice as illustrated in [Figures D-6\(a\)](#) and (b). The result of flipping a coin can only be one of two possibilities: heads or tails. [Figure D-6\(a\)](#) indicates this PDF and shows that the probability (likelihood) is equal to one-half for both heads and tails. That is, we have an equal chance of the coin side facing up being heads or tails. The sum of those two probability values is one, meaning that there's a 100 percent probability that either a head or a tail will occur.

Figure D-6 Simple probability density functions: (a) probability of flipping a single coin; (b) probability of a particular sum of the upper faces of two dice; (c) probability of the order of birth of the girl and her sibling.



[Figure D-6\(b\)](#) shows the probability of a particular sum of the upper faces when we throw a pair of dice. This probability function is not uniform because, for example, we're six times more likely to have the die faces sum to seven than sum to two (snake eyes).

We can say that after tossing the dice a large number of times, we should expect that $6/36 = 16.7$ percent of those tosses would result in sevens, and $1/36 = 2.8$ percent of the time we'll get snake eyes. The sum of those 11 probability values in [Figure D-6\(b\)](#) is also one, telling us that this PDF accounts for all (100 percent) of the possible outcomes of throwing the dice.

The fact that PDFs must account for all possible result conditions is emphasized in an interesting way in [Figure D-6\(c\)](#). Suppose a woman says, "Of my two children, one is a girl. What's the probability that my daughter has a sister?" Be careful now—curiously enough, the answer to this controversial question is not a 50-50 chance. There are more possibilities to consider than the girl just having a brother or a sister. We can think of all the possible combinations of birth order of two children such that one child is a girl. Because we don't know the gender of the first-born child, there are three gender order possibilities: girl, then boy; boy, then girl; and girl, then girl as shown in [Figure D-6\(c\)](#). So the possibility of the daughter having a sister is $1/3$ instead of $1/2$! (Believe it.) Again, the sum of

those three 1/3rd probability values is one.

Two important features of PDFs are illustrated by the examples in [Figure D-6](#): PDFs are always positive and the area under their *curves* must be equal to unity. The very concept of PDFs make them a positive *likelihood* that a particular result will occur, and the fact that some result must occur is equivalent to saying that there's a probability of one (100 percent chance) that we'll have that result. For continuous probability density functions, $p(x)$, we indicate these two characteristics by

$$(D-24)$$

$$\text{PDF values are never negative: } \rightarrow p(x) \geq 0,$$

and

$$(D-25)$$

$$\text{The sum of all the PDF values is one: } \rightarrow \int_{-\infty}^{\infty} p(x) dx = 1.$$

In [Section D.1](#) we illustrated how to calculate the average (mean) and variance of discrete samples. We can also determine these statistical measures for a random function x if we know the PDF of that function. Using μ_x to denote the average of a random function of x , μ_x is defined as

$$(D-26)$$

$$\mu_x = \int_{-\infty}^{\infty} x \cdot p(x) dx$$

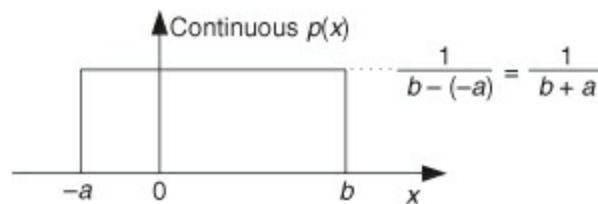
and the variance of x is defined as[\[4\]](#)

$$(D-27)$$

$$\sigma_x^2 = \int_{-\infty}^{\infty} (x - \mu_x)^2 \cdot p(x) dx = \int_{-\infty}^{\infty} x^2 \cdot p(x) dx - \mu_x^2.$$

In digital signal processing, we'll encounter continuous probability density functions that are uniform in value similar to the examples in [Figure D-3](#). In these cases it's easy to use [Eqs. \(D-26\)](#) and [\(D-27\)](#) to determine their average and variance. [Figure D-7](#) illustrates a uniform continuous PDF indicating a random function whose values have an equal probability of being anywhere in the range from $-a$ to b .

Figure D-7 Continuous uniform probability density function.



From [Eq. \(D-25\)](#) we know that the area under the curve must be unity (i.e., the probability is 100 percent that the value will be somewhere under the curve). So the amplitude of $p(x)$ must be the area divided by the width, or $p(x) = 1/(b + a)$. From [Eq. \(D-26\)](#) the average of this $p(x)$ is

$$(D-28)$$

$$\begin{aligned}\mu_x &= \int_{-\infty}^{\infty} x \cdot p(x) dx = \int_{-\infty}^{\infty} x \cdot \frac{1}{b+a} dx = \frac{1}{b+a} \int_{-a}^b x dx \\ &= \frac{1}{b+a} \left[\frac{x^2}{2} \right]_{-a}^b = \frac{b^2 - a^2}{2(b+a)} = \frac{(b+a)(b-a)}{2(b+a)} = \frac{b-a}{2},\end{aligned}$$

which happens to be the midpoint in the range from $-a$ to b . The variance of the PDF in [Figure D-7](#) is

(D-29)

$$\begin{aligned}\sigma_x^2 &= \int_{-\infty}^{\infty} x^2 \cdot p(x) dx - \mu_x^2 = \int_{-a}^b x^2 \frac{1}{b+a} dx - \frac{(b-a)^2}{4} \\ &= \frac{1}{b+a} \left[\frac{x^3}{3} \right]_{-a}^b - \frac{(b-a)^2}{4} = \frac{1}{3(b+a)} \cdot (b^3 + a^3) - \frac{(b-a)^2}{4} \\ &= \frac{(b+a)(b^2 - ab + a^2)}{3(b+a)} - \frac{b^2 - 2ab + a^2}{4} \\ &= \frac{b^2 + 2ab + a^2}{12} = \frac{(b+a)^2}{12}.\end{aligned}$$

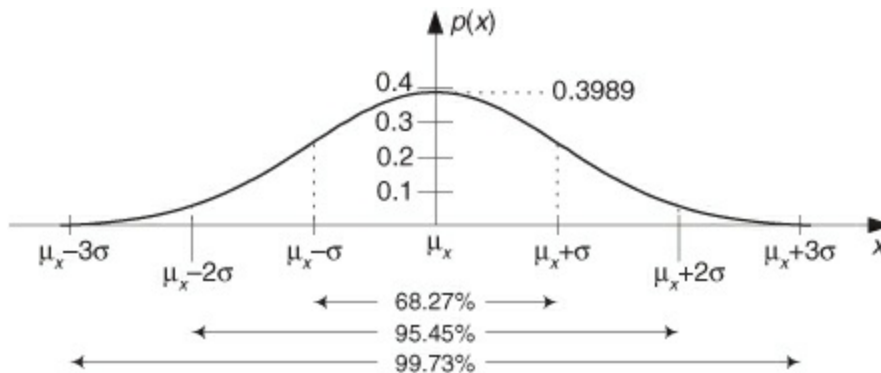
We use the results of [Eqs. \(D-28\)](#) and [\(D-29\)](#) in [Chapter 9](#) to analyze the errors induced by quantization from analog-to-digital converters, and the effects of finite word lengths of hardware registers.

D.7 The Normal Probability Density Function

A probability density function (PDF) that's so often encountered in nature deserves our attention. This function is so common that it's actually called the *normal* PDF and is also sometimes called the *Gaussian* PDF. (A scheme for generating discrete data to fit this function is discussed in [Section 13.12](#).)

This function, whose shape is shown in [Figure D-8](#), is important because random data having this distribution is very useful in testing both software algorithms and hardware processors. The normal PDF is defined mathematically by

Figure D-8 A normal PDF with mean $= \mu_x$ and standard deviation $= \sigma$.



(D-30)

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu_x)^2/2\sigma^2}.$$

The area under the curve is one and the percentages at the bottom of [Figure D-8](#) tell us that, for random functions having a normal distribution, there's a 68.27 percent chance that any particular value of x will differ from the mean by $\leq \sigma$. Likewise, 99.73 percent of all the x data values will be within 3σ of the mean μ_x .

References

- [1] Papoulis, A. *Probability Random Variables, and Stochastic Processes*, McGraw-Hill, New York, 1965, pp. 189, pp. 266–268.
- [2] Miller, Irwin, and Freund, John. *Probability and Statistics for Engineers*, 2nd ed., Prentice Hall, Englewood Cliffs, New Jersey, 1977.
- [3] Meyer, B. *Introductory Probability and Statistical Applications*, Addison-Wesley, Reading, Massachusetts, 1965, pp. 122–125.
- [4] Bendat, Julius, and Piersol, Allen. *Measurement and Analysis of Random Data*, John Wiley and Sons, New York, 1972.

Appendix E. Decibels (dB and dBm)

This appendix introduces the logarithmic function used to improve the magnitude resolution of frequency-domain plots when we evaluate signal spectra, digital filter magnitude responses, and window function magnitude responses. When we use a logarithmic function to plot signal levels in the frequency domain, the vertical axis unit of measure is *decibels*.

E.1 Using Logarithms to Determine Relative Signal Power

In discussing decibels, it's interesting to see how this unit of measure evolved. When comparing continuous (analog) signal levels, early specialists in electronic communications found it useful to define a measure of the difference in powers of two signals. If that difference was treated as the logarithm of a ratio of powers, it could be used as a simple additive measure to determine the overall gain or loss of cascaded electronic circuits. The positive logarithms associated with system components having gain could be added to the negative logarithms of those components having loss quickly to determine the overall gain or loss of the system. With this in mind, the difference between two signal power levels (P_1 and P_2), measured in bels, was defined as the base 10 logarithm of the ratio of those powers, or

(E-1)

$$\text{Power difference} = \log_{10} \left(\frac{P_1}{P_2} \right) \text{bels.}^\dagger$$

[†] The dimensionless unit of measure *bel* was named in honor of Alexander Graham Bell.

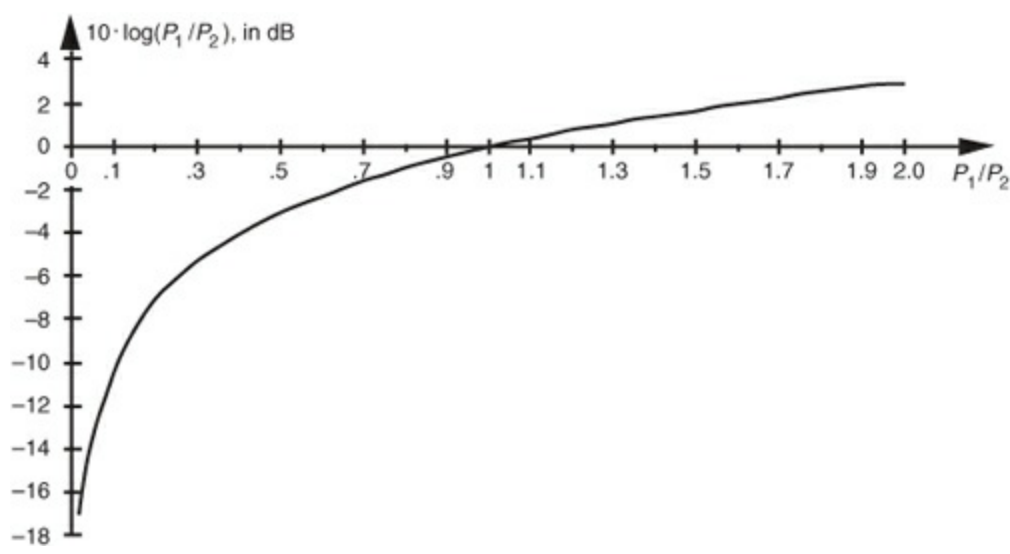
The use of [Eq. \(E-1\)](#) led to another evolutionary step because the unit of bel was soon found to be inconveniently small. For example, it was discovered that the human ear could detect audio power level differences of one-tenth of a bel. Measured power differences smaller than one bel were so common that it led to the use of the decibel (bel/10), effectively making the unit of bel obsolete. The decibel (dB), then, is a unit of measure of the relative power difference of two signals defined as

(E-2)

$$\text{Power difference} = 10 \cdot \log_{10} \left(\frac{P_1}{P_2} \right) \text{dB.}$$

The logarithmic function $10 \cdot \log_{10}(P_1/P_2)$, plotted in [Figure E-1](#), doesn't seem too beneficial at first glance, but its application turns out to be very useful. Notice the large change in the function's value when the power ratio (P_1/P_2) is small, and the gradual change when the ratio is large. The effect of this nonlinearity is to provide greater resolution when the ratio P_1/P_2 is small, giving us a good way to recognize very small differences in the power levels of signal spectra, digital filter responses, and window function frequency responses.

Figure E-1 Logarithmic decibel function of [Eq. \(E-2\)](#).



Let's demonstrate the utility of the logarithmic function's variable resolution. First, remember that the power of any frequency-domain sequence representing signal magnitude $|X(m)|$ is proportional to $|X(m)|$ squared. For convenience, the proportionality constant is assumed to be one, so we say the power of $|X(m)|$ is

$$(E-3)$$

$$\text{discrete power spectrum of } X(m) = |X(m)|^2.$$

Although [Eq. \(E-3\)](#) may not actually represent power (in watts) in the classical sense, it's the squaring operation that's important here, because it's analogous to the traditional magnitude squaring operation used to determine the power of continuous signals. (Of course, if $X(m)$ is complex, we can calculate the power spectrum sequence using $|X(m)|^2 = X_{\text{real}}(m)^2 + X_{\text{imag}}(m)^2$.) Taking ten times the log of [Eq. \(E-3\)](#) allows us to express a power spectrum sequence $X_{\text{dB}}(m)$ in dB as

$$(E-4)$$

$$X_{\text{dB}}(m) = 10 \cdot \log_{10}(|X(m)|^2) \text{ dB}.$$

Because $\log(x^2) = \log(x) + \log(x) = 2\log(x)$, we can eliminate the squaring operation in [Eq. \(E-4\)](#) by doubling the factor of ten and represent the power spectrum sequence by the expression

$$(E-5)$$

$$X_{\text{dB}}(m) = 20 \cdot \log_{10}(|X(m)|) \text{ dB}.$$

Without the need for the squaring operation, [Eq. \(E-5\)](#) is a more convenient way than [Eq. \(E-4\)](#) to calculate the $X_{\text{dB}}(m)$ power spectrum sequence from the $X(m)$ sequence.

[Equations \(E-4\)](#) and [\(E-5\)](#), then, are the expressions used to convert a linear magnitude axis to a logarithmic magnitude-squared, or power, axis measured in dB. What we most often see in the literature are normalized log magnitude spectral plots where each value of $|X(m)|^2$ is divided by the first $|X(0)|^2$ power value (for $m = 0$), as

$$(E-6)$$

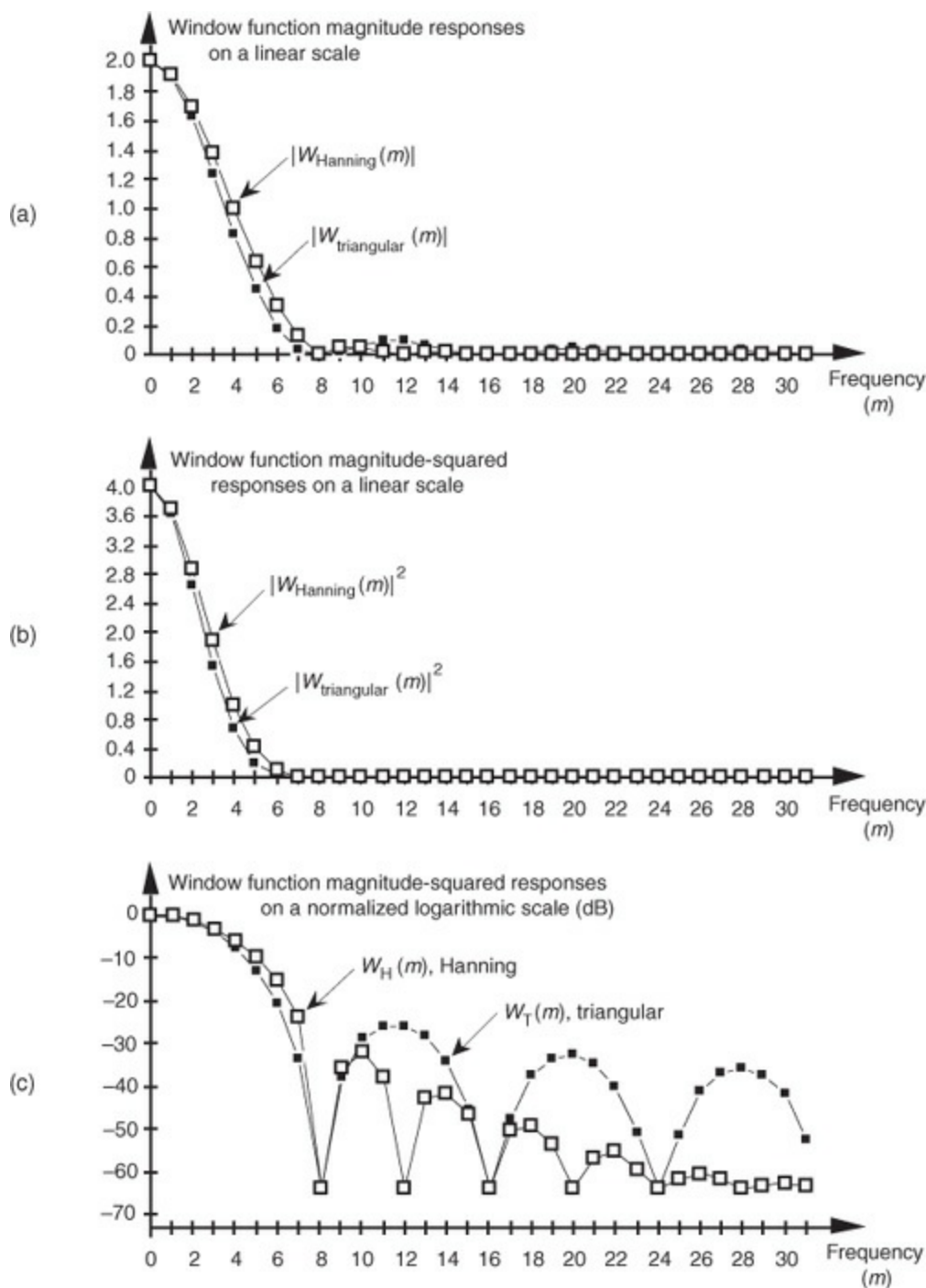
$$\text{normalized } X_{\text{dB}}(m) = 10 \cdot \log_{10}\left(\frac{|X(m)|^2}{|X(0)|^2}\right) = 20 \cdot \log_{10}\left(\frac{|X(m)|}{|X(0)|}\right) \text{ dB}.$$

The division by the $|X(0)|^2$ or $|X(0)|$ value always forces the first value in the normalized log

magnitude sequence $X_{\text{dB}}(m)$ equal to 0 dB.[‡] This makes it easy for us to compare multiple log magnitude spectral plots. To illustrate, let's look at the frequency-domain representations of the Hanning and triangular window functions. The magnitudes of those frequency-domain functions are plotted on a linear scale in [Figure E-2\(a\)](#) where we've arbitrarily assigned their peak values to be 2. Comparing the two linear scale magnitude sequences, $W_{\text{Hanning}}(m)$ and $W_{\text{triangular}}(m)$, we can see some minor differences between their magnitude values. If we're interested in the power associated with the two window functions, we square the two magnitude functions and plot them on a linear scale as in [Figure E-2\(b\)](#). The difference between the two window functions' power sequences is impossible to see above the frequency of, say, $m = 8$ in [Figure E-2\(b\)](#). Here's where the dB scale helps us out. If we plot the normalized log magnitude versions of the two magnitude-squared sequences on a logarithmic dB scale using [Eq. \(E-6\)](#), the difference between the two functions will become obvious.

[‡] That's because $\log_{10}(|X(0)|/|X(0)|) = \log_{10}(1) = 0$.

Figure E-2 Hanning (white squares) and triangular (black squares) window functions in the frequency domain: (a) magnitude responses using a linear scale; (b) magnitude-squared responses using a linear scale; (c) log magnitude responses using a normalized dB scale.



Normalization, in the case of the Hanning window, amounts to calculating the log magnitude sequence normalized over $|W_{\text{Hanning}}(0)|$ as

(E-7)

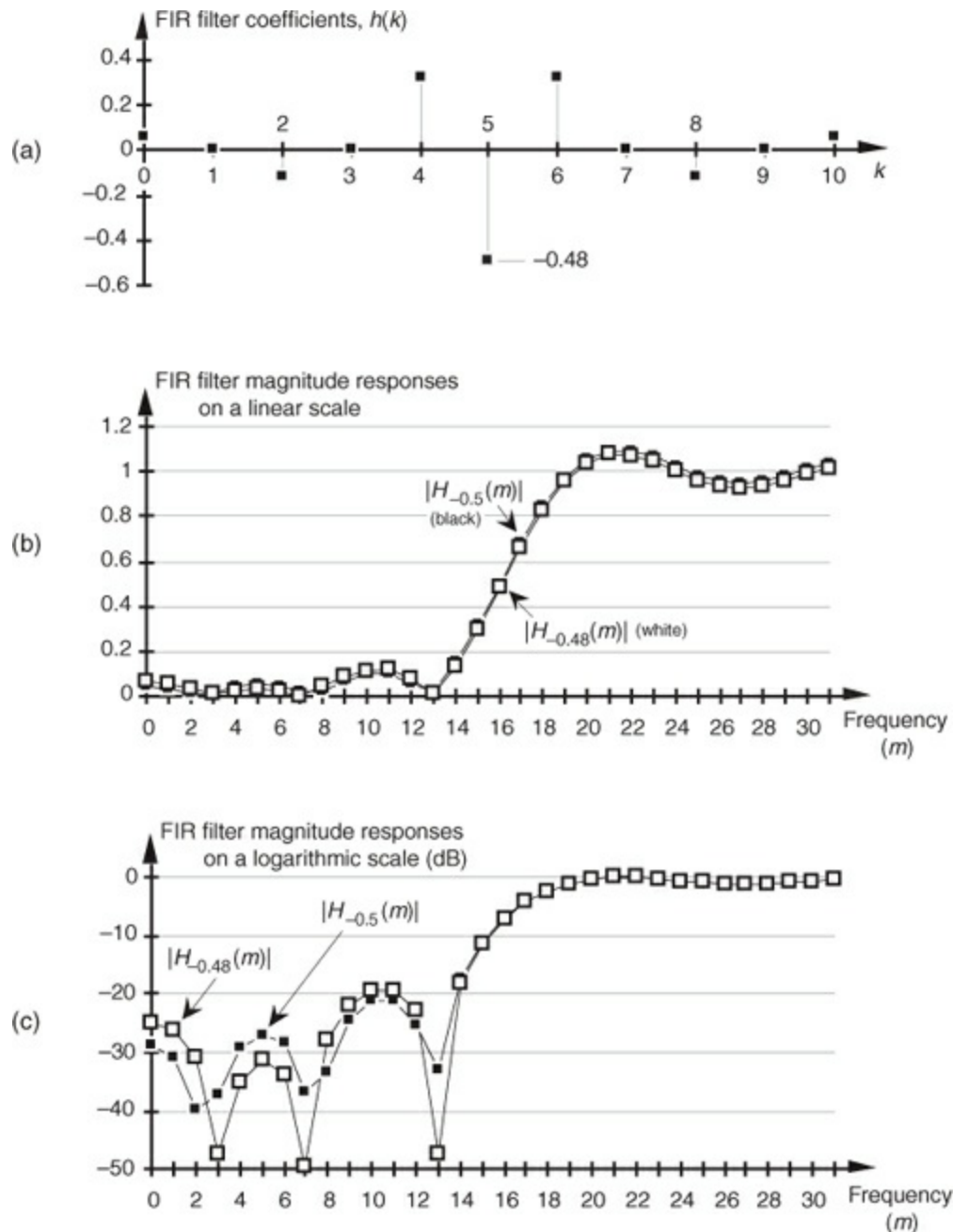
$$W_H(m) = 10 \cdot \log_{10} \left(\frac{|W_{\text{Hanning}}(m)|^2}{|W_{\text{Hanning}}(0)|^2} \right) = 20 \cdot \log_{10} \left(\frac{|W_{\text{Hanning}}(m)|}{|W_{\text{Hanning}}(0)|} \right) \text{ dB}.$$

The normalized log magnitude sequences are plotted in [Figure E-2\(c\)](#). We can now clearly see the difference in the magnitude-squared window functions in [Figure E-2\(c\)](#) as compared to the linear plots in [Figure E-2\(b\)](#). Notice how normalization forced the peak values for both log magnitude functions in [Figure E-2\(c\)](#) to be zero dB. (The dots in [Figure E-2](#) are connected by lines to emphasize the sidelobe features of the two log magnitude sequences.)

Although we've shown the utility of dB plots using window function frequency responses as examples, the dB scale is equally useful when we're plotting signal-power spectra or digital filter

frequency responses. We can further demonstrate the dB scale using a simple digital filter example. Let's say we're designing an 11-tap highpass FIR filter whose coefficients are shown in [Figure E-3\(a\)](#). If the center coefficient $h(5)$ is -0.48 , the filter's frequency magnitude response $|H_{-0.48}(m)|$ can be plotted as the white dots on the linear scale in [Figure E-3\(b\)](#). Should we change $h(5)$ from -0.48 to -0.5 , the new frequency magnitude response $|H_{-0.5}(m)|$ would be the black dots in [Figure E-3\(b\)](#). It's difficult to see much of a difference between $|H_{-0.48}(m)|$ and $|H_{-0.5}(m)|$ on a linear scale. If we used [Eq. \(E-6\)](#) to calculate two normalized log magnitude sequences, they could be plotted as shown in [Figure E-3\(c\)](#), where the filter sidelobe effects of changing $h(5)$ from -0.48 to -0.5 are now easy to see.

Figure E-3 FIR filter magnitude responses: (a) FIR filter time-domain coefficients; (b) magnitude responses using a linear scale; (c) log magnitude responses using the dB scale.



E.2 Some Useful Decibel Numbers

If the reader uses dB scales on a regular basis, there are a few constants worth committing to memory. A power difference of 3 dB corresponds to a power factor of two; that is, if the magnitude-squared

ratio of two different frequency components is 2, then from [Eq. \(E-2\)](#),

(E-8)

$$\text{power difference} = 10 \cdot \log_{10}\left(\frac{2}{1}\right) = 10 \cdot \log_{10}(2) = 3.01 \approx 3 \text{ dB}.$$

Likewise, if the magnitude-squared ratio of two different frequency components is 1/2, then the relative power difference is -3 dB because

(E-9)

$$\text{power difference} = 10 \cdot \log_{10}\left(\frac{1}{2}\right) = 10 \cdot \log_{10}(0.5) = -3.01 \approx -3 \text{ dB}.$$

[Table E-1](#) lists several magnitude and power ratios versus dB values that are worth remembering. Keep in mind that decibels indicate only relative power relationships. For example, if we're told that signal A is 6 dB above signal B, we know that the power of signal A is four times that of signal B, and that the magnitude of signal A is twice the magnitude of signal B. We may not know the absolute power of signals A and B in watts, but we do know that the power ratio is $P_A/P_B = 4$.

Table E-1 Some Useful Logarithmic Relationships

Magnitude ratio	Magnitude-squared power (P_1/P_2) ratio	Relative dB (approximate)	
$10^{-1/2}$	10^{-1}	-10	$\leftarrow P_1$ is one-tenth P_2
2^{-1}	$2^{-2} = 1/4$	-6	$\leftarrow P_1$ is one-fourth P_2
$2^{-1/2}$	$2^{-1} = 1/2$	-3	$\leftarrow P_1$ is one-half P_2
2^0	$2^0 = 1$	0	$\leftarrow P_1$ is equal to P_2
$2^{1/2}$	$2^1 = 2$	3	$\leftarrow P_1$ is twice P_2
2^1	$2^2 = 4$	6	$\leftarrow P_1$ is four times P_2
$10^{1/2}$	$10^1 = 10$	10	$\leftarrow P_1$ is ten times P_2
10^1	$10^2 = 100$	20	$\leftarrow P_1$ is one hundred times P_2
$10^{3/2}$	$10^3 = 1000$	30	$\leftarrow P_1$ is one thousand times P_2

E.3 Absolute Power Using Decibels

Let's discuss another use of decibels that the reader may encounter in the literature. It's convenient for practitioners in the electronic communications field to measure continuous signal-power levels referenced to a specific absolute power level. In this way, they can speak of absolute power levels in watts while taking advantage of the convenience of decibels. The most common absolute power reference level used is the milliwatt. For example, if P_2 in [Eq. \(E-2\)](#) is a reference power level of one milliwatt, then

(E-10)

$$\text{absolute power of } P_1 = 10 \cdot \log_{10}\left(\frac{P_1}{P_2}\right) = 10 \cdot \log_{10}\left(\frac{P_1 \text{ in watts}}{1 \text{ milliwatt}}\right) \text{ dBm}.$$

The dBm unit of measure in [Eq. \(E-10\)](#) is read as "dB relative to a milliwatt." Thus, if a continuous signal is specified as having a power of 3 dBm, we know that the signal's absolute power level is 2 times one milliwatt, or 2 milliwatts. Likewise, a -10 dBm signal has an absolute power of 0.1 milliwatts.[‡]

† Other absolute reference power levels can be used. People involved with high-power transmitters sometimes use a single watt as their reference power level. Their unit of power using decibels is the dBW, read as “dB relative to a watt.” In this case, for example, 3 dBW is equal to a 2-watt power level.

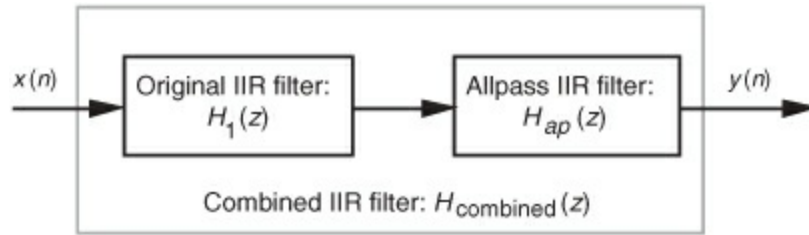
The reader should take care not to inadvertently use dB and dBm interchangeably. They mean very different things. Again, dB is a relative power level relationship, and dBm is an absolute power level in milliwatts.

Appendix F. Digital Filter Terminology

The first step in becoming familiar with digital filters is to learn to speak the language used in the filter business. Fortunately, the vocabulary of digital filters corresponds very well to the mother tongue used for continuous (analog) filters—so we don't have to unlearn anything that we already know. This appendix is an introduction to the terminology of digital filters.

Allpass filter—an IIR filter whose magnitude response is unity over its entire frequency range, but whose phase response is variable. Allpass filters are typically appended in a cascade arrangement following a standard IIR filter, $H_1(z)$, as shown in [Figure F-1](#).

Figure F-1 Typical use of an allpass filter.



An allpass filter, $H_{ap}(z)$, can be designed so that its phase response compensates for, or *equalizes*, the nonlinear phase response of an original IIR filter[1–3]. Thus, the phase response of the combined filter, $H_{combined}(z)$, is more linear than the original $H_1(z)$, and this is particularly desirable in communications systems. In this context, an allpass filter is sometimes called a *phase equalizer*.

Allpass filters have the property that the numerator polynomial coefficients in the filter's $H(z)$ transfer function are a reverse-order version of the denominator polynomial coefficients. For example, the following transfer function describes a 2nd-order allpass filter:

(F-1)

$$H_{\text{allpass}}(z) = \frac{B + Az^{-1} + z^{-2}}{1 + Az^{-1} + Bz^{-2}}$$

where the numerator polynomial coefficients are $[B, A, 1]$ and the denominator polynomial coefficients are $[1, A, B]$.

Attenuation—an amplitude loss, usually measured in dB, incurred by a signal after passing through a digital filter. Filter attenuation is the ratio, at a given frequency, of the signal amplitude at the output of the filter divided by the signal amplitude at the input of the filter, defined as

(F-2)

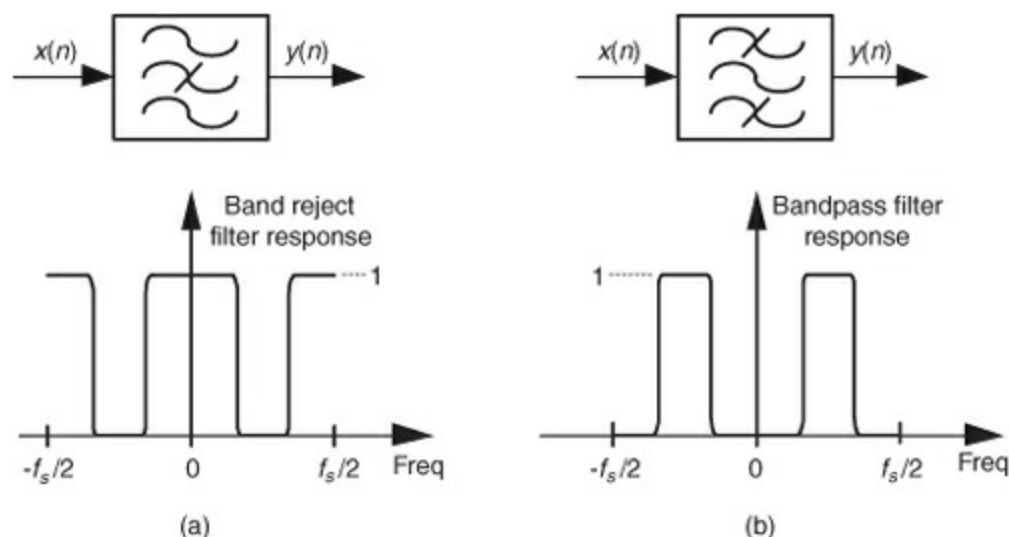
$$\text{attenuation} = 20 \cdot \log_{10} \left(\frac{a_{\text{out}}}{a_{\text{in}}} \right) \text{ dB}.$$

For a given frequency, if the output amplitude of the filter is smaller than the input amplitude, the ratio in [Eq. \(F-2\)](#) is less than one, and the attenuation is a negative number.

Band reject filter—a filter that rejects (attenuates) one frequency band and passes both a lower- and a higher-frequency band. [Figure F-2\(a\)](#) depicts the frequency response of an ideal band reject

filter. This filter type is sometimes called a [notch filter](#).

Figure F-2 Filter symbols and frequency responses: (a) band reject filter; (b) bandpass filter.



Bandpass filter—a filter, as shown in [Figure F-2\(b\)](#), that passes one frequency band and attenuates frequencies above and below that band.

Bandwidth—the frequency width of the passband of a filter. For a lowpass filter, the bandwidth is equal to the cutoff frequency. For a bandpass filter, the bandwidth is typically defined as the frequency difference between the upper and lower 3 dB points.

Bessel function—a mathematical function used to produce the most linear phase response of all IIR filters with no consideration of the frequency magnitude response. Specifically, filter designs based on Bessel functions have maximally constant group delay.

Butterworth function—a mathematical function used to produce maximally flat filter magnitude responses with no consideration of phase linearity or group delay variations. Filter designs based on a Butterworth function have no amplitude ripple in either the passband or the stopband. Unfortunately, for a given filter order, Butterworth designs have the widest transition region of the most popular filter design functions.

Cascaded filters—a filtering *system* where multiple individual filters are connected in series; that is, the output of one filter drives the input of the following filter as illustrated in [Figures F-1](#) and [6-37\(a\)](#).

Center frequency (f_0)—the frequency lying at the midpoint of a bandpass filter. [Figure F-2\(b\)](#) shows the f_0 center frequency of a bandpass filter.

Chebyshev function—a mathematical function used to produce passband or stopband ripples constrained within fixed bounds. There are families of Chebyshev functions based on the amount of ripple, such as 1 dB, 2 dB, and 3 dB of ripple. Chebyshev filters can be designed to have a frequency response with ripples in the passband and a flat stopband (Chebyshev Type I), or flat passbands and ripples in the stopband (Chebyshev Type II). Chebyshev filters cannot have ripples in both the passband and the stopband. Digital filters based upon Chebyshev functions have steeper transition region roll-off but more nonlinear-phase response characteristics than, say, Butterworth filters.

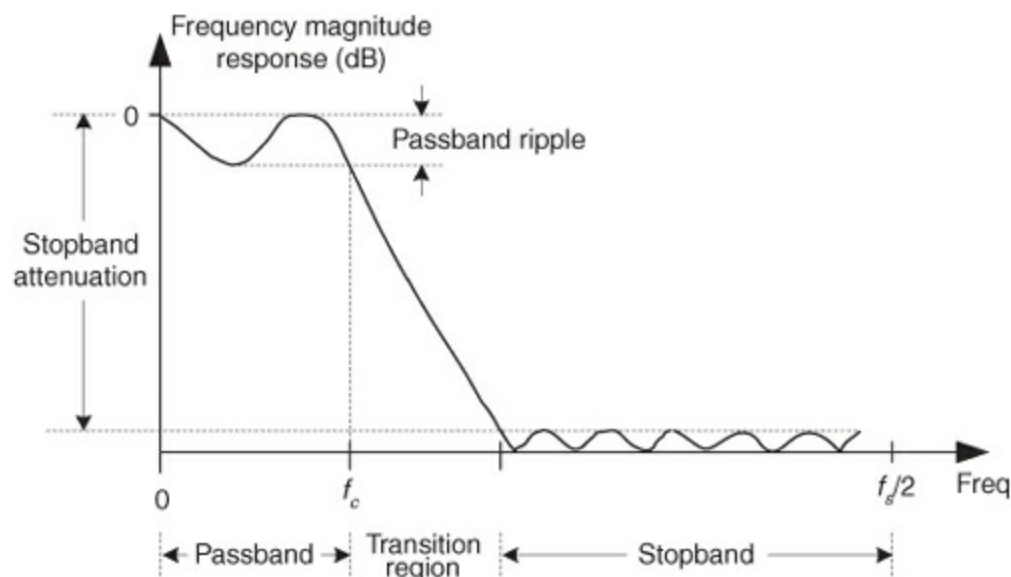
CIC filter—cascaded integrator-comb filter. CIC filters are computationally efficient, linear-phase,

recursive, FIR, lowpass filters used in sample rate change applications. Those filters are discussed in [Chapter 10](#).

Coefficients—see [filter coefficients](#).

Cutoff frequency—the highest passband frequency for lowpass filters (and the lower passband frequency for highpass filters) where the magnitude response is within the peak-peak passband ripple region. [Figure F-3](#) illustrates the f_c cutoff frequency of a lowpass filter.

Figure F-3 A lowpass digital filter frequency response. The stopband relative amplitude is -20 dB.



Decibels (dB)—a unit of attenuation, or gain, used to express the relative voltage or power between two signals. For filters, we use decibels to indicate cutoff frequencies (-3 dB) and stopband signal levels (-20 dB) as illustrated in [Figure F-3](#). [Appendix E](#) discusses decibels in more detail.

Decimation filter—a lowpass digital FIR filter where the output sample rate is less than the filter's input sample rate. As discussed in [Section 10.1](#), to avoid aliasing problems, the output sample rate must not violate the Nyquist criterion.

Digital filter—computational process, or algorithm, transforming a discrete sequence of numbers (the input) into another discrete sequence of numbers (the output) having a modified frequency-domain spectrum. Digital filtering can be in the form of a software routine operating on data stored in computer memory or can be implemented with dedicated hardware.

Elliptic function—a mathematical function used to produce the sharpest roll-off for a given number of filter taps. However, filters designed by using elliptic functions, also called *Cauer filters*, have the poorest phase linearity of the most common IIR filter design functions. The ripples in the passband and stopband are equal with elliptic filters.

Envelope delay—see [group delay](#).

Filter coefficients—the set of constants, also called [tap weights](#), used to multiply against delayed signal sample values within a digital filter structure. Digital filter design is an exercise in determining the filter coefficients that will yield the desired filter frequency response. For an FIR filter, by definition, the filter coefficients are the impulse response of the filter.

Filter order—a number describing the highest exponent in either the numerator or denominator of the z -domain transfer function of a digital filter. For tapped-delay line FIR filters, there is no

denominator in the transfer function and the filter order is merely the number of delay elements used in the filter structure. Generally, the larger the filter order, the better the frequency-domain performance, and the higher the computational workload, of the filter.

Finite impulse response (FIR) filter—defines a class of digital filters that have only zeros on the z -plane. The key implications of this are: (1) FIR filter impulse responses have finite time durations, (2) FIR filters are always stable, and (3) FIR filters can have exactly linear phase responses (so long as the filters' impulse response samples are symmetrical, or antisymmetrical). For a given filter order, digital FIR filters have a much more gradual transition region roll-off (poorer performance) than digital IIR filters. FIR filters can be implemented with both nonrecursive (tapped-delay line) and recursive (CIC filters, for example) structures.

Frequency magnitude response—a frequency-domain description of how a filter interacts with input signals. The frequency magnitude response in [Figure F-3](#) is a curve of filter attenuation (in dB) versus frequency. Associated with a filter's magnitude response is a phase response.

Group delay—the negative of the derivative of a filter's frequency-domain phase response with respect to frequency, $G(\omega) = -d(H_\phi(\omega))/d(\omega)$. If a filter's complex frequency response is represented in polar form as

(F-3)

$$H(\omega) = |H(\omega)| e^{jH_\phi(\omega)}$$

where digital frequency ω is continuous and ranges from $-\pi$ to π radians/sample, corresponding to a cyclic frequency range of $-f_s/2$ to $f_s/2$ Hz, then the filter's group delay is defined as

(F-4)

$$G(\omega) = \frac{-d(H_\phi(\omega))}{d(\omega)}.$$

Because the dimensions of $H_\phi(\omega)$ are radians, and the dimensions of ω are radians/sample, the dimensions of group delay $G(\omega)$ are time measured in samples.

If a filter's complex frequency response is expressed in terms of a normalized frequency variable as

(F-5)

$$H(f) = |H(f)| e^{jH_\phi(f)}$$

where frequency f is continuous and is in the range of $-0.5 \leq f \leq 0.5$, then the group delay $G(f)$ is defined as

(F-6)

$$G(f) = \frac{-1}{2\pi} \cdot \frac{d(H_\phi(f))}{d(f)}.$$

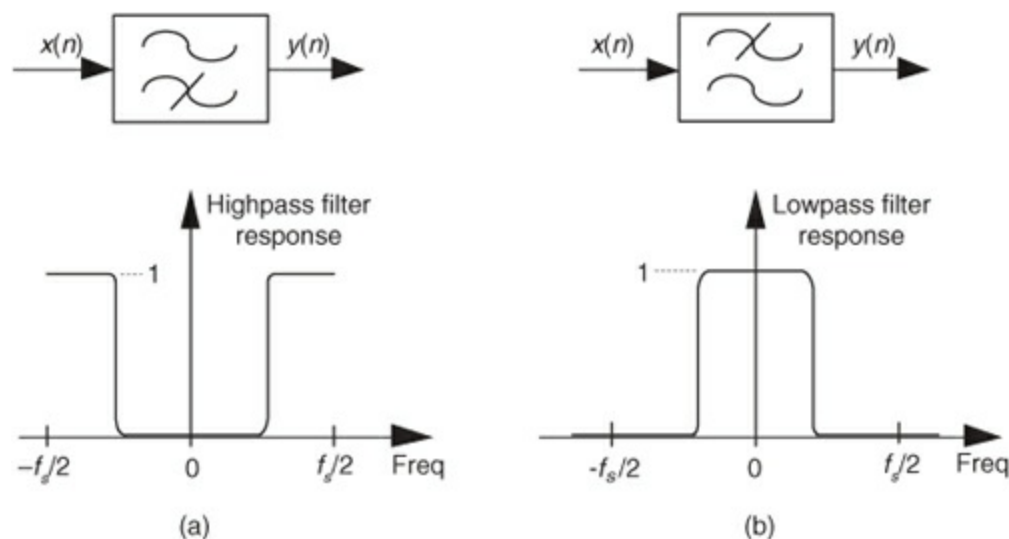
The concept of group delay deserves additional explanation beyond a simple algebraic definition. For an ideal lowpass filter, for example, the frequency-domain phase response will be linear and the group delay would be constant. Group delay can also be thought of as the propagation time delay of the envelope (the information) of an amplitude-modulated (AM) signal as it passes

through a digital filter. (In this context, group delay is often called [envelope delay](#).) If a filter's passband group delay is not constant (a nonlinear-phase filter), then *group delay distortion* occurs because signals at different frequencies take different amounts of time (a different number of sample time intervals) to pass through the filter.

Half-band filter—a type of FIR filter whose transition region is centered at one-quarter of the sampling rate, or $f_s/4$. Specifically, the end of the passband and the beginning of the stopband are equally spaced about $f_s/4$. Due to their frequency-domain symmetry, half-band filters are often used in decimation filtering schemes because half of their time-domain coefficients are zero. This reduces the number of necessary filter multiplications, as described in [Section 5.7](#).

Highpass filter—a filter that passes high frequencies and attenuates low frequencies, as shown in [Figure F-4\(a\)](#). We've all experienced a kind of highpass filtering in our living rooms. Notice what happens when we turn up the treble control (or turn down the bass control) on our home stereo systems. The audio amplifier's normally flat frequency response changes to a kind of analog highpass filter, giving us that sharp and tinny sound as the high-frequency components of the music are being accentuated.

Figure F-4 Filter symbols and frequency responses: (a) highpass filter; (b) low pass filter.



Impulse response—a digital filter's time-domain output sequence when the input is a single unity-valued sample (impulse) preceded and followed by zero-valued samples. A digital filter's frequency-domain response can be calculated by taking the discrete Fourier transform of the filter's time-domain impulse response[\[4\]](#).

Infinite impulse response (IIR) filter—a class of digital filters that may have both zeros and poles on the z -plane. As such, IIR filters are not guaranteed to be stable and almost always have nonlinear phase responses. For a given filter order (number of IIR feedback taps), IIR filters have a much steeper transition region roll-off than digital FIR filters.

Linear-phase filter—a filter that exhibits a constant change in phase angle (degrees) as a function of frequency. The resultant filter phase plot versus frequency is a straight line. As such, a linear-phase filter's group delay is a constant. To preserve the integrity of their information-carrying signals, linear phase is an important criterion for filters used in communications systems.

Lowpass filter—a filter that passes low frequencies and attenuates high frequencies as shown in [Figure F-4\(b\)](#). By way of example, we experience lowpass filtering when we turn up the bass

control (or turn down the treble control) on our home stereo systems, giving us that dull, muffled sound as the high-frequency components of the music are being attenuated.

Nonrecursive filter—a digital filter implementation where no filter output sample is ever retained for later use in computing a future filter output sample. Such filters have no “feedback” signal paths.

Notch filter—see [band reject filter](#).

Order—see [filter order](#).

Passband—that frequency range over which a filter passes signal energy with minimum attenuation, usually defined as the frequency range where the magnitude response is within the peak-peak passband ripple region, as depicted in [Figure F-3](#).

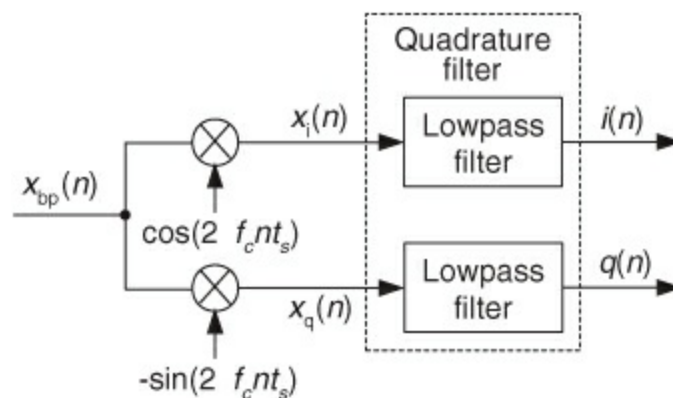
Passband ripple—peak-peak fluctuations, or variations, in the frequency magnitude response within the passband of a filter as illustrated in [Figure F-3](#).

Phase response—the difference in phase, at a particular frequency, between an input sinewave and the output sinewave at that frequency. The phase response, sometimes called *phase delay*, is usually depicted by a curve showing the filter’s phase shift versus frequency. [Section 5.8](#) discusses digital filter phase response in more detail.

Phase wrapping—an artifact of arctangent software routines, used to calculate phase angles, that causes apparent phase discontinuities. When a true phase angle is in the range of -180° to -360° , some software routines automatically convert those angles to their equivalent positive angles in the range of 0° to $+180^\circ$. [Section 5.8](#) illustrates an example of phase wrapping when the phase of an FIR filter is calculated.

Quadrature filter—a dual-path digital filter operating on complex signals, as shown in [Figure F-5](#). One filter operates on the in-phase $i(n)$ data, and the other filter processes the quadrature-phase $q(n)$ signal data. Quadrature filtering is normally performed on complex signals, whose spectra are centered at zero Hz, using lowpass digital filters.

Figure F-5 Two lowpass filters used to implement quadrature filtering.



Recursive filter—a digital filter implementation where current filter output samples are retained for later use in computing future filter output samples. Such filters have “feedback” signal paths.

Relative attenuation—attenuation measured relative to the largest magnitude value. The largest signal level (minimum attenuation) is typically assigned the reference level of zero dB, as depicted in [Figure F-3](#), making all other magnitude points on a frequency-response curve negative dB

values.

Ripple—refers to fluctuations (measured in dB) in the passband, or stopband, of a filter's frequency-response curve. Elliptic and Chebyshev-based filters have equiripple characteristics in that their ripple is constant across their passbands. Bessel- and Butterworth-derived filters have no ripple in their passband responses. Ripples in the stopband response are sometimes called *out-of-band ripple*.

Roll-off—a term used to describe the steepness, or slope, of the filter response in the transition region from the passband to the stopband. A particular digital filter may be said to have a roll-off of 12 dB/octave, meaning that the second-octave frequency would be attenuated by 24 dB, and the third-octave frequency would be attenuated by 36 dB, and so on.

Shape factor—a term used to indicate the steepness of a filter's roll-off. Shape factor is normally defined as the ratio of a filter's passband width divided by the passband width plus the transition region width. The smaller the shape factor value, the steeper the filter's roll-off. For an ideal filter with a transition region of zero width, the shape factor is unity. The term *shape factor* is also used to describe analog filters.

Stopband—that band of frequencies attenuated by a digital filter. [Figure F-3](#) shows the stopband of a lowpass filter.

Structure—a fancy term meaning the block diagram, the signal-flow implementation, of a digital filter. For example, lowpass moving average filters may be built (implemented) with both nonrecursive structures and recursive structures.

Tap—a multiplication operation inside a digital filter that computes the product of a single data value times a single filter coefficient.

Tap weights—*see* [filter coefficients](#).

Chebyshev function—*see* [Chebyshev function](#).

Transfer function—a mathematical expression of the ratio of the output of a digital filter divided by the input of the filter as expressed in a transform domain (e.g., z -domain, Laplace, frequency). Given the transfer function, we can determine the filter's frequency magnitude and phase responses.

Transition region—the frequency range over which a filter transitions from the passband to the stopband. [Figure F-3](#) illustrates the transition region of a lowpass filter. The transition region is sometimes called the *transition band*.

Transversal filter—in the field of digital filtering, *transversal filter* is another name for FIR filters implemented with the nonrecursive structures described in [Chapter 5](#).

Zero-phase filter—an off-line (because it operates on a block of filter input samples) filtering method which cancels the nonlinear phase response of an IIR filter. [Section 13.12](#) details this non-real-time filtering technique.

References

- [1] Rabiner, L. R., and Gold, B. *The Theory and Application of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1975, pp. 206, 273, and 288.
- [2] Oppenheim, A. V., and Schaffer, R. W. *Discrete Time Signal Processing*, Prentice Hall,

Englewood Cliffs, New Jersey, 1989, pp. 236 and 441.

- [3] Laakso, T. I., et al. "Splitting the Unit Delay," *IEEE Signal Processing Magazine*, January 1996, p. 46.
- [4] Pickerd, J. "Impulse-Response Testing Lets a Single Test Do the Work of Thousands," *EDN*, April 27, 1995.

Appendix G. Frequency Sampling Filter Derivations

While much of the algebra related to frequency sampling filters is justifiably omitted in the literature, several derivations are included here for two reasons: first, to validate the equations used in [Section 7.5](#); and second, to show the various algebraic acrobatics that may be useful in your future digital signal processing analysis efforts.

G.1 Frequency Response of a Comb Filter

The frequency response of a comb filter is $H_{\text{comb}}(z)$ evaluated on the unit circle. We start by substituting $e^{j\omega}$ for z in $H_{\text{comb}}(z)$ from [Eq. \(7-37\)](#), because $z = e^{j\omega}$ defines the unit circle, giving

(G-1)

$$H_{\text{comb}}(e^{j\omega}) = H_{\text{comb}}(z) \big|_{z=e^{j\omega}} = (1 - e^{-jN\omega}).$$

Factoring out the half-angled exponential $e^{-j\omega N/2}$, we have

(G-2)

$$H_{\text{comb}}(e^{j\omega}) = e^{-j\omega N/2} (e^{j\omega N/2} - e^{-j\omega N/2}).$$

Using Euler's identity, $2j\sin(\alpha) = e^{j\alpha} - e^{-j\alpha}$, we arrive at

(G-3)

$$H_{\text{comb}}(e^{j\omega}) = e^{-j\omega N/2} [2j\sin(\omega N/2)].$$

Replacing j with $e^{j\pi/2}$, we have

(G-4)

$$H_{\text{comb}}(e^{j\omega}) = e^{-j(\omega N - \pi)/2} 2\sin(\omega N/2).$$

Determining the maximum magnitude response of a filter is useful in DSP. Ignoring the phase shift term (complex exponential) in [Eq. \(G-4\)](#), the frequency-domain magnitude response of a comb filter is

(G-5)

$$|H_{\text{comb}}(e^{j\omega})| = 2 |\sin(\omega N/2)|,$$

with the maximum magnitude being 2.

G.2 Single Complex FSF Frequency Response

The frequency response of a single-section complex FSF is $H_{\text{ss}}(z)$ evaluated on the unit circle. We start by substituting $e^{j\omega}$ for z in $H_{\text{ss}}(z)$, because $z = e^{j\omega}$ defines the unit circle. Given an $H_{\text{ss}}(z)$ of

(G-6)

$$H_{\text{ss}}(z) = (1 - z^{-N}) \frac{H(k)}{1 - [e^{j2\pi k/N}]z^{-1}},$$

we replace the z terms with $e^{j\omega}$, giving

(G-7)

$$\begin{aligned}
H_{ss}(e^{j\omega}) &= H_{ss}(z) \big|_{z=e^{j\omega}} = (1 - e^{-jN\omega}) \frac{H(k)}{1 - [e^{j2\pi k/N}]e^{-j\omega}} \\
&= H(k) \frac{1 - e^{-jN\omega}}{1 - e^{-j(\omega - 2\pi k/N)}}.
\end{aligned}$$

Factoring out the half-angled exponentials $e^{-j\omega N/2}$ and $e^{-j(\omega/2 - \pi k/N)}$, we have

(G-8)

$$H_{ss}(e^{j\omega}) = H(k) \frac{e^{-j\omega N/2} (e^{j\omega N/2} - e^{-j\omega N/2})}{e^{-j(\omega/2 - \pi k/N)} (e^{j(\omega/2 - \pi k/N)} - e^{-j(\omega/2 - \pi k/N)})}.$$

Using Euler's identity, $2j\sin(\alpha) = e^{j\alpha} - e^{-j\alpha}$, we arrive at

(G-9)

$$H_{ss}(e^{j\omega}) = H(k) \frac{e^{-j\omega N/2} [2j\sin(\omega N/2)]}{e^{-j\omega/2} e^{j\pi k/N} [2j\sin(\omega/2 - \pi k/N)]}.$$

Canceling common factors and rearranging terms in preparation for our final form, we have the desired frequency response of a single-section complex FSF:

(G-10)

$$H_{ss}(e^{j\omega}) = e^{-j\omega(N-1)/2} e^{-j\pi k/N} H(k) \frac{\sin(\omega N/2)}{\sin(\omega/2 - \pi k/N)}.$$

Next we derive the maximum amplitude response of a single-section FSF when its pole is on the unit circle and $H(k) = 1$. Ignoring those phase shift factors (complex exponentials) in [Eq. \(G-10\)](#), the amplitude response of a single-section FSF is

(G-11)

$$H_{ss,amp}(e^{j\omega}) = \frac{\sin(\omega N/2)}{\sin(\omega/2 - \pi k/N)}.$$

We want to know the value of [Eq. \(G-11\)](#) when $\omega = 2\pi k/N$, because that's the value of ω at the pole locations, but $|H_{ss}(e^{j\omega})|_{\omega=2\pi k/N}$ is indeterminate as

(G-12)

$$H_{ss,amp}(e^{j\omega}) \big|_{\omega=2\pi k/N} = \frac{\sin(\pi k)}{\sin(\pi k/N - \pi k/N)} = \frac{\sin(\pi k)}{\sin(0)} = \frac{0}{0}.$$

Applying the Marquis de L'Hopital's Rule to [Eq. \(G-11\)](#) yields

(G-13)

$$\begin{aligned}
H_{ss,amp}(e^{j\omega}) \big|_{\omega \rightarrow 2\pi k/N} &= \frac{d[\sin(\omega N/2)]/d\omega}{d[\sin(\omega/2 - \pi k/N)]/d\omega} \bigg|_{\omega \rightarrow 2\pi k/N} \\
&= \frac{N/2}{1/2} \frac{\cos(\omega N/2)}{\cos(\omega/2 - \pi k/N)} \bigg|_{\omega \rightarrow 2\pi k/N} = \frac{N \cos(\pi k)}{\cos(\pi k/N - \pi k/N)} = N(-1)^k.
\end{aligned}$$

The phase factors in [Eq. \(G-10\)](#), when $\omega = 2\pi k/N$, are

(G-14)

$$e^{-j\omega(N-1)/2} e^{-j\pi k/N} \bigg|_{\omega=2\pi k/N} = e^{-j\pi k} e^{j\pi k/N} e^{-j\pi k/N} = (-1)^k.$$

Combining the result of [Eqs. \(G-13\)](#) and [\(G-14\)](#) with [Eq. \(G-10\)](#), we have

$$(G-15)$$

$$|H_{ss,amp}(e^{j\omega})|_{\omega=2\pi k/N} = |H(k)| N(-1)^{2k} = |H(k)| N.$$

So the maximum magnitude response of a single-section complex FSF at resonance is $|H(k)|N$, independent of k .

G.3 Multisection Complex FSF Phase

This appendix shows how the $(-1)^k$ factors arise in [Eq. \(7-48\)](#) for an even- N multisection linear-phase complex FSF. Substituting the positive-frequency, $0 \leq k \leq (N/2)-1$, $|H(k)|e^{j\phi(k)}$ gain factors, with $\phi(k)$ phase values from [Eq. \(7-46\)](#), into [Eq. \(7-45\)](#) gives

$$(G-16)$$

$$H_{cplx,lp,pf}(e^{j\omega}) = e^{-j\omega(N-1)/2} \sin(\omega N/2) \sum_{k=0}^{(N/2)-1} \frac{|H(k)| e^{-jk\pi(N-1)/N} e^{-jk\pi/N}}{\sin(\omega/2 - \pi k/N)},$$

where the subscript “pf” means positive frequency. Focusing only on the numerator inside the summation in [Eq. \(G-16\)](#), it is

$$(G-17)$$

$$\begin{aligned} \text{Numerator}_{pf} &= |H(k)| e^{-jk\pi(N-1)/N} e^{-jk\pi/N} = |H(k)| e^{-jk\pi/N(N-1+1)} \\ &= |H(k)| e^{-jk\pi} = |H(k)| (e^{-j\pi})^k = |H(k)| (-1)^k, \end{aligned}$$

showing how the $(-1)^k$ factors occur within the first summation of [Eq. \(7-48\)](#). Next we substitute the negative-frequency $|H(k)|e^{j\phi(k)}$ gain factors, $(N/2)+1 \leq k \leq N-1$, with $\phi(k)$ phase values from [Eq. \(7-46\)](#), into [Eq. \(7-45\)](#), giving

$$(G-18)$$

$$H_{cplx,lp,nf}(e^{j\omega}) = e^{-j\omega(N-1)/2} \sin(\omega N/2) \sum_{k=(N/2)+1}^{N-1} \frac{|H(k)| e^{j\pi(N-k)(N-1)/N} e^{-jk\pi/N}}{\sin(\omega/2 - \pi k/N)}$$

where the subscript “nf” means negative frequency. Again, looking only at the numerator inside the summation in [Eq. \(G-18\)](#), it is

$$(G-19)$$

$$\begin{aligned} \text{Numerator}_{nf} &= |H(k)| e^{j\pi(N-k)(N-1)/N} e^{-jk\pi/N} = |H(k)| e^{-j\pi[k - (N-k)(N-1)]/N} \\ &= |H(k)| e^{-j\pi[N + kN - N^2]/N} = |H(k)| e^{-j\pi[1 + k - N]} = |H(k)| (e^{-j\pi})(e^{-jk\pi})(e^{j\pi N}). \end{aligned}$$

That $e^{j\pi N}$ factor in [Eq. \(G-19\)](#) is equal to 1 when N is even, so we write

$$(G-20)$$

$$\begin{aligned} \text{Numerator}_{nf} &= |H(k)| (-1)(e^{-jk\pi})(1) = -|H(k)| (e^{-jk\pi}) \\ &= -|H(k)| (e^{-j\pi})^k = -|H(k)| (-1)^k, \end{aligned}$$

establishing both the negative sign before, and the $(-1)^k$ factor within, the second summation of [Eq. \(7-48\)](#). To account for the single-section for the $k = N/2$ term (this is the Nyquist, or $f_s/2$, frequency, where $\omega = \pi$), we plug the $|H(N/2)|e^{j0}$ gain factor, and $k = N/2$, into [Eq. \(7-43\)](#), giving

(G-21)

$$\begin{aligned}
H_{\text{cplx,lp,pf}}(e^{j\omega}) \big|_{\omega=\pi} &= e^{-j\omega(N-1)/2} e^{-j\pi(N/2)/N} |H(N/2)| e^{-j0} \frac{\sin(\omega N / 2)}{\sin(\omega / 2 - \pi(N / 2) / N)} \\
&= e^{-j\omega(N-1)/2} e^{-j\pi/2} |H(N/2)| \frac{\sin(\omega N / 2)}{\sin(\omega / 2 - \pi / 2)} .
\end{aligned}$$

G.4 Multisection Complex FSF Frequency Response

The frequency response of a guaranteed-stable complex N -section FSF, when $r < 1$, is $H_{\text{gs,cplx}}(z)$ with the z variable in [Eq. \(7-53\)](#) replaced by $e^{j\omega}$, giving

(G-22)

$$H_{\text{gs,cplx}}(e^{j\omega}) = H_{\text{gs,cplx}}(z) \big|_{z=e^{j\omega}} = (1 - r^N e^{-jN\omega}) \sum_{k=0}^{N-1} \frac{H(k)}{1 - [r e^{j2\pi k/N}] e^{-j\omega}} .$$

To temporarily simplify our expressions, we let $\theta = \omega - 2\pi k/N$, giving

(G-23)

$$H_{\text{gs,cplx}}(e^{j\omega}) = (1 - r^N e^{-jN\omega}) \sum_{k=0}^{N-1} \frac{H(k)}{1 - r e^{-j\theta}} .$$

Factoring out the half-angled exponentials, and accounting for the r factors, we have

(G-24)

$$\begin{aligned}
H_{\text{gs,cplx}}(e^{j\omega}) &= r^{N/2} e^{-jN\omega/2} (r^{-N/2} e^{jN\omega/2} - r^{N/2} e^{-jN\omega/2}) \\
&\times \sum_{k=0}^{N-1} \frac{H(k)}{r^{1/2} e^{-j\theta/2} (r^{-1/2} e^{j\theta/2} - r^{1/2} e^{-j\theta/2})} .
\end{aligned}$$

Converting all the terms inside parentheses to exponentials (we'll see why in a moment), we have

(G-25)

$$\begin{aligned}
H_{\text{gs,cplx}}(e^{j\omega}) &= r^{N/2} e^{-jN\omega/2} \{ e^{[N\ln(r)/2 - jN\omega/2]} - e^{[N\ln(r)/2 - jN\omega/2]} \} \\
&\times \sum_{k=0}^{N-1} \frac{H(k)}{r^{1/2} e^{-j\theta/2} \{ e^{-[\ln(r)/2 - j\theta/2]} - e^{-[\ln(r)/2 - j\theta/2]} \}} .
\end{aligned}$$

The algebra gets a little messy here because our exponents have both real and imaginary parts. However, hyperbolic functions to the rescue. Recalling when α is a complex number, $\sinh(\alpha) = (e^\alpha - e^{-\alpha})/2$, we have

(G-26)

$$\begin{aligned}
H_{\text{gs,cplx}}(e^{j\omega}) &= r^{N/2} e^{-jN\omega/2} \{-2\sinh[N\ln(r)/2 - jN\omega/2]\} \\
&\times \sum_{k=0}^{N-1} \frac{H(k)}{r^{1/2} e^{-j\theta/2} \{-2\sinh[\ln(r)/2 - j\theta/2]\}} .
\end{aligned}$$

Replacing angle θ with $\omega - 2\pi k/N$, canceling the -2 factors, we have

(G-27)

$$H_{\text{gs,cplx}}(e^{j\omega}) = r^{N/2} e^{-jN\omega/2} \sinh[N \ln(r)/2 - jN\omega/2] \\ \times \sum_{k=0}^{N-1} \frac{H(k)}{r^{1/2} e^{-j(\omega - 2\pi k/N)/2} \sinh[\ln(r)/2 - j(\omega - 2\pi k/N)/2]}.$$

Rearranging and combining terms, we conclude with

(G-28)

$$H_{\text{gs,cplx}}(e^{j\omega}) \\ = \sqrt{r^{N-1}} e^{-j\omega(N-1)/2} \sum_{k=0}^{N-1} \frac{H(k) e^{-j\pi k/N} \sinh[N \ln(r)/2 - jN\omega/2]}{\sinh[\ln(r)/2 - j(\omega - 2\pi k/N)/2]}.$$

(Whew! Now we see why this frequency response expression is not usually found in the literature.)

G.5 Real FSF Transfer Function

The transfer function equation for the real-valued multisection FSF looks a bit strange at first glance, so rather than leaving its derivation as an exercise for the reader, we show the algebraic acrobatics necessary in its development. To preview our approach, we'll start with the transfer function of a multisection complex FSF and define the $H(k)$ gain factors such that all filter poles are in conjugate pairs. This will lead us to real-FSF structures with real-valued coefficients. With that said, we begin with [Eq. \(7-53\)](#)'s transfer function of a guaranteed-stable N -section complex FSF of

(G-29)

$$H_{\text{gs,cplx}}(z) = (1 - r^N z^{-N}) \sum_{k=0}^{N-1} \frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}}.$$

Assuming N is even, and breaking [Eq. \(G-29\)](#)'s summation into parts, we can write

(G-30)

$$H_{\text{gs,cplx}}(z) = \\ (1 - r^N z^{-N}) \left[\frac{H(0)}{1 - rz^{-1}} + \frac{H(N/2)}{1 + rz^{-1}} + \sum_{k=1}^{N/2-1} \frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}} + \sum_{k=N/2+1}^{N-1} \frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}} \right].$$

The first two ratios inside the brackets account for the $k = 0$ and $k = N/2$ frequency samples. The first summation is associated with the positive-frequency range, which is the upper half of the z -plane's unit circle. The second summation is associated with the negative-frequency range, the lower half of the unit circle.

To reduce the clutter of our derivation, let's identify the two summations as

(G-31)

$$\text{Summations} = \sum_{k=1}^{N/2-1} \frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}} + \sum_{k=N/2+1}^{N-1} \frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}}.$$

We then combine the summations by changing the indexing of the second summation as

(G-32)

$$\text{Summations} = \sum_{k=1}^{N/2-1} \left[\frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}} + \frac{H(N-k)}{1 - [re^{j2\pi[N-k]/N}]z^{-1}} \right].$$

Putting those ratios over a common denominator and multiplying the denominator factors, and then forcing the $H(N-k)$ gain factors to be complex conjugates of the $H(k)$ gain factors, we write

(G-33)

$$\begin{aligned} & \text{Summations} \\ &= \sum_{k=1}^{N/2-1} \frac{H(k)(1 - re^{j2\pi[N-k]/N}z^{-1}) + H^*(k)(1 - re^{j2\pi k/N}z^{-1})}{1 - re^{j2\pi[N-k]/N}z^{-1} - re^{j2\pi k/N}z^{-1} + r^2e^{j2\pi[k+N-k]/N}z^{-2}}, \end{aligned}$$

where the “*” symbol means conjugation. Defining $H(N-k) = H^*(k)$ mandates that all poles will be conjugate pairs and, as we’ll see, this condition converts our complex FSF into a real FSF with real-valued coefficients. Plowing forward, because $e^{j2\pi[N-k]/N} = e^{-j2\pi N/N}e^{-j2\pi k/N} = e^{-j2\pi k/N}$, we make that substitution in [Eq. \(G-33\)](#), rearrange the numerator, and combine the factors of z^{-1} in the denominator to arrive at

(G-34)

$$\begin{aligned} & \text{Summations} \\ &= \sum_{k=1}^{N/2-1} \frac{H(k) + H^*(k) - [H(k)re^{-j2\pi k/N} + H^*(k)re^{j2\pi k/N}]z^{-1}}{1 - r[e^{-j2\pi k/N} + e^{j2\pi k/N}]z^{-1} + r^2z^{-2}}. \end{aligned}$$

Next we define each complex $H(k)$ in rectangular form with an angle ϕ_k , or $H(k) = |H(k)|[\cos(\phi_k) + j\sin(\phi_k)]$, and $H^*(k) = |H(k)|[\cos(\phi_k) - j\sin(\phi_k)]$. Realizing that the imaginary parts of the sum cancel so that $H(k) + H^*(k) = 2|H(k)|\cos(\phi_k)$ allows us to write

(G-35)

$$\begin{aligned} & \text{Summations} \\ &= \sum_{k=1}^{N/2-1} \frac{2|H(k)|\cos(\phi_k) - |H(k)|[re^{j[\phi_k - 2\pi k/N]} + re^{-j[\phi_k - 2\pi k/N]}]z^{-1}}{1 - r[e^{-j2\pi k/N} + e^{j2\pi k/N}]z^{-1} + r^2z^{-2}}. \end{aligned}$$

Recalling Euler’s identity, $2\cos(\alpha) = e^{j\alpha} + e^{-j\alpha}$, and combining the $|H(k)|$ factors leads to the final form of our summation:

(G-36)

$$\begin{aligned} & \text{Summations} \\ &= \sum_{k=1}^{N/2-1} \frac{2|H(k)|[\cos(\phi_k) - r\cos(\phi_k - 2\pi k/N)z^{-1}]}{1 - [2r\cos(2\pi k/N)]z^{-1} + r^2z^{-2}}. \end{aligned}$$

Substituting [Eq. \(G-36\)](#) for the two summations in [Eq. \(G-30\)](#), we conclude with the desired transfer function

(G-37)

$$\begin{aligned} H_{\text{gs,real}}(z) &= (1 - r^N z^{-N}) \left[\frac{H(0)}{1 - rz^{-1}} + \frac{H(N/2)}{1 + rz^{-1}} \right. \\ &\quad \left. + \sum_{k=1}^{N/2-1} \frac{2|H(k)|[\cos(\phi_k) - r\cos(\phi_k - 2\pi k/N)z^{-1}]}{1 - [2r\cos(2\pi k/N)]z^{-1} + r^2z^{-2}} \right], \end{aligned}$$

where the subscript “real” means a real-valued multisection FSF.

G.6 Type-IV FSF Frequency Response

The frequency response of a single-section even- N Type-IV FSF is its transfer function evaluated on the unit circle. To begin that evaluation, we set [Eq. \(7-58\)](#)’s $|H(k)| = 1$, and denote a Type-IV FSF’s single-section transfer function as

(G-38)

$$H_{\text{Type-IV,ss}}(z) = (1 - r^N z^{-N}) \frac{1 - r^2 z^{-2}}{1 - 2r \cos(2\pi k / N) z^{-1} + r^2 z^{-2}}$$

where the “ss” subscript means single-section. Under the assumption that the damping factor r is so close to unity that it can be replaced with 1, we have the simplified FSF transfer function

(G-39)

$$H_{\text{Type-IV,ss}}(z) = (1 - z^{-N}) \frac{1 - z^{-2}}{1 - 2 \cos(2\pi k / N) z^{-1} + z^{-2}}.$$

Letting $\omega_r = 2\pi k / N$ to simplify the notation and factoring $H_{\text{Type-IV,ss}}(z)$ ’s denominator gives

(G-40)

$$\begin{aligned} H_{\text{Type-IV,ss}}(z) \\ = (1 - z^{-N}) \frac{1 - z^{-2}}{1 - 2 \cos(\omega_r) z^{-1} + z^{-2}} &= \frac{(1 - z^{-N})(1 - z^{-2})}{(1 - e^{j\omega_r} z^{-1})(1 - e^{-j\omega_r} z^{-1})} \end{aligned}$$

in which we replace each z term with $e^{j\omega}$, as

(G-41)

$$\begin{aligned} H_{\text{Type-IV,ss}}(e^{j\omega}) \\ = \frac{(1 - e^{-j\omega N})(1 - e^{-j2\omega})}{(1 - e^{j\omega_r} e^{-j\omega})(1 - e^{-j\omega_r} e^{-j\omega})} &= \frac{(1 - e^{-j\omega N})(1 - e^{-j2\omega})}{(1 - e^{-j(\omega - \omega_r)})(1 - e^{-j(\omega + \omega_r)})}. \end{aligned}$$

Factoring out the half-angled exponentials, we have

(G-42)

$$\begin{aligned} H_{\text{Type-IV,ss}}(e^{j\omega}) \\ = \frac{e^{-j\omega N / 2} (e^{j\omega N / 2} - e^{-j\omega N / 2}) e^{-j\omega} (e^{j\omega} - e^{-j\omega})}{e^{-j(\omega - \omega_r) / 2} (e^{j(\omega - \omega_r) / 2} - e^{-j(\omega - \omega_r) / 2}) e^{-j(\omega + \omega_r) / 2} (e^{j(\omega + \omega_r) / 2} - e^{-j(\omega + \omega_r) / 2})}. \end{aligned}$$

Using Euler’s identity, $2j \sin(\alpha) = e^{j\alpha} - e^{-j\alpha}$, we obtain

(G-43)

$$\begin{aligned} H_{\text{Type-IV,ss}}(e^{j\omega}) \\ = \frac{e^{-j\omega N / 2} [2j \sin(\omega N / 2)] e^{-j\omega} [2j \sin(\omega)]}{e^{-j(\omega - \omega_r) / 2} [2j \sin([\omega - \omega_r] / 2)] e^{-j(\omega + \omega_r) / 2} [2j \sin([\omega + \omega_r] / 2)]}. \end{aligned}$$

Canceling common factors, and adding like terms, we have

(G-44)

$$H_{\text{Type-IV,ss}}(e^{j\omega}) = \frac{e^{-j\omega N/2} e^{-j\omega}}{e^{-j(\omega-\omega_r)/2} e^{-j(\omega+\omega_r)/2}} \frac{\sin(\omega N/2) \sin(\omega)}{\sin([\omega-\omega_r]/2) \sin([\omega+\omega_r]/2)}$$

$$= e^{-j\omega N/2} \frac{\cos(\omega N/2 - \omega) - \cos(\omega N/2 + \omega)}{\cos(\omega_r) - \cos(\omega)}.$$

Plugging $2\pi k/N$ back in for ω_r , the single-section frequency response is

$$(G-45)$$

$$H_{\text{Type-IV,ss}}(e^{j\omega}) = e^{-j\omega N/2} \frac{\cos(\omega N/2 - \omega) - \cos(\omega N/2 + \omega)}{\cos(2\pi k/N) - \cos(\omega)}.$$

Based on [Eq. \(G-45\)](#), the frequency response of a multisection even- N Type-IV FSF is

$$(G-46)$$

$$H_{\text{Type-IV}}(e^{j\omega})$$

$$= e^{-j\omega N/2} \sum_{k=0}^{N/2} \frac{(-1)^k |H(k)| [\cos(\omega N/2 - \omega) - \cos(\omega N/2 + \omega)]}{\cos(2\pi k/N) - \cos(\omega)}.$$

To determine the amplitude response of a single section, we ignore the phase shift terms (complex exponentials) in [Eq. \(G-45\)](#) to yield

$$(G-47)$$

$$H_{\text{Type-IV,amp}}(e^{j\omega}) = \frac{\cos(\omega N/2 - \omega) - \cos(\omega N/2 + \omega)}{\cos(2\pi k/N) - \cos(\omega)}.$$

To find the maximum amplitude response at resonance we evaluate [Eq. \(G-47\)](#) when $\omega = 2\pi k/N$, because that's the value of ω at the FSF's pole locations. However, that ω causes the denominator to go to zero, causing the ratio to go to infinity. We move on with one application of L'Hopital's Rule to [Eq. \(G-47\)](#) to obtain

$$(G-48)$$

$$H_{\text{Type-IV,amp}}(e^{j\omega}) \big|_{\omega \rightarrow 2\pi k/N} = \frac{d[\cos(\omega N/2 - \omega) - \cos(\omega N/2 + \omega)] / d\omega}{d[\cos(2\pi k/N) - \cos(\omega)] / d\omega} \bigg|_{\omega \rightarrow 2\pi k/N}$$

$$= \frac{-\sin(\omega N/2 - \omega)(N/2 - 1) + \sin(\omega N/2 + \omega)(N/2 + 1)}{\sin(\omega)} \bigg|_{\omega \rightarrow 2\pi k/N}$$

$$= \frac{-[(N-2)/2] \sin(\pi k - 2\pi k/N) + [(N+2)/2] \sin(\pi k + 2\pi k/N)}{\sin(2\pi k/N)}.$$

Eliminating the πk terms by using trigonometric reduction formulas $\sin(\pi k - \alpha) = (-1)^k [-\sin(\alpha)]$ and $\sin(\pi k + \alpha) = (-1)^k [\sin(\alpha)]$, we have a maximum amplitude response of

$$(G-49)$$

$$H_{\text{Type-IV,amp}}(e^{j\omega}) \big|_{\omega = 2\pi k/N}$$

$$= \frac{[(N-2)/2](-1)^k \sin(2\pi k/N) + [(N+2)/2](-1)^k \sin(2\pi k/N)}{2 \sin(2\pi k/N)}$$

$$= \frac{N(-1)^k \sin(2\pi k/N)}{\sin(2\pi k/N)} = N(-1)^k, \quad k = 1, 2, \dots, \frac{N}{2} - 1.$$

[Equation \(G-49\)](#) is only valid for $1 \leq k \leq (N/2)-1$. Disregarding the $(-1)^k$ factors, we have a

magnitude response at resonance, as a function of k , of

(G-50)

$$|H_{\text{Type-IV}}(e^{j\omega})|_{\omega=2\pi k/N} = N, \quad k = 1, 2, \dots, \frac{N}{2} - 1.$$

To find the resonant gain at 0 Hz (DC) we set $k = 0$ in [Eq. \(G-47\)](#), apply L'Hopital's Rule (the derivative with respect to ω) twice, and set $\omega = 0$, giving

(G-51)

$$|H_{\text{Type-IV}}(e^{j\omega})|_{\omega=0} = 2N.$$

To obtain the resonant gain at $f_s/2$ Hz we set $k = N/2$ in [Eq. \(G-47\)](#), again apply L'Hopital's Rule twice, and set $\omega = \pi$, yielding

(G-52)

$$|H_{\text{Type-IV}}(e^{j\omega})|_{\omega=\pi} = 2N.$$

Appendix H. Frequency Sampling Filter Design Tables

In [Section 7.5](#) we described the so-called Type-IV frequency sampling filter (FSF). The tables in this appendix provide a list of optimum transition coefficient values for the Case I (see [Figure 7-44](#)) Type-IV lowpass FSFs of various passband bandwidths, over a range of values of N . [Table H-1](#) provides the $H(k)$ single transition coefficient and two transition coefficients for even values of N . [Table H-2](#) provides the $H(k)$ three transition coefficients for even N . [Table H-3](#) provides the $H(k)$ single transition coefficient and two transition coefficients for odd values of N , while [Table H-4](#) provides the $H(k)$ three transition coefficients for odd N .

Table H-1 Lowpass Type-IV FSF for Even N (One and Two Coefficients)

BW	Atten	T1	BW	Atten	T1	T2
$N = 16$			$N = 16$			
1	-44.9	0.41924081	1	-76.5	0.56626687	0.07922718
2	-45.8	0.38969818	2	-77.2	0.55487263	0.08012238
3	-47.3	0.36942214	3	-81.2	0.53095099	0.07087993
4	-49.6	0.34918551	4	-87.7	0.49927622	0.05813368
$N = 24$			$N = 24$			
1	-44	0.42452816	1	-73.6	0.57734042	0.08641861
2	-44.1	0.40042889	2	-72.5	0.57708274	0.09305238
3	-44.9	0.38622106	3	-72.9	0.56983709	0.09177956
4	-45.7	0.37556064	4	-73.8	0.55958351	0.08770698
5	-46.5	0.36663149	5	-75.6	0.54689579	0.08202772
$N = 32$			$N = 32$			
1	-43.6	0.42638815	1	-72.6	0.58109341	0.08892320
2	-43.6	0.40407598	2	-71	0.58466392	0.09771906
3	-44	0.39197681	3	-70.8	0.58101218	0.09823378
4	-44.5	0.38377786	5	-71.6	0.57002693	0.09442029
6	-45.5	0.37172559	7	-73.2	0.55593774	0.0879846

7	-45.8	0.38912443	9	-76.2	0.53661082	0.07884406
8	-46.6	0.36087271	13	-106	0.43242657	0.03643965
N = 48			N = 48			
1	-43.4	0.42772741	1	-72	0.58385966	0.09079945
2	-43.2	0.40654471	2	-70	0.58999731	0.10107095
3	-43.5	0.39569517	3	-69.5	0.58898579	0.1030238
4	-43.8	0.38879556	5	-69.6	0.58356869	0.10204926
6	-44.4	0.37994174	7	-70	0.57749269	0.09959325
8	-44.9	0.37394938	9	-70.6	0.57143299	0.09683486
10	-45.3	0.3690437				
N = 64			N = 64			
1	-43.3	0.42815077	1	-71.8	0.58480329	0.09144087
2	-43.1	0.40742967	2	-69.7	0.59178518	0.10221701
3	-43.4	0.39690507	3	-69.2	0.59168291	0.10467406
4	-43.6	0.39043339	4	-68.9	0.5899207	0.10496398
5	-43.9	0.38583162	5	-68.9	0.58788109	0.10457886
6	-44.1	0.38244173	9	-69.4	0.58010661	0.10157302
10	-44.7	0.37382147	13	-70.2	0.57272483	0.09810828
14	-45.3	0.36813961	17	-71.1	0.56417336	0.09386963
N = 96			N = 96			
1	-43.2	0.42856954	1	-71.6	0.585424	0.09185794
2	-43	0.40790815	2	-69.5	0.59305878	0.10302346
3	-43.3	0.3977603	3	-68.8	0.59332978	0.10571202
4	-43.5	0.39154291	4	-68.4	0.59202942	0.10623854
5	-43.7	0.38719365	5	-68.3	0.59062246	0.10623287
6	-43.9	0.38409245	9	-68.5	0.58514671	0.10445521
10	-44.4	0.37686993	13	-68.9	0.58087019	0.10253761
14	-44.7	0.37270333	17	-69.3	0.57751103	0.10097157
18	-45	0.36984146	21	-65.3	0.59419612	0.11180709
N = 128			N = 128			
1	-43.2	0.42864273	1	-71.6	0.58574352	0.09208128
2	-43	0.40823844	2	-69.4	0.59357535	0.10335703
3	-43.2	0.39811024	3	-68.6	0.59383385	0.10601544
4	-43.4	0.39188378	4	-68.3	0.59279342	0.10671242
5	-43.7	0.38774353	6	-68.1	0.59024028	0.10655014
7	-44	0.38236389	9	-68.2	0.5868017	0.10542034
10	-44.3	0.37771128	17	-68.7	0.58071332	0.10271656
18	-44.8	0.3717883	25	-69.2	0.57652818	0.1006787
26	-45.1	0.36862161	33	-69.8	0.57265344	0.09871098
N = 192			N = 192			
1	-43.2	0.42881027	1	-71.5	0.58589507	0.09218797
2	-43	0.40831822	2	-69.4	0.59383365	0.1035231
3	-43.2	0.39830476	3	-68.6	0.59433749	0.10635017
4	-43.4	0.39219024	4	-68.2	0.5933049	0.10702144
5	-43.6	0.38797095	6	-67.9	0.59098287	0.10700026

7	-43.9	0.3828125	9	-68	0.58790082	0.10604544
10	-44.2	0.37845008	17	-68.4	0.58295021	0.10399738
18	-44.7	0.37302517	25	-68.7	0.57978921	0.10242432
26	-44.9	0.37049755	33	-69	0.57773363	0.10143253
34	-45	0.36908526	41	-69.2	0.57597277	0.10055134
42	-45.2	0.36764286	49	-69.5	0.57407637	0.09954949
$N = 224$			$N = 224$			
1	-43.2	0.42874481	1	-71.5	0.58591935	0.09220841
2	-43	0.40836093	2	-69.4	0.593894	0.10355622
3	-43.2	0.39831237	3	-68.5	0.59435536	0.10634325
4	-43.4	0.3921651	4	-68.1	0.59354863	0.10719315
5	-43.6	0.38807204	6	-67.9	0.59106856	0.1070268
7	-43.9	0.38281226	9	-67.9	0.58816185	0.10619806
10	-44.2	0.37847605	10	-67.9	0.58726527	0.10582934
11	-44.3	0.37742038	17	-68.3	0.58317185	0.10407347
18	-44.6	0.37324982	33	-68.8	0.57860121	0.10189345
34	-45	0.36946431	49	-69.2	0.5757377	0.10043439
50	-45.2	0.36753866	57	-69.4	0.57440527	0.09975201
$N = 256$			$N = 256$			
1	-43.2	0.42874481	1	-71.5	0.58599793	0.09225917
2	-43	0.40844072	2	-69.4	0.59395199	0.10360852
3	-43.2	0.39839019	3	-68.5	0.59445009	0.10641045
4	-43.4	0.39231838	4	-68.1	0.59358715	0.10721801
5	-43.6	0.38814788	6	-67.9	0.59121865	0.10712227
7	-43.9	0.38296192	9	-67.9	0.58824601	0.10622746
10	-44.2	0.37855003	10	-67.9	0.58727091	0.10580213
11	-44.3	0.37756795	17	-68.2	0.58347729	0.10424412
18	-44.6	0.3733958	33	-68.7	0.57913354	0.1021842
34	-44.9	0.36982575	49	-69	0.57667852	0.10093887
50	-45.1	0.36804233	57	-69.2	0.57568486	0.10043603
58	-45.2	0.3673716	65	-69.3	0.57469205	0.09993526

Table H-2 Lowpass Type-IV FSF for Odd N (One and Two Coefficients)

BW	Atten	T_1	BW	Atten	T_1	T_2
$N = 15$			$N = 15$			
1	-45.1	0.41802444	1	-77.3	0.56378193	0.07767395
2	-46.2	0.38716426	2	-78.9	0.54831544	0.07646082
3	-48	0.36461603	3	-83.4	0.52139051	0.06627593
4	-51.4	0.34005655	4	-96.3	0.47732772	0.0487037

N = 23			N = 23			
1	-44	0.42412451	1	-73.8	0.57648809	0.08585489
2	-44.3	0.3995718	2	-72.7	0.57569102	0.09221862
3	-45.1	0.38497937	3	-73.2	0.56732401	0.09032595
4	-46	0.37367551	4	-74.6	0.55575797	0.08563268
5	-46.8	0.36445788	5	-76.5	0.54245814	0.07981527
N = 33			N = 33			
1	-43.6	0.42659097	1	-72.6	0.58141103	0.08913503
2	-43.5	0.40433257	2	-70.9	0.58529197	0.09811292
3	-44	0.39239983	3	-70.7	0.58187597	0.09874619
4	-44.4	0.3843389	5	-71.3	0.57154421	0.09525674
6	-45.3	0.37272147	7	-72.9	0.55826179	0.08918146
8	-46.4	0.36256798	9	-75.3	0.54128598	0.08116809
N = 47			N = 47			
1	-43.4	0.42768264	1	-72	0.58376507	0.09073592
2	-43.2	0.40649692	2	-70.1	0.58975381	0.1009182
3	-43.5	0.39553589	3	-69.6	0.5886934	0.10284475
4	-43.8	0.38859729	5	-69.6	0.58311582	0.10178496
6	-44.4	0.37968179	7	-70.1	0.57687412	0.09925266
8	-44.9	0.37362421	9	-70.7	0.57045477	0.0963056
10	-45.4	0.36853968	11	-71.6	0.56319305	0.09280396
N = 65			N = 65			
1	-43.3	0.4281872	1	-71.7	0.58480896	0.09144011
	-43.1	0.40743541	2	-69.7	0.59188395	0.10227409
3	-43.4	0.39694541	3	-69.1	0.59158717	0.10459977
4	-43.6	0.39043991	4	-68.9	0.59001405	0.10501496
5	-43.9	0.3859325	5	-68.8	0.58797401	0.10462176
6	-44.1	0.38249194	9	-69.4	0.58031079	0.10167681
10	-44.7	0.37399454	13	-70.1	0.57312044	0.09830111
14	-45.2	0.36841874	17	-71	0.56506463	0.0943328
18	-45.8	0.36324429	21	-72.5	0.55369626	0.08867447
N = 95			N = 95			
1	-43.2	0.42852251	1	-71.6	0.58559589	0.09199196
2	-43	0.40799464	2	-69.5	0.59306419	0.10302347
3	-43.3	0.39772511	3	-68.8	0.59323668	0.10563966
4	-43.5	0.39143867	4	-68.5	0.59207559	0.10628422
5	-43.7	0.38722579	5	-68.3	0.5905046	0.10614086
6	-43.9	0.38400287	9	-68.5	0.58506537	0.10440925
10	-44.4	0.3766755	13	-68.9	0.58094477	0.10262083
14	-44.8	0.37268027	17	-69.3	0.57725045	0.10081244
18	-45	0.369842	21	-69.8	0.57370707	0.09903691
N = 125			N = 125			

1	-43.2	0.42857276	1	-71.5	0.5856764	0.0920373
2	-43	0.40819419	2	-69.5	0.59346193	0.10328234
3	-43.2	0.39806479	3	-68.7	0.59389103	0.1060686
4	-43.5	0.39191493	5	-68.1	0.59144981	0.1067222
6	-43.8	0.38458541	7	-68.1	0.58887274	0.10611287
8	-44.1	0.38046599	9	-68.2	0.58670301	0.10535739
10	-44.3	0.37761366	17	-68.7	0.5805297	0.10262003
18	-44.8	0.37166577	25	-69.3	0.57610034	0.10041636
26	-45.1	0.36836415	33	-69.9	0.57206269	0.0984002
N = 191			N = 191			
1	-43.2	0.42865655	1	-71.5	0.5859143	0.09220684
2	-43	0.40839373	2	-69.4	0.5937664	0.10346999
3	-43.2	0.39822053	3	-68.5	0.59423193	0.1062604
4	-43.4	0.39214502	5	-68	0.59213475	0.1071269
6	-43.8	0.38503751	7	-67.9	0.589875	0.10671096
8	-44	0.38095089	9	-68	0.58788996	0.10603382
10	-44.2	0.37828083	17	-68.4	0.58282022	0.10390276
18	-44.7	0.37305805	25	-68.7	0.57971044	0.10236319
26	-44.9	0.37048161	33	-69	0.57760031	0.10133385
34	-45	0.36904678	41	-69.2	0.57578133	0.10042123
42	-45.1	0.3676021	49	-69.5	0.57404729	0.09954845
N = 223			N = 223			
1	-43.2	0.42874481	1	-71.5	0.58589267	0.0921919
2	-43	0.40836093	2	-69.4	0.59379277	0.10347913
3	-43.2	0.39835128	3	-68.5	0.59435536	0.10634325
4	-43.4	0.3921651	4	-68.1	0.59354863	0.10719315
5	-43.6	0.38807204	6	-67.9	0.59114264	0.10708575
7	-43.9	0.38288709	9	-67.9	0.58797899	0.10606475
10	-44.2	0.37851304	10	-67.9	0.58726527	0.10582934
11	-44.3	0.37742038	17	-68.3	0.58329541	0.10416575
18	-44.6	0.37324982	33	-68.8	0.57849661	0.10182631
34	-45	0.36946431	49	-69.2	0.57568307	0.10040604
50	-45.2	0.3674667	57	-69.4	0.5744086	0.09975962
N = 255			N = 255			
1	-43.2	0.42874481	1	-71.5	0.58590294	0.09218854
2	-43	0.40836093	2	-69.3	0.59392035	0.10356223
3	-43.2	0.39831237	3	-68.5	0.59440493	0.10637653
4	-43.4	0.39224174	4	-68.1	0.59363182	0.10725379
5	-43.6	0.38814788	6	-67.9	0.59121586	0.10712951
7	-43.9	0.38296192	9	-67.9	0.58822523	0.10621751
10	-44.2	0.37855003	10	-67.9	0.58747402	0.10596604
11	-44.3	0.37756795	17	-68.2	0.58351595	0.10427857
18	-44.6	0.3733958	33	-68.7	0.57906429	0.10213183
34	-44.9	0.36982575	49	-69	0.57660259	0.10088411
50	-45.1	0.36804233	57	-69.2	0.57562728	0.10040259
58	-45.2	0.3673716	65	-69.4	0.57461162	0.09987875

Table H-3 Lowpass Type-IV FSF for Even N (Three Coefficients)

BW	Atten	T_1	T_2	T_3
$N = 16$				
1	-112.6	0.64272445	0.15442399	0.00880089
2	-95.9	0.70487291	0.22419597	0.01947599
3	-100.7	0.70063089	0.21872748	0.01757096
4	-115.9	0.68531929	0.19831357	0.01270197
$N = 24$				
1	-103.8	0.6599002	0.17315143	0.01189889
2	-101.8	0.67718249	0.19461557	0.01429673
3	-95.5	0.69609682	0.21773826	0.01860944
4	-104.1	0.66830223	0.18959572	0.01339907
$N = 32$				
1	-99.6	0.6642114	0.17798254	0.01278046
2	-98.5	0.68804961	0.20639825	0.01646338
4	-87.4	0.73378289	0.26142233	0.02762054
6	-100.5	0.67913658	0.20169658	0.01554611
8	-105.3	0.65936975	0.18380663	0.01270743
$N = 48$				
1	-93.8	0.68361144	0.2010237	0.01735969
2	-96	0.69534463	0.21480253	0.01812435
4	-87.2	0.73314865	0.26098449	0.02762804
6	-86.4	0.73802064	0.26732823	0.02900775
8	-95	0.69703503	0.2211425	0.01909109
10	-90	0.71746809	0.24474881	0.02420421

N = 64				
1	-96.6	0.67620503	0.19208214	0.01551621
2	-94.9	0.69693984	0.21653685	0.01842226
3	-89.7	0.72079468	0.24569738	0.02432222
4	-92.3	0.7068141	0.22927121	0.02042893
8	-91.4	0.70957119	0.23498487	0.02215407
12	-93.8	0.7026052	0.22772953	0.02059288
16	-85.3	0.74439511	0.27543213	0.03085705

N = 96				
1	-98.5	0.6720933	0.18712559	0.01449609
2	-92.9	0.70471821	0.22591053	0.02048075
3	-93	0.70905096	0.23165702	0.02121954
4	-88.7	0.72625477	0.25269331	0.02574193
8	-90.8	0.71369108	0.23929089	0.02281527
12	-90.8	0.71110318	0.23715671	0.02248568
16	-85.2	0.74356072	0.27478153	0.03080406
20	-85.8	0.74022029	0.27104418	0.02999046

N = 128				
1	-98.3	0.67221636	0.18725564	0.01451885
2	-94.4	0.70015724	0.22042278	0.01929075
3	-92.6	0.70981704	0.23257905	0.02143209
5	-90.6	0.71933148	0.24480839	0.02391897
8	-89.8	0.72190475	0.24869701	0.02481883
16	-88.5	0.72569265	0.25405918	0.02615712
24	-87.4	0.7301942	0.25964746	0.02748522

N = 192				
1	-98.1	0.67216994	0.1871603	0.01447431
2	-94.3	0.70064573	0.22097713	0.01939796
3	-92.6	0.71046628	0.23329177	0.02156244
5	-90.6	0.71933299	0.24477507	0.0238993
8	-89.8	0.72185688	0.24857861	0.02477626
16	-88.5	0.72617255	0.2545026	0.02622728
24	-87.7	0.72957884	0.25880678	0.02726692
32	-90.9	0.71321929	0.24041037	0.02328586
40	-91.2	0.71133926	0.23853571	0.02293979
48	-91.4	0.70862489	0.2357226	0.0224067

N = 224				
1	-98.2	0.67256687	0.18767169	0.01459779
2	-94.2	0.70077254	0.22112728	0.01942992
3	-90.4	0.7026477	0.22304697	0.01885735
5	-91.1	0.71677647	0.24176238	0.0232377
8	-89.9	0.72168089	0.24837531	0.02473386
9	-89.8	0.71675825	0.24253218	0.02331464
16	-90.3	0.71805244	0.24514888	0.0241623
32	-90.6	0.71429115	0.24150812	0.0234885
48	-91.1	0.71133746	0.23857357	0.02295657

N = 256				
1	-95.7	0.67780153	0.19398356	0.01590119
2	-94	0.70138048	0.22187281	0.01959708
3	-90.3	0.70235664	0.22265441	0.01875372
5	-91	0.71654134	0.24139758	0.02311255
8	-89.9	0.72167623	0.24835995	0.02472548
9	-89.7	0.71676546	0.24249377	0.02328724
16	-90.2	0.71841628	0.24555225	0.02424786
32	-90.5	0.71523646	0.24257287	0.02372755
48	-90.8	0.71282545	0.2402303	0.02331467
56	-90.7	0.71353605	0.24104134	0.02347778

Table H-4 Lowpass Type-IV FSF for Odd N (Three Coefficients)

BW	Atten	T ₁	T ₂	T ₃
N = 15				
1	-99.1	0.67276446	0.18765467	0.01448603
2	-109	0.65109591	0.16879848	0.01032845
3	-103	0.64743501	0.16597437	0.00887322
4	-129.8	0.58430569	0.11830427	0.0041067
N = 23				
1	-98.6	0.67021235	0.18516808	0.01420518
2	-96.9	0.6596023	0.17408826	0.00996838
3	-95.3	0.64635467	0.16260027	0.0077623
4	-86.1	0.60390729	0.12509768	0.00296913
N = 33				
1	-98.4	0.67150869	0.18654613	0.01442309
2	-97.9	0.68975409	0.20844142	0.01686088
4	-93.8	0.70392025	0.22717012	0.02035858
6	-92.1	0.70836197	0.23374423	0.02185812
8	-92.9	0.70271751	0.22868478	0.02098636
N = 47				
1	-99.7	0.66933083	0.18386234	0.01384901
2	-94.7	0.69037782	0.20845236	0.01654889
4	-94.4	0.70435781	0.22714301	0.02019897
6	-94.5	0.70200706	0.22582668	0.01999782
8	-95.4	0.69662478	0.22082819	0.01907788
10	-97	0.69029654	0.21493063	0.01807728
12	-88.8	0.64107819	0.16254219	0.0076571

N = 65				
1	-98.8	0.67071168	0.18547676	0.01416929
2	-94.8	0.69743725	0.21725	0.01864255
3	-93.6	0.70659336	0.22882367	0.02062804
4	-93	0.70962871	0.23307976	0.02141978
8	-92.6	0.70884359	0.23403175	0.02173011
12	-88.7	0.72402053	0.25220517	0.02579854
16	-89.9	0.71679306	0.24461807	0.02427071

N = 95				
1	-98.6	0.67204252	0.18706788	0.0144861
2	-94.5	0.69934889	0.21945399	0.01908355
3	-93	0.70879388	0.23134381	0.02114855
4	-92.2	0.71277193	0.23666779	0.0221755
8	-89.2	0.72434568	0.25160197	0.02547937
12	-88.8	0.72479877	0.25277864	0.02583655
16	-90.1	0.71754976	0.24492389	0.02419794
20	-92.6	0.70606236	0.23250776	0.02165584

N = 125				
1	-98.3	0.67220923	0.18724753	0.01451708
2	-94.4	0.7001615	0.22041952	0.01928486
4	-90.8	0.71816438	0.24299337	0.02355056
6	-90.5	0.71950797	0.24538851	0.02405304
8	-90.5	0.71546288	0.2410237	0.02303409
16	-89.5	0.72139717	0.24910324	0.02505485
24	-89.9	0.71817491	0.24597097	0.02448715
32	-88.9	0.72170356	0.25030101	0.02550214

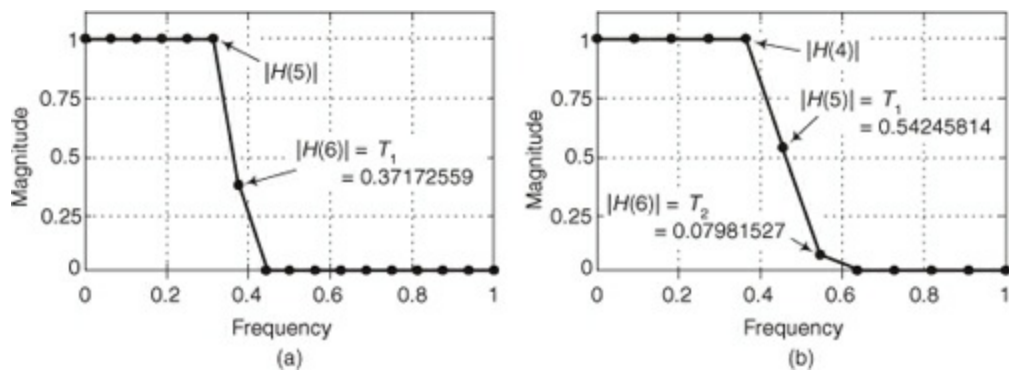
N = 191				
1	-98.1	0.67223344	0.1872747	0.01452257
2	-94.3	0.70075621	0.22112031	0.01943466
4	-90.5	0.7196321	0.24471727	0.02393159
6	-90.9	0.71816017	0.24379217	0.02370239
8	-90.4	0.71833134	0.24451461	0.02390559
16	-90.4	0.71734228	0.244362	0.02399541
24	-90.1	0.71838476	0.24599127	0.02444213
32	-89	0.72239379	0.25075541	0.02551853
40	-89.7	0.7188511	0.24690437	0.0247143
48	-91.6	0.70888027	0.23598149	0.02243978

N = 223				
1	-98.2	0.67257273	0.18768014	0.01460038
2	-94.2	0.70072996	0.22108243	0.01942305
3	-90.4	0.70262204	0.22302806	0.0188601
5	-91	0.71641456	0.24126979	0.02309323
8	-89.9	0.72166528	0.24835901	0.02473006
9	-89.8	0.71674758	0.24252389	0.02331481
16	-90.3	0.7179882	0.24507651	0.02414698
32	-90.6	0.71403002	0.24119297	0.02341017
48	-91.2	0.71143573	0.2386827	0.02297579
56	-90.7	0.71338506	0.24097504	0.02348867

$N = 255$				
1	-97.6	0.67176623	0.18670931	0.01441482
2	-94.2	0.70070534	0.22105163	0.01941457
3	-90.3	0.70233551	0.22263226	0.01874791
5	-91	0.71654995	0.24140965	0.02311617
8	-89.9	0.72164196	0.24831241	0.02471143
9	-89.7	0.71673449	0.2424607	0.02328277
16	-90.1	0.71885073	0.24604614	0.02435372
32	-89.6	0.71549778	0.24342724	0.02411907
48	-90.9	0.71268842	0.23999548	0.02323405
56	-90.7	0.71349454	0.240994	0.02346651
64	-91.3	0.71035623	0.23761455	0.02277658

The passband bandwidth in these tables, signified by the BW parameter, is the number of FSF sections having unity-valued $H(k)$ gain factors. For example, an $N = 32$ lowpass FSF using six passband sections and a single transition region coefficient (T_1) would have the $H(k)$ gain values shown in [Figure H-1\(a\)](#). In this case, the T_1 coefficient would be found in [Table H-1](#) for $N = 32$ at a bandwidth $BW = 6$. An $N = 23$ lowpass FSF with five passband sections and two transition region coefficients (T_1 and T_2) would have the $H(k)$ gain values shown in [Figure H-1\(b\)](#). In this case, the T_1 and T_2 coefficients are found in [Table H-2](#) for $N = 23$ at a bandwidth $BW = 5$. An additional parameter in the tables is the maximum stopband sidelobe attenuation levels (Atten).

Figure H-1 Transition coefficient examples: (a) one coefficient for $N = 32$ and $BW = 6$; (b) two coefficients for $N = 23$ and $BW = 5$.



Appendix I. Computing Chebyshev Window Sequences

Because detailed methods for computing Chebyshev window functions are not readily available in the literature of DSP, here we provide the steps for computing these useful window sequences.

Below we provide methods for computing two types of Chebyshev window sequences. The first window type yields symmetric window sequences, in which their first and last samples are equal. That type of window is used in the Window Design Method of tapped-delay line FIR filter design.

The second Chebyshev window computation method produces nonsymmetric window sequences, in which their first and last samples are not equal. That type of window is used for spectral leakage reduction in spectrum analysis applications. (This nonsymmetric type of window has a Fourier transform that is real-only.) I thank DSP guru Prof. Fredric J. Harris, San Diego State University, for his personal guidance enabling the creation of the following procedures.

I.1 Chebyshev Windows for FIR Filter Design

Symmetric Chebyshev window sequences, used in the Window Design Method of tapped-delay line FIR filters, are computed as follows:

1. Given a desired Chebyshev window sequence length of N , where N is an odd integer, define integer $M = N-1$.
2. Define the window's sidelobe-level control parameter as γ . The window's sidelobe peak levels will be -20γ dB below the main lobe's peak level. (For example, if we desire frequency-domain sidelobe levels to be 40 dB below the main lobe's peak level, then we set $\gamma = 2$.)
3. Compute parameter α as

(I-1)

$$\alpha = \cosh[\cosh^{-1}(10^{\gamma})/M].$$

4. Compute the M -length sequence $A(m)$ using

(I-2)

$$A(m) = |\alpha \cdot \cos(\pi m / M)|$$

where the index m is $0 \leq m \leq (M-1)$.

5. For each m , evaluate the M th-degree Chebyshev polynomial whose argument is $A(m)$ to generate a frequency-domain sequence $W(m)$. There are many ways to evaluate Chebyshev polynomials. Due to its simplicity of notation, we suggest the following:

(I-3)

$$W(m) = (-1)^m \cdot \cosh\{M \cdot \cosh^{-1}[A(m)]\}, \text{ when } |A(m)| > 1,$$

or

(I-4)

$$W(m) = (-1)^m \cdot \cos\{M \cdot \cos^{-1}[A(m)]\}, \text{ when } |A(m)| \leq 1,$$

depending on whether or not an individual $|A(m)|$ value is greater than unity. In theory the resultant $W(m)$ sequence is real-only, but our software's computational numerical errors may produce a

complex-valued $W(m)$ with very small imaginary parts. Those imaginary parts, if they exist, should be ignored. The above $(-1)^m$ factors are necessary because the frequency-domain index m is never less than zero. Note: If your software does not accommodate complex values, then you can avoid problems by replacing $A(m)$ with $|A(m)|$ in this step.

6. Compute a preliminary time-domain window sequence, $w(m)$, using

$$w(m) = \text{real part of the } M\text{-point inverse DFT of } W(m).$$

7. Replace $w(0)$, the first $w(m)$ time sample, with $w(0)/2$.

8. Append that new $w(0)$ sample value to the end of the M -point $w(m)$ sequence, $w(N-1) = w(0)$, creating the desired N -length $w(k)$ window sequence where the time index k is $0 \leq k \leq (N-1)$.

9. Normalize the amplitude of $w(k)$, to obtain a unity peak amplitude, by dividing each sample of $w(k)$ from Step 8 by the maximum sample value in $w(k)$.

The above procedure seems a bit involved but it's not really so bad, as the following Chebyshev window design example will show. Assume we need an $N = 9$ -sample Chebyshev window function whose frequency-domain sidelobes are 60 dB below the window's main lobe level. Given those requirements, $N = 9$, $M = 8$, $\gamma = 3$, and from [Eq. \(I-1\)](#)

$$\alpha = \cosh[\cosh^{-1}(10^3)/8] = 1.4863.$$

After the inverse DFT operation in the above Step 6, $w(m=0)/2 = 11.91$, thus we set $w(k=0) = w(k=8) = 11.91$. The maximum value of $w(k)$ is 229.6323, so we divide the entire $w(k)$ sequence by that value, yielding our final normalized 9-sample symmetric Chebyshev window sequence listed in the rightmost column of [Table I-1](#).

Table I-1 Nine-Point Symmetric Chebyshev Window Computations

m	$A(m)$	$W(m)$	$w(m)$	k	$w(k)$	$w(k)$ norm.
0	1.4863	1000.00	23.8214	0	11.911	0.0519
1	1.3732	-411.49	52.1550	1	52.1550	0.2271
2	1.0510	6.4074	123.5232	2	123.5232	0.5379
3	0.5688	-0.1276	197.5950	3	197.5950	0.8605
4	0.0000	1.000	229.6323	4	229.6323	1.0000
5	-0.5688	-0.1276	197.5950	5	197.5950	0.8605
6	-1.0510	6.4074	123.5232	6	123.5232	0.5379
7	-1.3732	-411.49	52.1550	7	52.1550	0.2271
				8	11.911	0.0519

I.2 Chebyshev Windows for Spectrum Analysis

Nonsymmetric Chebyshev window sequences, used for spectral leakage reduction in spectrum analysis applications, are computed using the above steps with the following changes:

- For a Q -length nonsymmetric Chebyshev window sequence, where Q is an even integer, in the above Step 1 set $M = Q$.
- Skip the above Step 8, retaining the Q -length nonsymmetric $w(k)$ sequence, where the time index k is $0 \leq k \leq (Q-1)$. Normalize the amplitude of the $w(k)$ sequence as described in the above Step 9.

If $Q = 8$, for example, our final $\gamma = 3$ normalized 8-sample nonsymmetric Chebyshev window sequence would be the samples listed in the rightmost column of [Table I-2](#).

Table I-2 Eight-Point Nonsymmetric Chebyshev Window Computations

<i>m</i>	<i>A(m)</i>	<i>W(m)</i>	<i>w(m)</i>	<i>k</i>	<i>w(k)</i>	<i>w(k)</i> norm.
0	1.4863	1000.00	23.8214	0	11.911	0.0519
1	1.3732	−411.49	52.1550	1	52.1550	0.2271
2	1.0510	6.4074	123.5232	2	123.5232	0.5379
3	0.5688	−0.1276	197.5950	3	197.5950	0.8605
4	0.0000	1.000	229.6323	4	229.6323	1.0000
5	−0.5688	−0.1276	197.5950	5	197.5950	0.8605
6	−1.0510	6.4074	123.5232	6	123.5232	0.5379
7	−1.3732	−411.49	52.1550	7	52.1550	0.2271

Index

A

Absolute value, [9](#). *See also* [Magnitude](#).

A/D converters, quantization noise

clipping, [706](#)

crest factor, [640](#)

dithering, [706–709](#)

effective bits, [641](#)

fixed-point binary word length, effects of, [634–642](#)

oversampling, [704–706](#)

reducing, [704–709](#)

SNR (signal-to-noise ratio), [637–642](#), [711–714](#)

triangular dither, [708](#)

A/D converters, testing techniques

A/D dynamic range, estimating, [714–715](#)

histogram testing, [711](#)

missing codes, detecting, [715–716](#)

quantization noise, estimating with the FFT, [709–714](#)

SFDR (spurious free dynamic range), [714–715](#)

SINAD (signal-to-noise-and-distortion), [711–714](#)

SNR (signal-to-noise ratio), [711–714](#)

Adaptive filters, [184](#)

Addition

block diagram symbol, [10](#)

complex numbers, [850](#)

Additive white noise (AWN), [380](#)

AGC (automatic gain control), [783–784](#)

Aliasing

definition, [36](#)

frequency-domain ambiguity, [33–38](#)

in IIR filters, [304–305](#)

All-ones rectangular functions

DFT for, [115–118](#)

Dirichlet kernel, [115–118](#), [120](#)

Allpass filters, definition, [893](#)

AM demodulation

filtering narrowband noise, [792–797](#)

Hilbert transforms, [484–485](#)

- Amplitude
 - definition, [8](#)
 - loss. *See* [Attenuation](#).
- Amplitude response, DFT
 - complex input, [73](#)
 - real cosine input, [83–84](#)
- Analog, definition, [2](#)
- Analog filters
 - approximating, [302](#)
 - vs. digital, [169](#)
- Analog signal processing, [2](#)
- Analog-to-digital (A/D) converters. *See* [A/D converters](#).
- Analytic signals
 - bandpass quadrature, [455](#)
 - definition, [483](#)
 - generation methods, comparing, [497–498](#)
 - half-band FIR filters, [497](#)
 - time-domain, generating, [495–497](#)
- Anti-aliasing filters, [42](#), [555–558](#)
- Anti-imaging filters, [555–558](#)
- Arctangent
 - approximation, [756–758](#)
 - vector rotation. *See* [Vector rotation with arctangents](#).
- Argand, Jean Robert, [848](#)
- Argand diagrams of complex numbers, [848](#)
- Argand plane, [440–441](#)
- Attenuation
 - CIC filters, improving, [557–558](#)
 - definition, [894](#)
- Automatic gain control (AGC), [783–784](#)
- Average, statistical measures of noise, [868–870](#)
- Average power in electrical circuits, calculating, [874–875](#)
- Averaging signals. *See* [Signal averaging](#).
- AWN (additive white noise), [380](#)

B

- Band reject filters, [894](#)
- Band-limited signals, [38](#)
- Bandpass design, for FIR filters, [201–203](#)
- Bandpass filters

comb filters, [400](#)

definition, [895](#)

from half-band FIR filters, [497](#)

multisection complex FSFs, [398–403](#)

Bandpass sampling

1st-order sampling, [46](#)

definition, [43](#)

optimum sampling frequency, [46](#)

positioning sampled spectra, [48](#)

real signals, [46](#)

sampling translation, [44](#)

SNR (signal-to-noise) ratio, [48–49](#)

spectral inversion, [46–47](#)

spectral replication, [44–45](#)

Bandpass signals

in the frequency-domain, [454–455](#)

interpolating, [728–730](#)

Bandwidth, definition, [895](#)

Bartlett windows. *See* [Triangular windows](#).

Base 8 (octal) numbers, [624–625](#)

Base 16 (hexadecimal) numbers, [625](#)

Bell, Alexander Graham, [885](#)

Bels, definition, [885](#)

Bessel functions

definition, [895](#)

Bessel-derived filters, ripples, [901](#)

Bessel's correction, [870–871](#)

Bias

DC, sources and removal, [761](#)

in estimates, [870–871](#)

fixed-point binary formats, [628](#)

in signal variance, computing, [797–799](#)

Bilateral Laplace transforms, [258](#)

Bilinear transform method, designing IIR filters

analytical methods, [302](#)

definition, [257](#)

example, [326–330](#)

frequency warping, [319](#), [321–325](#), [328–330](#)

mapping complex variables, [320–324](#)

process description, [324–326](#)

- Bin centers, calculating absolute frequency, [139–140](#)
- Binary points, [629](#)
- Binary shift multiplication/division, polynomial evaluation, [773–774](#)
- Biquad filters, [299](#)
- Bit normalization, [653](#)
- Bit reversals
 - avoiding, [158](#)
 - fast Fourier transform input/output data index, [149–151](#)
- Bits, definition, [623](#)
- Blackman windows
 - in FIR filter design, [195–201](#)
 - spectral leakage reduction, [686](#)
- Blackman windows (exact), [686](#), [733](#)
- Blackman-Harris windows, [686](#), [733](#)
- Block averaging, SNR (signal-to-noise ratio), [770](#)
- Block convolution. *See* [Fast convolution](#).
- Block diagrams
 - filter structure, [172–174](#)
 - quadrature sampling, [459–462](#)
 - symbols, [10–11](#)
 - uses for, [10](#)
- Block floating point, [656–657](#)
- Boxcar windows. *See* [Rectangular windows](#).
- Butterfly patterns in FFTs
 - description, [145–149](#)
 - optimized, [156](#)
 - radix-2 structures, [151–154](#)
 - single butterfly structures, [154–158](#)
 - wingless, [156](#)
- Butterworth function
 - definition, [895](#)
 - derived filters, ripples, [901](#)

C

- Cardano, Girolamo, [439](#)
- Carrier frequency, [44](#)
- Cartesian form, quadrature signals, [442](#)
- Cascaded filters, [295–299](#), [895](#)
- Cascaded integrators, [563](#)
- Cascaded-comb subfilters, [412–413](#)

- Cascade/parallel filter combinations, [295–297](#)
- Cauer filters, [896](#)
- Causal systems, [258](#)
- Center frequency, definition, [895](#)
- Central Limit Theory, [723](#)
- Central-difference differentiators, [363–366](#)
- CFT (continuous Fourier transform), [59](#), [98–102](#)
- Chebyshev function, definition, [895](#)
- Chebyshev windows, [197–201](#), [927–930](#)
- Chebyshev-derived filters, ripples, [900](#)
- CIC (cascaded integrator-comb) filters
 - cascaded integrators, [563](#)
 - comb section, [553](#)
 - compensation FIR filters, [563–566](#)
 - definition, [895](#)
 - implementation issues, [558–563](#)
 - nonrecursive, [765–768](#)
 - recursive running sum filters, [551–552](#)
 - structures, [553–557](#)
 - substructure sharing, [765–770](#)
 - transposed structures, [765–770](#)
 - two's complement overflow, [559–563](#)
- Circular buffers, IFIR filters, [388–389](#)
- Clipping A/D converter quantization noise, [706](#)
- Coefficients. *See* [Filter coefficients](#).
- Coherent sampling, [711](#)
- Coherent signal averaging. *See* [Signal averaging, coherent](#).
- Comb filters. *See also* [Differentiators](#).
 - alternate FSF structures, [416–418](#)
 - bandpass FIR filtering, [400](#)
 - cascaded-comb subfilters, [412–413](#)
 - with complex resonators, [392–398](#)
 - frequency response, [903–904](#)
 - second-order comb filters, [412–413](#)
- Comb section. CIC filters, [553](#)
- Commutative property, LTI, [18–19](#)
- Commutator model, polyphase filters, [524](#)
- Compensation FIR filters, CIC filters, [563–566](#)
- Complex conjugate, DFT symmetry, [73](#)
- Complex down-conversion

- decimation, in frequency translation, [782](#)
- quadrature signals, [455](#), [456–462](#)
- Complex exponentials, quadrature signals, [447](#)
- Complex frequency, Laplace variable, [258](#)
- Complex frequency response, filters, [277](#)
- Complex mixing, quadrature signals, [455](#)
- Complex multipliers, down-converting quadrature signals, [458](#)
- Complex number notation, quadrature signals, [440–446](#)
- Complex numbers. *See also* [Quadrature signals](#).
- Argand diagrams, [848](#)
- arithmetic of, [848–858](#)
- definition, [439](#)
- as a function of time, [446–450](#)
- graphical representation of, [847–848](#)
- rectangular form, definition, [848–850](#)
- rectangular form, vs. polar, [856–857](#)
- roots of, [853–854](#)
- trigonometric form, [848–850](#)
- Complex phasors, quadrature signals, [446–450](#)
- Complex plane, quadrature signals, [440–441](#), [446](#)
- Complex resonators
 - with comb filters, [392–398](#)
 - FSF (frequency sampling filters), [394–398](#)
- Complex signals. *See* [Quadrature signals](#).
- Conditional stability, Laplace transform, [268](#)
- Conjugation, complex numbers, [851–852](#)
- Constant-coefficient transversal FIR filters, [184](#)
- Continuous Fourier transform (CFT), [59](#), [98–102](#)
- Continuous lowpass filters, [41](#)
- Continuous signal processing
 - definition, [2](#)
 - frequency in, [5–6](#)
- Continuous signals, definition, [2](#)
- Continuous systems, time representation, [5](#)
- Continuous time-domain, Laplace transform, [258–259](#)
- Converting analog to digital. *See* [A/D converters](#).
- Convolution. *See also* [FIR \(finite impulse response\) filters, convolution](#).
 - fast, [716–722](#)
 - LTI, [19](#)
 - overlap-and-add, [720–722](#)

overlap-and-save, [718–720](#)

Cooley, J., [135](#)

CORDIC (COordinate Rotation DIgital Computer), [756–758](#)

Coupled quadrature oscillator, [787](#)

Coupled-form IIR filter, [834–836](#)

Crest factor, [640](#)

Critical Nyquist, [37](#)

Cutoff frequencies

definition, [896](#)

designing FIR filters, [186](#)

D

Data formats

base systems, [624](#)

definition, [623](#)

place value system, [624](#)

Data formats, binary numbers. *See also* [Fixed-point binary formats](#); [Floating-point binary formats](#).

1.15 fixed-point, [630–632](#)

block floating point, [656–657](#)

converting to hexadecimal, [625](#)

converting to octal, [624–625](#)

definition, [623](#)

dynamic range, [632–634](#)

precision, [632–634](#)

representing negative values, [625–626](#)

Data overflow. *See* [Overflow](#).

dB (decibels), definition, [886](#), [896](#)

dBm (decibels), definition, [892](#)

DC

bias, sources of, [761](#)

block-data DC removal, [762](#)

defined, [62](#)

from a time-domain signal, [812–815](#)

DC removal, real-time

using filters, [761–763](#)

noise shaping property, [765](#)

with quantization, [763–765](#)

Deadband effects, [293](#)

DEC (Digital Equipment Corp.), floating-point binary formats, [654–655](#)

Decibels

bels, definition, [885](#)

common constants, [889–891](#)

dB, definition, [886](#), [896](#)

dBm, definition, [892](#)

Decimation. *See also* [Interpolation](#).

combining with interpolation, [521–522](#)

definition, [508](#)

to implement down-conversion, [676–679](#)

multirate filters, [521–522](#)

sample rate converters, [521–522](#)

drawing downsampled spectra, [515–516](#)

frequency properties, [514–515](#)

magnitude loss in the frequency-domain, [515](#)

overview, [508–510](#)

time invariance, [514](#)

time properties, [514–515](#)

example, [512–513](#)

overview, [510–511](#)

polyphase decomposition, [514](#)

Decimation filters

choosing, [510](#)

definition, [896](#)

Decimation-in-frequency algorithms, FFTs

radix-2 butterfly structures, [151–154](#), [734–735](#)

Decimation-in-time algorithms, FFTs

index bit reversal, [149–151](#)

radix-2 butterfly structures, [151–154](#)

single butterfly structures, [154–158](#), [735–737](#)

Demodulation

AM, [484–485](#)

FM, [486](#)

quadrature signals, [453–455](#), [456–462](#)

Descartes, René, [439](#)

Detection

envelope, [784–786](#)

peak threshold, with matched filters, [377](#), [379–380](#)

quadrature signals, [453–454](#)

signal transition, [820–821](#)

single tone. *See* [Single tone detection](#).

DFT (discrete Fourier transform). *See also* [DTFT \(discrete-time Fourier transform\)](#); [SDFT \(sliding](#)

[DFT](#)).

analyzing FIR filters, [228–230](#)

computing large DFTs from small FFTs, [826–829](#)

definition, [60](#)

examples, [63–73](#), [78–80](#)

versus FFT, [136–137](#)

frequency axis, [77](#)

frequency granularity, improving. *See* [Zero padding](#).

frequency spacing, [77](#)

frequency-domain sampling, [98–102](#)

inverse, [80–81](#)

linearity, [75](#)

magnitudes, [75–76](#)

picket fence effect, [97](#)

rectangular functions, [105–112](#)

resolution, [77](#), [98–102](#)

scallop loss, [96–97](#)

shifting theorem, [77–78](#)

spectral estimation, improving. *See* [Zero padding](#).

time reversal, [863–865](#)

zero padding, [97–102](#)

DFT leakage. *See also* [Spectral leakage, FFTs](#).

cause, [82–84](#)

definition, [81](#)

description, [81–82](#)

predicting, [82–84](#)

sinc functions, [83](#), [89](#)

wraparound, [86–88](#)

DFT leakage, minimizing

Chebyshev windows, [96](#)

Hamming windows, [89–93](#)

Hanning windows, [89–97](#)

Kaiser windows, [96](#)

rectangular windows, [89–97](#)

triangular windows, [89–93](#)

windowing, [89–97](#)

DFT processing gain

average output noise-power level, [103–104](#)

inherent gain, [102–105](#)

integration gain, [105](#)

- multiple DFTs, [105](#)
- output signal-power level, [103–104](#)
- single DFT, [102–105](#)
- SNR (signal-to-noise ratio), [103–104](#)
- DIF (decimation-in-frequency), [734–735](#)
- Difference equations
 - example, [5](#)
 - IIR filters, [255–256](#)
- Differentiators
 - central-difference, [363–366](#)
 - differentiating filters, [364](#)
 - first-difference, [363–366](#)
 - narrowband, [366–367](#)
 - optimized wideband, [369–370](#)
 - overview, [361–363](#)
 - performance improvement, [810–812](#)
 - wideband, [367–369](#)
- Digital differencer. *See* [Differentiators](#).
- Digital Equipment Corp. (DEC), floating-point binary formats, [654–655](#)
- Digital filters. *See also specific [filters](#)*.
 - vs. analog, [169](#)
 - definition, [896](#)
- Digital signal processing, [2](#)
- Direct Form I filters, [275–278](#), [289](#)
- Direct Form II filters, [289–292](#)
- Direct Form implementations, IIR filters, [292–293](#)
- Dirichlet, Peter, [108](#)
- Dirichlet kernel
 - all-ones rectangular functions, [115–118](#), [120](#)
 - general rectangular functions, [108–112](#)
 - symmetrical rectangular functions, [113–114](#)
- Discrete convolution in FIR filters. *See also* [FIR \(finite impulse response\) filters, convolution](#).
 - description, [214–215](#)
 - in the time domain, [215–219](#)
- Discrete Fourier transform (DFT). *See* [DFT \(discrete Fourier transform\)](#).
- Discrete Hilbert transforms. *See* [Hilbert transforms](#).
- Discrete linear systems, [12–16](#)
- Discrete systems
 - definition, [4](#)
 - example, [4–5](#)

- time representation, [5](#)
- Discrete-time expression, [4](#)
- Discrete-time Fourier transform (DTFT), [101](#), [120–123](#)
- Discrete-time signals
 - example of, [2](#)
 - frequency in, [5–6](#)
 - sampling, frequency-domain ambiguity, [33–38](#)
 - use of term, [2](#)
- Discrete-time waveforms, describing, [8](#)
- Dispersion, statistical measures of noise, [869](#)
- DIT (decimation-in-time), [735–737](#)
- Dithering
 - A/D converter quantization noise, [706–709](#)
 - with filters, [294](#)
 - triangular, [708](#)
- Dolph-Chebyshev windows in FIR filter design, [197](#)
- Down-conversion
 - Delay/Hilbert transform filter, [817–818](#), [819–820](#)
 - filtering and decimation, [676–679](#)
 - folded FIR filters, [818](#)
 - frequency translation, without multiplication, [676–679](#)
 - half-band filters, [817–818](#)
 - single-decimation technique, [819–820](#)
- Down-conversion, quadrature signals
 - complex, [455](#), [456–462](#)
 - complex multipliers, [458](#)
 - sampling with digital mixing, [462–464](#)
- Downsampling, decimation
 - drawing downsampled spectra, [515–516](#)
 - frequency properties, [514–515](#)
 - magnitude loss in the frequency-domain, [515](#)
 - overview, [508–510](#)
 - time invariance, [514](#)
 - time properties, [514–515](#)
- DTFT (discrete-time Fourier transform), [101](#), [120–123](#). *See also* [DFT \(discrete Fourier transform\)](#).
- Dynamic range
 - binary numbers, [632–634](#)
 - floating-point binary formats, [656–658](#)
 - SFDR (spurious free dynamic range), [714–715](#)

- Elliptic functions, definition, [896](#)
- Elliptic-derived filters, ripples, [900](#)
- Envelope delay. *See* [Group delay](#).
- Envelope detection
 - approximate, [784–786](#)
 - Hilbert transforms, [483–495](#)
- Equiripple filters, [418](#), [901](#)
- Estrin's Method, polynomial evaluation, [774–775](#)
- Euler, Leonhard, [442](#), [444](#)
- Euler's equation
 - bilinear transform design of IIR filters, [322](#)
 - DFT equations, [60](#), [108](#)
 - impulse invariance design of IIR filters, [315](#)
 - quadrature signals, [442–443](#), [449](#), [453](#)
- Exact Blackman windows, [686](#)
- Exact interpolation, [778–781](#)
- Exponent, floating-point binary format, [652](#)
- Exponential averagers, [608–612](#)
- Exponential moving averages, [801–802](#)
- Exponential signal averaging. *See* [Signal averaging, exponential](#).
- Exponential variance computation, [801–802](#)

F

- Fast convolution, [716–722](#)
- FFT (fast Fourier transform)
 - averaging multiple, [139](#)
 - constant-geometry algorithms, [158](#)
 - convolution. *See* [Fast convolution](#).
 - decimation-in-frequency algorithms, [151–154](#)
 - decimation-in-time algorithms, [149–158](#)
 - versus* DFT, [136–137](#)
 - exact interpolation, [778–781](#)
 - fast FIR filtering, [716–722](#)
 - hints for using, [137–141](#)
 - history of, [135](#)
 - interpolated analytic signals, computing, [781](#)
 - interpolated real signals, interpolating, [779–780](#)
 - interpreting results, [139–141](#)
 - inverse, computing, [699–702](#), [831–833](#)
 - in place algorithm, [157](#)

- radix-2 algorithm, [141–149](#)
- radix-2 butterfly structures, [151–158](#)
- signal averaging, [600–603](#)
- single tone detection, [737–738](#), [740–741](#)
- vs. single tone detection, [740–741](#)
- software programs, [141](#)
- time-domain interpolation, [778–781](#)
- Zoom FFT, [749–753](#)
- FFT (fast Fourier transform), real sequences
 - a $2N$ -point real FFT, [695–699](#)
 - two N -point real FFTs, [687–694](#)
- FFT (fast Fourier transform), twiddle factors
 - derivation of the radix-2 FFT algorithm, [143–149](#)
- DIF (decimation-in-frequency), [734–735](#)
- DIT (decimation-in-time), [735–737](#)
- Fibonacci, [450–451](#)
- Filter coefficients
 - definition, [897](#)
 - for FIRs. *See* [Impulse response](#).
 - flipping, [493–494](#)
 - for FSF (frequency sampling filters), [913–926](#)
 - quantization, [293–295](#)
- Filter order, [897](#)
- Filter taps, estimating, [234–235](#), [386–387](#)
- Filters. *See also* [FIR \(finite impulse response\) filters](#); [IIR \(infinite impulse response\) filters](#); [Matched filters](#); *specific filters*.
 - adaptive filters, [184](#)
 - allpass, [893](#)
 - analog vs. digital, [169](#)
 - band reject, [894](#)
 - bandpass, [895](#)
 - cascaded, [895](#)
 - Cauer, [896](#)
 - CIC, [895](#)
 - DC-removal, [762–763](#)
 - decimation, [896](#)
 - differentiating, [364](#). *See also* [Differentiators](#).
 - digital, [896](#)
 - down-conversion, [676–679](#)
 - equiripple, [418](#)

- highpass, [898](#)
- linear phase, [899](#)
- lowpass, [899](#)
- narrowband noise, [792–797](#)
- nonrecursive, [226–230](#), [290–291](#), [899](#)
- optimal FIR, [418](#)
- overview, [169–170](#)
- parallel, [295–297](#)
- passband, [900](#)
- process description, [169–170](#)
- prototype, [303](#)
- quadrature, [900](#)
- real-time DC removal, [762–763](#)
- recursive, [290–291](#), [900](#)
- recursive running sum, [551–552](#)
- Remez Exchange, [418](#)
- sharpening, [726–728](#)
- structure, diagramming, [172–174](#)
- time-domain slope detection, [820–821](#)
- transposed structure, [291–292](#)
- transversal, [173–174](#). *See also* [FIR \(finite impulse response\) filters](#).
- zero-phase, [725](#), [902](#)

Filters, analytic signals

- half-band FIR filters, [497](#)
- I-channel filters, [496](#)
- in-phase filters, [496](#)
- Q-channel filters, [496](#)
- quadrature phase filters, [496](#)
- time-domain FIR filter implementation, [489–494](#)

Finite-word-length errors, [293–295](#)

FIR (finite impulse response) filters. *See also* [FSF \(frequency sampling filters\)](#); [IFIR \(interpolated FIR\) filters](#); [IIR \(infinite impulse response\) filters](#).

- coefficients. *See* [Impulse response](#).
- constant coefficients, [184](#)
- definition, [897](#)
- fast FIR filtering using the FFT, [716–722](#)
- folded structure. *See* [Folded FIR filters](#).
- frequency magnitude response, determining, [179](#)
- frequency-domain response, determining, [179](#)
- group delay, [211–212](#)

half-band. *See* [Half-band FIR filters](#).

vs. IIR filters, [332–333](#)

impulse response, [177–179](#)

narrowband lowpass. *See* [IFIR \(interpolated FIR\) filters](#).

nonrecursive, analyzing, [226–230](#)

phase response in, [209–214](#)

phase unwrapping, [210](#)

phase wrapping, [209](#), [900](#)

polyphase filters, [522–527](#)

sharpening, [726–728](#)

signal averaging. *See* [Signal averaging, with FIR filters](#).

signal averaging with, [178](#), [180–184](#)

stopband attenuation, improving, [726–728](#)

tapped delay, [181–182](#)

transient response, [181–182](#)

z-transform of, [288–289](#)

FIR (finite impulse response) filters, analyzing

with DFTs, [228–230](#)

estimating number of, [234–235](#)

fractional delay, [233](#)

group delay, [230–233](#)

passband gain, [233–234](#)

stopband attenuation, [234–235](#)

symmetrical-coefficient FIR filters, [232–233](#)

FIR (finite impulse response) filters, convolution

description, [175–186](#)

discrete, description, [214–215](#)

discrete, in the time domain, [215–219](#)

fast convolution, [716–722](#)

impulse response, [177–178](#)

inputs, time order reversal, [176](#)

signal averaging, [175–176](#)

theorem, applying, [222–226](#)

theorem, description, [219–222](#)

time-domain aliasing, avoiding, [718–722](#)

time-domain convolution vs. frequency-domain multiplication, [191–194](#)

FIR (finite impulse response) filters, designing

bandpass method, [201–203](#)

cutoff frequencies, [186](#)

with forward FFT software routines, [189](#)

Fourier series design method. *See* [Window design method, FIR filters](#).

Gibbs's phenomenon, [193](#)

highpass method, [203–204](#)

low-pass design, [186–201](#)

magnitude fluctuations, [190–194](#)

Optimal design method, [204–207](#)

Parks-McClellan Exchange method, [204–207](#)

passband ripples, minimizing, [190–194](#), [204–207](#). *See also* [Windows](#).

Remez method, [204–207](#)

stopband ripples, minimizing, [204–207](#)

time-domain coefficients, determining, [186–194](#)

time-domain convolution vs. frequency-domain multiplication, [191–194](#)

very high performance filters, [775–778](#)

window design method, [186–194](#)

windows used in, [194–201](#)

1st-order IIR filters, signal averaging, [612–614](#)

1st-order sampling, [46](#)

First-difference differentiators, [363–366](#)

Fixed-point binary formats. *See also* [Floating-point binary formats](#).

1.15 format, [630–632](#)

bias, [628](#)

binary points, [629](#)

decimal numbers, converting to 1.5 binary, [632](#)

fractional binary numbers, [629–632](#)

hexadecimal (base 16) numbers, [625](#)

integer plus fraction, [629](#)

lsb (least significant bit), [624](#)

msb (most significant bit), [624](#)

octal (base 8) numbers, [624–625](#)

offset, [627–628](#)

overflow, [629](#)

Q30 format, [629](#)

radix points, [629](#)

representing negative values, [625–626](#)

sign extend operations, [627](#)

sign-magnitude, [625–626](#)

two's complement, [626–627](#), [629](#)

Fixed-point binary formats, finite word lengths

A/D converter best estimate values, [635](#)

A/D converter quantization noise, [634–642](#)

A/D converter vs. SNR, [640–642](#)

convergent rounding, [651](#)

crest factor, [640](#)

data overflow, [642–646](#)

data rounding, [649–652](#)

effective bits, [641](#)

round off noise, [636–637](#)

round to even method, [651](#)

round-to-nearest method, [650–651](#)

truncation, [646–649](#)

Floating-point binary formats. *See also* [Fixed-point binary formats](#).

bit normalization, [653](#)

common formats, [654–655](#)

DEC (Digital Equipment Corp.), [654–655](#)

description, [652](#)

dynamic range, [656–658](#)

evaluating, [652](#)

exponent, [652](#)

fractions, [653](#)

gradual underflow, [656](#)

hidden bits, [653](#)

IBM, [654–655](#)

IEEE Standard P754, [654–655](#)

mantissa, [652](#)

MIL-STD 1750A, [654–655](#)

min/max values, determining, [656–657](#)

unnormalized fractions, [656](#)

word lengths, [655](#)

FM demodulation

algorithms for, [758–761](#)

filtering narrowband noise, [792–797](#)

Hilbert transforms, [486](#)

Folded FIR filters

designing Hilbert transforms, [493](#)

down-conversion, [818](#)

frequency translation, without multiplication, [678](#)

half-band filters, sample rate conversion, [548](#)

Hilbert transforms, designing, [493](#)

multipliers, reducing, [702–704](#)

nonrecursive, [419–420](#)

tapped-delay line, [389](#)

Folding frequencies, [40](#)

Forward FFT

computing, [831–833](#)

software routines for designing FIR filters, [189](#)

Fourier series design FIR filters. *See* [Window design method, FIR filters](#).

Fourier transform pairs, FIR filters, [178–179](#)

Fractional binary numbers, [629–632](#)

Fractional delay, FIR filters, [233](#)

Frequency

continuous *vs.* discrete systems, [5](#)

of discrete signals, determining. *See* [DFT \(discrete Fourier transform\)](#).

discrete-time signals, [5–6](#)

properties, interpolation, [519](#)

resolution, improving with FIR filters, [228–230](#)

units of measure, [2–3](#)

Frequency attenuation, FIR filters, [182](#)

Frequency axis

definition, [77](#)

DFT, [77](#)

in Hz, [118](#)

normalized angle variable, [118](#)

in radians/seconds, [118–119](#)

rectangular functions, [118–120](#)

with zero padding, [100](#)

Frequency domain

definition, [6](#)

Hamming windows, [683–686](#)

Hanning windows, [683–686](#)

listing sequences, [7](#)

performance. IIR filters, [282–289](#)

quadrature signals, [451–454](#)

spectral leak reduction, [683–686](#)

windowing in, [683–686](#)

windows, [683–686](#)

Frequency magnitude response

definition, [897](#)

determining with FIR filters, [179](#)

Frequency response

LTI, determining, [19](#)

- for Mth-order IIR filter, [275–276](#)
- Frequency response, FIR filters
 - determining, [179–186](#)
 - factors affecting, [174](#)
 - modifying, [184–186](#)
- Frequency sampling design method vs. FSF, [393–394](#)
- Frequency sampling filters. *See* [FSF \(frequency sampling filters\)](#).
- Frequency translation, bandpass sampling, [44](#)
- Frequency translation, with decimation
 - complex down-conversion, [782](#)
 - complex signals, [781–783](#)
 - real signals, [781](#)
- Frequency translation, without multiplication
 - by 1/2 the sampling rate, [671–673](#)
 - by 1/4 the sampling rate, [674–676](#)
 - down-conversion, [676–679](#)
 - inverting the output spectrum, [678–679](#)
- Frequency translation to baseband, quadrature signals, [319](#)
- Frequency warping, [319](#), [321–325](#), [328–330](#)
- FSF (frequency sampling filters). *See also* [FIR \(finite impulse response\) filters](#).
 - complex resonators, [394–398](#)
 - designing, [423–426](#)
 - frequency response, single complex FSF, [904–905](#)
 - history of, [392–394](#)
 - linear-phase multisection real-valued, [409–410](#)
 - modeling, [413–414](#)
 - multisection complex, [398–403](#)
 - multisection real-valued, [406–409](#)
 - vs. Parks-McClellan filters, [392](#)
 - real FSF transfer function, [908–909](#)
 - stability, [403–406](#)
 - stopband attenuation, increasing, [414–416](#)
 - stopband sidelobe level suppression, [416](#)
 - transition band coefficients, [414–416](#)
 - Type IV example, [419–420](#), [423–426](#)

G

- Gain. *See also* [DFT processing gain](#).
 - AGC (automatic gain control), [783–784](#)
 - IIR filters, scaling, [300–302](#)

integration, signal averaging, [600–603](#)

passband, [233–234](#)

windows, [92](#)

Gauss, Karl, [439](#), [444](#)

Gaussian PDFs, [882–883](#)

General numbers, [446](#). *See also* [Complex numbers](#).

Geometric series, closed form, [107](#), [859–861](#)

Gibbs's phenomenon, [193](#)

Goertzel algorithm, single tone detection

advantages of, [739](#)

description, [738–740](#)

example, [740](#)

vs. the FFT, [740–741](#)

stability, [838–840](#)

Gold-Rader filter, [834–836](#)

Gradual underflow, floating-point binary formats, [656](#)

Gregory, James, [23](#)

Group delay

definition, [897–898](#)

differentiators, [365](#)

filters, computing, [830–831](#)

FIR filters, [211–212](#), [230–233](#)

H

Half Nyquist, [37](#)

Half-band FIR filters

analytic signals, [497](#)

as complex bandpass filters, [497](#)

definition, [898](#)

description, [207–209](#)

down-conversion, [817–818](#)

frequency translation, [802–804](#)

Half-band FIR filters, sample rate conversion

fundamentals, [544–546](#)

implementation, [546–548](#)

overview, [543](#)

Hamming, Richard, [366](#)

Hamming windows

in the frequency domain, [683–686](#)

spectral peak location, [733](#)

Hann windows. *See* [Hanning windows](#).

Hanning windows

- description, [89–97](#)

- DFT leakage, minimizing, [89–97](#)

- in the frequency domain, [683–686](#)

- spectral peak location, [733](#)

Harmonic sampling. *See* [Bandpass sampling](#).

Harmonics of discrete signals, determining. *See* [DFT \(discrete Fourier transform\)](#).

Harris, Fred, [791](#)

Heaviside, Oliver, [257](#)

Hertz, [3](#)

Hertz, Heinrich, [3](#)

Hexadecimal (base 16) numbers, [625](#)

Hidden bits, floating-point binary formats, [653](#)

Highpass filters, definition, [898](#)

Highpass method, designing FIR filters, [203–204](#)

Hilbert, David, [479](#)

Hilbert transformers, designing

- common mistake, [493–494](#)

- even-tap transformers, [493](#)

- frequency-domain transformers, [494–495](#)

- half-band filter coefficient modification, [804–805](#)

- half-band filter frequency translation, [802–804](#)

- odd-tap transformers, [493](#)

- time-domain FIR filter implementation, [489–494](#)

- time-domain transformers, [489–494](#)

Hilbert transforms

- AM demodulation, [484–485](#)

- definition, [480](#)

- envelope detection, [483–495](#)

- example, [481–482](#)

- FM demodulation, [486](#)

- impulse response, [487–489](#)

- one-sided spectrum, [483](#)

- signal envelope, [483–495](#)

Hilbert transforms, analytic signals

- definition, [483](#)

- generation methods, comparing, [497–498](#)

- half-band FIR filters, [497](#)

- time-domain, generating, [495–497](#)

Histogram testing, A/D converter techniques, [711](#)

Homogeneity property, [12](#)

Horner, William, [773](#)

Horner's Rule, [772–774](#)

Human ear, sensitivity to decibels, [886](#)

I

IBM, floating-point binary formats, [654–655](#)

I-channel filters, analytic signals, [496](#)

IDFT (inverse discrete Fourier transform), [80–81](#)

IEEE Standard P754, floating-point binary formats, [654–655](#)

IF sampling. *See* [Bandpass sampling](#).

IFIR (interpolated FIR) filters. *See also* [FIR \(finite impulse response\) filters](#).

- computational advantage, [384–385](#), [391](#)

- definition, [381](#)

- expansion factor M , [381](#), [385–386](#)

- filter taps, estimating, [386–387](#)

- image-reject subfilter, [382–384](#), [390](#)

- implementation issues, [388–389](#)

- interpolated, definition, [384](#)

- interpolators. *See* [Image-reject subfilter](#).

- lowpass design example, [389–391](#)

- optimum expansion factor, [386](#)

- performance modeling, [387–388](#)

- prototype filters, [382](#)

- shaping subfilters, [382](#), [385](#)

IIR (infinite impulse response) filters. *See also* [FIR \(finite impulse response\) filters](#); [FSF \(frequency sampling filters\)](#).

- allpass, [893](#)

- analytical design methods, [302](#)

- coupled-form, [834–836](#)

- definition, [899](#)

- design techniques, [257](#). *See also specific techniques*.

- difference equations, [255–256](#)

- vs. FIR filters, [253](#), [332–333](#)

- frequency domain performance, [282–289](#)

- infinite impulse response, definition, [280](#)

- interpolated, example, [837–838](#)

- phase equalizers. *See* [Allpass filters](#).

- poles, [284–289](#)

- recursive filters, [290–291](#)

scaling the gain, [300–302](#)

SNR (signal-to-noise ratio), [302](#)

stability, [263–270](#)

z-domain transfer function, [282–289](#)

zeros, [284–289](#)

z-plane pole / zero properties, [288–289](#)

z-transform, [270–282](#)

IIR (infinite impulse response) filters, pitfalls in building
coefficient quantization, [293–295](#)

deadband effects, [293](#)

Direct Form implementations, [292–293](#)

dither sequences, [294](#)

finite word length errors, [293–295](#)

limit cycles, [293](#)

limited-precision coefficients, [293](#)

overflow, [293–295](#)

overflow oscillations, [293](#)

overview, [292–293](#)

rounding off, [293](#)

IIR (infinite impulse response) filters, structures

biquad filters, [299](#)

cascade filter properties, [295–297](#)

cascaded, [295–299](#)

cascade/parallel combinations, [295–297](#)

changing, [291–292](#)

Direct Form I, [275–278](#), [289](#)

Direct Form II, [289–292](#)

optimizing partitioning, [297–299](#)

parallel filter properties, [295–297](#)

transposed, [291–292](#)

transposed Direct Form II, [289–290](#)

transposition theorem, [291–292](#)

Imaginary numbers, [439](#), [446](#)

Imaginary part, quadrature signals, [440](#), [454–455](#)

Impulse invariance method, designing IIR filters

aliasing, [304–305](#)

analytical methods, [302](#)

definition, [257](#)

Method 1, description, [305–307](#)

Method 1, example, [310–313](#)

Method 2, description, [307–310](#)

Method 2, example, [313–319](#)

preferred method, [317](#)

process description, [303–310](#)

prototype filters, [303](#)

Impulse response

convolution in FIR filters, [177–178](#)

definition, [898–899](#)

FIR filters, [177–179](#)

Hilbert transforms, [487–489](#)

Incoherent signal averaging. *See* [Signal averaging, incoherent](#).

Infinite impulse response (IIR) filters. *See* [IIR \(infinite impulse response\) filters](#).

Integer plus fraction fixed-point binary formats, [629](#)

Integration gain, signal averaging, [600–603](#)

Integrators

CIC filters, [553](#)

overview, [370](#)

performance comparison, [373–376](#)

rectangular rule, [371–372](#)

Simpson's rule, [372](#), [373–376](#)

Tick's rule, [373–376](#)

trapezoidal rule, [372](#)

Intermodulation distortion, [16](#)

Interpolated analytic signals, computing, [781](#)

Interpolated FIR (IFIR) filters. *See* [IFIR \(interpolated FIR\) filters](#).

Interpolated real signals, interpolating, [779–780](#)

Interpolation. *See also* [Decimation](#).

accuracy, [519](#)

bandpass signals, [728–730](#)

combining with decimation, [521–522](#)

definition, [384](#), [508](#)

drawing upsampled spectra, [520–521](#)

exact, [778–781](#)

frequency properties, [519](#)

history of, [519](#)

linear, [815–816](#)

multirate filters, [521–522](#)

overview, [516–518](#)

sample rate converters, [521–522](#)

time properties, [519](#)

time-domain, [778–781](#)

unwanted spectral images, [519](#)

upsampling, [517–518](#), [520–521](#)

zero stuffing, [518](#)

Interpolation filters, [518](#)

Inverse DFT, [80–81](#)

Inverse discrete Fourier transform (IDFT), [80–81](#)

Inverse FFT, [699–702](#), [831–833](#)

Inverse of complex numbers, [853](#)

Inverse sinc filters, [563–566](#)

I/Q demodulation, quadrature signals, [459–462](#)

J

Jacobsen, Eric, [775](#)

j -operator, quadrature signals, [439](#), [444–450](#)

K

Kaiser, James, [270](#)

Kaiser windows, in FIR filter design, [197–201](#)

Kaiser-Bessel windows, in FIR filter design, [197](#)

Kelvin, Lord, [60](#)

Kootsookos, Peter, [603](#), [724](#)

Kotelnikov, V., [42](#)

L

Lanczos differentiators, [366–367](#)

Laplace transfer function

conditional stability, [268](#)

description, [262–263](#)

determining system stability, [263–264](#), [268](#)

impulse invariance design, Method 1, [305–307](#), [310–313](#)

impulse invariance design, Method 2, [307–310](#), [313–319](#)

in parallel filters, [295–297](#)

second order, [265–268](#)

Laplace transform. *See also* Z-transform.

bilateral transform, [258](#)

causal systems, [258](#)

conditional stability, [268](#)

for continuous time-domain, [258–259](#)

description, [257–263](#)

development of, [257](#)

- one-sided transform, [258](#)
- one-sided/causal, [258](#)
- poles on the s-plane, [263–270](#)
- stability, [263–270](#)
- two-sided transform, [258](#)
- zeros on the s-plane, [263–270](#)
- Laplace variable, complex frequency, [258](#)
- Leakage. *See* [DFT leakage](#).
- Leaky integrator, [614](#)
- Least significant bit (lsb), [624](#)
- l'Hopital's Rule, [110](#)
- Limit cycles, [293](#)
- Linear, definition, [12](#)
- Linear differential equations, solving. *See* [Laplace transform](#).
- Linear interpolation, [815–816](#)
- Linear phase filters, [899](#)
- Linear systems, example, [13–14](#)
- Linear time-invariant (LTI) systems. *See* [LTI \(linear time-invariant\) systems](#).
- Linearity, DFT, [75](#)
- Linear-phase filters
 - DC removal, [812–815](#)
 - definition, [899](#)
- Logarithms
 - and complex numbers, [854–856](#)
 - measuring signal power, [191](#)
- Lowpass design
 - designing FIR filters, [186–201](#)
 - IFIR filters, example, [389–391](#)
- Lowpass filters, definition, [899](#)
- Lowpass signals
 - definition, [38](#)
 - sampling, [38–42](#)
- lsb (least significant bit), [624](#)
- LTI (linear time-invariant) systems
 - analyzing, [19–21](#)
 - commutative property, [18–19](#)
 - convolution, [19](#)
 - DFT (discrete Fourier transform), [19](#)
 - discrete linear systems, [12–16](#)
 - frequency response, determining, [19](#)

- homogeneity property, [12](#)
- intermodulation distortion, [16](#)
- internally generated sinusoids, [16](#)
- linear, definition, [12](#)
- linear system, example, [13–14](#)
- nonlinear system, example, [14–16](#)
- output sequence, determining, [19](#)
- overview, [12](#)
- proportionality characteristic, [12](#)
- rearranging sequential order, [18–19](#)
- time-invariant systems, [17–18](#)
- unit impulse response, [19–20](#)

M

MAC (multiply and accumulate) architecture

- polynomial evaluation, [773](#)
- programmable DSP chips, [333](#)

Magnitude

- approximation (vector), [679–683](#)
- of complex numbers, [848](#)
- definition, [8–9](#)
- DFT, [75–76](#)

Magnitude and angle form of complex numbers, [848–850](#)

Magnitude response of DFTs

- aliased sinc function, [108](#)
- all-ones rectangular functions, [115–118](#)
- fluctuations. *See* [Scalloping](#).
- general rectangular functions, [106–112](#)
- overview, [105–106](#)
- sidelobe magnitudes, [110–111](#)
- symmetrical rectangular functions, [112–115](#)

Magnitude response of DFTs, Dirichlet kernel

- all-ones rectangular functions, [115–118](#), [120](#)
- general rectangular functions, [108–112](#)
- symmetrical rectangular functions, [113–114](#)

Magnitude-angle form, quadrature signals, [442](#)

Mantissa, floating-point binary formats, [652](#)

Matched filters

- definition, [376](#)
- example, [378–380](#)

- implementation considerations, [380](#)
- peak detection threshold, [377](#), [379–380](#)
- properties, [376–378](#)
- purpose, [376](#)
- SNR (signal-power-to-noise-power ratio), maximizing, [376](#)
- McClellan, James, [206](#). *See also* [Parks-McClellan algorithm](#).
- Mean (statistical measure of noise)
 - definition, [868–869](#)
 - PDF (probability density function), [879–882](#)
 - of random functions, [879–882](#)
- Mean (statistical average), of random functions, [879–882](#)
- Mehrnia, A., [386](#)
- MIL-STD 1750A, floating-point binary formats, [654–655](#)
- Missing
 - A/D conversion codes, checking, [715–716](#)
 - sample data, recovering, [823–826](#). *See also* [Interpolation](#).
- Mixing. *See* Frequency translation.
- Modeling FSF (frequency sampling filters), [413–414](#)
- Modulation, quadrature signals, [453–454](#)
- Modulus of complex numbers, [848](#)
- Most significant bit (msb), [624](#)
- Moving averages
 - CIC filters, [551–552](#)
 - as digital lowpass filters, [20–21](#), [173](#), [231](#)
 - sample rate conversion, CIC filters, [551–552](#)
- Moving averages, coherent signal averaging
 - exponential moving averages, computing, [801–802](#)
 - exponential signal averaging, [801–802](#)
 - moving averages, computing, [799–801](#)
 - nonrecursive moving averagers, [606–608](#)
 - recursive moving averagers, [606–608](#)
 - time-domain averaging, [604–608](#)
- msb (most significant bit), [624](#)
- Multiplication
 - block diagram symbol, [10](#)
 - CIC filters, simplified, [765–770](#)
 - complex numbers, [850–851](#)
- Multirate filters
 - decimation, [521–522](#)
 - interpolation, [521–522](#)

Multirate systems, sample rate conversion

filter mathematical notation, [534–535](#)

signal mathematical notation, [533–534](#)

z-transform analysis, [533–535](#)

Multirate systems, two-stage decimation, [511](#)

N

Narrowband differentiators, [366–367](#)

Narrowband noise filters, [792–797](#)

Natural logarithms of complex numbers, [854](#)

Negative frequency, quadrature signals, [450–451](#)

Negative values in binary numbers, [625–626](#)

Newton, Isaac, [773](#)

Newton's method, [372](#)

Noble identities, polyphase filters, [536](#)

Noise

definition, [589](#)

measuring. *See* [Statistical measures of noise](#).

random, [868](#)

Noise shaping property, [765](#)

Nonlinear systems, example, [14–16](#)

Nonrecursive CIC filters

description, [765–768](#)

prime-factor-R technique, [768–770](#)

Nonrecursive filters. *See* [FIR filters](#)

Nonrecursive moving averagers, [606–608](#)

Normal distribution of random data, generating, [722–724](#)

Normal PDFs, [882–883](#)

Normalized angle variable, [118–119](#)

Notch filters. *See* [Band reject filters](#).

Nyquist, H., [42](#)

Nyquist criterion, sampling lowpass signals, [40](#)

O

Octal (base 8) numbers, [624–625](#)

Offset fixed-point binary formats, [627–628](#)

1.15 fixed-point binary format, [630–632](#)

Optimal design method, designing FIR filters, [204–207](#)

Optimal FIR filters, [418](#)

Optimization method, designing IIR filters

definition, [257](#)

description, [302](#)

iterative optimization, [330](#)

process description, [330–332](#)

Optimized butterflies, [156](#)

Optimized wideband differentiators, [369–370](#)

Optimum sampling frequency, [46](#)

Order

of filters, [897](#)

polyphase filters, swapping, [536–537](#)

Orthogonality, quadrature signals, [448](#)

Oscillation, quadrature signals, [459–462](#)

Oscillator, quadrature

coupled, [787](#)

overview, [786–789](#)

Taylor series approximation, [788](#)

Overflow

computing the magnitude of complex numbers, [815](#)

fixed-point binary formats, [629](#), [642–646](#)

two's complement, [559–563](#)

Overflow errors, [293–295](#)

Overflow oscillations, [293](#)

Oversampling A/D converter quantization noise, [704–706](#)

P

Parallel filters, Laplace transfer function, [295–297](#)

Parks-McClellan algorithm

designing FIR filters, [204–207](#)

vs. FSF (frequency sampling filters), [392](#)

optimized wideband differentiators, [369–370](#)

Parzen windows. *See* [Triangular windows](#).

Passband, definition, [900](#)

Passband filters, definition, [900](#)

Passband gain, FIR filters, [233–234](#)

Passband ripples

cascaded filters, estimating, [296–297](#)

definition, [296](#), [900](#)

IFIR filters, [390](#)

minimizing, [190–194](#), [204–207](#)

PDF (probability density function)

Gaussian, [882–883](#)

- mean, calculating, [879–882](#)
- mean and variance of random functions, [879–882](#)
- normal, [882–883](#)
- variance, calculating, [879–882](#)
- Peak correlation, matched filters, [379](#)
- Peak detection threshold, matched filters, [377](#), [379–380](#)
- Periodic sampling
 - aliasing, [33–38](#)
 - frequency-domain ambiguity, [33–38](#)
- Periodic sampling
 - 1st-order sampling, [46](#)
 - anti-aliasing filters, [42](#)
 - bandpass, [43–49](#)
 - coherent sampling, [711](#)
 - definition, [43](#)
 - folding frequencies, [40](#)
 - Nyquist criterion, [40](#)
 - optimum sampling frequency, [46](#)
 - real signals, [46](#)
 - sampling translation, [44](#)
 - SNR (signal-to-noise) ratio, [48–49](#)
 - spectral inversion, [46–47](#)
 - undersampling, [40](#)
- Phase angles, signal averaging, [603–604](#)
- Phase delay. *See* [Phase response](#).
- Phase response
 - definition, [900](#)
 - in FIR filters, [209–214](#)
- Phase unwrapping, FIR filters, [210](#)
- Phase wrapping, FIR filters, [209](#), [900](#)
- Pi, calculating, [23](#)
- Picket fence effect, [97](#)
- Pisa, Leonardo da, [450–451](#)
- Polar form
 - complex numbers, vs. rectangular, [856–857](#)
 - quadrature signals, [442](#), [443–444](#)
- Poles
 - IIR filters, [284–289](#)
 - on the s-plane, Laplace transform, [263–270](#)
- Polynomial curve fitting, [372](#)

Polynomial evaluation

binary shift multiplication/division, [773–774](#)

Estrin's Method, [774–775](#)

Horner's Rule, [772–774](#)

MAC (multiply and accumulate) architecture, [773](#)

Polynomial factoring, CIC filters, [765–770](#)

Polynomials, finding the roots of, [372](#)

Polyphase decomposition

CIC filters, [765–770](#)

definition, [526](#)

diagrams, [538–539](#)

two-stage decimation, [514](#)

Polyphase filters

benefits of, [539](#)

commutator model, [524](#)

implementing, [535–540](#)

issues with, [526](#)

noble identities, [536](#)

order, swapping, [536–537](#)

overview, [522–528](#)

polyphase decomposition, [526](#), [538–539](#)

prototype FIR filters, [522](#)

uses for, [522](#)

Power, signal. *See also* [Decibels](#).

absolute, [891–892](#)

definition, [9](#)

relative, [885–889](#)

Power spectrum, [63](#), [140–141](#)

Preconditioning FIR filters, [563–566](#)

Prewarp, [329](#)

Prime decomposition, CIC filters, [768–770](#)

Prime factorization, CIC filters, [768–770](#)

Probability density function (PDF). *See* [PDF \(probability density function\)](#).

Processing gain or loss. *See* [DFT processing gain](#); [Gain](#); [Loss](#).

Prototype filters

analog, [303](#)

FIR polyphase filters, [522](#)

IFIR filters, [382](#)

Q

- Q30 fixed-point binary formats, [629](#)
- Q-channel filters, analytic signals, [496](#)
- Quadratic factorization formula, [266](#), [282](#)
- Quadrature component, [454–455](#)
- Quadrature demodulation, [455](#), [456–462](#)
- Quadrature filters, definition, [900](#)
- Quadrature mixing, [455](#)
- Quadrature oscillation, [459–462](#)
- Quadrature oscillator
 - coupled, [787](#)
 - overview, [786–789](#)
 - Taylor series approximation, [788](#)
- Quadrature phase, [440](#)
- Quadrature processing, [440](#)
- Quadrature sampling block diagram, [459–462](#)
- Quadrature signals. *See also* [Complex numbers](#).
 - analytic, [455](#)
 - Argand plane, [440–441](#)
 - bandpass signals in the frequency-domain, [454–455](#)
 - Cartesian form, [442](#)
 - complex exponentials, [447](#)
 - complex mixing, [455](#)
 - complex number notation, [440–446](#)
 - complex phasors, [446–450](#)
 - complex plane, [440–441](#), [446](#)
 - decimation, in frequency translation, [781–783](#)
 - definition, [439](#)
 - demodulation, [453–454](#)
 - detection, [453–454](#)
 - down-conversion. *See* [Down-conversion, quadrature signals](#).
 - Euler's identity, [442–443](#), [449](#), [453](#)
 - exponential form, [442](#)
 - in the frequency domain, [451–454](#)
 - generating from real signals. *See* [Hilbert transforms](#).
 - generation, [453–454](#)
 - imaginary part, [440](#), [454–455](#)
 - in-phase component, [440](#), [454–455](#)
 - I/Q demodulation, [459–462](#)
 - j -operator, [439](#), [444–450](#)
 - magnitude-angle form, [442](#)

- mixing to baseband, [455](#)
- modulation, [453–454](#)
- negative frequency, [450–451](#)
- orthogonality, [448](#)
- polar form, [442](#), [443–444](#)
- positive frequency, [451](#)
- real axis, [440](#)
- real part, [440](#), [454–455](#)
- rectangular form, [442](#)
- representing real signals, [446–450](#)
- sampling scheme, advantages of, [459–462](#)
- simplifying mathematical analysis, [443–444](#)
- three-dimensional frequency-domain representation, [451–454](#)
- trigonometric form, [442](#), [444](#)
- uses for, [439–440](#)

Quantization

- coefficient/errors, [293–295](#)
- noise. *See* [A/D converters](#), [quantization noise](#).
- real-time DC removal, [763–765](#)

R

- Radix points, fixed-point binary formats, [629](#)

Radix-2 algorithm, FFT

- butterfly structures, [151–154](#)
- computing large DFTs, [826–829](#)
- decimation-in-frequency algorithms, [151–154](#)
- decimation-in-time algorithms, [151–154](#)
- derivation of, [141–149](#)
- FFT (fast Fourier transform), [151–158](#)
- twiddle factors, [143–149](#)

- Raised cosine windows. *See* [Hanning windows](#).

Random data

- Central Limit Theory, [723](#)
- generating a normal distribution of, [722–724](#)

- Random functions, mean and variance, [879–882](#)

- Random noise, [868](#). *See also* [SNR \(signal-to-noise ratio\)](#).

Real numbers

- definition, [440](#)
- graphical representation of, [847–848](#)

- Real sampling, [46](#)

Real signals

- bandpass sampling, [46](#)

- decimation, in frequency translation, [781](#)

- generating complex signals from. *See* [Hilbert transforms](#).

- representing with quadrature signals, [446–450](#)

Rectangular form of complex numbers

- definition, [848–850](#)

- vs. polar form, [856–857](#)

Rectangular form of quadrature signals, [442](#)

Rectangular functions

- all ones, [115–118](#)

- DFT, [105–112](#)

- frequency axis, [118–120](#)

- general, [106–112](#)

- overview, [105–106](#)

- symmetrical, [112–115](#)

- time axis, [118–120](#)

Rectangular windows, [89–97](#), [686](#)

Recursive filters. *See* [IIR filters](#)

Recursive moving averagers, [606–608](#)

Recursive running sum filters, [551–552](#)

Remez Exchange, [204–207](#), [418](#)

Replications, spectral. *See* [Spectral replications](#).

Resolution, DFT, [77](#), [98–102](#)

Ripples

- in Bessel-derived filters, [901](#)

- in Butterworth-derived filters, [901](#)

- in Chebyshev-derived filters, [900](#)

- definition, [900–901](#)

- designing FIR filters, [190–194](#)

- in Elliptic-derived filters, [900](#)

- equiripple, [418](#), [901](#)

- out-of-band, [901](#)

- in the passband, [900](#)

- in the stopband, [901](#)

rms value of continuous sinewaves, [874–875](#)

Roll-off, definition, [901](#)

Roots of

- complex numbers, [853–854](#)

- polynomials, [372](#)

Rosetta Stone, [450](#)

Rounding fixed-point binary numbers

- convergent rounding, [651](#)

- data rounding, [649–652](#)

- effective bits, [641](#)

- round off noise, [636–637](#)

- round to even method, [651](#)

- round-to-nearest method, [650–651](#)

Roundoff errors, [293](#)

S

Sample rate conversion. *See also* [Polyphase filters](#).

- decreasing. *See* [Decimation](#).

- definition, [507](#)

- with IFIR filters, [548–550](#)

- increasing. *See* [Interpolation](#).

- missing data, recovering, [823–826](#). *See also* [Interpolation](#).

- by rational factors, [540–543](#)

Sample rate conversion, multirate systems

- filter mathematical notation, [534–535](#)

- signal mathematical notation, [533–534](#)

- z-transform analysis, [533–535](#)

Sample rate conversion, with half-band filters

- folded FIR filters, [548](#)

- fundamentals, [544–546](#)

- implementation, [546–548](#)

- overview, [543](#)

Sample rate converters, [521–522](#)

Sampling, periodic. *See* [Periodic sampling](#).

Sampling translation, [44](#)

Sampling with digital mixing, [462–464](#)

Scaling IIR filter gain, [300–302](#)

Scalloping loss, [96–97](#)

SDFT (sliding DFT)

- algorithm, [742–746](#)

- overview, [741](#)

- stability, [746–747](#)

SFDR (spurious free dynamic range), [714–715](#)

Shannon, Claude, [42](#)

Shape factor, [901](#)

- Sharpened FIR filters, [726–728](#)
- Shifting theorem, DFT, [77–78](#)
- Shift-invariant systems. *See* [Time-invariant systems](#).
- Sidelobe magnitudes, [110–111](#)
- Sidelobes
 - Blackman window and, [194–197](#)
 - DFT leakage, [83](#), [89](#)
 - FIR (finite impulse response) filters, [184](#)
 - ripples, in low-pass FIR filters, [193–194](#)
- Sign extend operations, [627](#)
- Signal averaging. *See also* [SNR \(signal-to-noise ratio\)](#).
 - equation, [589](#)
 - frequency-domain. *See* [Signal averaging, incoherent](#).
 - integration gain, [600–603](#)
 - mathematics, [592–594](#), [599](#)
 - multiple FFTs, [600–603](#)
 - phase angles, [603–604](#)
 - postdetection. *See* [Signal averaging, incoherent](#).
 - quantifying noise reduction, [594–597](#)
 - rms. *See* [Signal averaging, incoherent](#).
 - scalar. *See* [Signal averaging, incoherent](#).
 - standard deviation, [590](#)
 - time-domain. *See* [Signal averaging, coherent](#).
 - time-synchronous. *See* [Signal averaging, coherent](#).
 - variance, [589–590](#)
 - video. *See* [Signal averaging, incoherent](#).
- Signal averaging, coherent
 - exponential averagers, [608–612](#)
 - exponential moving averages, computing, [801–802](#)
 - exponential smoothing, [608](#)
 - filtering aspects, [604–608](#)
 - moving averagers, [604–608](#)
 - moving averages, computing, [799–801](#)
 - nonrecursive moving averagers, [606–608](#)
 - overview, [590–597](#)
 - recursive moving averagers, [606–608](#)
 - reducing measurement uncertainty, [593](#), [604–608](#)
 - time-domain filters, [609–612](#)
 - true signal level, [604–608](#)
 - weighting factors, [608](#), [789](#)

Signal averaging, exponential

- 1st-order IIR filters, [612–614](#)

- dual-mode technique, [791](#)

- example, [614](#)

- exponential smoothing, [608](#)

- frequency-domain filters, [612–614](#)

- moving average, computing, [801–802](#)

- multiplier-free technique, [790–791](#)

- overview, [608](#)

- single-multiply technique, [789–790](#)

Signal averaging, incoherent

- 1st-order IIR filters, [612–614](#)

- example, [614](#)

- frequency-domain filters, [612–614](#)

- overview, [597–599](#)

Signal averaging, with FIR filters

- convolution, [175–176](#)

- example, [170–174](#), [183–184](#)

- as a lowpass filter, [180–182](#)

- performance improvement, [178](#)

Signal envelope, Hilbert transforms, [483–495](#)

Signal power. *See also* [Decibels](#).

- absolute, [891–892](#)

- relative, [885–889](#)

Signal processing

- analog, [2](#). *See also* [Continuous signals](#).

- definition, [2](#)

- digital, [2](#)

- operational symbols, [10–11](#)

Signal transition detection, [820–821](#)

Signal variance

- biased and unbiased, computing, [797–799](#), [799–801](#)

- definition, [868–870](#)

- exponential, computing, [801–802](#)

- PDF (probability density function), [879–882](#)

- of random functions, [879–882](#)

- signal averaging, [589–590](#)

Signal-power-to-noise-power ratio (SNR), maximizing, [376](#)

Signal-to-noise ratio (SNR). *See* [SNR \(signal-to-noise ratio\)](#).

Sign-magnitude, fixed-point binary formats, [625–626](#)

- Simpson, Thomas, [372](#)
- SINAD (signal-to-noise-and-distortion), [711–714](#)
- Sinc filters. *See* [CIC \(cascaded integrator-comb\) filters](#).
- Sinc functions, [83](#), [89](#), [116](#)
- Single tone detection, FFT method
 - drawbacks, [737–738](#)
 - vs. Goertzel algorithm, [740–741](#)
- Single tone detection, Goertzel algorithm
 - advantages of, [739](#)
 - description, [738–740](#)
 - example, [740](#)
 - vs. the FFT, [740–741](#)
 - stability, [838–840](#)
- Single tone detection, spectrum analysis, [737–741](#)
- Single-decimation down-conversion, [819–820](#)
- Single-multiply technique, exponential signal averaging, [789–790](#)
- Single-stage decimation, vs. two-stage, [514](#)
- Single-stage interpolation, vs. two-stage, [532](#)
- Sliding DFT (SDFT). *See* [SDFT \(sliding DFT\)](#).
- Slope detection, [820–821](#)
- Smoothing impulsive noise, [770–772](#)
- SNDR. *See* [SINAD \(signal-to-noise-and-distortion\)](#).
- SNR (signal-to-noise ratio)
 - vs. A/D converter, fixed-point binary finite word lengths, [640–642](#)
 - A/D converters, [711–714](#)
 - bandpass sampling, [48–49](#)
 - block averaging, [770](#)
 - corrected mean, [771](#)
 - DFT processing gain, [103–104](#)
 - IIR filters, [302](#)
 - measuring. *See* [Statistical measures of noise](#).
 - reducing. *See* [Signal averaging](#).
 - smoothing impulsive noise, [770–772](#)
- SNR (signal-power-to-noise-power ratio), maximizing, [376](#)
- Software programs, fast Fourier transform, [141](#)
- Someya, I., [42](#)
- Spectral inversion
 - around signal center frequency, [821–823](#)
 - bandpass sampling, [46–47](#)
- Spectral leakage, FFTs, [138–139](#), [683–686](#). *See also* [DFT leakage](#).

Spectral leakage reduction

A/D converter testing techniques, [710–711](#)

Blackman windows, [686](#)

frequency domain, [683–686](#)

Spectral peak location

estimating, algorithm for, [730–734](#)

Hamming windows, [733](#)

Hanning windows, [733](#)

Spectral replications

bandpass sampling, [44–45](#)

sampling lowpass signals, [39–40](#)

Spectral vernier. *See* [Zoom FFT](#).

Spectrum analysis. *See also* [SDFT \(sliding DFT\)](#); [Zoom FFT](#).

center frequencies, expanding, [748–749](#)

with SDFT (sliding DFT), [748–749](#)

single tone detection, [737–741](#)

weighted overlap-add, [755](#)

windowed-presum FFT, [755](#)

Zoom FFT, [749–753](#)

Spectrum analyzer, [753–756](#)

Spurious free dynamic range (SFDR), [714–715](#)

Stability

comb filters, [403–404](#)

conditional, [268](#)

FSF (frequency sampling filters), [403–406](#)

IIR filters, [263–270](#)

Laplace transfer function, [263–264](#), [268](#)

Laplace transform, [263–270](#)

SDFT (sliding DFT), [746–747](#)

single tone detection, [838–840](#)

z-transform and, [272–274](#), [277](#)

Stair-step effect, A/D converter quantization noise, [637](#)

Standard deviation

of continuous sinewaves, [874–875](#)

definition, [870](#)

signal averaging, [590](#)

Statistical measures of noise

average, [868–870](#)

average power in electrical circuits, [874–875](#)

Bessel's correction, [870–871](#)

- biased estimates, [870–871](#)
- dispersion, [869](#)
- fluctuations around the average, [869](#)
- overview, [867–870](#). *See also* [SNR \(signal-to-noise ratio\)](#).
- of real-valued sequences, [874](#)
- rms value of continuous sinewaves, [874–875](#)
- of short sequences, [870–871](#)
- standard deviation, definition, [870](#)
- standard deviation, of continuous sinewaves, [874–875](#)
- summed sequences, [872–874](#)
- unbiased estimates, [871](#)

Statistical measures of noise, estimating SNR

- for common devices, [876](#)
- controlling SNR test signals, [879](#)
- in the frequency domain, [877–879](#)
- overview, [875–876](#)
- in the time domain, [876–877](#)

Statistical measures of noise, mean

- definition, [868–869](#)
- PDF (probability density function), [879–882](#)
- of random functions, [879–882](#)

Statistical measures of noise, variance. *See also* [Signal variance](#).

- definition, [868–870](#)
- PDF (probability density function), [879–882](#)
- of random functions, [879–882](#)

Steinmetz, Charles P., [446](#)

Stockham, Thomas, [716](#)

Stopband, definition, [901](#)

Stopband ripples

- definition, [901](#)
- minimizing, [204–207](#)

Stopband sidelobe level suppression, [416](#)

Structure, definition, [901](#)

Structures, IIR filters

- biquad filters, [299](#)
- cascade filter properties, [295–297](#)
- cascaded, [295–299](#)
- cascade/parallel combinations, [295–297](#)
- changing, [291–292](#)
- Direct Form 1, [275–278](#), [289](#)

Direct Form II, [289–292](#)

optimizing partitioning, [297–299](#)

parallel filter properties, [295–297](#)

transposed, [291–292](#)

transposed Direct Form II, [289–290](#)

transposition theorem, [291–292](#)

Sub-Nyquist sampling. *See* [Bandpass sampling](#).

Substructure sharing, [765–770](#)

Subtraction

block diagram symbol, [10](#)

complex numbers, [850](#)

Summation

block diagram symbol, [10](#)

description, [11](#)

equation, [10](#)

notation, [11](#)

Symbols

block diagram, [10–11](#)

signal processing, [10–11](#)

Symmetrical rectangular functions, [112–115](#)

Symmetrical-coefficient FIR filters, [232–233](#)

Symmetry, DFT, [73–75](#)

T

Tacoma Narrows Bridge collapse, [263](#)

Tap, definition, [901](#)

Tap weights. *See* [Filter coefficients](#).

Tapped delay, FIR filters, [174](#), [181–182](#)

Taylor series approximation, [788](#)

Tchebyshev function, definition, [902](#)

Tchebyshev windows, in FIR filter design, [197](#)

Time data, manipulating in FFTs, [138–139](#)

Time invariance, decimation, [514](#)

Time properties

decimation, [514–515](#)

interpolation, [519](#)

Time representation, continuous vs. discrete systems, [5](#)

Time reversal, [863–865](#)

Time sequences, notation syntax, [7](#)

Time-domain

aliasing, avoiding, [718–722](#)

analytic signals, generating, [495–497](#)

coefficients, determining, [186–194](#)

convolution, matched filters, [380](#)

convolution vs. frequency-domain multiplication, [191–194](#)

equations, example, [7](#)

FIR filter implementation, [489–494](#)

Hilbert transforms, designing, [489–494](#)

interpolation, [778–781](#)

slope filters, [820–821](#)

Time-domain data, converting

from frequency-domain data. *See* [IDFT \(inverse discrete Fourier transform\)](#).

to frequency-domain data. *See* [DFT \(discrete Fourier transform\)](#).

Time-domain filters

coherent signal averaging, [609–612](#)

exponential signal averaging, [609–612](#)

Time-domain signals

amplitude, determining, [140](#)

continuous, Laplace transform for, [258](#)

DC removal, [812–815](#)

definition, [4](#)

vs. frequency-domain, [120–123](#)

Time-invariant systems. *See also* [LTI \(linear time-invariant\) systems](#).

analyzing, [19–21](#)

commutative property, [18–19](#)

definition, [17–18](#)

example of, [17–18](#)

Tone detection. *See* [Single tone detection](#).

Transfer functions. *See also* [Laplace transfer function](#).

definition, [902](#)

real FSF, [908–909](#)

z-domain, [282–289](#)

Transient response, FIR filters, [181–182](#)

Transition region, definition, [902](#)

Translation, sampling, [44](#)

Transposed Direct Form II filters, [289–290](#)

Transposed Direct Form II structure, [289–290](#)

Transposed filters, [291–292](#)

Transposed structures, [765–770](#)

Transposition theorem, [291–292](#)

Transversal filters, [173–174](#). *See also* [FIR \(finite impulse response\) filters](#).

Triangular dither, [708](#)

Triangular windows, [89–93](#)

Trigonometric form, quadrature signals, [442](#), [444](#)

Trigonometric form of complex numbers, [848–850](#)

Truncation, fixed-point binary numbers, [646–649](#)

Tukey, J., [135](#)

Two's complement

fixed-point binary formats, [626–627](#), [629](#)

overflow, [559–563](#)

Two-sided Laplace transform, [258](#)

Type-IV FSF

examples, [419–420](#), [423–426](#)

frequency response, [910–912](#)

optimum transition coefficients, [913–926](#)

U

Unbiased estimates, [871](#)

Unbiased signal variance, computing, [797–799](#), [799–801](#)

Undersampling lowpass signals, [40](#). *See also* [Bandpass sampling](#).

Uniform windows. *See* [Rectangular windows](#).

Unit circles

definition, [271](#)

z -transform, [271](#)

Unit circles, FSF

forcing poles and zeros inside, [405](#)

pole / zero cancellation, [395–398](#)

Unit delay

block diagram symbol, [10](#)

description, [11](#)

Unit impulse response, LTI, [19–20](#)

Unnormalized fractions, floating-point binary formats, [656](#)

Unwrapping, phase, [210](#)

Upsampling, interpolation, [517–518](#), [520–521](#)

V

Variance. *See* [Signal variance](#).

Vector, definition, [848](#)

Vector rotation with arctangents

to the 1st octant, [805–808](#)

division by zero, avoiding, [808](#)

jump address index bits, [807](#)

overview, [805](#)

by $\pm\pi/8$, [809–810](#)

rotational symmetries, [807](#)

Vector-magnitude approximation, [679–683](#)

von Hann windows. *See* [Hanning windows](#).

W

Warping, frequency, [319](#), [321–325](#), [328–330](#)

Weighted overlap-add spectrum analysis, [755](#)

Weighting factors, coherent signal averaging, [608](#), [789](#)

Wideband compensation, [564](#)

Wideband differentiators, [367–370](#)

Willson, A., [386](#)

Window design method, FIR filters, [186–194](#)

Windowed-presum FFT spectrum analysis, [755](#)

Windows

Blackman, [195–201](#), [686](#), [733](#)

Blackman-Harris, [686](#), [733](#)

exact Blackman, [686](#)

FFTs, [139](#)

in the frequency domain, [683–686](#)

magnitude response, [92–93](#)

mathematical expressions of, [91](#)

minimizing DFT leakage, [89–97](#)

processing gain or loss, [92](#)

purpose of, [96](#)

rectangular, [89–97](#), [686](#)

selecting, [96](#)

triangular, [89–93](#)

Windows, Hamming

description, [89–93](#)

DFT leakage reduction, [89–93](#)

in the frequency domain, [683–686](#)

spectral peak location, [733](#)

Windows, Hanning

description, [89–97](#)

DFT leakage, minimizing, [89–97](#)

in the frequency domain, [683–686](#)

spectral peak location, [733](#)

Windows used in FIR filter design

- Bessel functions, [198–199](#)

- Blackman, [195–201](#)

- Chebyshev, [197–201](#), [927–930](#)

- choosing, [199–201](#)

- Dolph-Chebyshev, [197](#)

- Kaiser, [197–201](#)

- Kaiser-Bessel, [197](#)

- Tchebyshev, [197](#)

Wingless butterflies, [156](#)

Wraparound leakage, [86–88](#)

Wrapping, phase, [209](#), [900](#)

Z

z-domain expression for Mth-order IIR filter, [275–276](#)

z-domain transfer function, IIR filters, [282–289](#)

Zero padding

- alleviating scalloping loss, [97–102](#)

- FFTs, [138–139](#)

- FIR filters, [228–230](#)

- improving DFT frequency granularity, [97–102](#)

- spectral peak location, [731](#)

Zero stuffing

- interpolation, [518](#)

- narrowband lowpass filters, [834–836](#)

Zero-overhead looping

- DSP chips, [333](#)

- FSF (frequency sampling filters), [422–423](#)

- IFIR filters, [389](#)

Zero-phase filters

- definition, [902](#)

- techniques, [725](#)

Zeros

- IIR filters, [284–289](#)

- on the s-plane, Laplace transform, [263–270](#)

Zoom FFT, [749–753](#)

Zoom FFT, [749–753](#)

z-plane pole / zero properties, IIR filters, [288–289](#)

z-transform. *See also* [Laplace transform](#).

- definition, [270](#)

description of, [270–272](#)

FIR filters, [288–289](#)

IIR filters, [270–282](#)

infinite impulse response, definition, [280](#)

polar form, [271](#)

poles, [272–274](#)

unit circles, [271](#)

zeros, [272–274](#)

z-transform, analyzing IIR filters

digital filter stability, [272–274](#), [277](#)

Direct Form 1 structure, [275–278](#)

example, [278–282](#)

frequency response, [277–278](#)

overview, [274–275](#)

time delay, [274–278](#)

z-domain transfer function, [275–278](#), [279–280](#)