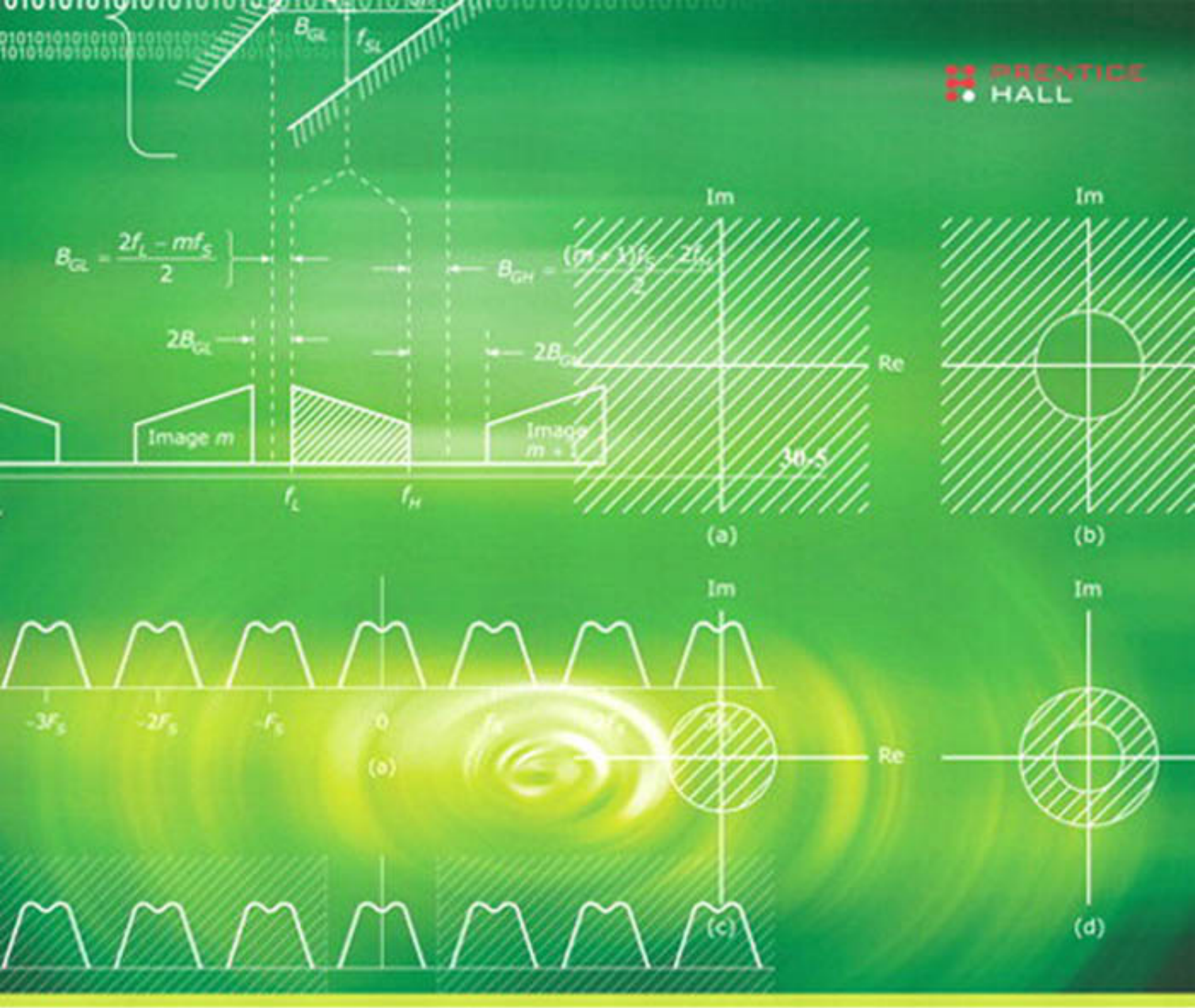


NOTES ON DIGITAL SIGNAL PROCESSING

Practical Recipes for Design, Analysis, and Implementation

C. BRITTON RORABAUGH

Notes on Digital Signal Processing

This page intentionally left blank

Notes on Digital Signal Processing

Practical Recipes for Design, Analysis, and Implementation

C. Britton Rorabaugh

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

Library of Congress Cataloging-in-Publication Data

Rorabaugh, C. Britton.
Notes on digital signal processing : practical recipes for design, analysis, and implementation / C. Britton Rorabaugh.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-158334-4 (hardcover : alk. paper)
1. Signal processing—Digital techniques. I. Title.
TK5102.9.R673 2011
621.382'2--dc22

2010040954

ISBN-13: 978-0-13-158334-4
ISBN-10: 0-13-158334-4

Text printed in the United States on recycled paper at Edwards Brothers in Ann Arbor, Michigan
First printing, November 2010

To Joyce

This page intentionally left blank

Contents

Part III Fast Fourier Transform Techniques

Part IV Window Techniques

Part V Classical Spectrum Analysis

Part VI FIR Filter Design

Part V Analog Prototype Filters

Part VI *z*-Transform Analysis

Part VII IIR Filter Design

Part VIII Multirate Signal Processing

Part IX Bandpass and Quadrature Techniques

Part X Statistical Signal Processing

Preface

Standard advice for writing a preface tells the author to begin by answering the question, "Why did you write this book?" The published answers almost always include an explanation of how something is still missing in the already vast body of existing literature, and how the book in question represents a valiant attempt to fill the void at least partially.

This book is no exception. There still is a dearth of *good* collections of step-by-step procedures, or *recipes,* for design and implementation of anything beyond just the most elementary DSP procedures. This book *is* an attempt to fill this void—at least partially. However, the tagline for this book is most definitely not meant to be, "Get a result without really gaining much understanding along the way." Here, the focus is clearly on the recipes, but supporting explanations and mathematical material are also provided. This supporting material is set off in such a way so that it is easily bypassed if the reader so desires.

This book provides an opportunity to delve deeper into the nuances of certain interesting topics within DSP. A good alternative title might be *Exploring the Nooks and Crannies of Digital Signal Processing*. As with all books, every reader will not resonate with every topic, but I'm confident that each reader will share an interest in a large subset of the topics presented.

Note 1, Navigating the DSP Landscape, provides diagrams that map the relationships among all the book's various topics. One diagram is dedicated to processing techniques that operate on real-valued digital signals to modify in some way the properties of those signals while leaving their fundamental real-valued and digital natures intact. A second diagram is dedicated to processing techniques that are concerned primarily with conversion between real-valued digital signals and other entities such as analog signals, complex-valued signals, and estimated spectra.

Many of the Notes include examples that demonstrate an actual application of the technique being presented. Most sections use MATLAB tools for routine tasks such as designing the digital filters that are used in the reference designs. When appropriate, the

use of these tools is discussed in the text. The results provided in Note 66, Generating I and Q Channels Digitally: Generalization of Rader's Approach, were generated by a modified version of the `PracSim` simulation package that is described in *Simulating Wireless Communication Systems* (Prentice Hall, 2004). However, the filter coefficients used in the simulation were generated using MATLAB. The examples make heavy use of MATLAB as a convenience. However, it is not my intent to make this a MATLAB "workbook" with projects and exercises that *require* the reader to use MATLAB, because I want the book to remain useful and attractive to readers who do not have access to MATLAB. The m-files for the MATLAB programs discussed in the book, as well as for programs used to generate some of the illustrations, can be found at the websitewww.informit.com/ph.

This book is not the best choice for a first book from which to learn DSP if you're starting from scratch. For this task, I recommend *Understanding Digital Signal Processing* by Richard Lyons (Prentice Hall, 2004). This book is, however, a good *N*+1st book for anyone—from novice to expert—with an interest in DSP. Its contents comprise an assortment of interesting tidbits, unique insights, alternative viewpoints, and rarely published techniques. The following are some examples.

- The set of five techniques for generating analytic signals presented in Notes 60 through 64 do not appear together in any single text.
- The visualization techniques used in Note 22 probably are not discussed anywhere else, because I came up with them while writing this book. These techniques follow directly from first principles, but I've never seen them explicitly presented elsewhere.
- Natural sampling, as discussed in Note 6, usually can be found only in older texts that cover traditional (that is, analog) communication theory.

My overarching goal was to write an easy-to-read book loaded with easy-to-access information and easy-to-use recipes. I hope I have succeeded.

About the Author

C. BRITTON RORABAUGH has a B.S. and M.S. in electrical engineering from Drexel University and currently holds the position of Chief Scientist for a company that develops and manufactures specialized military communications equipment. He is an associate editor for *IEEE Signal Processing Magazine*, and the author of several publications on topics such as wireless communications, DSP, digital filters, and error coding, with experience in communication system design and analysis, geolocation, radar processing, real-time software, numerical methods, computer graphics, C++, C, MATLAB, and assembly languages for various microprocessors and DSP devices.

This page intentionally left blank

Note 1

Navigating the DSP Landscape

Digital signal processing (DSP) is based on the notions that an analog signal can be digitized and that mathematical operations can effectively take the place of (or even surpass) electronic manipulations performed on the original analog signal. In the earliest days of DSP, its applications were limited to sonar and seismology because these fields utilized low-bandwidth signals that could be sampled at adequate rates using the available technology. As digital processing circuits and analog-to-digital converters have become faster and faster, the number of applications for DSP has exploded.

Hundreds of techniques (and variations thereof) are used in DSP, and it can be difficult to see the big picture—how all these various techniques relate to each other and to a particular application at hand. Rather than a comprehensive treatment of all the possible topics within DSP, this book is an attempt to document in-depth explorations of some of the "nooks and crannies" in DSP that often are glossed over in traditional texts. Figures 1.1 and 1.2 show diagramatically the realtionships among the various processing techniques explored in this book. The topic areas are arbitrarily split into two groups

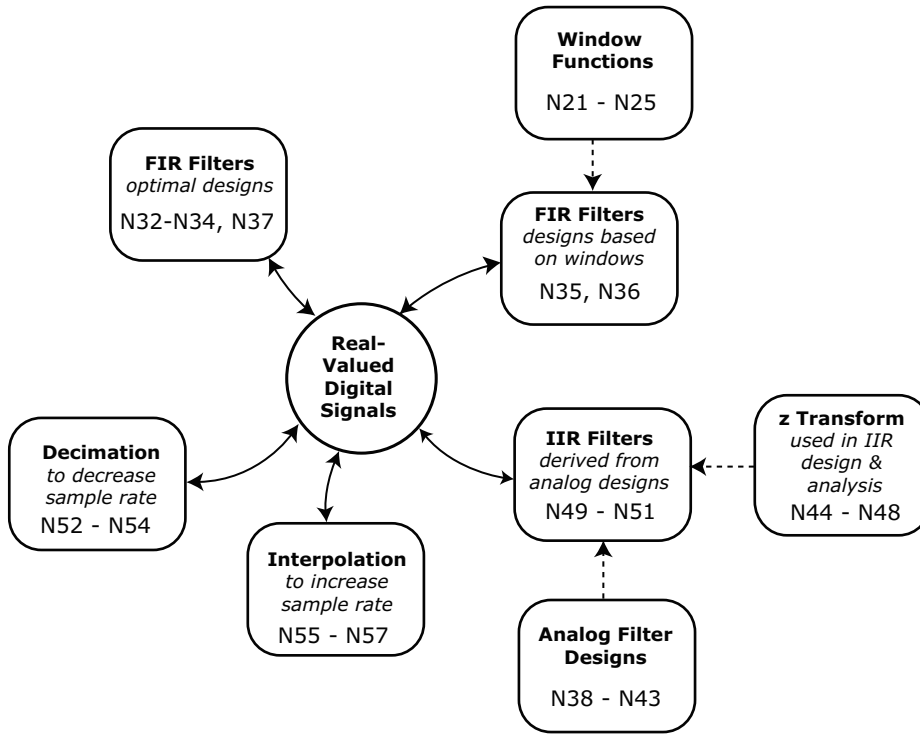


Figure 1.1 Processing techniques that modify the properties of real-valued digital signals. The numbers "Nnn" indicate the Notes in which each technique is discussed. Solid paths indicate "run-time" data connections. Dashed paths indicate "design-time" connections.

for organizational purposes. Figure 1.1 shows those techniques that are concerned primarily with operating on real-valued digital signals to modify in some way the properties of those signals while leaving their fundamental real-valued and digital natures intact. Given that complex-valued signals are really just quadrature pairs of real-valued signals, most of these techniques are easily extended to corresponding complex cases. Figure 1.2 shows those techniques that are concerned primarily with conversion between real-valued digital signals and other entities such as analog signals, complex-valued signals, and spectrum estimates.

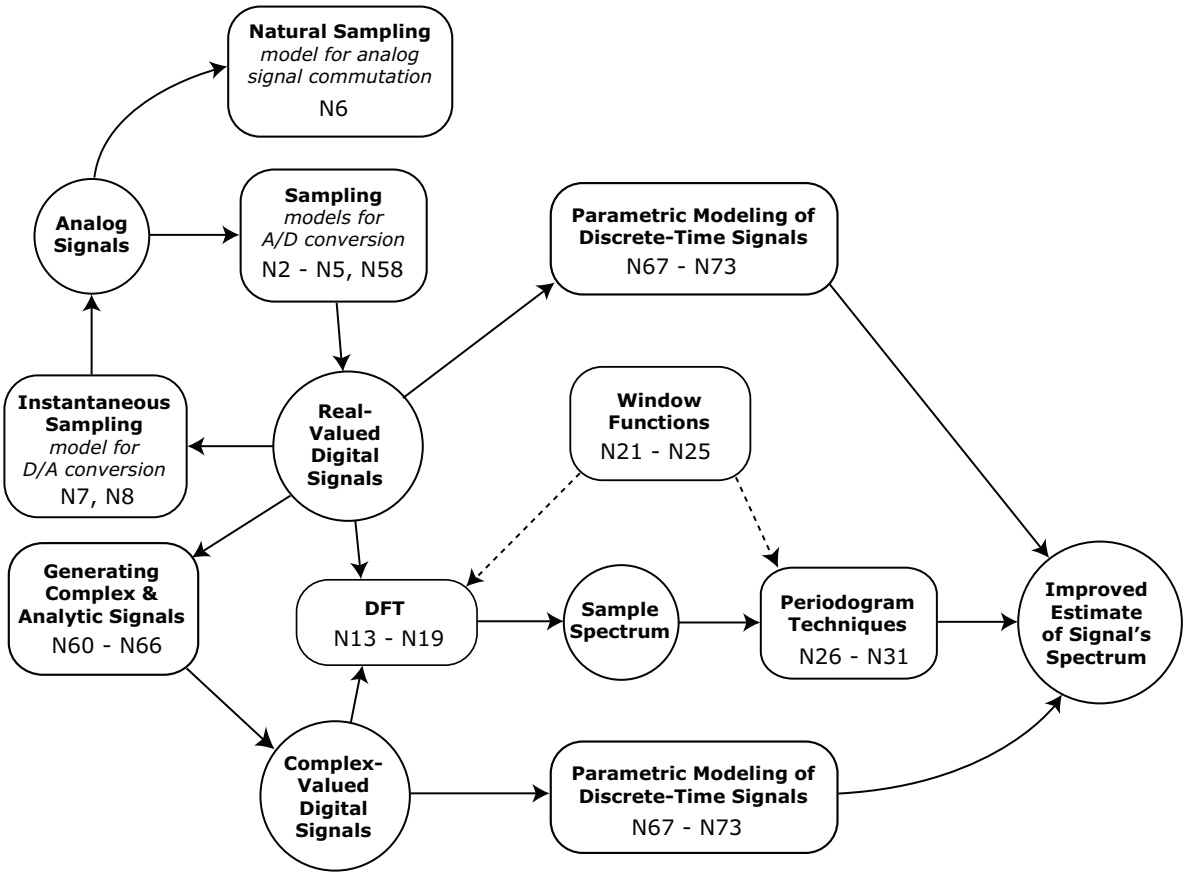


Figure 1.2 Processing techniques that convert real-valued digital signals to or from other things such as analog signals, complex-valued digital signals, or spectral estimates. The numbers "Nnn" indicate the Notes in which each technique is discussed. Solid paths indicate "run-time" data connections. Dashed paths indicate "design-time" connections.

Note 2

Overview of Sampling Techniques

This note discusses the difference between *implicit* and *explicit* sampling. It introduces three different mathematical models of explicit sampling techniques—*ideal* sampling, *natural* sampling, and *instantaneous* sampling.

Digitization of an analog signal is the one process above all others that makes DSP such a useful technology. If it were limited to working with only those signals that orginate in digital form, DSP would be just an academic curiosity. Digitization actually comprises two distinct operations: sampling and quantization, which are usually analyzed separately.

2.1 Implicit Sampling Techniques

In *implicit sampling*, a sample measurement is triggered by the signal attaining some specified value or crossing some specified threshold. Recording the times at which zero-crossings occur in a bipolar signal is an example of implict sampling.

2.2 Explicit Sampling Techniques

Unlike implict sampling, in which samples are triggered by some aspect of signal behavior, in *explicit sampling*, signal values are measured at specified times without regard to the signal's behavior. Consider the continuous-time sinusoidal signal and its two-sided magnitude spectrum depicted in Figure 2.1. There are three explicit sampling techniques—*natural sampling*, *instantaneous sampling*, and *ideal sampling*—that can be used to sample such a signal. The results produced by these techniques, and the corresponding impacts on the signal's spectrum are compared in Key Concept 2.1.

Ideal Sampling

As depicted in Key Concept 2.1, zero-width samples take on instantaneous values of the analog signal. Neglecting quantization and timing errors, the sequence of values produced by an analog-to-digital converter can be modeled as the output of

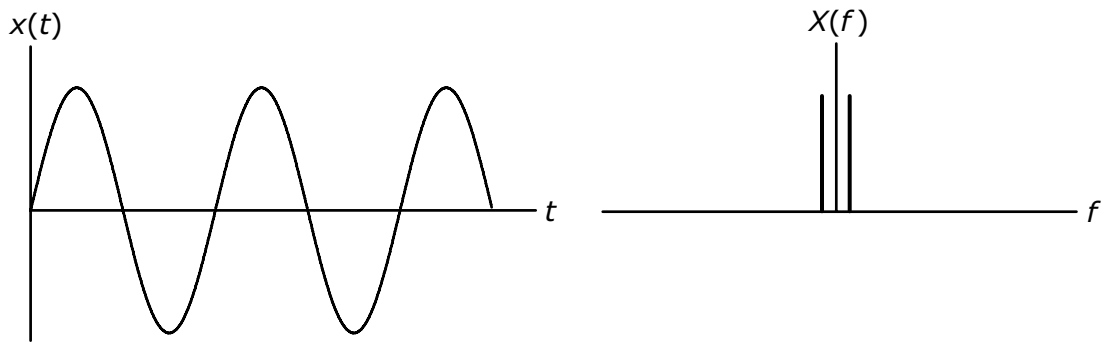


Figure 2.1 Continuous-time sinusoid and its two-sided magnitude spectrum

an ideal sampling process. Ideal sampling is discussed further in Note 3.

Natural Sampling

Nonzero-width samples have time-varying amplitudes that follow the contours of the analog signal, as shown in Key Concept 2.1. Commutator systems for time-division multiplexing of telegraph signals, first proposed in 1848, used an approximation to natural sampling. The sample pulses were created by gating a signal with rotating mechanical contacts. This multiplexing technique was subsequently applied to telephone signals in 1891. It was the application of natural sampling to telephony that first led to consideration of just how rapidly a continuous-time signal needed to be sampled in order to preserve fidelity and ensure the ability to reconstruct exactly the original, unsampled signal. Natural sampling is explored further in Note 6.

Instantaneous Sampling

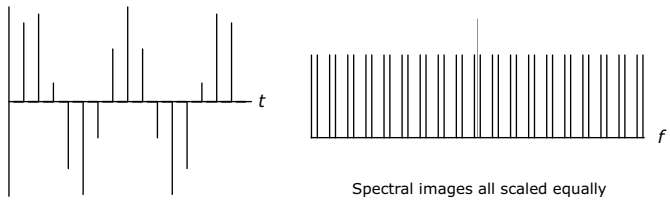
In instantaneous sampling, nonzero-width samples each have a constant amplitude that corresponds to the instantaneous value of the analog signal at the beginning of the sample. The sample values are held constant long enough to create flat-topped sample pulses. The output of a digital-to-analog converter (DAC) can be modeled as the output of an instantaneous sampling process, often as the limiting case in which the sample width equals the sampling interval. As discussed in Note 7, the results of the instantaneous sampling model play a key role in the specification of the analog filter used to smooth the DAC output.

Key Concept 2.1

Explicit Sampling Techniques

Ideal Sampling

Models the output of an *A/D converter*

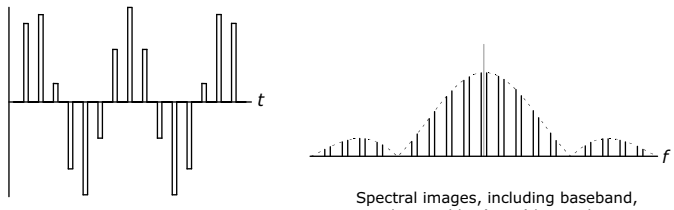


Zero width samples take on instantaneous values of the analog signal.

Spectral images all scaled equally

Instantaneous Sampling

Models the output of a *sample and hold amplifier*, *zero order data hold*, or *D/A converter*

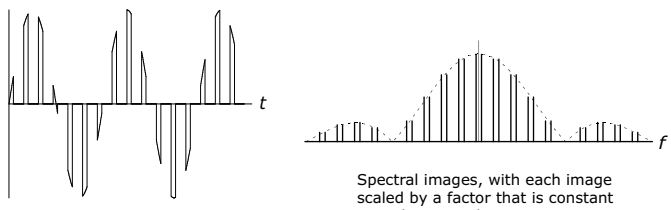


Non zero width samples, each with amplitude held constant over the width of the pulse

Spectral images, including baseband, are distorted by (sin *x*)/*x* envelope.

Natural Sampling

Models the output of an *analog demultiplexer*



Non zero width samples with time varying amplitudes that follow the contours of the analog signal

Spectral images, with each image scaled by a factor that is constant over the image but varies image to image

Reference

1. H. D. Lüke, "The Origins of the Sampling Theorem," *IEEE Communications,* April 1999, pp. 106–108.

Note 3

Ideal Sampling

The concept of ideal sampling can be viewed as the foundation upon which the rest of DSP is built. The simplest way to view ideal sampling is as a mapping process, such as the one depicted in Figure 3.1, that "grabs" uniformly spaced instantaneous values of the continuous-time function, $x(t)$, and uses these values to construct the discrete-time sequence, $x[n]$:

$$x[n] \leftarrow x(nT) \tag{3.1}$$

where n is the is the integer-valued sample index and T is the real-valued sampling interval. In order to depict ideal sampling in a block diagram, some authors adopt the concept of an *ideal sampler,* which is usually depicted as a time-driven switch, as in Figure 3.2(a). Oppenheim and Schafer [1] introduce a more elegant depiction in the form of an ideal continuous-to-discrete (C/D) converter, as shown in Figure 3.2(b).

Without some augmentation, the ideal sampler and C/D converter concepts are both just notational conveniences to indicate that "sampling happens here." They don't provide any mathematical basis for the spectral images that are known to arise due to sampling. It is possible to successfully practice DSP at the journeyman level without this mathematical basis, and many introductory texts simply present the facts concerning spectral images and aliasing without delving into the mathematics needed to derive these facts. However, the mathematical background is useful for exploring advanced topics like *sampling jitter* and *nonuniform sampling*.

The formal mathematical model for ideal sampling is presented in Note 5. The important result

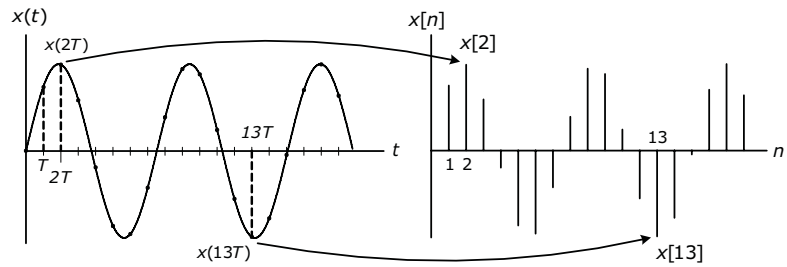


Figure 3.1 Ideal sampling viewed as a simple mapping from a continuous-time function to a discrete-time sequence

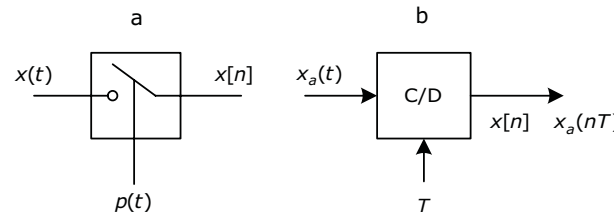


Figure 3.2 Block diagram representation of an ideal sampling process: (a) ideal sampler, (b) ideal continuous-to-discrete converter

produced by this model shows that sampling in the time domain can be expected to create periodic *images* of the original signal's spectrum in the frequency domain. Physical measurements confirm that creation of images predicted by the mathematical model does in fact occur in the real world.

Consider an arbitrary continuous-time signal, $x(t)$, having a spectrum that is bandlimited to the frequency range, $\pm f_H$, as shown in Figure 3.3. The spectrum for the ideally sampled version of the signal will consist of copies or images of the spectrum for $x(t)$ periodically repeated along the frequency axis with a center-to-center spacing that is equal to the sampling rate, as shown in Figure 3.4. The image corresponding to the original signal's spectrum is often called the *baseband*.

The sample signal's spectrum can be expressed in terms of the original signal's spectrum, $X(f)$, as

$$X_s(f) = \frac{1}{T}\sum_{m=-\infty}^{\infty} X\left(f - \frac{m}{T}\right) = f_s \sum_{m=-\infty}^{\infty} X(f - mf_s) \tag{3.2}$$

A few texts (such as [2]) redefine the sampling relation of Eq. (3.1) to be

$$x[n] \leftarrow Tx(nT)$$

thereby causing Eq. (3.2) to become

$$X_s(f) = \sum_{m=-\infty}^{\infty} X\left(f - \frac{m}{T}\right)$$

The creation of the images is the important idea—the presence or absence of the $1/T$ scaling factor is usually lost in the overall scaling strategy of most real-world sampling implementations.

3.1 Aliasing

Figure 3.4 shows the case in which $f_s > 2f_H$. The original baseband spectrum and the images are all distinct, and theoretically the original signal could be recovered exactly via ideal lowpass filtering to completely remove all of the images and leave the undistorted baseband spectrum. The sampling theorem, presented in the next section, provides a formal statement of this property, as well as a formula for reconstructing the original signal from the ideal samples.

In the case where $f_s < 2f_H$, the spectral images overlap, as shown in Figure 3.5. This overlap creates the condition known as *aliasing*. Components in the original signal at frequencies greater than $f_s/2$ will appear at frequencies below $f_s/2$ after the

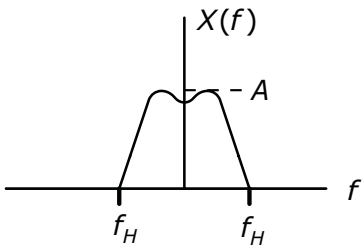


Figure 3.3 Bandlimited spectrum of an arbitrary analog signal

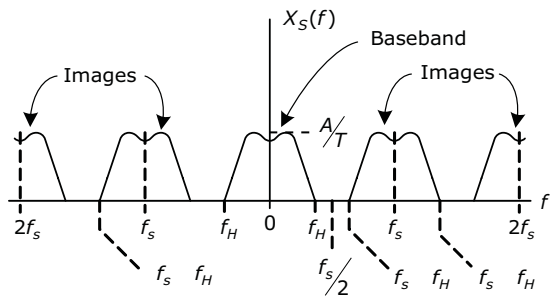


Figure 3.4 Frequency relationships for sampling-induced images when $f_s > 2f_H$

sampling is performed. As depicted in Figure 3.6, the result is as though the portion of the spectrum above $f_s/2$ folds over the line at $f = f_s/2$ and adds into the spectrum immediately below $f_s/2$. Based on this viewpoint, $f_s/2$ is sometimes called the *folding frequency*.

Note 4 shows how the concept of ideal sampling is applied to real-world signals in practical applications.

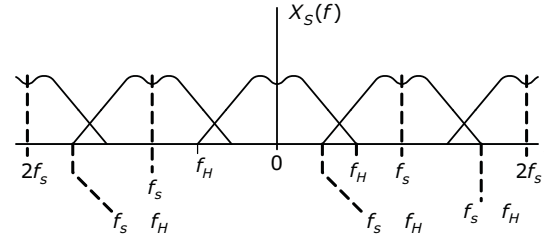


Figure 3.5 Frequency relationships for sampling-induced images when $f_s < 2f_H$

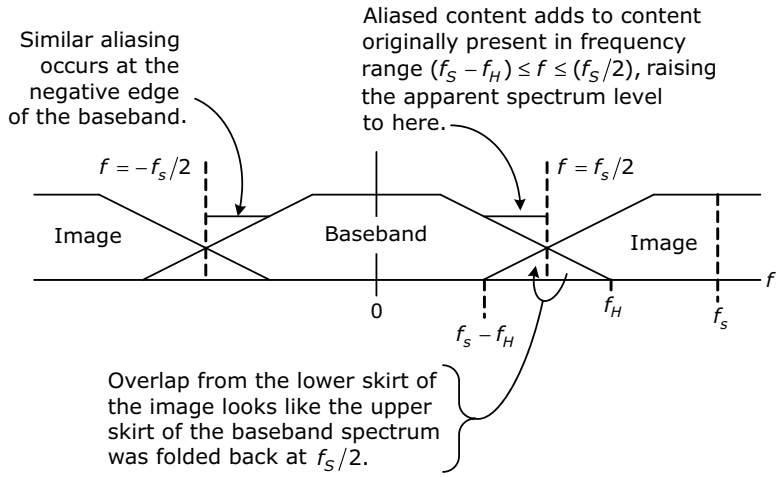


Figure 3.6 Aliasing viewed as spectral folding at $f = f_s/2$

References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing,* Prentice Hall, 1989.
2. R. A. Roberts and C.T. Mullis, *Digital Signal Processing,* Addison-Wesley, 1987.

Practical Application of Ideal Sampling

This note shows how theoretical results from Note 3 are used to develop a practical sampling strategy that minimizes the effects of aliasing.

In the discussions of aliasing in Note 3, the frequency f_H is portrayed as an absolute upper frequency. Prior to sampling, the signal of interest has zero spectral content at frequencies greater than f_H. Under these conditions, sampling at rates greater than $2f_H$ would result in no aliasing. However, this ideal situation is impossible to achieve in practical applications, where there will always be some aliasing. In many cases, even to achieve extremely low levels of aliasing, f_H would have to be set so high that, in most cases, sampling at a rate of $2f_H$ would be prohibitively difficult and expensive. In practical situations, rather than trying to avoid all aliasing, the design goal is to minimize the effects of aliasing while recognizing that they can not be completely eliminated.

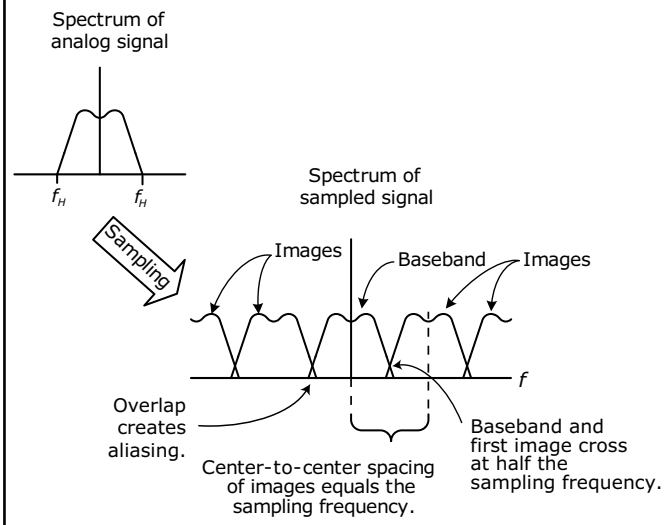
Practical sampling is performed at a rate greater than $2f_H$, where the signal of interest is known to have negligible (rather than zero) spectral content above some upper frequency, f_H. The definition of "negligible" varies based on the particular application. In some cases, f_H might be set so restrictively that less than 0.01 percent of the signal's energy is at frequencies greater than f_H. In less demanding applications in which the analog-to-digital converter (ADC) cost may need to be kept low, f_H may be set lower to allow up to 5 percent of the signal's energy to be at frequencies greater than f_H.

In most applications, the signal is passed through an anti-aliasing filter prior to being digitized. The purpose of that filter is to ensure

Key Concept 4.1

Aliasing

The sampling process creates periodic images of the original signal's spectrum in the frequency domain. Overlap between these images creates a condition called *aliasing*, in which components in the original signal at frequencies greater than half the sampling rate will appear, or *alias*, at frequencies below half the sampling rate in the spectrum of the sampled signal.

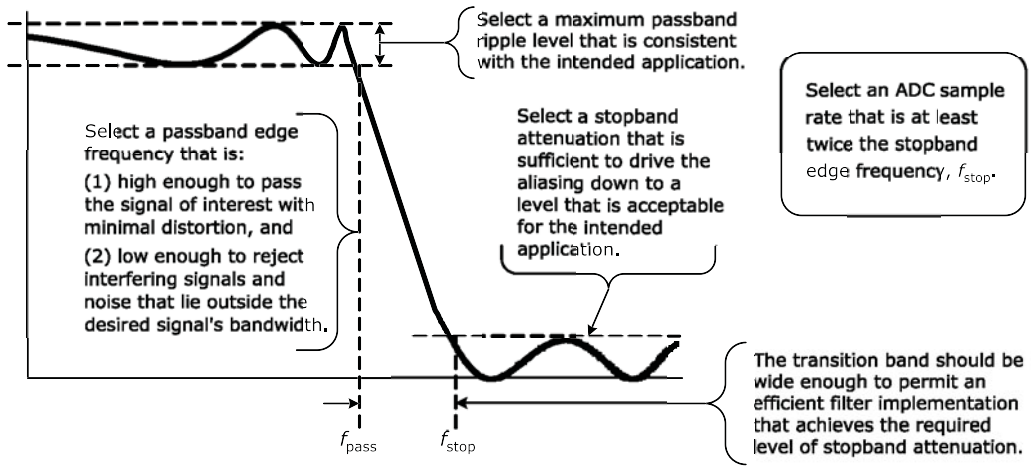


In practical systems, a lowpass *anti-aliasing filter* is typically used prior to the sampling operation in order to attenuate components at frequencies greater than half the sampling rate and thereby minimize the effects of aliasing.

Design Strategy 4.1

Setting the Anti-Aliasing Filter Specification

For the *anti-aliasing filter*, the designer's challenge is to strike a good compromise between sampling rate and filter complexity.



that aliasing is held to tolerably low levels. A value for f_H can be chosen based on the properties of the signal of interest and the requirements of the application, but in real-world situations, it is generally not possible to guarantee that the chosen value for f_H bounds the frequency extent of the actual signal that is presented to the ADC. There are several reasons why this is so:

- The signal of interest may be contaminated with wideband additive noise.
- The signal of interest may be contaminated with an unanticipated interfering signal having a bandwidth that extends beyond f_H.
- The signal of interest may contain spurious self-interference caused by nonlinear processing in a mixer or a saturated amplifier prior to sampling. Because nonlinear processing creates components at new frequencies, some of this self-interference may occur at frequencies above f_H.

The selection of the sampling rate, f_s, and the design of the anti-aliasing filter are coordinated, observing the guidelines called out in Design Strategy 4.1, so that the filter produces minimal attenuation or distortion for frequencies below f_H, but provides severe attenuation for all frequencies above $f_s/2$.

References

1. A.V. Oppenheim and R.W. Schafer, *Discrete-Time Signal Processing,* Prentice Hall, 1989.
2. R.A. Roberts and C.T. Mullis, *Digital Signal Processing,* Addison-Wesley, 1987.

Delta Functions and the Sampling Theorem

When ideal sampling is viewed as a direct mapping process, as it is in Note 3, the creation of spectral images is simply stated as a fact and accepted without mathematical justification. To generate mathematical support for the existence of these images, a more complicated mathematical model of the sampling process must be adopted. In addition to correctly predicting the appearance of spectral images, such a model can also be used to derive the discrete-time Fourier transform (DTFT) and the discrete Fourier transform (DFT) from the "usual" continuous-time Fourier transform (CTFT). (See Math Boxes 12.1 and 13.2.)

If the result of ideal sampling is considered in the continuous-time domain, each sample exists for a single instant on the continuous-time axis, and the value of the sampled waveform is zero between sample instants. This differs from the approach corresponding to Eq. (3.1) in Note 3 in which the result of ideal sampling is viewed in the discrete-time domain where the result is defined only at the sample instants—between samples the result is not zero, it is simply not defined at all.

An ideal sampling process that produces a continuous-time result can be constructed using a generalized function: The impulse is sometimes called the *delta function* (due to its usual notation) or the *Dirac delta function,* in honor of the English physicist Paul Dirac (1902–1984), who made extensive use of impulse functions in his work on quantum mechanics.

Math Box 5.1

Sampling Theorem

If the spectrum, $X(f)$, of a function, $x(t)$, vanishes beyond an upper frequency of f_H Hz, then $x(t)$ can be completely determined by its values at uniform intervals of less than $1/(2f_H)$.

Reconstruction Formula

If a function is sampled at a rate, $f_s = T^{-1}$, that satisfies the sampling theorem, the original function, $x(t)$, can be reconstructed from the samples as

$$x(t) = \sum_{n=-\infty}^{\infty} x(nT) \frac{\sin\left[\pi\left(\frac{t}{T} - n\right)\right]}{\pi\left(\frac{t}{T} - n\right)} = \sum_{n=-\infty}^{\infty} x(nT)\,\mathrm{sinc}_{\pi}\left(\frac{t}{T} - n\right) \quad \text{(MB 5.1)}$$

5.1 Dirac Delta Function

As far as mathematicians are concerned, the Dirac delta function is not a function in the usual sense, and its use in engineering is sometimes controversial. Nevertheless, the delta function provides a convenient way to relate the spectra of ideal sampling, natural

sampling, and instantaneous sampling that are discussed in Notes 3, 6, and 7.

A number of nonrigorous approaches for defining the impulse function can be found throughout the literature. A *unit impulse* can be described loosely as having zero width and an infinite amplitude at the origin such that the total area under the impulse is equal to unity.

The impulse function is usually denoted as $\delta(t)$ and is depicted as a vertical arrow at the origin. The rigorous definition of $\delta(t)$, introduced in 1950 by Laurent Schwartz [1], rejects the notion that the impulse is an ordinary function and instead defines it as a *distribution*. The rigorous definition notwithstanding, most engineers are more comfortable defining the impulse in an *operational* sense. Specifically, $\delta(t)$ is taken as that function which exhibits the so-called *sifting property*:

$$\int_{-\infty}^{\infty} \delta(t) f(t) dt = f(0) \tag{5.1}$$

It has been shown [2] that the impulse function exhibits the properties listed in Math Box 5.2.

Math Box 5.2

Properties of the Dirac Delta Function

$$\int_{-\infty}^{\infty} \delta(t) dt = 1 \tag{MB 5.2}$$

$$\int_{-\infty}^{\infty} \delta(t - t_0) f(t) dt = f(t_0) \tag{MB 5.3}$$

$$\delta(at) = \frac{1}{|a|} \delta(t) \tag{MB 5.4}$$

$$\delta(t - t_0) f(t) = \delta(t - t_0) f(t_0) \tag{MB 5.5}$$

$$\delta(t) \overset{\text{FT}}{\longleftrightarrow} 1 \tag{MB 5.6}$$

5.2 Comb

In DSP, the delta function is most often encountered in the form of an infinite periodic sequence of impulses:

$$\sum_{k=-\infty}^{\infty} \delta(t - kT) \tag{5.2}$$

The DSP literature is about evenly split between two different names and notations for the function represented by Eq. (5.2). Many texts refer to this function as a *Dirac comb* and use the notation $\delta_T(t)$—or sometimes $\Delta_T(t)$—to indicate an impulse sequence having a period of T seconds:

$$\delta_T(t) \quad \sum_{k=-\infty}^{\infty} \delta(t - kT) \tag{5.3}$$

Other texts refer to this function as the *shah function* and denote it using the Cyrillic letter shah, which somewhat resembles the graph of the function

$$\text{ш}_T(t) \quad \sum_{k=-\infty}^{\infty} \delta(t - kT) \tag{5.4}$$

5.3 Sampling Model

A sequence of samples can be modeled as the result of multiplying the original signal and a comb of Dirac impulses:

$$x_s(t) = x_a(t)\delta_T(t)$$

where

$$\delta_T(t) \quad \sum_{n=-\infty}^{\infty} \delta(t - nT)$$

Figure 5.1 shows a C/D converter that has been modified to include a representation of this multiplication.

As indicated by Eqs. (MB 5.10) and (MB 5.11), the Fourier transform of a Dirac comb with a spacing of T in time is a second Dirac comb with a spacing in frequency of T^{-1} Hz, or $2\pi/T$ radians per second:

$$\begin{aligned} \mathcal{F}\{\delta_T(t)\} &= f_s \delta_{f_s}(f) \\ &= \omega_s \delta_{\omega_s}(\omega) \end{aligned} \qquad \textbf{(5.5)}$$

where

$$\delta_{f_s}(f) \quad \sum_{m=-\infty}^{\infty} \delta(f - mf_s)$$

$$\delta_{\omega_s}(\omega) \quad \sum_{m=-\infty}^{\infty} \delta(\omega - m\omega_s)$$

Multiplication in time corresponds to convolution in frequency, so the spectrum of the sampled signal can be obtained by convolving the original signal's spectrum with the right-hand sides of Eq. (5.5):

$$X_s(f) = X(f)\left[f_s\, \delta_{f_s}(f)\right]$$

$$X_s(\omega) = \frac{1}{2\pi}\left(X(\omega) \cdot [\omega_s\, \delta_{\omega_s}(\omega)]\right)$$

Finally, exploiting the shifting property of the delta function yields

$$X_s(f) \quad f_s \sum_{m=-\infty}^{\infty} X(f - mf_s)$$

$$X_s(\omega) \quad \frac{\omega_s}{2\,\pi} \sum_{m=-\infty}^{\infty} \delta(\omega - m\omega_s)$$

Math Box 5.3

Properties of the Dirac Comb

$$\delta_T(t)x(t) = \sum_{k=-\infty}^{\infty} x(kt)\delta(t - kT) \qquad \textbf{(MB 5.7)}$$

$$\delta_T(t) \otimes x(t) = \sum_{k=-\infty}^{\infty} x(t - kT) \qquad \textbf{(MB 5.8)}$$

$$\delta_T(t) = \frac{1}{T} \sum_{k=-\infty}^{\infty} \exp(j2\pi nt / T) \qquad \textbf{(MB 5.9)}$$

$$\delta_T(t) \xrightarrow{\text{FT}} \frac{1}{T} \sum_{k=-\infty}^{\infty} \delta\left(f - \frac{k}{T}\right) \qquad \textbf{(MB 5.10)}$$

$$\delta_T(t) \xrightarrow{\text{FT}} \frac{2\pi}{T} \sum_{k=-\infty}^{\infty} \delta\left(\omega - k\frac{2\pi}{T}\right) \qquad \textbf{(MB 5.11)}$$

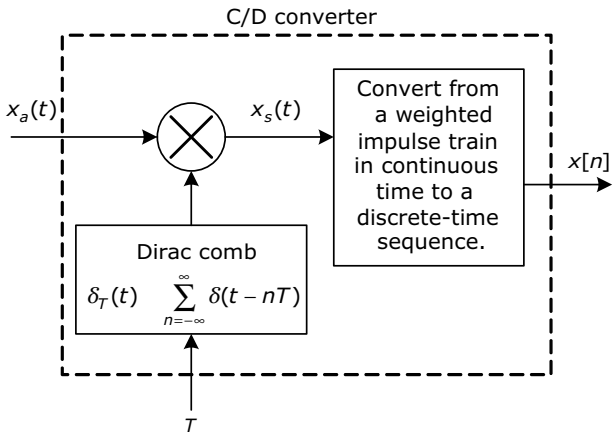


Figure 5.1 Continuous-to-discrete converter with two separate processing steps

Historical Note 5.1

Development of the Sampling Theorem

The early history of sampling theory is detailed in a 1999 article by Lüke [3]. It was the application of multiplexing in telephony that first led to consideration of just how rapidly a continuous-time signal needed to be sampled in order to preserve fidelity and ensure the ability to reconstruct exactly the original unsampled signal.

The early practitioners in telephony published vague or incorrect statements regarding the required minimum sampling rate for a given signal. In some cases, a sample rate was deemed adequate based on intelligibility of the reconstructed signal rather than on the absolute accuracy of the reconstruction.

The result, now known as the uniform sampling theorem or sometimes Shannon's sampling theorem, was independently formulated and published by H. Nyquist in 1928 [4], V. A. Kotelnikov in 1933 [6], and by H. Raabe in 1939 [7]. Raabe's work was cited by Bennett in 1941 [8], and Bennett was subsequently cited by Shannon in 1949 [5].

Shannon makes no claim to discovering the sampling theorem, having written that the sampling theorem result "is a fact which is common knowledge in the communications art." Shannon's name is associated with the sampling theorem because he published the first proof of the theorem in English, and he was the first to publicize the theorem widely within the communications engineering community.

References

1. L. Schwartz, *Théorie des distributions, Herman & Cie,* Paris, 1950.
2. R. N. Bracewell, *The Fourier Transform and Its Applications,* 3rd ed., McGraw-Hill, 2000.
3. H. D. Lüke, "The Origins of the Sampling Theorem," *IEEE Communications,* April 1999, pp. 106–108.
4. H. Nyquist, "Certain topics in telegraph transmission theory," *Trans. AIEE,* vol. 47, Apr. 1928, pp. 617–644, republished in *Proc. IEEE,* vol. 90, no. 2, Feb. 2002, pp. 280–305.
5. C. E. Shannon, "Communications in the presence of noise," *Proc. IRE,* vol. 37, 1949, pp. 10–21.
6. V. A. Kotelnikov, "O propusknoj sposobnosti 'efira' i provoloki v elektrosvjazi," ("On the transmission capacity of 'ether' and wire in electro-communications"), *First All-Union Conference on Questions of Communication,* Jan. 14, 1933.
7. H. Raabe, "Untersuchungen an der wechselzeitigen Mehrfachubertragung (Multiplexubertragung)," *Elektrische Nachrichtentechnik,* vol. 16, 1939, pp. 213–228.
8. W. R. Bennett, "Time division multiplex systems," *Bell Syst. Tech. J.,* vol. 20, 1941, pp. 199–221.

Natural Sampling

In natural sampling, an analog signal is gated in such a way that the resulting signal consists of pulses with time-varying amplitudes that follow the contours of the original waveform as shown in Figure 6.1. In this example, the original signal is a sinusoid with a period of T_x, and the sampled signal has a sampling interval of T, and a sample width of τ. Natural sampling is mathematically equivalent to multiplying the original signal with a train of unit-amplitude rectangular sampling pulses. Therefore, the spectrum of a naturally sampled signal can be determined by convolving the original signal's spectrum with the spectrum of the train of sampling pulses.

Figure 6.2 summarizes the relationships between the original signal (labeled T4), the naturally sampled signal (labeled T5), and the corresponding spectra (labeled F4 and F5, respectively). The train of sampling pulses (labeled T3) can be generated by convolving a single pulse of width τ (labeled T1) with a Dirac comb having impulses spaced at intervals of T (labeled T2). The resulting pulse train has a magnitude spectrum like the one shown in Figure 6.3.

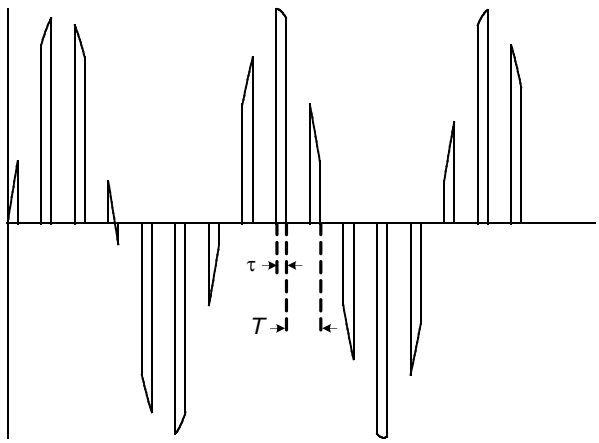
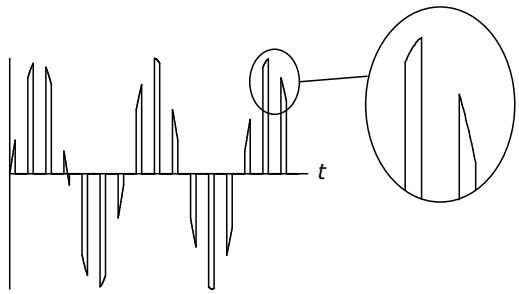


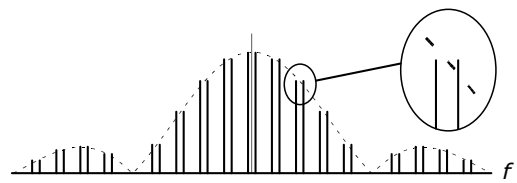
Figure 6.1 In natural samping, the amplitudes of the sample pulses follow the varying amplitudes of the original function.

At a Glance

- *Natural sampling* is a mathematical model that can be used to analyze the impacts of analog multiplexing and signal commutation and decommutation.
- In natural sampling, the samples have nonzero width and the amplitude of the sample varies across the width of the sample to match the amplitude of the analog signal.



- In the magnitude spectrum of a naturally sampled signal, each image is scaled by a factor that is constant over the image but that varies from image to image according to the magnitude of a sinc envelope.



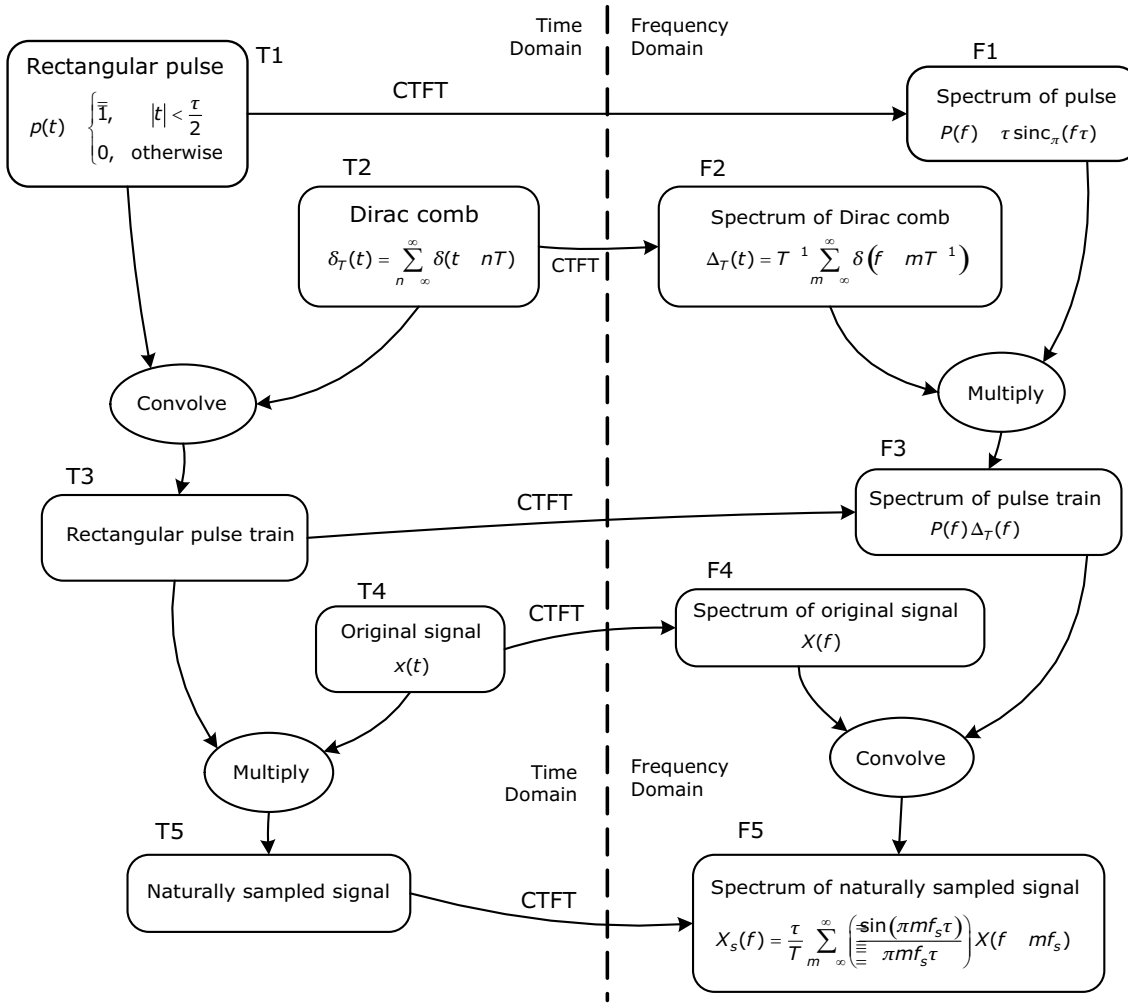


Figure 6.2 Natural sampling is mathematically equivalent to multiplying the original spectrum with a train of rectangular sampling pulses. Therefore, the spectrum of a naturally sampled signal can be determined by convolving the original signal's spectrum with the spectrum of the train of sampling pulses. (The multiply operation that creates spectrum F3 is not mathematically rigorous, but it is consistent with typical engineering use of Dirac delta functions as discussed in Note 5. "CTFT" indicates the continuous-time Fourier transform.)

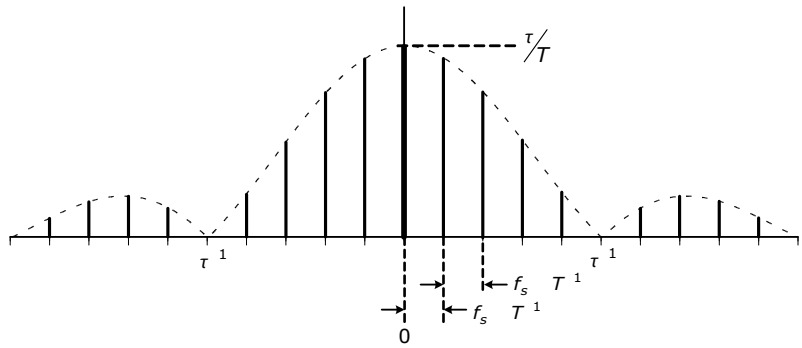


Figure 6.3 Magnitude spectrum of rectangular pulse train (corresponds to block F3 in Figure 6.2)

The magnitude spectrum for a naturally sampled sinusoid is shown in Figure 6.4. The spacing of the images is equal to the reciprocal of the sampling interval, and each image is amplitude scaled by the value of $\tau T^{-1}|\text{sinc}_\pi(f\tau)|$ at the frequency corresponding to the center of the image. In other words, the image centered at $f = nf_s$ is scaled by $\tau T^{-1}|\text{sinc}_\pi(nf_s\tau)|$. This factor changes from image to image, but remains constant across the width of each image.

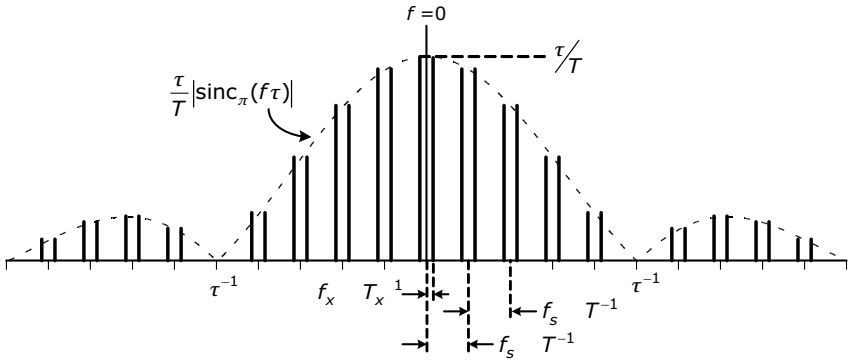


Figure 6.4 Magnitude spectrum of naturally sampled sinusoid (corresponds to block F5 in Figure 6.2)

Reference

1. W. R. Bennett, "Time division multiplex systems," *Bell Syst. Tech. J.*, vol. 20, 1941, pp. 199–221.

Instantaneous Sampling

In *instantaneous sampling*, the value of the analog signal is captured at the sampling instant and held constant for the duration of the sample pulse, as shown in Figure 7.1. Instantaneous sampling is sometimes called "flat-topped" sampling or *zero-order-data-hold* sampling.

Instantaneous sampling is mathematically equivalent to convolving a single rectangular sampling pulse with an ideally sampled version of the original signal. Therefore, the spectrum of an instantaneously sampled signal can be determined by multiplying the ideally sampled signal's spectrum with the spectrum of a single sample pulse.

Figure 7.2 summarizes the relationships between the original signal (labeled T2), the instantaneously sampled signal (labeled T5) and the corresponding spectra (labeled F2 and F5 respectively). A single pulse, $p(t)$, (labeled T4 in the figure) defined by

$$p(t) = \begin{cases} 1, & |t| < \frac{\tau}{2} \\ 0, & \text{otherwise} \end{cases}$$

has the Fourier transform, $P(f)$, (labled F4) given by

$$P(f) \quad \tau \operatorname{sinc}_{\pi}(f\tau)$$

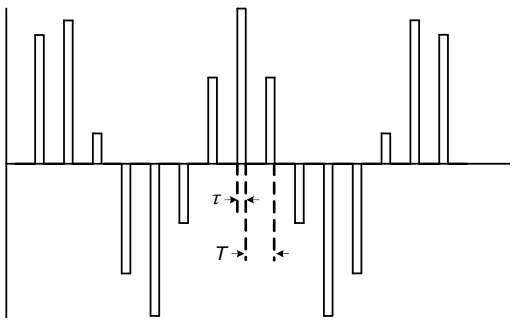
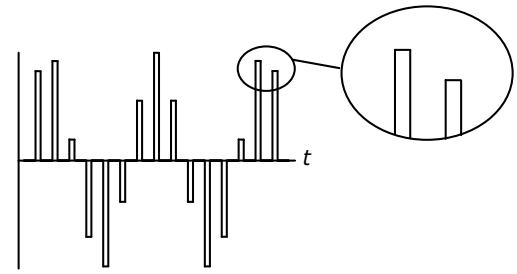


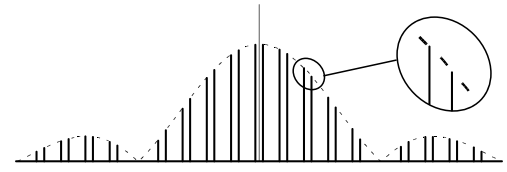
Figure 7.1 In instantaneous sampling, the amplitude of each sample remains constant over its width.

At a Glance

- *Instantaneous sampling* (also known as *zero-order-data-hold* (ZODH) sampling) is a mathematical model that can be used to analyze the impact on signal quality of digital-to-analog conversion, as shown in Note 8.
- In instantaneous sampling, the samples have nonzero width and the amplitude of the sample remains constant across the width but varies from sample to sample to match the amplitude of the analog signal at each sample's starting instant.



- In the magnitude spectrum of an instantaneously sampled signal, all spectral components are scaled by the magnitude of a sinc envelope. Because the scaling varies across the width of each spectral image, the signal is distorted.



The magnitude spectrum of an ideally sampled sinusoid is shown in Figure 7.3. Multiplying this spectrum (F3 in Figure 7.2) by $P(f)$ yields the result shown in Figure 7.4. The scaling factor does not remain constant across each image, as it does in the spectrum for natural sampling. If the spectrum of the original spectrum is represented as $X(f)$, then the spectrum of the corresponding instantaneously sampled signal is given by

$$X_s(f) = \frac{\tau}{T}\left(\frac{\sin(\pi f \tau)}{\pi f \tau}\right)\sum_{n=-\infty}^{\infty} X\left(f - nT^{-1}\right)$$

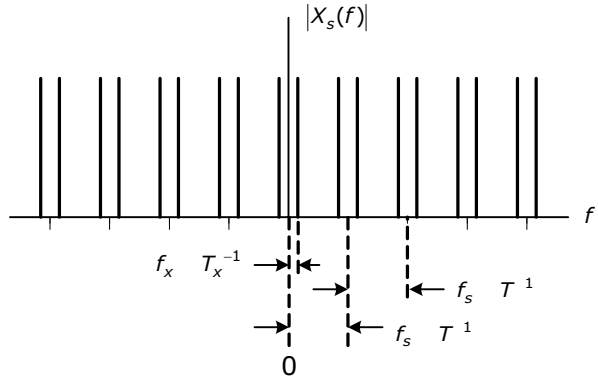


Figure 7.3 Magnitude spectrum of ideally sampled sinusoid (corresponds to block F3 in Figure 7.2).

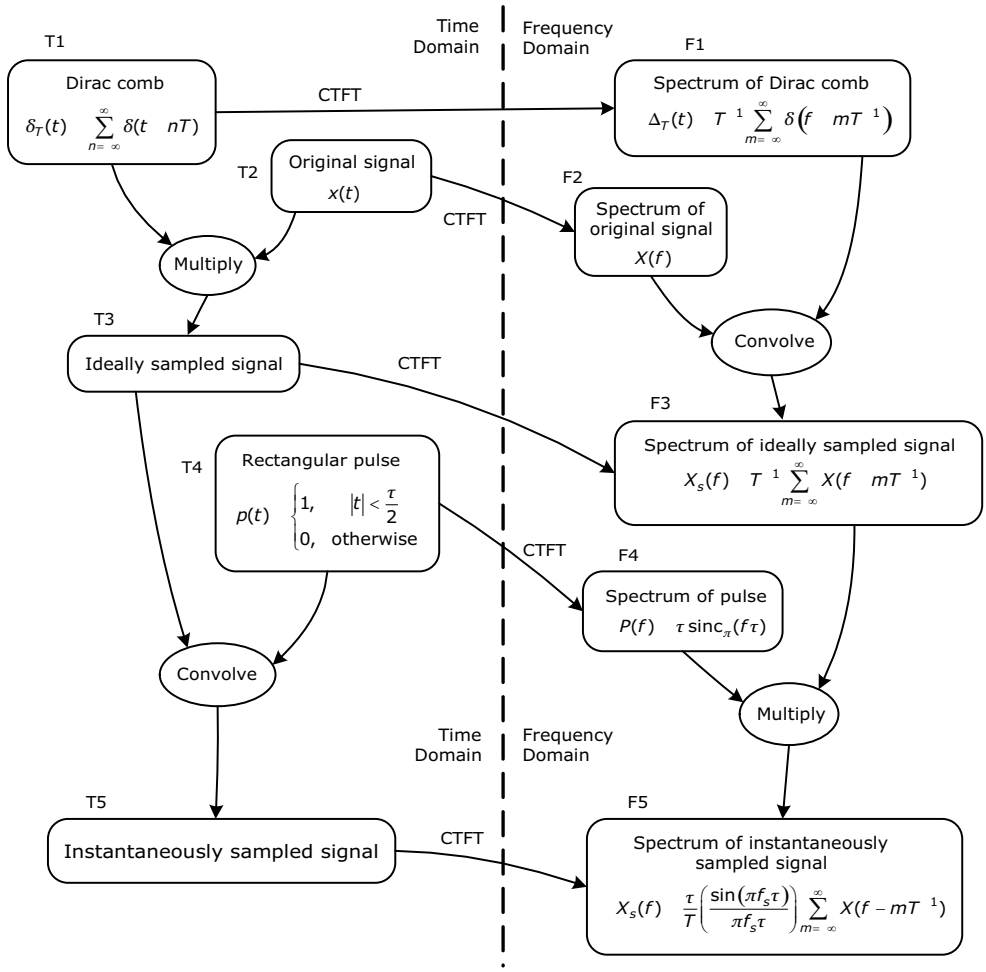


Figure 7.2 Instantaneous sampling is mathematically equivalent to convolving a single rectangular sampling pulse with an ideally sampled version of the original signal. Therefore, the spectrum of an instantaneously sampled signal can be determined by multiplying the ideally sampled signal's spectrum with the spectrum of a single sample pulse. ("CTFT" indicates the continuous-time Fourier transform.)

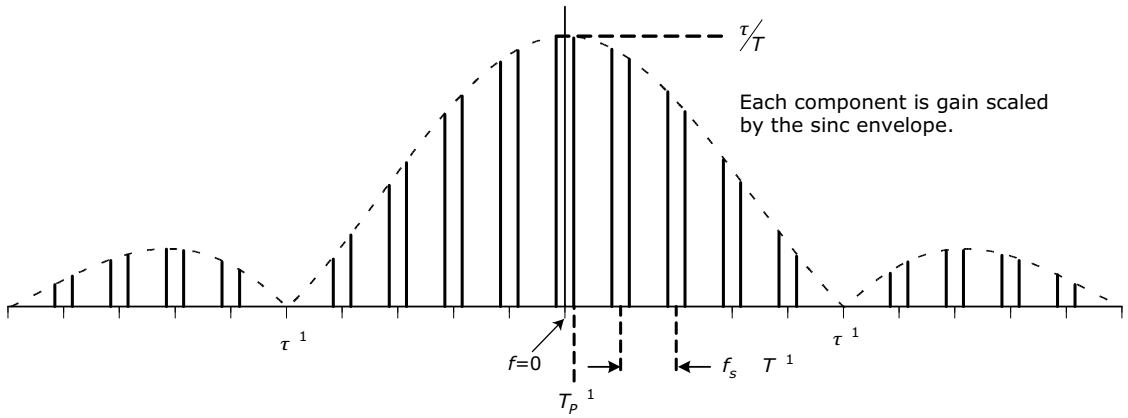


Figure 7.4 Magnitude spectrum of instantaneously sampled sinusoid (corresponds to block F5 in Figure 7.2)

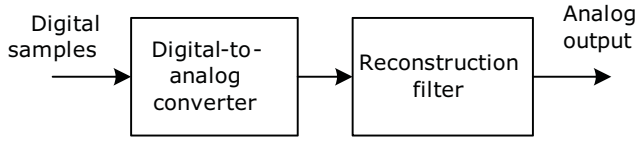
Reference

1. W. R. Bennett, "Time division multiplex systems," *Bell Syst. Tech. J.*, vol. 20, 1941, pp. 199–221.

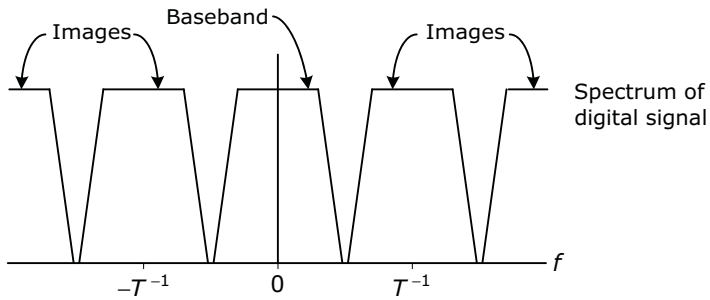
Reconstructing Physical Signals

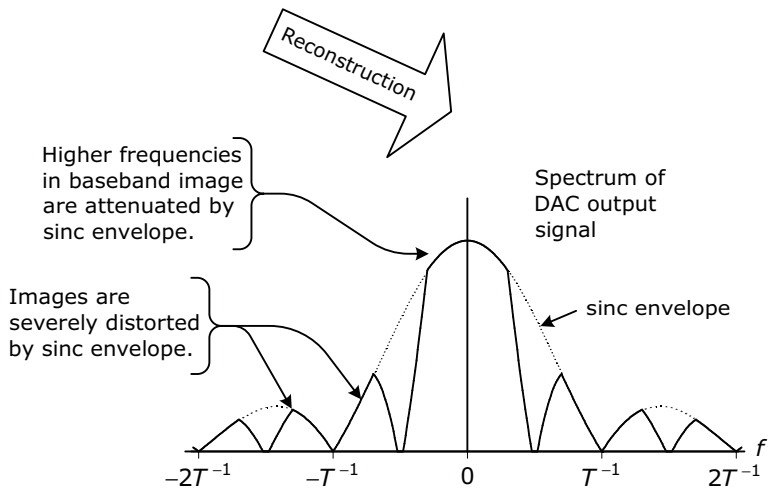
The Basic Idea

An analog signal is typically reconstructed from a discrete-time signal using a *digital-to-analog converter* (DAC) followed by a *reconstruction filter*.



The spectrum of the DAC output signal is distorted by a sinc envelope, with the nulls of the sinc envelope falling in the center of each image other than the baseband.





The reconstruction filter must remove the images and correct for the sinc distortion in the baseband. The close spacing between the baseband and the adjacent images can make the design of this filter relatively difficult.

The mathematical signal reconstruction techniques presented in Note 5 do have practical uses, but these techniques are not really suited for converting a sequence of digital signal values back into a continuously time-varying voltage that can be used to drive a speaker or a pair of headphones. Reconstruction of a physical analog signal is usually accomplished using a digital-to-analog converter (DAC). The input to the DAC is a sequence of digital words, and the output is a time-varying voltage that is proportional to the sequence of values represented by the input words. Each output voltage is held constant until the input value changes.

The output of the DAC can be viewed as a special case of the instantaneously sampled signal described in Note 7. In Note 7, each voltage pulse is depicted as being significantly narrower than the sampling interval. The DAC output is a special case in that the DAC typically holds each output value for an entire sampling interval, thereby generating a "stair-step" signal, such as the one shown in Figure 8.2, in which the sample width equals the sampling interval.

The spectrum of the DAC output contains images and is multiplied by a $(\sin x)/x$ envelope, as discussed in Note 7. However, the illustrations in Note 7 depict the case in which the sampling interval, T, is several times larger than the sample width, τ. When the sample width is equal or nearly equal to the sampling interval, the distortion effects caused by the $(\sin x)/x$ envelope are much more severe.

Assume that the ideally sampled signal inside the processing computer has a simple trapezoidal baseband spectrum, as depicted in Figure 8.3. The corresponding DAC output has a spectrum,

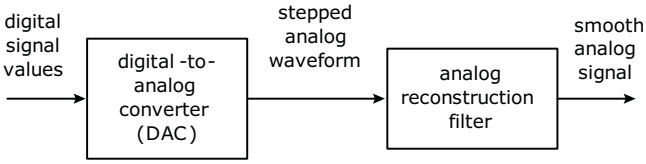


Figure 8.1 Block diagram of the signal reconstruction process

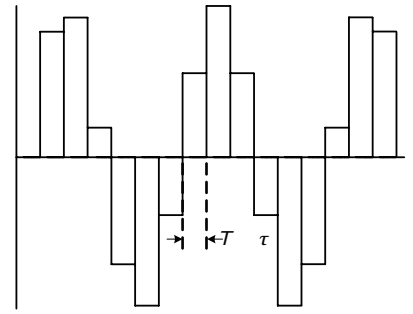


Figure 8.2 Output of DAC modeled as the limiting case of instantaneous sampling

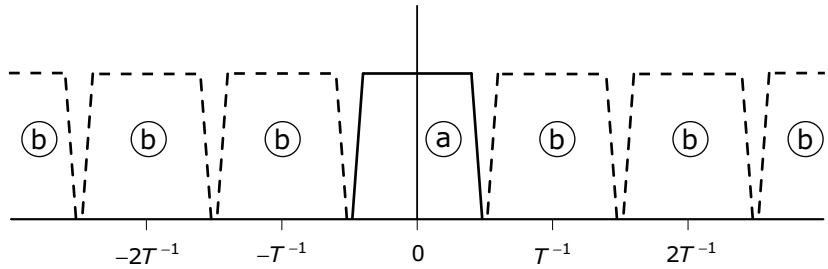


Figure 8.3 Idealized trapezoidal spectrum for a sampled signal, showing (a) the baseband spectrum of the original signal, and (b) spectral images created by the sampling process

as shown in Figure 8.4, with the main lobe of the $(\sin x)/x$ envelope having a null-to-null width of just twice the sampling rate. All of the images are severely distorted by the side lobes. Because it occupies such a large portion of the main lobe, the baseband component of the spectrum also experiences distortion from the $(\sin x)/x$ envelope.

The reconstruction filter that follows the DAC needs to have both a stopband response that severely attenuates the spectral images and a passband response that is designed to correct the $(\sin x)/x$ distortion present on the baseband spectrum. When the sample rate equals exactly twice the highest frequency component in the signal's original spectrum, the signal is said to be *critically sampled*. When the signal is critically sampled, as in the case depicted in Figure 8.4, the images are close together, thus making it almost impossible to design a filter that can both remove the images and compensate for $(\sin x)/x$ distortion. Most filter designs that can remove the images under these conditions are likely to introduce phase distortion into the baseband signal.

Design and implementation of the reconstruction filter can be made easier by modifying the DAC output signal in a way that separates the spectral images, as shown in Figure 8.5. In many practical systems, the sample rate is already significantly higher than twice the signal bandwidth, which causes the images to be spread farther apart and simplifies the filter design task. In other applications, such as audio CD players, it is necessary to take explicit steps to make the reconstruction problem easier to manage by increasing the *Nyquist bandwidth* (which is equal to one-half the sample rate) without increasing the utilized bandwidth, as shown in Figure 8.5.

Audio CD players use a sample rate of 44.1 kHz to support a utilized bandwidth of about 20 kHz, so there would be a gap of about 4 kHz between images in the DAC output. Design of an acceptable reconstruction filter is not impossible, and there were many early CD players built that used direct reconstruction of the 44.1 kHz sample stream. However, CD players are consumer products, and there is constant pressure to make them smaller, lighter, cheaper, and better-sounding. Increasing the CD sample rate would make it easier to build cheaper reconstruction filters with good performance, but an increased sample rate for the recorded signal would require more samples for each

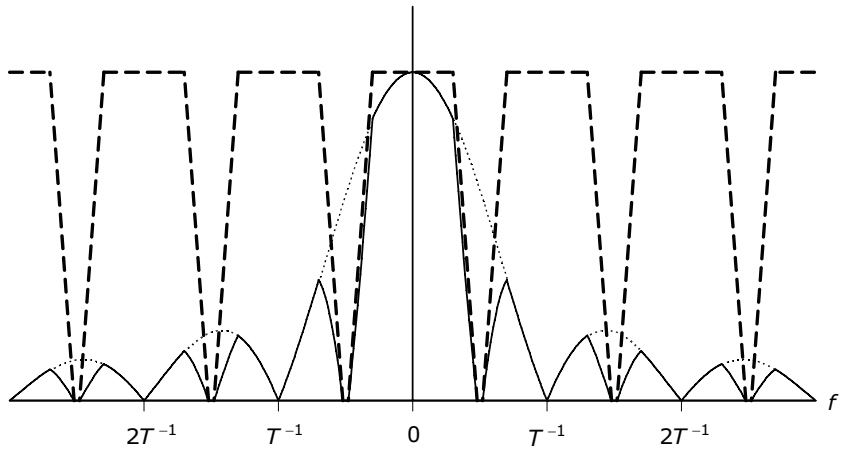


Figure 8.4 Spectrum at output of DAC (solid trace). Shown for comparison are the undistorted images of the trapezoidal spectrum (dashed trace) and sinc envelope (dotted trace).

second of audio, thus resulting in reduced playing time for a disc having a given total bit capacity.

Most newer CD players advertise 4×, 8×, or even 16× oversampling, but the increased sample rate is not used for the recorded signal. Instead, the digital signal is interpolated to create new sample values in between the sample values that are actually read from the disc. This type of oversampling does not increase the utilized bandwidth; that is, it does not increase the bandwidth of the recorded signal or the reconstructed signal. What it does do is increase the Nyquist bandwidth, which moves the spectral images farther apart so that it becomes relatively easy to design a reconstruction filter that will reject all of the non-baseband images while simultaneously compensating for the $(\sin x)/x$ distortion in the baseband spectral component. As shown in Figure 8.5, this oversampling will also have the effect of widening the main lobe of the sinc function so that the utilized bandwidth coincides with a more central, flatter portion of the main lobe, thus lessening the severity of the distortion that the reconstruction filter must correct.

If oversampling is carried throughout the digital processing and if the utilized bandwidth has not already been limited by this processing, then it may be prudent to perform digital filtering to limit the utilized bandwidth just prior to sending the signal to the DAC. Because of the increased Nyquist bandwidth (due to the higher sampling rate), earlier processing may have inadvertantly introduced components outside of the intended signal bandwidth and thereby increased the utilized bandwidth, reducing the effectiveness of the oversampling strategy. On the other hand, if the oversampling is introduced by an interpolation process just prior to sending the signal to the DAC, there is no opportunity to inadvertently increase the utilized bandwidth, and additional filtering would not be necessary.

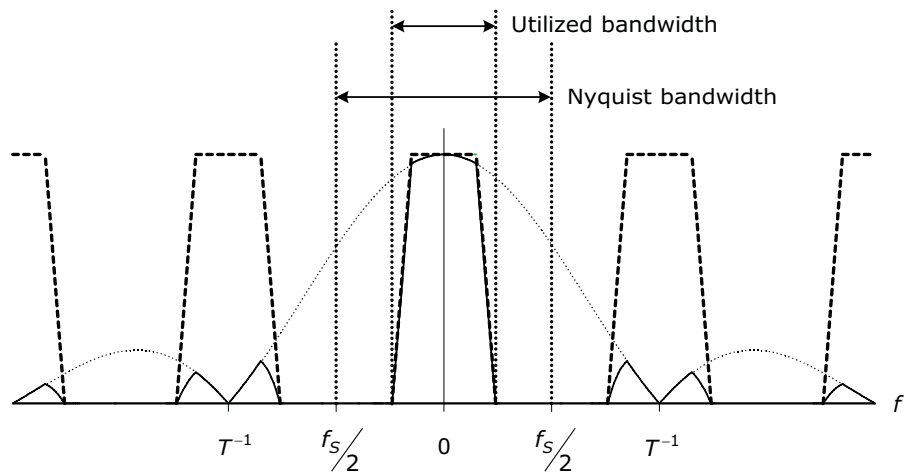


Figure 8.5 Spectrum at output of DAC with 2× oversampling

Overview of Fourier Analysis

Fourier analysis is based on the notion that signals can be represented and analyzed as weighted sums of sinusoidal components over a range of different frequencies. The collection of amplitudes and phases for the sinusoids needed to completely represent the signal is usually called the spectrum of the signal. Depending upon the nature of the signal being analyzed, the spectrum can span either a continuum of frequencies or a countable (but possibly infinite) set of discrete frequencies.

9.1 Fourier Series

Periodic continuous-time signals have finite power (but infinite energy), and can be analyzed using the *Fourier series* (FS) defined by

$$X[n] = \frac{1}{T}\int_{T} x(t)\exp\left(-jn\omega_0 t\right)dt$$

$$\text{for } n = 0, \pm 1, \pm 2, \pm 3, \ldots, \pm\infty$$

where ω_0 is the fundamental radian frequency of $x(t)$, T is the period of $x(t)$, and the single limit of T on the integral indicates that the integration is performed over one period of $x(t)$.

The spectrum produced by the Fourier series is a function of discrete frequency—or in other words, the spectrum has nonzero values only at discrete frequencies that correspond to integer multiples (including zero) of the periodic time signal's fundamental frequency. The Fourier series is discussed in more detail in Note 10.

Major Categories of Fourier Analysis

Fourier Series

- Signal: periodic function of continuous time
- Spectrum: nonperiodic function of discrete frequency
- Detailed in Note 10

Fourier Transform

- Signal: nonperiodic function of continuous time
- Spectrum: nonperiodic function of continuous frequency
- Detailed in Note 11

Discrete-Time Fourier Transform

- Signal: nonperiodic function of discrete time
- Spectrum: represented as a periodic function of continuous frequency
- Detailed in Note 12

Discrete Fourier Transform

- Signal: periodic function of discrete time
- Spectrum: periodic function of discrete frequency
- Detailed in Notes 13 through 16

Fast Fourier Transform

- Collection of techniques for efficient implementation of the discrete Fourier transform
- Detailed in Notes 17 through 19

9.2 Fourier Transform

Mathematically defined continuous time signals having finite energy can be analyzed using the *Fourier transform* (FT, or sometimes CTFT to emphasize the continuous-time nature of the input) defined by

$$X(\omega) = \int_{-\infty}^{\infty} x(t)\exp(-j\omega t)dt$$

where ω represents continuous radian frequency. There are also some "special" signals that have a Fourier transform even though the signals do not have finite energy. The spectrum produced by the Fourier transform is a function of continuous frequency. Details of the Fourier transform are discussed in Note 11.

9.3 Discrete-Time Fourier Transform

Discrete-time signals that have finite energy can be analyzed using the *discrete-time Fourier transform* (DTFT), defined by

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]\exp(-j\omega nT)$$

The spectrum produced by the DTFT is a function of continuous frequency and is periodic, with a period equal to the reciprocal of the time-domain sampling interval. If the discrete-time signal is viewed as having been created via sampling of a properly bandlimited continuous-time signal, then one period of the DTFT's periodic spectrum can be interpreted as the spectrum of the original analog signal. The DTFT is discussed in Note 12.

9.4 Discrete Fourier Transform

Periodic discrete-time signals have finite power and are analyzed using the *discrete Fourier transform* (DFT), defined by

$$X[m] = \sum_{n=0}^{N-1} x[n]\exp(-j2\pi mFnT)$$

The spectrum produced by the DFT is discrete in frequency and periodic, with a period equal to the reciprocal of the time-domain sampling interval. Many practical applications do not involve periodic discrete-time signals—in such applications, a finite-duration segment of the sampled signal is assumed to represent exactly one period of a periodic (and infinite duration) signal and analyzed using the DFT. If the segment of the discrete-time signal is obtained by sampling a properly bandlimited analog signal, then one period of the DFT's periodic spectrum can be interpreted as a frequency-sampled estimate of the original analog signal's spectrum. The DFT and its properties are discussed in Notes 13 through 16.

9.5 Fast Fourier Transforms

Direct computation of an N-sample DFT involves a number of arithmetic operations that is proportional to N^2. There are a number of different algorithms that exploit periodicities in the sine and cosine functions to compute the DFT using significantly fewer arithmetic operations. These algorithms are collectively referred to as the *fast Fourier transform* (FFT). The most commonly used FFT algorithms require a number of arithmetic operations that is proportional to $N\log_2 N$. Fast Fourier transform algorithms are presented in Notes 17 through 19. Using FFT algorithms to perform fast convolution is discussed in Note 20.

Fourier Series

The Fourier series forms a mathematical link between periodic continuous-time signals and their discrete-frequency spectra. In many books and engineering courses, the Fourier series is often introduced merely as a stepping stone on the way to the Fourier transform. However, within DSP, the Fourier series plays an important role in the design of *finite-impulse-response* (FIR) digital filters, which are discussed in Note 32.

10.1 Classical Form

The "classical" presentation of the Fourier series (FS) is given in a form that emphasizes that the time signal is being represented by a weighted sum of discrete sinusoids:

$$x(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t)] \tag{10.1}$$

where

$$a_0 = \frac{2}{T} \int_{-T/2}^{T/2} x(t)\,dt$$

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} x(t)\cos(n\omega_0 t)\,dt$$

$$b_n = \frac{2}{T} \int_{-T/2}^{T/2} x(t)\sin(n\omega_0 t)\,dt$$

$$T = \text{period of } x(t)$$

$$\beta = \cosh\left[\frac{1}{N}\cos\right.$$

Using a few trigonometric identities, the FS can be put in the magnitude-and-phase form given by

$$x(t) = c_0 + \sum_{n=1}^{\infty} c_n \cos(n\omega_0 t - \theta_n) \tag{10.2}$$

where the c_n and θ_n can be obtained from a_n and b_n as

$$c_0 = \frac{a_0}{2}$$

$$c_n = \sqrt{a_n^2 + b_n^2}$$

$$\theta_n = \tan^{-1}\left(\frac{b_n}{a_n}\right)$$

10.2 Modern Form

Mathematical manipulations are often made more convenient if the FS is put into the exponential form given by

$$x(t) = \sum_{n=-\infty}^{\infty} X[n]\exp(j2\pi n f t_0) \tag{10.3}$$

where

$$X[n] = \frac{1}{T}\int_{T} x(t)\exp(-j2\pi n f t_0)\,dt \tag{10.4}$$

In general, the values of X[n] are complex, and they are often displayed in the form of a magnitude spectrum, $|X[n]|$, and a phase spectrum, $\arg\{X[n]\}$ given by

$$\left|X[n]\right| = \sqrt{\left(\text{Re}\{X[n]\}\right)^2 + \left(\text{Im}\{X[n]\}\right)^2}$$

$$\arg\left(X[n]\right) = \tan^{-1}\left(\frac{\text{Im}\{X[n]\}}{\text{Re}\{X[n]\}}\right)$$

Notice that the spectra represented by Eqs. (10.1) and (10.2) are each "one-sided" in that their coefficients are only defined for values of $n \geq 0$. The spectrum represented by Eq. (10.4) is "two-sided" in that $X[n]$ is defined for all integer values of n. The two-sided spectrum of $X[n]$ can be related to the one-sided spectrum of (a_n, b_n) using

$$X[n] = \begin{cases} \dfrac{a_{-n} + jb_{-n}}{2} & n < 0 \\ a_0 & n = 0 \\ \dfrac{a_n - jb_n}{2} & n > 0 \end{cases}$$

10.3 Dirichlet Conditions

The Fourier series can be applied to most periodic signals of practical interest. However, there are a few mathematical functions for which the series will not converge. The FS coefficients exist, and the series will converge uniformly if $x(t)$ satisfies the following:

1. $x(t)$ is a single-valued function.
2. $x(t)$ has, at most, a finite number of discontinuities within each period.
3. $x(t)$ has, at most, a finite number of extrema within each period.
4. $x(t)$ is absolutely integrable over a period:

$$\int_T |x(t)| dt < 0 \tag{10 5}$$

These conditions are known as the Dirichlet conditions in honor of Peter Gustav Lejeune Dirichlet (1805–1859), who first published them in 1828.

Table 10.1 Properties of the Fourier Series

Property	Time Function	Frequency Function
Homogeneity	$ax(t)$	$aX[n]$
Additivity	$x(t)+y(t)$	$X[n]+Y[n]$
Linearity	$ax(t)+by(t)$	$aX[n]+bY[n]$
Multiplication	$x(t)y(t)$	$\sum_{m=-\infty}^{\infty} X[n-m]Y[m]$
Convolution	$\frac{1}{T}\int_0^T x(t-\tau)y(\tau)d\tau$	$X[n]Y[n]$
Time shifting	$x(t-\tau)$	$\exp\left(\frac{-j2\pi n\tau}{T}\right)X[n]$
Frequency shifting	$\exp\left(\frac{-j2\pi mt}{T}\right)x(t)$	$X[n-m]$

Fourier Transform

Mathematically defined continuous-time signals having finite energy can be analyzed using the Fourier transform (FT), defined in Math Box 11.1. The FT can be expressed as a function of either cyclic frequency, f, or radian frequency, $\omega = 2\pi f$. In some contexts, the Fourier transform is referred to as the *continuous-time Fourier transform* (CTFT) to emphasize the distinction between it and the *discrete-time Fourier transform* that is presented in Note 12.

A number of frequently encountered Fourier transform pairs are listed in Table 11.1, and a number of useful FT properties are listed in Table 11.2.

Math Box 11.1

Fourier Transform

$$X(f) = \int_{-\infty}^{\infty} x(t)\exp(-j2\pi ft)\,dt \qquad \text{(MB 11.1)}$$

$$X(\omega) = \int_{-\infty}^{\infty} x(t)\exp(-j\omega t)\,dt \qquad \text{(MB 11.2)}$$

Inverse Transform

$$x(t) = \int_{-\infty}^{\infty} X(f)\exp(j2\pi ft)\,df = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)\exp(j\omega t)\,d\omega \qquad \text{(MB 11.3)}$$

Table 11.1 Fourier transform pairs

#	$x(t)$	$X(f)$	$X(\omega)$
1.	1	$\delta(f)$	$2\pi\delta(\omega)$
2.	$u_1(t)$	$\frac{1}{2\pi f}+\frac{1}{2}\delta(f)$	$\frac{1}{j\omega}+\pi\delta(\omega)$
3.	$\delta(t)$	1	1
4.	t^n	$\left(\frac{j}{2\pi}\right)^n\delta^{(n)}(f)$	$2\pi j^n\delta^{(n)}(\omega)$
5.	$\sin\omega_0 t$	$\frac{j}{2}\left[\delta(f+f_0)-\delta(f-f_0)\right]$	$j\pi\left[\delta(\omega+\omega_0)-\delta(\omega-\omega_0)\right]$
6.	$\cos\omega_0 t$	$\frac{1}{2}\left[\delta(f+f_0)+\delta(f-f_0)\right]$	$\pi\left[\delta(\omega+\omega_0)+\delta(\omega-\omega_0)\right]$
7.	$e^{-at}u_1(t)$	$\frac{1}{j2\pi f+a}$	$\frac{1}{j\omega+a}$
8.	$u_1(t)e^{-at}\sin\omega_0 t$	$\frac{2\pi f_0}{(a+j2\pi f)^2+(2\pi f_0)^2}$	$\frac{\omega_0}{(a+j\omega)^2+\omega_0^2}$
9.	$u_1(t)e^{-at}\cos\omega_0 t$	$\frac{a+j2\pi f}{(a+j2\pi f)^2+(2\pi f_0)^2}$	$\frac{a+j\omega}{(a+j\omega)^2+\omega_0^2}$
10.	$\begin{cases}1 & \lvert t\rvert\le\frac{1}{2}\\ 0 & \text{otherwise}\end{cases}$	$\operatorname{sinc} f$	$\operatorname{sinc}\left(\frac{\omega}{2\pi}\right)$
11.	$\operatorname{sinc} t\quad\frac{\sin\pi t}{\pi t}$	$\begin{cases}1 & \lvert f\rvert\le\frac{1}{2}\\ 0 & \text{otherwise}\end{cases}$	$\begin{cases}1 & \lvert\omega\rvert\le\pi\\ 0 & \text{otherwise}\end{cases}$
12.	$\begin{cases}at\exp(-at) & t>0\\ 0 & \text{otherwise}\end{cases}$	$\frac{a}{(a+j2\pi f)^2}$	$\frac{a}{(a+j\omega)^2}$
13.	$\exp(-a\lvert t\rvert)$	$\frac{2a}{a^2+4\pi^2 f^2}$	$\frac{2a}{a^2+\omega^2}$
14.	$\text{signum}\, t\quad\begin{cases}1 & t>0\\ 0 & t=0\\ -1 & t<0\end{cases}$	$\frac{1}{j\pi f}$	$\frac{2}{j\omega}$

Table 11.2 Fourier transform properties

#	Property	Time function	Transform
1.	Homogeneity	$ax(t)$	$aX(f)$
2.	Additivity	$x(t)+y(t)$	$X(f)+Y(f)$
3.	Linearity	$ax(t)+by(t)$	$aX(f)+bY(f)$
4.	Time shifting	$x(t-\tau)$	$e^{-j2\pi ft}X(f)$
5.	Frequency shifting	$\exp(-j2\pi f_0 t)x(t)$	$X(f+f_0)$
6.	Multiplication	$x(t)y(t)$	$\int_{-\infty}^{\infty} X(\lambda)Y(f-\lambda)d\lambda$
7.	Convolution	$\int_{-\infty}^{\infty} h(t-\tau)x(\tau)d\tau$	$H(f)X(f)$
8.	Sine modulation	$x(t)\sin(2\pi f_0 t)$	$\frac{1}{2}j\left[X(f-f_0)+X(f+f_0)\right]$
9.	Cosine modulation	$x(t)\cos(2\pi f_0 t)$	$\frac{1}{2}\left[X(f-f_0)-X(f+f_0)\right]$
10.	Time and frequency scaling	$x\left(\frac{t}{a}\right)\quad a>0$	$aX(af)$
11.	Duality	$X(t)$	$x(-f)$
12.	Conjugation	$x^*(t)$	$X^*(-f)$
13.	Real part	$\mathrm{Re}\left[x(t)\right]$	$\frac{1}{2}\left[X(f)+X^*(-f)\right]$
14.	Imaginary part	$\mathrm{Im}\left[x(t)\right]$	$\frac{1}{2}j\left[X(f)-X^*(-f)\right]$
15.	Differentiation	$\frac{d^n}{dt^n}x(t)$	$(j2\pi f)^n X(f)$
16.	Integration	$\int_{-\infty}^{t} x(\tau)d\tau$	$\frac{X(f)}{j2\pi f}+\frac{1}{2}X(0)\delta(f)$

Discrete-Time Fourier Transform

The *discrete-time Fourier transform* (DTFT) is the appropriate Fourier technique to use in order to obtain a continuous-frequency spectrum for a signal that is a function of discrete time. The continuous-frequency spectrum obtained from the DTFT is periodic, with a period equal to T^{-1}, where T is the discrete-time sampling interval. The DTFT finds widespread use within DSP, primarily because a digital filter's unit sample response and frequency response comprise a DTFT pair.

The DTFT is defined by

$$X(e^{j\omega T}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega nT} \qquad (12.1)$$

and the corresponding inverse is given by

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega nT}\,d\omega \qquad (12.2)$$

where ω is the continuous radian frequency and T is the discrete-time sampling interval. The z transform is defined in Note 44 as

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \qquad (12.3)$$

If $e^{j\omega T}$ is substituted for z in this definition, the result is identical to Eq. (12.1). This result indicates that the DTFT is equal to the z transform of $x[n]$ evaluated on the unit circle in the z-plane.

Math Box 12.1

Deriving the Discrete-Time Fourier Transform

The delta function model of ideal sampling can be used to derive the discrete-time Fourier transform (DTFT) from the continuous-time Fourier transform (CTFT). Begin with the delta function model of ideal sampling applied to the continuous-time function, $x(t)$:

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

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

The CTFT for the sum in the equation above can be computed term by term. For any particular value of n, $x(nT)$ is a constant, so the CTFT can be written as

$$\mathcal{F}\{x_s(t)\} = \sum_{n=-\infty}^{\infty} \left(x(t)\mathcal{F}\{\delta(t-nT)\}\right)$$

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

Replacing $x(nT)$ with the discrete-time sequence notation $x[n]$ yields

$$X_s(\omega) = \sum_{n=-\infty}^{\infty} x[n]\exp(-j\omega nT)$$

which agrees with Eq. (12.1).

Table 12.1 Discrete-time Fourier transform pairs

$x[n]$	DTFT
1	$\sum_{k=-\infty}^{\infty} 2\pi\delta(\omega+2\pi k)$
$\delta[n]$	1
$u[n]$	$\frac{1}{1-e^{-j\omega}}+\sum_{k=-\infty}^{\infty}\pi\delta(\omega+2\pi k)$
$\exp(j\omega_0 n)$	$\sum_{k=-\infty}^{\infty} 2\pi\delta(\omega-\omega_0+2\pi k)$
$a^n u[n],\quad \lvert a\rvert<1$	$\frac{1}{1-a\exp(-j\omega)}$
$\frac{\sin\omega_c n}{\pi n}$	$\begin{cases}1, & 0\le\lvert\omega\rvert\le\omega_c\\ 0, & \omega_c<\lvert\omega\rvert\le\pi\end{cases}$
$w_{\text{rect}}[n]=\begin{cases}1, & -M\le n\le M\\ 0, & \text{otherwise}\end{cases}$	$\frac{\sin\left[\pi f(2M+1)\right]}{\sin(\pi f)}$

Table 12.2 Properties of the DTFT

Property	Time Sequence	DTFT
	$x[n]$	$X(e^{j\omega})$
	$y[n]$	$Y(e^{j\omega})$
Linearity	$ax[n]+by[n]$	$aX(e^{j\omega})+bY(e^{j\omega})$
Time shift	$x[n-m]$	$e^{-j\omega m}X(e^{j\omega})$
Time reversal	$x[-n]$	$X(e^{-j\omega})$
Frequency shift	$e^{j\omega_0 n}x[n]$	$X(e^{j(\omega-\omega_0)})$
Convolution	$x[n]\otimes y[n]$	$X(e^{j\omega})Y(e^{j\omega})$
Differentiation of transform	$nx[n]$	$j\frac{d}{d\omega}X\left(e^{j\omega}\right)$

Discrete Fourier Transform

The *discrete Fourier transform* (DFT) is perhaps the single most important mathematical tool in all of DSP. Unlike many other Fourier analysis techniques that can be applied only to signals that are expressed in the form of mathematical functions, the DFT can be implemented in practical systems to analyze sampled real-world signals numerically. The DFT also plays a role in certain types of filter design as well as in the efficient implementation of filter banks, transmultiplexers, and demodulators for multi-frequency communication signals such as *orthogonal frequency division multiplexing* (OFDM).

Direct computation of an N-point DFT involves a number of arithmetic operations that is proportional to N^2. There are many different algorithms that exploit periodicity, symmetry, and phase relationships in the sine and cosine functions to compute the DFT using significantly fewer arithmetic operations. These algorithms are collectively referred to as the *fast Fourier transform*

Essential Facts

- The DFT operates on an input record of N samples. The spectral interpretation of the DFT output assumes that the input samples are uniformly spaced in the input domain (usually time). If the input samples are not uniformly spaced, the conventional interpretation of the output will be incorrect.
- The mathematical derivation of the DFT is based on an assumption that the N-sample input record comprises exactly one period taken from a periodic input signal.
- The DFT produces an output record of N samples that are uniformly spaced in the output domain (usually frequency). The sampled nature of the output can be viewed mathematically as either a cause or a consequence of the assumed periodicity of the input.
- The DFT's output record represents exactly one period of an infinite periodic spectrum. The periodicity in frequency can be viewed mathematically as either a cause or a consequence of sampling in time.

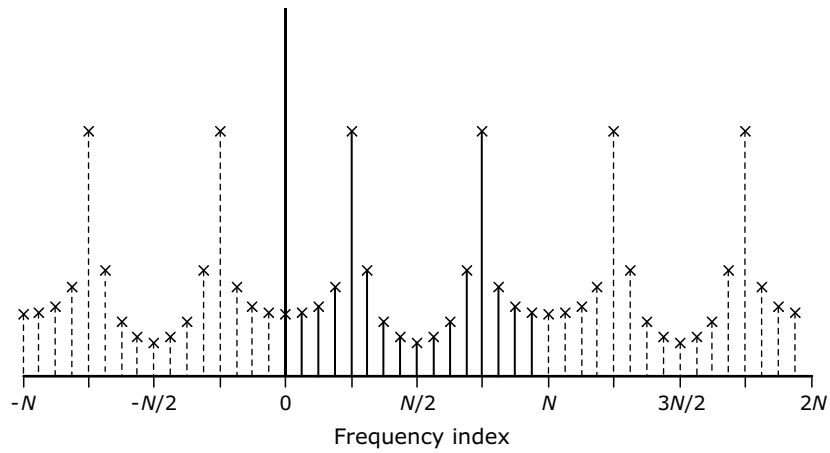


Figure 13.1 Implicit periodicity in the DFT spectrum. Solid drop lines indicate the "main" spectrum for $0 \le m \le (N-1)$. Dashed drop lines indicate periodic replications of the "main" spectrum.

(FFT). The most commonly used FFT algorithms require a number of arithmetic operations that is proportional to $N \log_2 N$. The existence of these computationally efficient FFT algorithms makes real-time DFT analysis a viable choice for many DSP applications. This note deals with exploiting fundamental DFT behaviors and properties that are independent of the specific FFT mechanization that is ultimately selected.

The *inverse* DFT, or IDFT, is sometimes called the *synthesis* DFT because it can be used to synthesize a time-domain signal from a frequency-domain specification. Similarly, the forward DFT is sometimes called the *analysis* DFT because it can be used to analyze the frequency content of a time-domain signal.

13.1 DFT Periodicity in the Frequency Domain

As discussed in Note 3, sampling in the time domain creates periodic spectral images in the frequency domain. How is this physical reality reflected in the mathematics of the DFT? In the equations that define the DFT, the frequency sequence, $X[m]$, is defined only

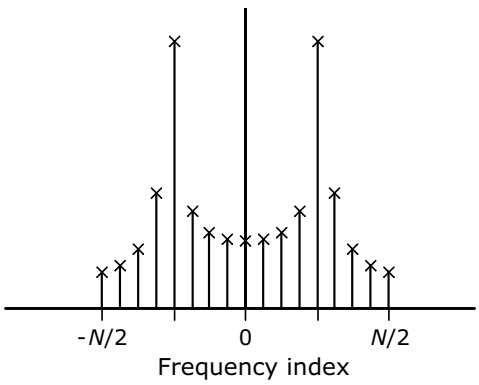


Figure 13.2 Two-sided presentation of DFT spectrum

Math Box 13.1

Discrete Fourier Transform

For a signal represented by an N-point sequence of samples, $x[n]$, for $n = 0, 1, \ldots, N-1$, having a sample interval of T, the *discrete Fourier transform* (DFT) can be used to compute an estimate of the signal's spectral content at the N discrete frequencies of $0, F, 2F, \ldots, (N-1)F$, where $F = (NT)^{-1}$. The DFT is defined by

$$X[m] = \sum_{n=0}^{N-1} x[n]\exp(-j2\pi mFnT)$$

$$= \sum_{n=0}^{N-1} x[n]\cos(2\pi mFnT) - j\sum_{n=0}^{N-1} x[n]\sin(2\pi mFnT)$$

$$\text{for } m = 0,1,\ldots,N-1 \qquad \textbf{(MB 13.1)}$$

Inverse DFT

Given all N samples of the frequency sequence, $X[m]$, it is possible to recover the original sequence, $x[n]$, using the *inverse* DFT (IDFT) defined by

$$x[n] = \sum_{n=0}^{N-1} X[m]\exp(j2\pi mFnT) \qquad \textbf{(MB 13.2)}$$

Alternate DFT Definition

Because $FT = N^{-1}$, the DFT from Eq. (MB 13.1) can be recast in terms of the sequence length N without any explicit appearance of the sampling interval T, or the frequency increment F:

$$X[m] = \sum_{n=0}^{N-1} x[n]\exp\left(\frac{-j2\pi mn}{N}\right)$$

$$= \sum_{n-0}^{N-1} x[n]\cos\left(\frac{2\pi mn}{N}\right) \qquad \textbf{(MB 13.3)}$$

$$- j\sum_{n-0}^{N-1} x[n]\sin\left(\frac{2\pi mn}{N}\right)$$

This formulation allows construction of generic DFT software routines that depend only upon the sequence length. In general, the frequency sequence, $X[m]$, is complex-valued even in cases where the time sequence, $x[n]$, is real-valued. However, when $x[n]$ is an even-symmetric function in time, the corresponding DFT, $X[m]$, is even-symmetric and real-valued. The roles of complex-valued time sequences are discussed in Note 60.

for $0 \le m < N$. What happens if we try to generate $X[m]$ for $m \ge N$ or for $m < 0$? Applying a few trigonometric identities and some algebraic manipulation to the DFT equations reveals that $X[m + kN] = X[m]$ for k, an integer. In other words, if the index m is not constrained to the interval $0 \le m < N$, the DFT produces a discrete frequency sequence that is periodic with a period of N samples, as illustrated in Figure 13.1. This periodicity is consistent with the physical results observed in real-world sampling experiments. The DFT in Eqs. (MB 13.1) and (MB 13.3) *could* be restated as being valid for all integer values of m. However, all of the useful spectral information is contained in a single period—the choice of the particular period corresponding to $0 \le m < N$ is somewhat arbitrary.

Despite an arbitrary choice of the period over which the DFT results are typically defined, periodicity in the results allows for some flexibility in how they are displayed. Consistent with the definitions (MB 13.1) and (MB 13.3), DFT results are often displayed for frequency indices from zero through $N-1$. This format requires that the usual mathematical definitions of even and odd symmetry be recast to accommodate the fact that, in this format, the "negative" frequencies correspond to the indices, m, for $(N/2) \le m < N$. It is also fairly common to display results for frequency indices in the range $\pm N/2$ using a "two-sided" format, as illustrated in Figure 13.2. This two-sided format is sometimes a more intuitive way to look at the frequency-domain symmetries. It should be noted that a two-sided plot involves $N+1$ points because the point at $m = N/2$ is duplicated for $m = -N/2$.

Math Box 13.2

Deriving the DFT from the DTFT

It is possible to create a DFT by applying the *discrete-time Fourier transform* (DTFT) to a finite-duration, discrete-time sequence, and evaluating the DTFT for only a finite set of specific discrete frequencies. The DTFT is defined in Note 12 as

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]\exp(-jn\omega T) \qquad \text{(MB 13.4)}$$

If we assume that $x[n]$ is nonzero for only the N samples corresponding to $n = 0, 1, \ldots, N-1$, then the summation limits in (MB 13.4) can be changed to yield

$$X(\omega) = \sum_{n=0}^{N-1} x[n]\exp(-jn\omega T) \qquad \text{(MB 13 5)}$$

However, $X(\omega)$ is still a function of continuous frequency, ω. If we attempt to define a DFT by evaluating Eq. (MB 13.5) at an arbitrary set of discrete frequencies, the result is generally not inverse-transformable back into the original sequence, $x[n]$. However, if we relax the constraints on $x[n]$ somewhat, it becomes possible to obtain an *invertible* spectral representation that involves nonzero values at only N discrete frequencies. Specifically, we define $x_P[n]$ to be the *periodic extension* of $x[n]$, such that the original N samples of $x[n]$ comprise one period of the new periodic signal:

$$x_P[n] = x[k \bmod N] \quad \text{for all } k \in \mathbb{Z}$$

As discussed previously, sampling in the time domain leads to periodicity in the frequency domain, and periodicity in the time domain leads to sampling in the frequency domain. The DTFT of $x_P[n]$ is periodic, with a period equal to the original sampling frequency. Furthermore, due to the time-domain periodicity of $x_P[n]$, the DTFT is nonzero at only N discrete frequencies per period. Specifically, the discrete frequencies for the period occupying the interval $0 \le \omega < T^{-1}$ is

$$\omega_m = \frac{2\pi m}{NT} \quad \text{for } m = 0, 1, \ldots, N-1$$

Substituting ω_m for ω in Eq. (MB 13.5) and converting to discrete sequence notation yields

$$X[m] = \sum_{n=0}^{N-1} x[n]\exp\left(-j\frac{2\pi nm}{N}\right) \quad \text{for } m = 0, 1, \ldots, N-1$$

which is the DFT defined by (MB 13.1).

13.2 Periodicity in the Time Domain

In the DSP literature, the DFT is described as operating on a time signal that is both discrete in time and also periodic, with a period of N samples. However, in the equations that define the DFT, values of the time sequence $x[n]$ are needed only for values of n from 0 through $N-1$; so why does it matter how $x[n]$ behaves outside the interval $0 \le n < N$? Furthermore, as given in Eq. (MB 13.2), the definition of the inverse DFT seems to imply that $x[n]$ is defined only for values of n inside this interval. What does it really mean to say that the time signal that serves as input to the DFT is periodic?

The results produced by the DFT are as though the original N samples of $x[n]$ have been duplicated along the time axis, as shown in Figure 13.3. In this periodic version of the signal, the sample at $n = 0$ is immediately preceded by a copy of the sample from $n = N-1$, and the sample at $n = N-1$ is immediately followed by a copy of the sample from $n = 0$. If there are discontinuities at these juxtaposed samples, the spectrum computed by the DFT may exhibit significant high-frequency content not present in the original signal. This spurious high-frequency content is sometimes called *leakage*. The usual technique for dealing with leakage is to multiply the time sequence by a *window* function that tapers the input sequence to values of zero or near zero at each end. This windowing technique effectively removes the discontinuities that cause leakage. Techniques for analyzing leakage are presented in Note 14, and leakage-reducing windows are discussed in Note 24.

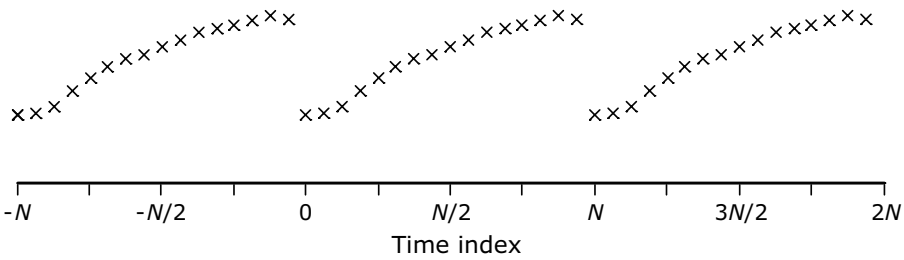


Figure 13.3 Periodically extended presentation of DFT input record

Analyzing Signal Truncation

An N-point DFT is often used to analyze a sampled signal that is much longer in duration than N samples. Truncating the DFT input sequence usually introduces spurious frequency content, called leakage, that is not present in the original signal. In order to analyze the effects of truncation, we need to start with a mathematical model of the truncation process. The relationships among the various signals and spectra discussed in this note are summarized in Figure 14.1, below. These relationships are used in Note 15 to explore DFT leakage for sinusoidal input signals.

Key Points

- Truncating the DFT input sequence to a length of N samples can be mathematically modeled as multiplying the sequence by a rectangular window sequence that has non-zero values only for sample indices 0 through $N-1$.
- The spectrum changes due to truncation in the time domain can be assessed by convolving the spectrum of the untruncated input signal with the spectrum of the rectangular window sequence.
- The spectrum of the rectangular window sequence can be defined in terms of the Dirichlet kernel, D_N, as

$$W(f) = N \exp\left[-j\pi f(N-1)\right] D_N(2\pi f)$$

where $D_N(x)$ can be obtained via the MATLAB call `diric(x,N)`.

14.1 Truncation

Truncating the original signal sequence (block 1 in Figure 14.1) to a length of just N samples can be viewed as multiplying the longer sequence with an N-point rectangular window sequence (block 2 in

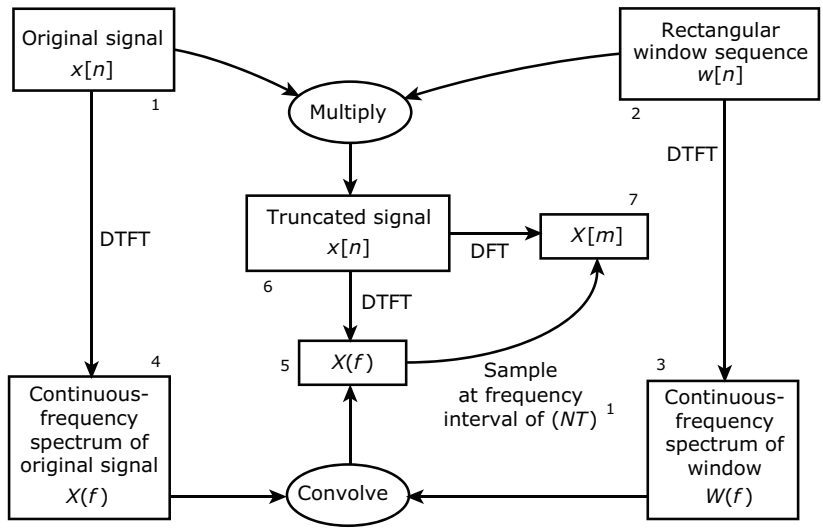


Figure 14.1 Relationships among the various signals and spectra involved in the analysis of leakage in the DFT

Figure 14.1), defined by

$$w[n]=\begin{cases}1 & 0\le n<N\\ 0 & \text{otherwise}\end{cases} \tag{14.1}$$

The multiplication indicated in the diagram is not actually performed—it is just a mathematical model of truncation that can be used to predict how the truncated signal's spectrum will differ from the original signal's spectrum.

Both the DTFT and DFT exhibit the property that multiplication of two functions in the time domain is equivalent to convolving the two functions' DTFTs, or DFTs in the frequency domain. Therefore, one way to assess the frequency-domain impact of truncating a time sequence is to convolve the DTFT (block 3 in Figure 14.1) of an N-point rectangular window with the spectrum (block 4) of the untruncated time sequence. The result of this convolution will be the DTFT spectrum (block 5) of the truncated signal (block 6) that was provided as input to the DFT. The DFT output (block 7) can also be obtained by sampling the DTFT spectrum (block 5) at a frequency interval of $F = (NT)^{-1}$.

Using the relationships depicted in Figure 14.1 to analyze the spectral impact of truncation in the time domain depends on being able to compute the DTFT of a discrete-time rectangular window.

14.2 DTFT of a Rectangular Window

The DTFT of the rectangular window defined by Eq. (14.1) is given by

$$W(f)=\exp\left[-j\pi f(N-1)\right]\frac{\sin(N\pi f)}{\sin(\pi f)} \tag{14.2}$$

(Note: All of the equations in this note follow the common convention of using a normalized sampling interval, $T = 1$.)

The exponential factor in Eq. (14.2) has unity magnitude for all values of f, and represents a simple linear phase shift. This phase shift is a consequence of defining the window over the interval $0 \le n < N$. If the rectangular window is defined to be symmetric about the origin, then the DTFT of the window will be

$$W_{\text{sym}}(f)=\frac{\sin(N\pi f)}{\sin(\pi f)} \tag{14.3}$$

The ratio of sine terms in Eqs. (14.2) and (14.3) is called the *Dirichlet kernel*[1] and can be computed using the `diric` function in MATLAB. A call to `diric(x,N)` returns $D_N(x)$ as given by

$$D_N(f)=\frac{\sin(Nx/2)}{N\sin(x/2)} \tag{14.4}$$

The DTFT of the rectangular window can be expressed in terms of D_N as

$$W(f)=N\exp\left[-j\pi f(N-1)\right]D_N(2\pi f) \tag{14.5}$$

When N is odd, a rectangular window symmetric about $n = 0$ can be defined as

$$w_{\text{sym}}[n]=\begin{cases}1 & \dfrac{-(N-1)}{2}\le n\le\dfrac{N-1}{2}\\ 0 & \text{otherwise}\end{cases} \tag{14.6}$$

and D_N is periodic with a period of 2π, as shown in Figure 14.2. When N is even, D_N is periodic with a period of 4π, as shown in Figure 14.3.

A period of 4π appears to violate the rule that a DTFT spectrum is always periodic with a period of T^{-1} Hz, or $2\pi/T$ radians per second. In trying to resolve this apparent violation, we discover that defining an even-length window that is symmetric about $n = 0$ is problematic. For even N, the closest we can get to symmetry would be to have

1. The MATLAB help files refer to D_N as the "Dirichlet function," but in nearly all of the mathematical literature, D_N (or minor variants thereof) is called the "Dirichlet kernel." The term "Dirichlet function" is usually reserved for the function defined as

$$f(x)=\lim_{m\to\infty}\lim_{n\to\infty}\cos^{2n}(m!\pi x)=\begin{cases}1 & \text{for } x \text{ rational}\\ 0 & \text{for } x \text{ irrational}\end{cases}$$

one sample at $n = 0$, with $N/2$ samples to one side of zero, and $(N/2) - 1$ samples to the other side of zero. Rather than attempting to define a symmetric window with samples at odd multiples of $T/2$ instead of at integer multiples of T, we simply note that any practical use of a an even-length rectangular window involves the window defined over the interval $0 \le n < N$, as in Eq. (14.1), and having the DTFT given by Eq. (14.5). For even N, the exponential factor in Eq. (14.5) corresponds to delaying a symmetric window (with samples at odd multiples of $T/2$) by $(N - 1)/2$ sample times. The samples of the delayed result occur at integer multiples of T, thereby "rectifying" the spectrum and making it periodic in 2π, as shown in Figure 14.4.

The Fourier transform of a rectangular pulse in continuous-time is a sinc function, and one period of the Dirichlet kernel is similar in appearance to a sinc function. Some DSP texts incorrectly claim that the DTFT for a discrete-time rectangular window is a "sinc function in frequency." However, the sinc function and the Dirichlet kernel really are different beasts. Some sources accentuate the difference but acknowledge the similarities between the discrete-time and continuous-time cases by referring to the Dirichlet kernel as the *aliased sinc function*.

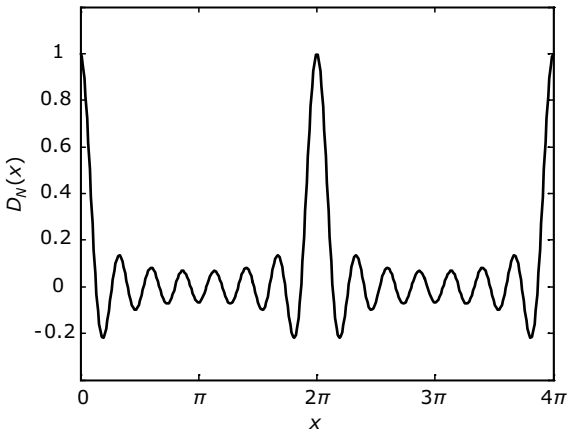


Figure 14.2 Dirichlet kernel for N = 15

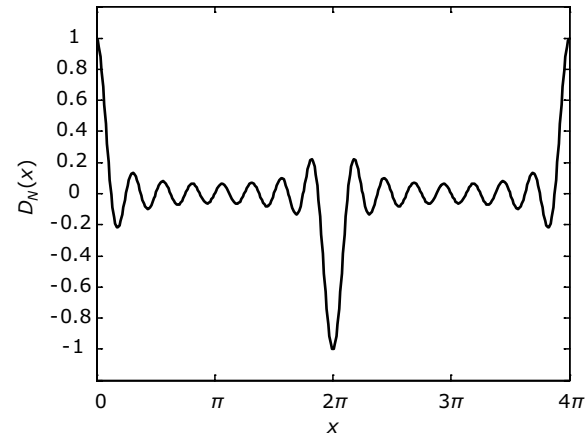


Figure 14.3 Dirichlet kernel for $N = 16$

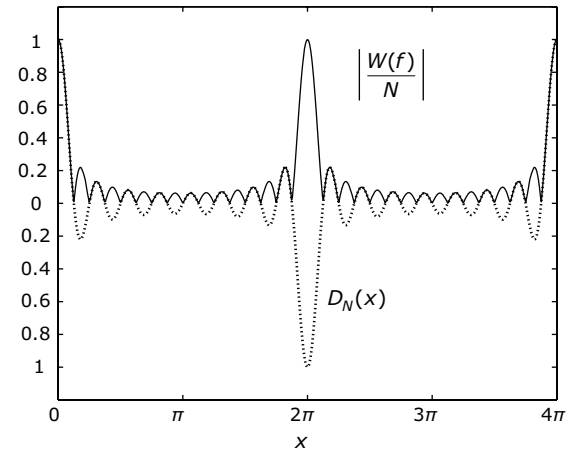


Figure 14.4 Comparison between raw Dirichlet kernel for $N = 16$ (dotted trace) and the same kernel after phase shifting via the exponential factor from Eq. (14.2) (solid trace)

Exploring DFT Leakage

This note uses the model of signal truncation developed in Note 14 and applies this model to sinusoidal signals to demonstrate how leakage can corrupt a DFT result—often to the point where the DFT results can be misleading when used to estimate signal frequency. The insights provided by this demonstration are the foundation for the introduction and analysis of windowing techniques in Notes 23 through 27.

The effects of truncation are absent in cases where the input signal is a sinusoid having a frequency that exactly matches one of the DFT bin frequencies. For example:

- Consider a DFT with $N = 32$ and $T = 1$.
- The signal of interest is $x(t) = \cos(2\pi f_c t)$, where $f_c = 4F$.
- Figure 15.1 shows the DFT magnitude spectrum for this signal; all of the response is concentrated in the bins corresponding to $\pm 4F$, as expected.

However, in general, the effects of truncation are very noticeable. Again, consider a DFT with N = 32 and T = 1:

- The signal of interest is changed to $x(t) = \cos(2\pi f_c t)$, where $f_c = 4.5F$.
- Intuition suggests that the postive-frequency response would be split equally between bins 4 and 5, and that the negative-frequency response would be split equally between bins 27 and 28.
- Figure 15.2 shows the DFT magnitude spectrum for this signal. The responses in bins 4 and 5 are not equal. Furthermore, there is a non-zero response in each of the 32 bins.

How can a small change in input frequency result in Figures 15.1 and 15.2 being so different? To explore leakage in a quantitative fashion, we need to draw upon the results of Note 14, as described in Recipe 15.1 and demonstrated in Example 15.1.

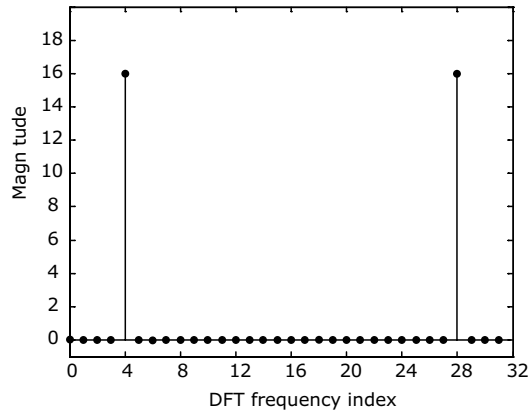


Figure 15.1 DFT magnitude spectrum of a rectangularly windowed, 32-sample segment of a sinusoid having a frequency of $f_c = 4F$.

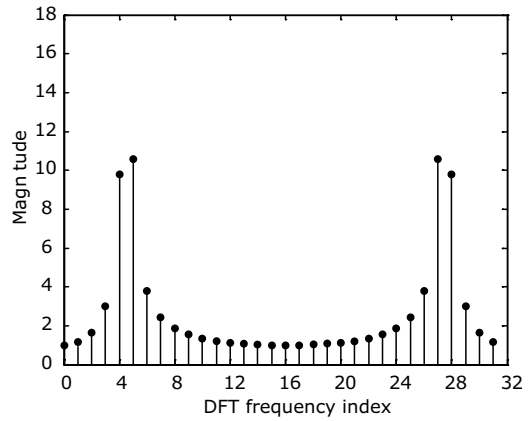


Figure 15.2 DFT magnitude spectrum of a rectangularly windowed, 32-point segment of a sinusoid having a frequency of $f_c = 4.5F$

Example 15.1

DTFT for Sinusoidal Pulses

Consider a DFT with $N = 32$ and $T = 1$. The signal of interest is $x(t) = \cos 2\pi f_c t$. Apply Recipe 15.1 for the cases of $f_c = 4F$ and $f_c = 4.5F$.

- The effects of truncation are hidden in cases where the frequency of the input sinusoid exactly matches one of the DFT bin frequencies. Specifically, in the case of $f_c = 4F = 4/(NT)$, the magnitude spectrum has a null at every bin frequency except for the bins corresponding to $\pm 4F$, as shown in Figure 15.3. (Note: Figures 15.3 and 15.4 are unusual in that they each contain both a DFT result having a discrete-frequency abscissa and a DTFT result having a continuous-frequency abscissa. See Math Box 15.1 for a discussion of how to set up the equations so that the DFT and DTFT results plot correctly on the same set of axes.)
- When the sinusoid's frequency does not exactly equal one of the DFT bin frequencies, there is generally some nonzero magnitude response in every bin. This is illustrated in Figure 15.4 for $f_c = 4.5F$. In this case, the main lobes of $X(f)$ are wide enough to each span two DFT bins, thus making it difficult to discern the frequency of the sinusoid.
- Intuition might suggest that a sinusoid at $fc = 4.5F$ should have equal magnitude responses in the bins for $f = 4F$ and $f = 5F$, but examination of Figure 15.4 reveals that this is not the case. The responses in bins 4 and 5 are unequal because of asymmetries in the side lobe that take place in the interval going from bin 5 to bin 27 and in the interval going from bin 28 wrapping around to bin 4.

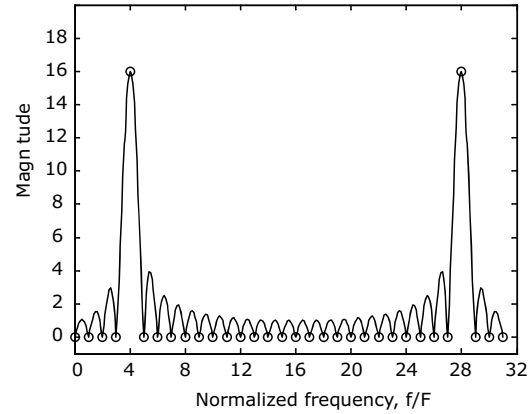


Figure 15.3 Fourier spectra of a rectangularly windowed segment of a sinusoid having a frequency of $f_c = 4F$. The solid trace represents the magnitude of the DTFT, and the small circles indicate the magnitudes produced by a 32-point DFT.

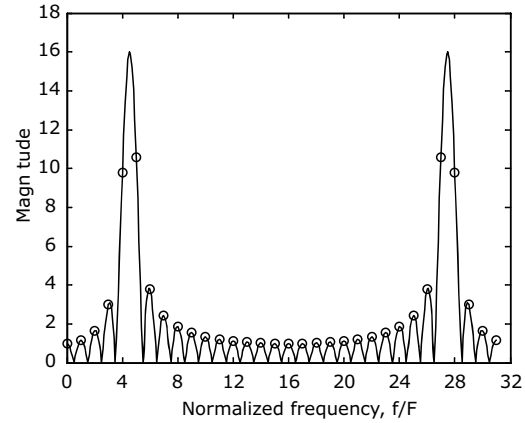


Figure 15.4 Fourier spectra of a rectangularly windowed segment of a sinusoid having a frequency of $f_c = 4.5F$. The solid trace represents the magnitude of the DTFT, and the small circles indicate the magnitudes produced by a 32-point DFT.

Math Box 15.1

Plotting DFT and DTFT Results Together in the Same Graph

Figures 15.3 and 15.4 are unusual in that they each contain both a DFT result and a DTFT result. These two types of transform results are usually plotted on diffrently scaled axes.

- The result of a DTFT is typically plotted against a real-valued frequency axis. Because the DTFT result is periodic with a period of T^{-1}, only a single period of the result is usually plotted. For a two-sided plot format, the period plotted is either for cyclic frequencies in the interval $[-\frac{1}{2}T^{-1}, \frac{1}{2}T^{-1}]$ or for radian frequencies in the interval $[-\pi T^{-1}, \pi T^{-1}]$. For a single-sided plot format, the period plotted is either for cyclic frequencies in the interval $[0, T^{-1}]$ or for radian frequencies in the interval $[0, 2\pi T^{-1}]$. When the conventional normalization of $T = 1$ is followed, the plotted frequency interval becomes either $-\frac{1}{2} \le f \le \frac{1}{2}$ or $-\pi \le \omega \le \pi$, for two-sided plots, and either $0 \le f \le 1$ or $0 \le \omega \le 2\pi$, for one-sided plots.
- On the other hand, DFT results are usually plotted against a sequence of integer-valued *frequency indices* or "bin numbers" $m = 0, 1, 2, \ldots, N-1$. The cyclic frequency corresponding to DFT bin m is given by

$$f_n = mF = \frac{m}{NT}$$

  To cover the same $[0, T^{-1}]$ frequency interval as the DTFT, the DFT frequency indices would need to run for $N + 1$ values from $m = 0$ through $m = N$.
- Because this particular note is concerned with leakage in the DFT, and the DTFT is playing only a supporting role, it seems logical to keep the plotting format consistent with typical DFT plots. Therefore, for Figures 15.3 and 15.4, the values of this normalized frequency are numerically equal to the corresponding DFT bin numbers.

Recipe 15.1

Computing the DTFT for Sinusoidal Pulses

- An ideal sinusoid of cyclic frequency f_c has a continuous-time Fourier transform whose magnitude consists of two Dirac delta functions, one located at $f = f_c$ and the other located at $f = -f_c$. The phasing of these two delta functions depends upon the phase of the sinusoid.
  - For $x(t) = \cos 2\pi f_c t$, the delta functions are both real and positive.
  - For $x(t) = \sin 2\pi f_c t$, the delta functions are both purely imaginary; the delta function at $f = f_c$ is negative and the other is positive.
- If $x(t)$ is multiplied by a time domain window, $w(t)$, the DTFT for the resulting product is equal to the convolution of $X(f)$ and $W(f)$.
- The result of convolving $W(f)$ with a delta function, $\delta(f_c)$, shifts $W(f)$ so that its peak is centered at f_c. The Fourier spectrum for a rectangularly windowed sinusoid is the sum of two shifted instances of $W(f)$, with one instance centered at $f = f_c$ and one instance centered at $f = -f_c$:

$$\begin{aligned} Y(f) &= X(f) \otimes W(f) \\ &= uW(f - f_c) + vW(f + f_c) \end{aligned} \tag{15.1}$$

  where

$$(u, v) = \begin{cases} (1,1) & x(t) = \cos(2\pi f_c t) \\ (-j, j) & x(t) = \sin(2\pi f_c t) \end{cases} \tag{15.2}$$

- The DTFT for a rectangularly windowed, weighted sum of M sinusoids is given by

$$Y(f) = \sum_{m=1}^{M} A_m \left[u_m W(f - f_m) + v_m W(f + f_m) \right]$$

Exploring DFT Resolution

This note provides demonstrations of a technique called *zero padding,* which is often used to obtain a finer frequency-domain increment when using a discrete Fourier transform (DFT). These demonstrations show that although zero padding improves the *observability* of some frequency domain features, it does not improve the *frequency resolution* of a DFT.

Sometimes important features of a signal's spectrum fall at frequencies between the DFT bin frequencies, making potentially important features of a signal's spectrum difficult to detect or observe. This inability to see critical details that fall at frequencies that are not integer multiples of F = NT – 1 is sometimes called the picket fence effect because it is similar to trying to view the details of a scene while looking through a picket fence—some details are clearly visible, while others are hidden by the slats of the fence. The picket fence effect can be mitigated to some extent by reducing the size of the frequency increment, F, but the effect can never be completely eliminated as long as we continue to operate in the discrete frequency domain.

In a DFT, the relationship $FNT = 1$ always holds. Therefore, the only way that F can be reduced is by increasing T or N. In the design of most signal processing strategies, there is usually not much "wiggle room" when it comes to increasing the value of T. Any viable approach for reducing F must therefore involve increasing N.

Key Points

- Sometimes important features of a signal's spectrum fall at frequencies between DFT bin frequencies, making these features difficult to detect or observe. This phenomenon is sometimes called the *picket fence effect*.
- One approach for mitigating the picket fence effect involves padding the DFT input sequence with a number of zero-valued samples so that a longer DFT with more finely spaced frequency bins can be used. This approach is called *zero-padding*.
- When a DFT input sequence is zero-padded, the DFT results no longer represent samples of the unpadded signal's DTFT. Thus, zero padding may make it easier to observe the approximate frequency of a spectral peak, but the observed amplitude of the peak generally is incorrect.
- With regard to the DFT, *resolution* is the ability to distinguish two closely spaced features in a signal's spectrum. Despite some claims to the contrary, zero padding does not improve the DFT's resolution. In applications where resolution is important, the only viable choice is to use a larger DFT that processes a longer sequence of the original signal.

16.1 Zero Padding

A classic approach used for decreasing F is called *zero padding*. This technique involves padding the original N-point input sequence with p zero-valued samples to create an input sequence of length $L = N + p$. The straightforward approach is to simply extend the DFT to some longer length, L, and collect this new larger number of input samples before computing the L-length DFT.

Example 16.1

Zero Padding Improves Observability

The signal of interest (SOI) is a sinusoid having a frequency of $f_c = 0.140625$ Hz. For the case of $T = 1$ and $N = 32$, the sinusoid's frequency can be expressed in terms of F as $f_c = 4.5F$.

- The positive-frequency half of a 32-point DFT magnitude spectrum for the SOI is shown in Figure 16.1, along with the corresponding DTFT magnitude spectrum.[1] The peak in the DTFT occurs at $f_c = 4.5F$, but the DFT provides outputs only at integer multiples of F, making it impossible to observe the peak directly in the DFT result.
- Note that the result in Figure 16.1 runs contrary to intuition, which expects the responses at $f = 4F$ and $f = 5F$ to be equal if the signal's true frequency lies exactly halfway in between $4F$ and $5F$. This observed asymmetry is because the sum of the Dirichlet kernels for $f = f_c$ and for $f = -f_c$ are not symmetric about f_c.
- In an attempt to better observe the frequency of the spectral peak, we increase the DFT length from $N = 32$ to $L = 64$ by appending 32 zero-valued samples to the original input sequence. The frequency increment is halved ($F' = F/2$), and the frequency corresponding to bin 9 in the new, longer DFT exactly equals the frequency of the sinusoid:

$$f_c = 4.5F = 4.5(2F') = 9F' \qquad (16.1)$$

As shown in Figure 16.2, the result for bin 9 is a peak, but the amplitude of this peak does not match the peak amplitude of the DTFT result, which is plotted for comparison. In fact, none of the 64-point DFT bins appear to be samples of the DTFT result. The DTFT plotted in the figure is the DTFT for a 64-sample segment of the SOI. However, the 64-point DFT results do agree with the DTFT for a 32-sample segment of the SOI, as shown in Figure 16.2. This observation should not come as a surprise; the segment of SOI is only 32 samples long whether zero-padding is employed or not.

1. For the figures in this note, the DTFT results and the DFT results are each plotted against a different abscissa, as explained in Math Box 16.1.

Zero padding does improve observability of some spectrum features, but contrary to claims often made in the DSP literature, zero padding does not improve frequency resolution in the frequency domain.

As already stated, the zero padding technique involves extending the original N-point input sequence by appending p zero-valued samples to create an input sequence of length $L = N + p$. An L-point DFT is then computed instead of an N-point DFT. The frequency domain values then fall at integer multiples of $F' = (LT)^{-1}$ rather than at multiples of $F = (NT)^{-1}$. Because $L > N$, the new frequency increment of F' is finer than the original increment of F, thereby improving observability, as demonstrated in Example 16.1.

Because zero padding is able to provide improved observability in the discrete-frequency domain, some texts incorrectly claim that zero padding increases the resolution of the DFT. Resolution is the ability to distinguish two closely spaced features in a signal's spectrum. As shown in Example 16.2, zero padding does not improve the DFT's resolution.

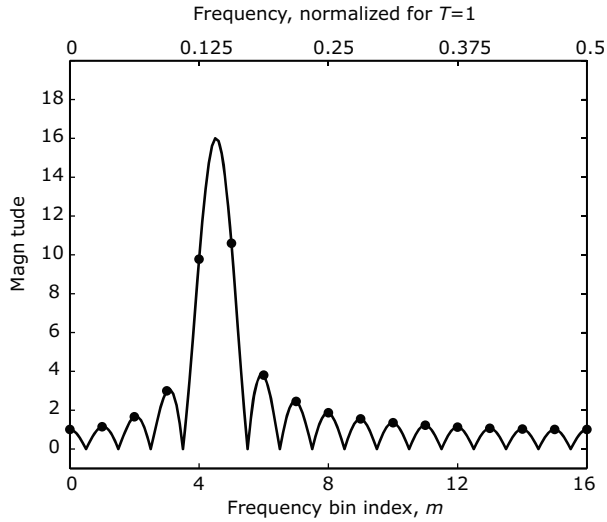


Figure 16.1 Fourier spectra from Example 16.1. The spectra are for a rectangularly windowed segment of a sinusoid having a frequency of $f_c = 4.5F$. The solid trace represents the magnitude of the 32-point DTFT, and the dots indicate the magnitude produced by a 32-point DFT.

Math Box 16.1

Frequency Scaling in Figures 16.1 through 16.6

- The DTFT results in Note 9 are plotted against a normalized frequency f/F, for which integer values correspond to DFT numbers.
- In this note, results for DFTs having different values of F are plotted along with DTFT results. Therefore, to avoid confusion regarding the value of F used for normalization, the DTFT results are plotted against a normalized frequency axis, which for $T = 1$ has a value of 0.5 at the frequency bin corresponding to DFT bin number N/2.
- Because frequencies scaled in this way are not numerically equal to DFT bin numbers (as they are in Note 9), each plot has two frequency axes. The solid trace, representing the DTFT result, is plotted against the top axis, and the dots, representing the DFT result, are plotted against the bottom axis in each figure.

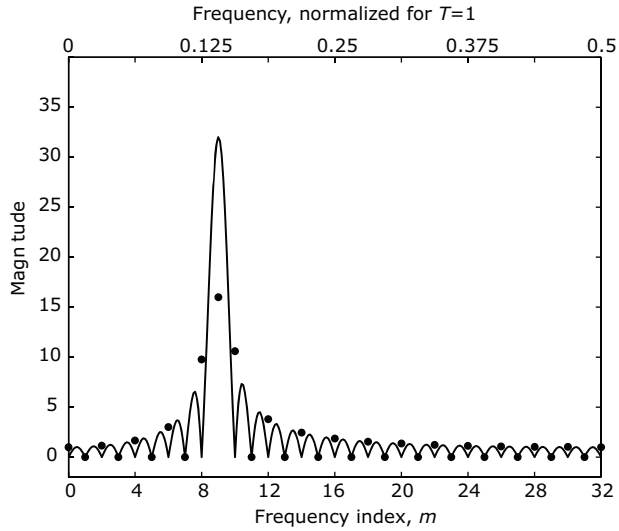


Figure 16.2 Fourier spectra from Example 16.1. The spectra are for a rectangularly windowed segment of a sinusoid having a frequency of $f_c = 4.5F$. The dots indicate the magnitudes produced by a 64-point DFT of a 32-sample signal segment that has been padded with 32 zero-valued samples. The solid trace represents the magnitude of the DTFT for a 64-point segment of the original sinusoid.

Example 16.2

Zero Padding Does Not Improve Resolution

The signal of interest (SOI) is the sum of two sinusoids—one having a frequency of $f_1 = 0.146875$ Hz and the other having a frequency of $f_2 = 0.165625$ Hz. For the case of $T = 1$ and $N = 32$, the sinusoids' frequencies can be expressed in terms of F as $f_1 = 4.7F$ and $f_2 = 5.3F$.

- The positive-frequency half of a 32-point DFT magnitude spectrum for the SOI is shown in Figure 16.4, along with the corresponding DTFT magnitude spectrum. There is a peak in both the DTFT response and in the DFT response at $f = 5F$, which is midway between $4.7F$ and $5.3F$. We are not able to distinguish the two different frequency components in this result.
- In an attempt to improve the resolution of the results, we increase the DFT length from $N = 32$ to $L = 64$ by appending 32 zero-valued samples to the original input sequence. As shown in Figure 16.5, the DFT result still consists of samples from the DTFT for a 32-sample segment of the SOI. All that zero padding has been able to accomplish is to cause the DFT bins to be more closely spaced in frequency.

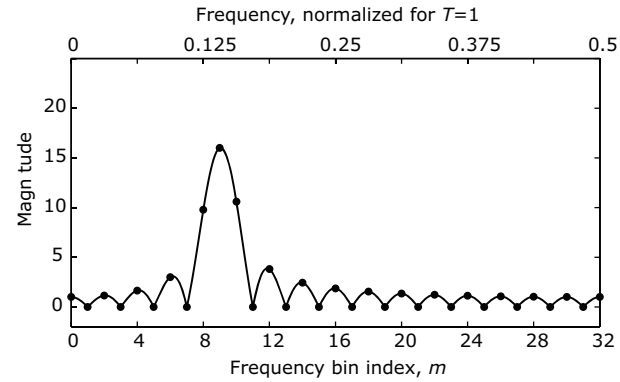


Figure 16.3 Fourier spectra from Example 16.1. The spectra are for a rectangularly windowed segment of a sinusoid having a frequency of $f_c = 4.5F$. The dots indicate the magnitudes produced by a 64-point DFT of a 32-sample signal segment that has been padded with 32 zero-valued samples. The solid trace represents the magnitude of the DTFT for a 32-point segment of the original sinusoid.

16.2 Lengthening the DFT

The straightforward approach to improving both observability and resolution in the DFT result is simply to extend the DFT to some longer length, L, and collect this new, larger number of input samples before computing the DFT. Example 16.3 applies this approach to the two-tone signal from Example 16.2 .

Example 16.3

Lengthening the DFT Improves Resolution

The signal of interest (SOI) is the sum of two sinusoids—one having a frequency of $f_1 = 0.146875$ Hz and the other having a frequency of $f_2 = 0.165625$. For the case of $T = 1$ and $N = 32$, the sinusoids' frequencies can be expressed in terms of F as $f_1 = 4.7F$ and $f_2 = 5.3F$.

- The positive-frequency half of a 32-point DFT magnitude spectrum for the SOI is shown in Figure 16.4, along with the corresponding DTFT magnitude spectrum. There is a peak in both the DTFT response and in the DFT response at $f = 5F$, which is midway between $4.7F$ and $5.3F$. We are not able to distinguish the two different frequency components in this result.
- In an attempt to improve the resolution of the results, we increase the DFT length from $N = 32$ to $L = 64$, and using 64 samples of the SOI, compute the magnitude response shown in Figure 16.6. The DFT result shows two distinct peaks at bins 9 and 11, with a notch between the peaks at bin 10.

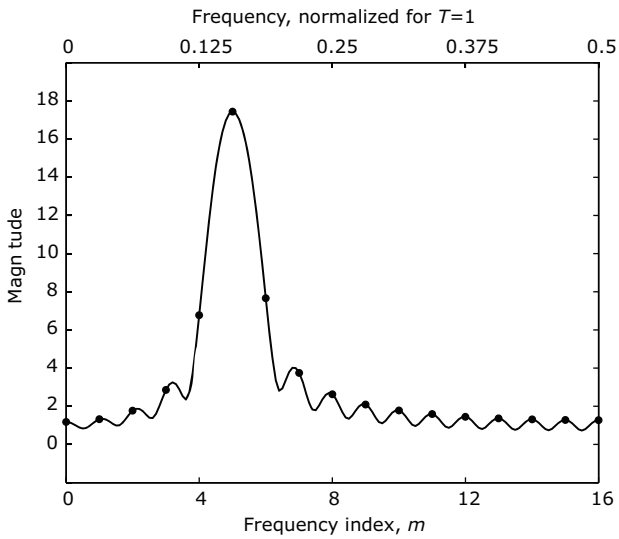


Figure 16.4 Fourier spectra from Examples 16.2 and 16.3. The spectra are for a rectangularly windowed segment of a signal consisting of two sinusoids having frequencies of $f_1 = 0.146875 = 4.7F$ and $f_2 = 0.165625 = 5.3F$. The solid trace represents the magnitude of the 32-point DTFT, and the dots indicate the magnitudes produced by a 32-point DFT.

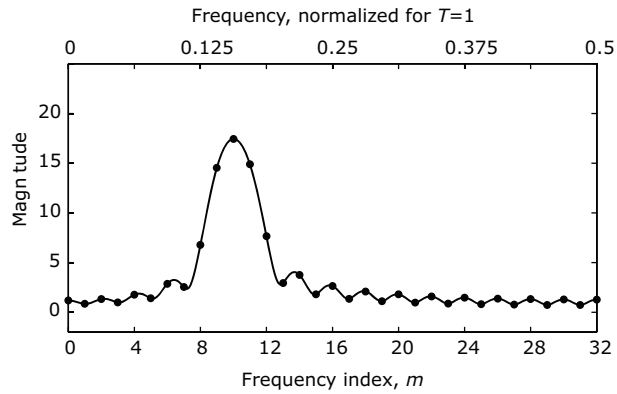


Figure 16.5 Fourier spectra from Example 16.2. The spectra are for a rectangularly windowed segment of a signal consisting of two sinusoids having frequencies of $f_1 = 0.146875 = 4.7F$ and $f_2 = 0.165625 = 5.3F$. The solid trace represents the magnitude of the DTFT for a 32-sample segment of the signal. The dots indicate the magnitudes produced by a 64-point DFT of a 32-point segment of the signal padded with 32 zero-valued samples.

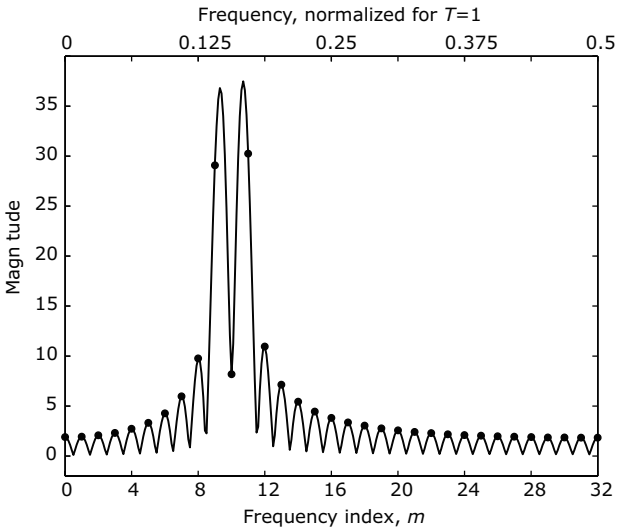


Figure 16.6 Fourier spectra from Example 16.3. Spectra are for a rectangularly windowed segment of a signal consisting of two sinusoids having frequencies of $f_1 = 0.146875 = 4.7F$ and $f_2 = 0.165625 = 5.3F$. The solid trace represents the magnitude of the DTFT for a 64-sample segment of the SOI, and the dots indicate the magnitudes produced by a 64-point DFT.

FFT: Decimation-in-Time Algorithms

Consider the discrete Fourier transform for an N-point sequence

$$W_N = \exp\left(\frac{-j2\pi}{N}\right) \tag{17.1}$$

where

$$X[m] = \sum_{n=0}^{N-1} x[n] W_N^{mn}$$

For even N, the DFT summation can be split into two separate summations—one for the even-indexed samples of $x[n]$ and one for the odd-indexed samples.

$$\begin{aligned} X[m] &= \sum_{n=0}^{N} x[n] W_N^{mn} \\ &= \sum_{n=0}^{N/2-1} x[2n] W_N^{2nm} + \sum_{n=0}^{N/2-1} x[2n+1] W_N^{(2n+1)m} \\ &= \sum_{n=0}^{N/2-1} x[2n] W_N^{2nm} + W_N^m \sum_{n=0}^{N/2-1} x[2n+1] W_N^{2nm} \\ &= \sum_{n=0}^{N/2-1} x_{\text{even}}[n] W_{N/2}^{nm} + W_N^m \sum_{n=0}^{N/2-1} x_{\text{odd}}[n] W_{N/2}^{nm} \end{aligned} \tag{17.2}$$

Each of the summations in the final line of Eq. (17.2) is in the form of an $(N/2)$-point DFT. The *signal flow graph* (SFG) corresponding to the final line of Eq. (17.2) is shown in Figure 17.1 for the specific case of $N = 8$. In a circuit analysis context, SFGs often include curved lines, and the direction of signal flow is almost always indicated by an arrow. In a DSP context, it is conventional to use only straight edges, with signals flowing from left to right through the edges, unless indicated otherwise by the presence of an arrowhead. Each node represents a signal formed by the sum of all the inbound edges incident to the node.

The node for $X[0]$ in the upper right-hand corner of the figure has two incident edges. The lower edge, coming from node $X_{4,1}[0]$, has a weight of W^0, as indicated by the annotation near the arrowhead. The upper edge, coming from node $X_{4,0}[0]$, has no weight indicated. This configuration of the node and the two incident edges indicates that

$$X[0] = X_{4,0}[0] + W^0 X_{4,1}[0]$$

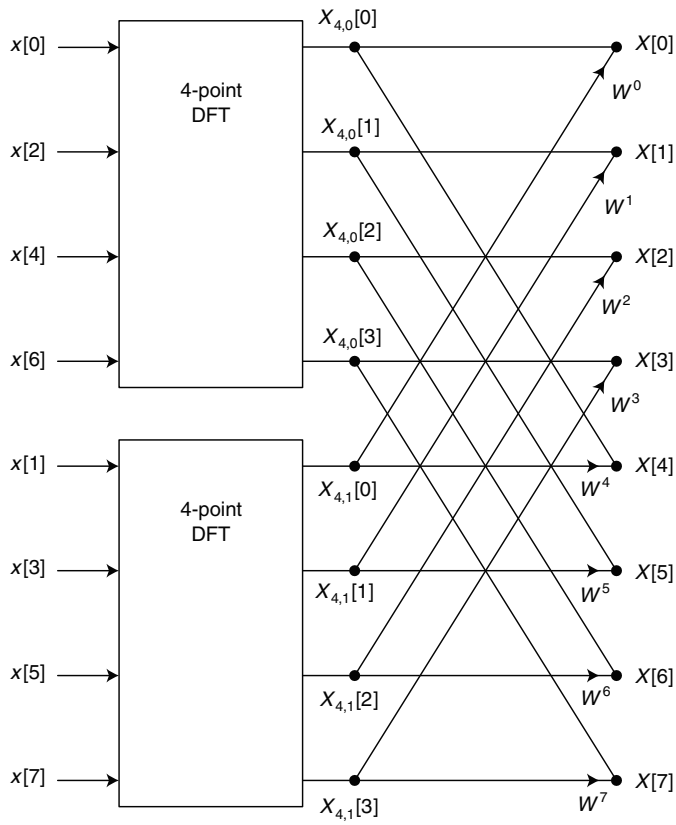


Figure 17.1 Signal flow graph depicting operations defined by the final line of Eq. (17.2)

The complete set of equations represented by Figure 17.1 is

$$\begin{aligned}
X[0] &= X_{4,0}[0] + W^0 X_{4,1}[0] \\
X[1] &= X_{4,0}[1] + W^1 X_{4,1}[1] \\
X[2] &= X_{4,0}[2] + W^2 X_{4,1}[2] \\
X[3] &= X_{4,0}[3] + W^3 X_{4,1}[3] \\
X[4] &= X_{4,0}[0] + W^4 X_{4,1}[4] \\
X[5] &= X_{4,0}[1] + W^5 X_{4,1}[5] \\
X[6] &= X_{4,0}[2] + W^6 X_{4,1}[6] \\
X[7] &= X_{4,0}[3] + W^7 X_{4,1}[7]
\end{aligned}$$

where the $X_{4,0}[m]$ are the output of the upper, or "even," 4-point DFT, and the $X_{4,1}[m]$ are the outputs of the lower, or "odd," 4-point DFT.

The weighting factors, W^k, have been called *twiddle factors* since the earliest days of digital signal processing. If $N/2$ is even, then the summations in Eq. (17.2) can each be split again. If $N = 2^L$, the summations at each stage can be split until the original transform is expressed as a combination of N 1-point DFTs. A 1-point transform is trivial:

$$\begin{aligned}
X[0] &= \sum_{n=0}^{0} x[0] W^0 \\
&= x[0]
\end{aligned}$$

After $L = \log_2 N$ stages of splitting, all of the computational burden has been moved out of the summations and into the operations needed to combine 1-point transforms into 2-point transforms, 2-point transforms into 4-point transforms, and so on. The SFG for $N = 8$ and $L = 3$ is shown in Figure 17.2. Pairs of input points from $x[n]$ are combined to form the eight values $X_{2,k}[m]$ for $k = 0, 1, 2, 3$ and $m = 0, 1$. One complex multiply-add operation is required for each of the eight values.

At the second stage, pairs of $X_{2,k}[m]$ values are combined to form the eight values $X_{4,k}[m]$ for $k = 0, 1$ and $m = 0, 1, 2, 3$. One complex multiply-add operation is required for computing each of the eight combinations.

Finally, pairs of $X_{4,k}[m]$ values are combined to form the eight values $X[m]$ for $m = 0, 1, \ldots, 7$. In general, computing an N-point transform using this approach entails $\log_2 N$ stages of combining with N complex multiply-add operations per stage for a total computational burden of only $N \log_2 N$ complex multiply-add operations.

The output values, $X[m]$, as shown in Figure 17.2, appear in "natural" order from $X[0]$ through $X[7]$. The input

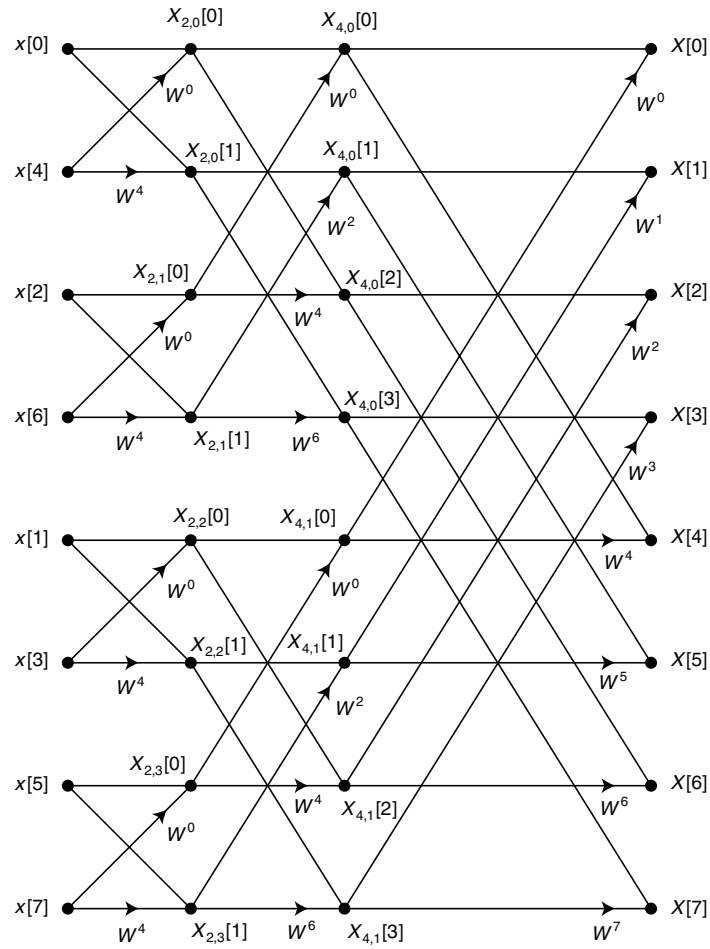


Figure 17.2 Signal flow graph for 8-point decimation-in-time FFT with permuted inputs and naturally ordered outputs

values, $x[n]$, appear in the permuted order $x[0]$, $x[4]$, $x[2]$, $x[6]$, $x[1]$, $x[5]$, $x[3]$, $x[7]$. This order is called *bit-reversed* order because the index for the kth input value can be obtained by representing k with $\log_2 N$ bits and reversing the order of these bits. For example, when $k = 3 = (011)_2$, the corresponding bit-reversed index is $(110)_2 = 6$. Thus, the input $x[6]$ appears in position 3 of the input sequence. The values $x[0]$ and $x[7]$ remain in their original positions because (000) and (111) are the same backward or forward. FFT algorithms of the type represented by Figure 17.2 are called *decimation-in-time, permuted input-natural output* (DIT-PINO) FFTs. It is possible to rearrange the nodes in Figure 17.2 without disturbing the connections between the nodes to obtain the SFG shown in Figure 17.3, where now the inputs are in natural order and the outputs are in bit-reversed order. This SFG represents a *decimation-in-time, natural input-permuted output* (DIT-NIPO) FFT.

17.1 Implementation Consideratons

For SFGs of FFTs that are drawn in "standard" form, the the edges connecting adjacent columns of nodes in can be grouped into patterns that are called *butterflies* because of their shape. Figure 17.4 shows the SFG of Figure 17.2, with one butterfly in

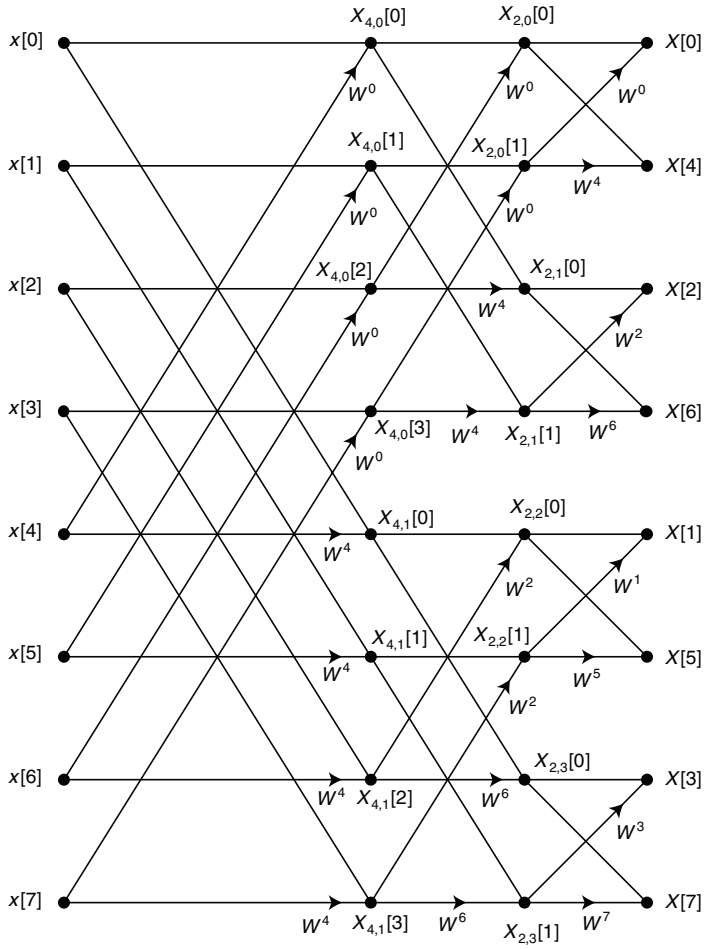


Figure 17.3 Signal flow graph for 8-point decimation-in-time FFT with naturally ordered inputs and permuted outputs

each stage highlighted in bold. Figure 17.5 defines some convenient terminology we can use in this series of notes whenever butterflies and the operations that they represent are discussed.

Let's take a closer look at the butterflies in Figure 17.4 to uncover the patterns that are essential to software mechanization of the underlying mathematical operations. The first butterfly in stage 2 of the SFG of Figure 17.2 has $X_{2,0}[0]$ and $X_{2,1}[0]$ as its input nodes and $X_{4,0}[0]$ and $X_{4,0}[2]$ as its output nodes. The rising diagonal is multiplied by W^0 and the bottom edge is multiplied by W^4. The top edge and falling diagonal each have unity gain. Notice that for every butterfly in the SFG, when the rising

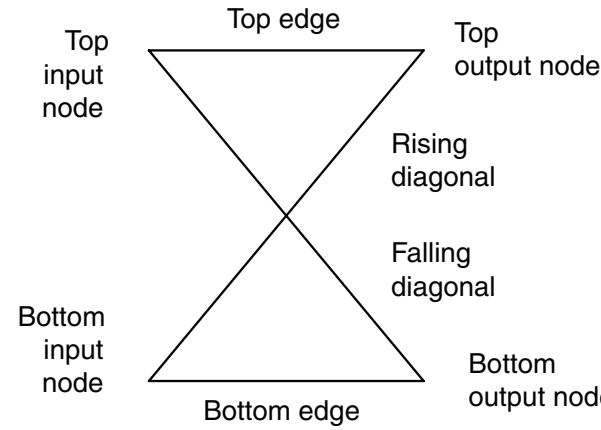


Figure 17.5 Butterfly terminology

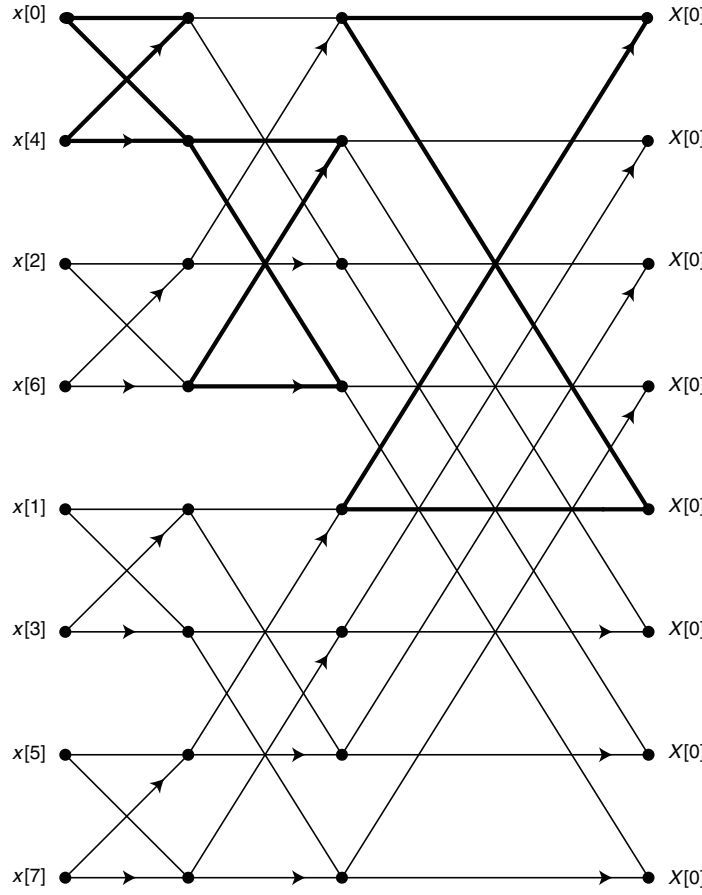


Figure 17.4 Bold edges depict the butterfly structures embedded in the SFG for an FFT.

diagonal is multiplied by W^k, the bottom edge is multiplied by $W^{k+N/2}$. Invoking the definition of W^k, it is easy to show that $W^{k+N/2} = -W^k$:

$$\begin{aligned} W^{k+N/2} &= \exp\left(\frac{-2j(k+N/2)\pi}{N}\right) \\ &= \exp(-j\pi)\exp\left(\frac{-2jk\pi}{N}\right) \\ &= -W^k \end{aligned}$$

Thus, each butterfly can be implemented using a single complex multiplication and two complex additions. The bottom input node is multiplied by W^k to produce an interim result that is added to the top input node to obtain the top output node. The interim result is subtracted from the top input node to obtain the bottom output node. In an efficient implementation, all of the input nodes for a given stage are stored in a single array. Once the array location holding the bottom input node is used for computing the interim result, it is no longer needed and can be overwritten with the value computed for the butterfly's bottom output node. The interim result can then be added to the array location holding the top input node to produce the top output node. The following fragment of C code summarizes this approach for butterfly implementation.

```
temp = array[bot_node_idx] * twiddle;
array[bot_node_idx]
       = aray[top_node_idx] - temp;
array[top_node] += temp;
```

The four butterflies in stage 1 of Figure 17.2 can all be computed in sequence using the single multiplier W^0. There are two viable approaches for doing the stage-2 computations. The first approach computes the butterflies in order from top to bottom, generating (or perhaps recalling from storage) the multipliers W^0, W^2, W^0, and W^2 in sequence as they are needed. The second approach generates W^0 only once, using it for each butterfly where it is needed before moving on to generate W^2. The first approach involves either extra trigonometry or storage for the multipliers. The second approach requires only slight additional complexity in the addressing scheme used to access locations in the node array.

FFT: Decimation-in-Frequency Algorithms

Decimation-in-time FFTs are based on repeatedly splitting the DFT summation into two summations—one for the decimated time sequence from which even-indexed samples have been removed and one for the decimated time sequence from which odd-indexed samples have been removed. As the name implies, decimation-in-frequency FFTs split the DFT summation in a way that produces decimated frequency sequences. The DFT summation can be specialized for computing only the even-indexed frequency samples:

$$X[2r] = \sum_{n=0}^{N-1} x[n] W_N^{2nr} \qquad r = 0,1,\ldots,(N/2)-1 \tag{18.1}$$

After some algebraic manipulations, Eq. (18.1) can be put in the form

$$X[2r] = \sum_{n=0}^{(N/2)-1} \left(x[n] + x\left[n + \frac{N}{2} \right] \right) W_{N/2}^{nr} = \sum_{n=0}^{(N/2)-1} a[n] W_{N/2}^{nr} \qquad r = 0,1,\ldots,(N/2)-1 \tag{18.2}$$

where $a[n]$ is the sequence formed by adding together corresponding samples from the first and second halves of the original time sequence:

$$a[n] = x[n] + x\left[n + \frac{N}{2} \right]$$

Thus, the even samples of the frequency sequence can be generated by performing an $(N/2)$-point DFT on combinations of samples from the original time sequence.

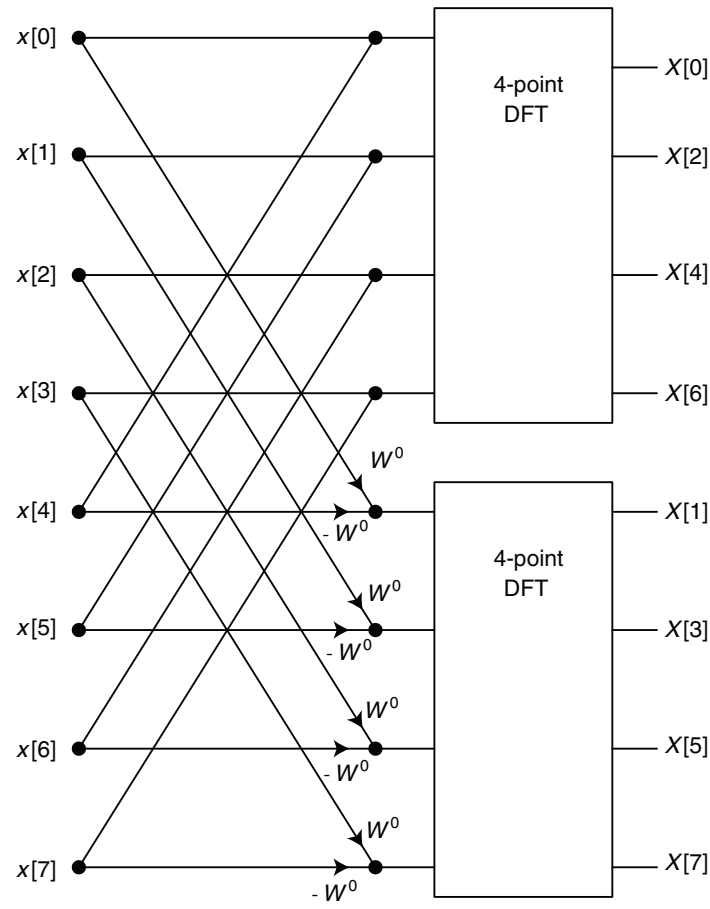


Figure 18.1 Signal flow graph depicting operations defined by Eqs. (18.2) and (18.3)

The DFT summation can also be specialized for computing only the odd-indexed frequency samples:

$$\begin{aligned} X[2r+1] &= \sum_{n=0}^{N-1} x[n] W_N^{n(2r+1)} \\ &= \sum_{n=0}^{(N/2)-1} \left(x[n] - x\left[n + \frac{N}{2}\right] \right) W_N^n W_{N/2}^{nr} \\ &= \sum_{n=0}^{(N/2)-1} b[n] W_N^n W_{N/2}^{nr} \end{aligned} \tag{18.3}$$

where $b[n]$ is the sequence formed by subtracting samples in the second half of the original time sequence from the corresponding samples in the first half of the original time sequence:

$$b[n] = x[n] - x\left[n + \frac{N}{2}\right]$$

Thus, the odd samples of the frequency sequence can be generated by performing an $(N/2)$-point DFT on weighted combinations of samples from the original time sequence. The SFG corresponding to Eqs. (18.2) and (18.3) for the specific case of $N = 8$ is shown in Figure 18.1. If $N = 2^L$, the summations at each stage can be split. The SFG for $N = 8$ and $L = 3$ is shown in Figure 18.2. The input values appear in natural order from $x[0]$ through $x[7]$. The output values appear in the bit-reversed order $X[0]$, $X[4]$, $X[2]$, $X[6]$, $X[1]$, $X[5]$, $X[3]$, $X[7]$. It is possible to rearrange the nodes in Figure 18.2 without disturbing the connections between the nodes to obtain the SFG shown in Figure 18.3, where the inputs are in bit-reversed order and the outputs are in natural order.

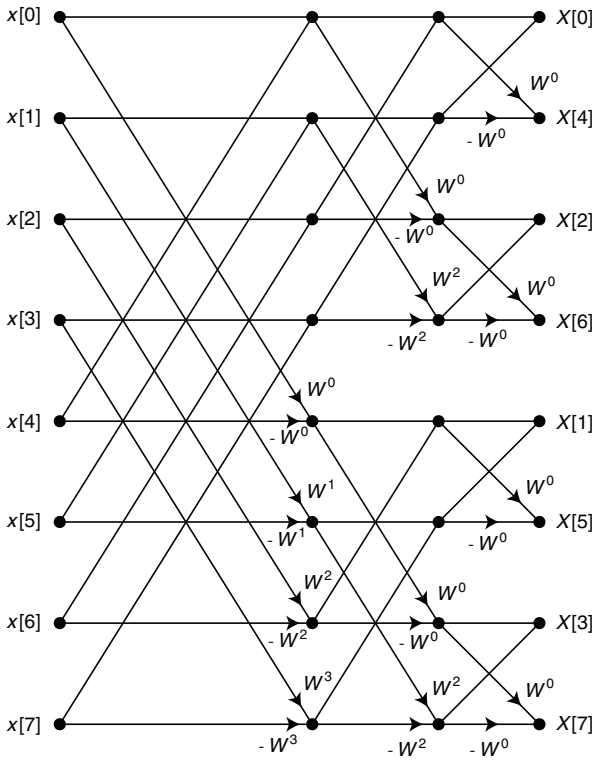


Figure 18.2 Signal flow graph for 8-point decimation-in-frequency FFT with naturally ordered inputs and permuted outputs

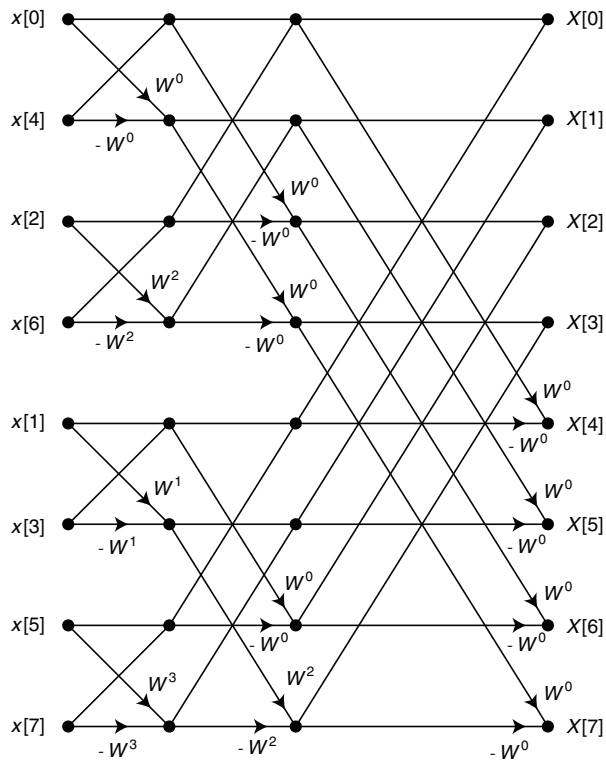


Figure 18.3 Signal flow graph for 8-point decimation-in-frequency FFT with permuted inputs and naturally ordered outputs

FFT: Prime Factor Algorithm

The radix-2 algorithms described in Notes 17 and 18 are very useful, but in certain situations, a DFT block size of 2^k is just not acceptable. The prime factor algorithm (PFA) is a technique for constructing fast transforms of length N, where N can be expressed as a product of mutually prime factors. A DFT of length $N = N_1 N_2$ can be restructured as an N_1 by N_2 two-dimensional DFT:

$$X_2[k_1,k_2] = \sum_{n_2=0}^{N_2-1} W_2^{n_2 k_2} \left(\sum_{n_1=0}^{N_1-1} x_2[n_1,n_2] W_1^{n_1 k_1} \right) \tag{19.1}$$

where

$$W_1 = \exp(-j2\pi / N_1)$$
$$W_2 = \exp(-j2\pi / N_2)$$

In order to make use of the restructured transform, the one-dimensional input sequence, $x[n]$, must be mapped into a two-dimensional sequence, $x_2[n_1, n_2]$, using the index transformation

$$n \equiv (N_1 n_2 + N_2 n_1) \text{ modulo } 2$$

where

$$n_1 = 0, 1, \quad , N_1 - 1$$
$$n_2 = 0, 1, \quad , N_2 - 1$$

After transformation, the two-dimensional result, $X_2[k_1, k_2]$, is mapped into the one-dimensional result using the index transformation

$$k \equiv (N_1 t_1 k_2 + N_2 t_2 k_1) \text{ modulo } 2$$

where t_1 and t_2 are chosen such that

$$N_1 t_1 = 1 \text{ modulo } N_2$$
$$N_2 t_2 = 1 \text{ modulo } N_1$$

19.1 Small-*N* Transforms

There are a number of very efficient specialized implementations for N-point DFTs when N is very small (see Math Boxes 19.1 through 19.3). These implememetations are ideally suited for use as component DFTs in the prime factor algorithm.

Math Box 19.1

3-point DFT Algorithm

$u = 2\pi/3$

$t_1 = x[1] + x[2]$

$m_0 = x[0] + t_1$

$m_1 = t_1(\cos u - 1)$

$m_2 = j(\sin u)(x[2] - x[1])$

$s_1 = m_0 + m_1$

$X[0] = m_0$

$X[1] = s_1 + m_2$

$X[2] = s_1 - m_2$

Math Box 19.2

5-point DFT Algorithm

$u = 2\pi/5$

$t_1 = x[1] + x[4]$ $\quad t_2 = x[2] + x[3]$

$t_3 = x[1] - x[4]$ $\quad t_4 = x[3] - x[2]$

$t_5 = t_1 + t_2$

$m_0 = t_5 + x[0]$ $\quad m_1 = t_5[(\cos u + \cos 2u)/2 - 1]$

$m_2 = (t_1 - t_2)[(\cos u - \cos 2u)/2]$ $\quad m_3 = -j(t_3 + t4)\sin u$

$m_4 = -jt_4(\sin u + \sin 2u)$ $\quad m_5 = jt_3(\sin u - \sin 2u)$

$s_1 = m_0 + m_1$ $\quad s_2 = s_1 + m_2$

$s_3 = m_3 - m_4$ $\quad s_4 = s_1 - m_2$

$s_5 = m_3 + m5$

$X[0] = m_0$ $\quad X[1] = s_2 + s_3$

$X[2] = s_4 + s_5$ $\quad X[3] = s_4 - s_5$

$X[4] = s_2 - s_3$

Math Box 19.3

7-point DFT Algorithm

$u = 2\pi/7$

$t_1 = x[1] + x[6]$
$t_2 = x[2] + x[5]$
$t_3 = x[3] + x[4]$
$t_4 = t_1 + t_2 + t_3$
$t_5 = x[1] - x[6]$
$t_6 = x[2] - x[5]$
$t_7 = x[4] - x[3]$
$t_8 = t_1 - t_3$
$t_9 = t_3 - t_2$
$t_{10} = t_5 + t_6 + t_7$
$t_{11} = t_7 - t_5$
$t_{12} = t_6 - t_7$
$t_{13} = -t_6 - t_9$
$t_{14} = -t_{11} - t_{12}$

$m_0 = t_4 + x[0]$
$m_1 = t_4[(\cos u + \cos 2u + \cos 3u)/3 - 1]$
$m_2 = t_8[(2\cos u - \cos 2u - \cos 3u)/3]$
$m_3 = t_9[(\cos u - 2\cos 2u + \cos 3u)/3]$
$m_4 = t_{13}[(\cos u + \cos 2u - 2\cos 3u)/3]$
$m_5 = -jt_{10}[(\sin u + \sin 2u - \sin 3u)/3]$
$m_6 = jt_{11}[(2\sin u - \sin 2u - \sin 3u)/3]$
$m_7 = jt_{12}[(\sin u - 2\sin 2u + \sin 3u)/3]$
$m_8 = jt_{14}[(\sin u + \sin 2u + 2\sin 2u)/3]$

$s_0 = -m_2 - m_3$
$s_1 = -m_2 - m_4$
$s_2 = -m_6 - m_7$
$s_3 = m_6 + m_8$
$s_4 = m_0 + m_1$
$s_5 = s_4 - s_0$
$s_6 = s_4 + s_1$
$s_7 = s_4 + s_0 - s_1$
$s_8 = m_5 - s_1$
$s_9 = m_5 - s_3$
$s_{10} = m_5 + s_2 + s_3$

$X[0] = m_0$
$X[1] = s_5 + s_8$
$X[2] = s_6 + s_9$
$X[3] = s_7 - s_{10}$
$X[4] = s_7 + s_{10}$
$X[5] = s_6 - s_9$
$X[6] = s_5 - s_8$

Fast Convolution Using the FFT

Consider an FIR filter having a unit-sample response, $h[n]$, that extends for N_R samples. Such a filter's output, $y[k]$, at time k is given by the discrete convolution

$$y[k] = \sum_{n=0}^{N_R-1} h[n]x[k-n] \qquad (20.1)$$

where $x[k]$ is the input sequence. In order to produce a block of N_B output samples, $y[0]$ through $y[N_B-1]$, Eq. (20.1) must process a block of N_B+N_R-1 input samples from $x[-N_R+1]$ through $x[\mathrm{N}_B-1]$. Assuming that $x[k]$ is valid only for $k \geq 0$, the first N_R-1 samples of the output will not be valid because computation of these values depends on samples of $x[k]$ before $k=0$. To produce a second block of N_B output samples, $y[N_B]$ through $y[2N_B-1]$, Eq. (20.1) must process a block of N_B+N_R-1 input samples from $x[N_B-N_R+1]$ through $x[2N_B-1]$. The final N_R-1 input samples processed in the first block will be the first N_R-1 input samples processed in the second block. This can be extended to say that the final N_R-1 input samples processed in block b will be the first N_R-1 input samples processed in block $b+1$. Therefore, to produce each block of N_B output samples, only N_B *new* input samples are needed, along with N_R-1 *old* input samples used in the previous block.

When the filtering is performed in the frequency domain using fast convolution, a technique called *overlap and save* can be used for managing the old and new input samples. The first step in implementing this technique is selecting appropriate values for the memory length, N_M; FFT length, N; and input and output block length, N_B, subject to the following constraints:

- The memory length, N_M, must be selected such that N_M+1 is equal to or greater than the filter's unit sample response.
- The FFT length, N, must be a highly composite number to facilitate the use of a "fast" algorithm for computing the DFT.
- The input/output block length, N_B, is equal to $N-N_M$.

The processing sequence for the overlap and save technique is listed in Recipe 20.1.

Recipe 20.1

Overlap-and-Save Fast Convolution

The following steps assume that values for N, N_M, and N_B have been selected based on the criteria discussed in the text.

1. Load the first N_B input samples into the first N_B locations in the FFT input buffer. Set the remaining $N-N_B$ locations in the FFT input buffer equal to zero.
2. Perform the FFT.
3. Multiply the FFT result by the N-point sampled response of the filter being implemented.
4. Perfrom the inverse FFT.
5. Take the first N_B samples of the IFFT output as the next N_B samples of the filter output.
6. Move the contents of locations (N_B-N_M) through (N_B-1) from the FFT input buffer into locations $(N-N_M)$ through $(N-1)$ of the same buffer.
7. Load the next N_B new input samples into locations 0 through (N_B-1) of the FFT input buffer.

Repeat steps 2 through 7 until filter operation is halted.

Figure 20.1 depicts the relationships between the input samples and output samples when Recipe 20.1 is used. For the specific case depicted in the figure, the FFT length is $N = 64$, the memory length is $N_M = 18$, and the input/output block length is $N_B = 46$. For pass k, the N_M input samples saved from the end of pass $k-1$ are placed at the end of the FFT input block, as shown at (a) in Figure 20.1. A block of $N_B = 46$ new input samples is read into locations zero through $N_B - 1$ of the FFT input block, as shown at (b) in the figure.

As discussed in Note 13, the N-point time sequence and N-point frequency sequence related by an N-point DFT are both implicitly periodic, with a period of N samples. Owing to this periodic extension, the N_M saved input samples at the end of the FFT block are effectively prepended to the beginning of the FFT block, as shown at (c) in the figure. Thus, as depicted at (d) in the figure, the filter is able to produce the first output sample of block k by effectively operating on a continuous block of $N_M + 1$ samples, comprising the N_M saved samples and the first new sample that is placed in location zero of the FFT input block.

The span of the filter's impulse response is successively moved one sample to the right, producing one output sample after each such move until the span reaches the position depicted at (e) in Figure 20.1. In this position, the filter is operating on the final N_M new input samples plus the oldest one of the saved input samples, and the output produced is not valid. Therefore, the final N_M samples (f) in the IFFT output block are discarded. The first N_B samples (g) in the IFFT output block are issued as

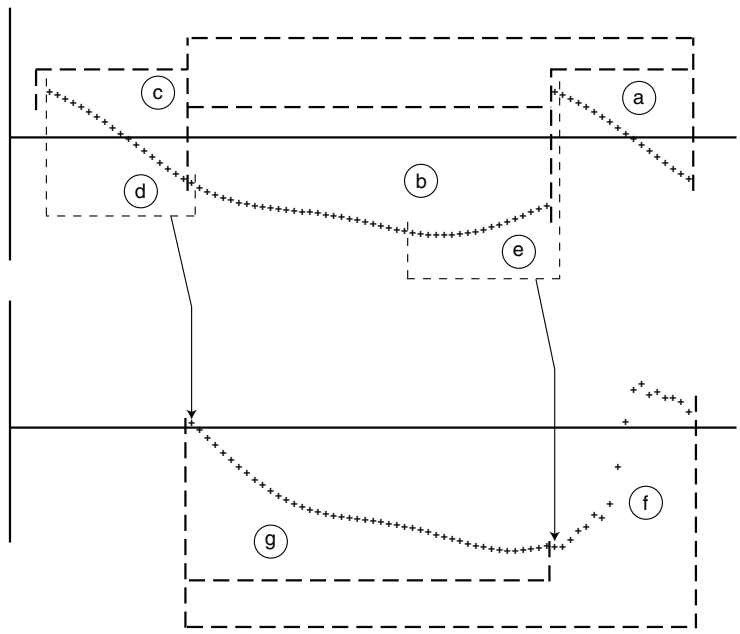


Figure 20.1 The overlap-and-save technique for filtering a long signal sequence via fast convolution: (a) the N_M samples from the previous pass, (b) the N_B new input samples for the current pass, (c) the periodic extension of the N_M saved samples, (d) the span of the filter's impulse response corresponding to the first output of sample of the current pass, (e) the span of the filter's impulse response response corresponding to the first invalid sample in the IFFT's output block, (f) discarded IFFT output samples, and (g) filter output samples.

the filter's output for pass k. In preparation for pass k+1, the first N_M input samples for pass k are then copied into the end of the FFT input block in accordance with step 6 of Recipe 20.1.

Figure 20.2 shows how a number of IFFT output blocks are reassembled to produce a continuous sequence of filter output samples that exactly match the sequence of samples that would be produced by a direct convolution performed using Eq. (20.1). Notice that in either approach, output samples $y[0]$ through $y[N_M-1]$ are not valid because there is insufficient input history for the filter until output sample $y[N_M]$.

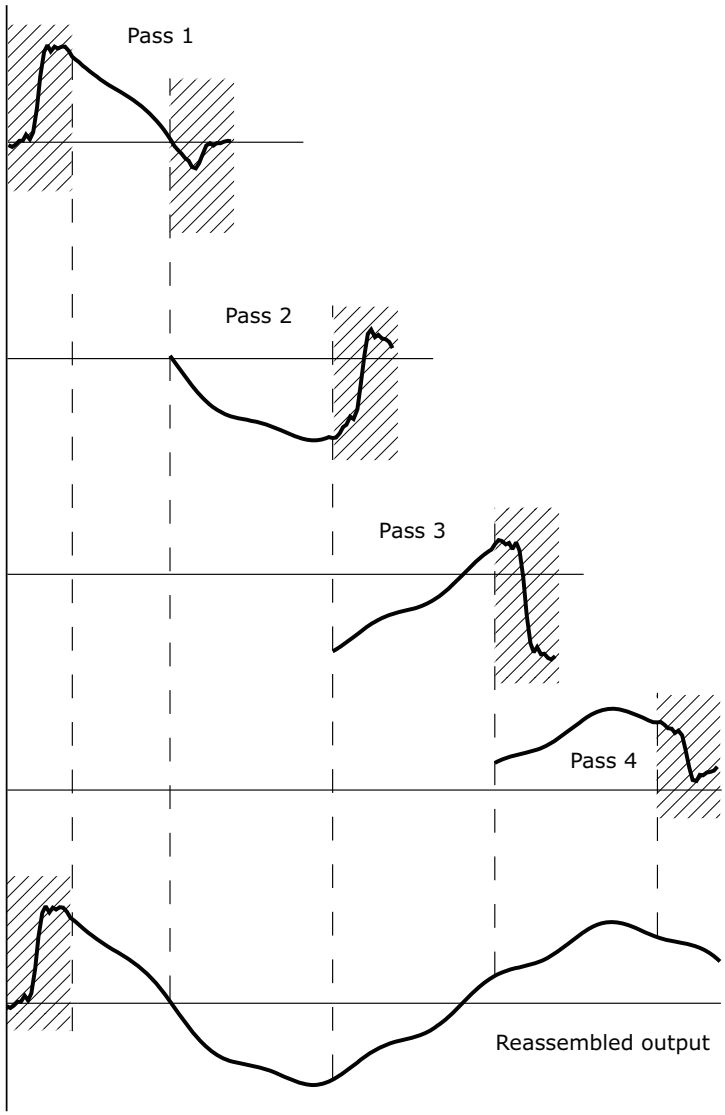


Figure 20.2 Continuous output sequence reassembled from IFFT output blocks using the overlap-and-save technique

Using Window Functions: Some Fundamental Concepts

Within DSP, window usage can be divided into two broad categories: (1) windowing of signal segments to reduce leakage in the DFT, and (2) designing FIR filters that are based on using windows to reduce the Gibb's phenomenon in finite approximations to ideal filters. Additional details concerning the use of windows for FIR filter design are discussed in Notes 23 and 24.

21.1 Elementary Windows

As discussed in Note 14, truncating a DFT input sequence to a length of N samples can be mathematically modeled as multiplying the sequence by a *rectangular window* sequence that has non-zero values only for sample indices 0 through $N-1$

$$w[n] = \begin{cases} 1 & 0 \le n < N \\ 0 & \text{otherwise} \end{cases} \qquad (21.1)$$

Rectangular windows are not particularly well suited for illustrating some of the time- and frequency-domain characteristics that make using windows so attractive. Therefore, we will begin by introducing a simple tapering window, called the *triangular window,* or sometimes, the *Bartlett* window.

Often, the mathematical descriptions of windows and their frequency responses are simplified if the window is defined centered around the zero index, as shown in Figure 21.1. A window in this position is sometimes called a *lag window*. It is natural to have an odd number of samples in a lag window so that one sample lies on the vertical axis and there are an equal number of samples to the left and to the right of the vertical axis. In this position, a triangular window can be defined as

$$w[n] = 1 - \frac{|2n - N + 1|}{N - 1} \quad \text{for} \quad \frac{-(N-1)}{2} \le n \le \frac{N-1}{2} \qquad (21\,2)$$

As defined, this window has a value of zero at each end and a peak value of 1 in the center. It could be argued that this window really has a length of only $N-2$ because of the two zero-valued end-points. However, it is convenient for some analyses of window properties to include the zero-valued endpoints as part of the window. There are a number of different window families with similar concerns. Hann and Blackman windows are other commonly used windows that have zero-valued endpoints.

Different authors adopt different conventions with regard to how zero-valued endpoints are treated. For the case of triangular

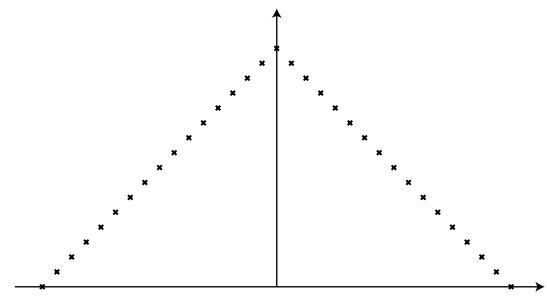


Figure 21.1 Triangular lag window with 33 points

windows, MATLAB supports both approaches—the function `triang(N)` generates a triangular window having N nonzero values, and the function `bartlett(N)` generates a triangular window with zero-valued endpoints and $N-2$ non-zero values. This support must be used with caution. For odd window lengths, as illustrated in Figure 21.2, for the case of $N=63$, `triang(N)` produces a window that exactly equals the center N points of the window produced by `bartlett(N+2)`. In other words, the result from `triang(N)` can be obtained just by removing the two zero-valued endpoints from the result produced by `bartlett(N+2)`. In each case, the slope for the left side of the triangle is 1/32. However, this relationship between the results from `triang(N)` and `bartlett(N+2)` is not maintained for the case when N is even. As shown in Figure 21.3(a), in the window produced by `bartlett(62)`, the peak of the triangle falls midway between sample indices 31 and 32. The left edge rises from a value of 0 at sample index 1, to a value of 1 at a point equivalent to sample index 31.5; thus the left-side slope is obtained as

$$\text{slope} = \frac{1-0}{31.5-1} = \frac{2}{61} = 0.0327869$$

As shown in Figure 21.3(b), in the window produced by `triang(60)`, the peak of the triangle falls midway between sample indices 30 and 31. Rather than having the first sample be equal to the first non-zero sample from `bartlett(62)` (which would locate the missing zero-valued samples at abscissae corresponding to sample indices 0 and 61), the implementers of MATLAB chose to construct the triangle in a way that places the zero values at abscissae corresponding to sample

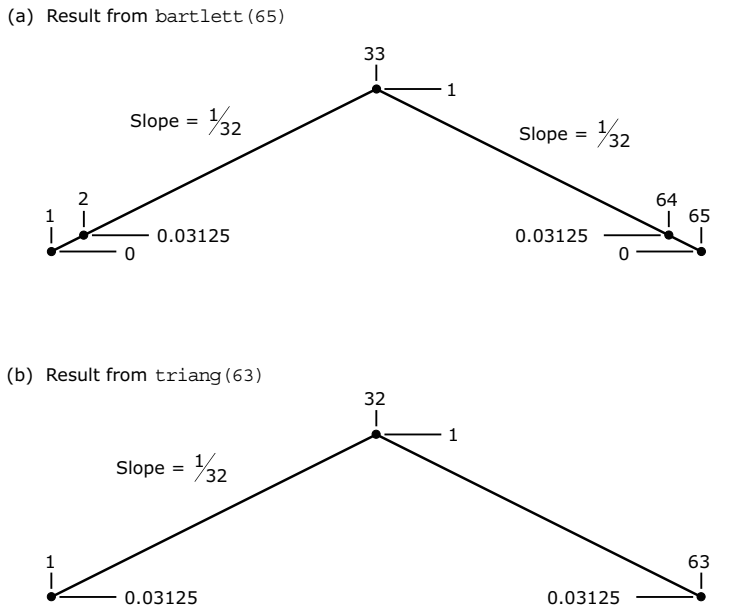


Figure 21.2 Comparison of windows produced by MATLAB functions `bartlett` and `triang` for odd N

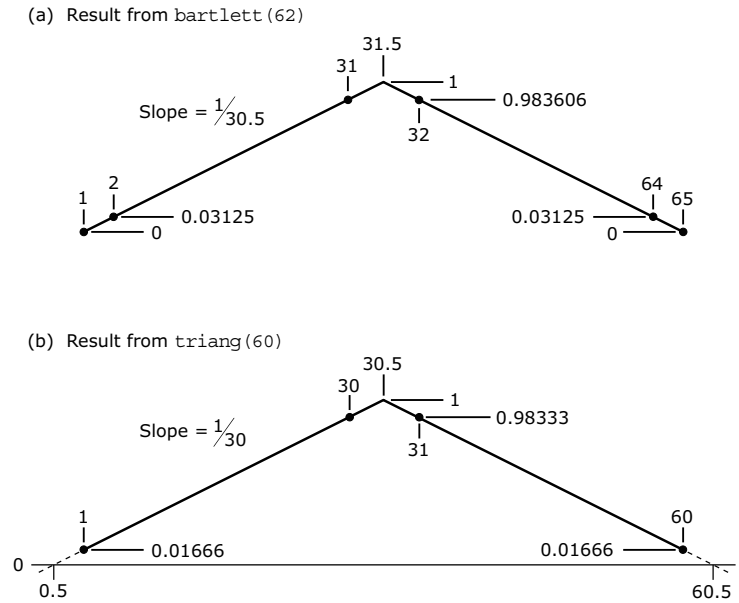


Figure 21.3 Comparison of windows produced by MATLAB functions `bartlett` and `triang` for even N

indices 0.5 and 60.5. This choice causes the left edge to rise from a value of 0 at a point equivalent to sample index 0.5, to a value of 1 at a point equivalent to sample index 30.5; thus the left-side slope is obtained as

$$\text{slope} = \frac{1-0}{30.5-0.5} = \frac{1}{30} = 0.033333$$

21.2 Even-Length Windows for DFT Applications

Even-length windows are most often used for reducing DFT leakage—the windows used in FIR filter design are usually odd in length. However, due to the implicit periodic extension inherent in the DFT, the correct way to generate the even-length windows used with even-length DFTs involves truncating the final point of an odd-length window. Figure 21.4(a) shows a symmetric 8-point triangular window and several of its images due to periodic extension. Figure 21.4(b) shows an asymmetric 8-point window that began as a symmetric 9-point window before the final point was truncated. As shown in the figure, the first point in each window image also fills the place of the missing final point for the preceding image.

Looking at Figure 21.4, the difference between the two approaches might not seem important, but proper generation of the DFT window is crucial if the DFT results are to represent samples of the DTFT result. Figure 21.5 shows the positive-frequency DTFT spectrum for a 33-point triangular window with zero-valued endpoints (i.e., the results from `bartlett(33)` in MATLAB). Superimposed on the continuous trace of the DTFT spectrum are circles marking values from the DFT of the asymmetric 32-point window that results from

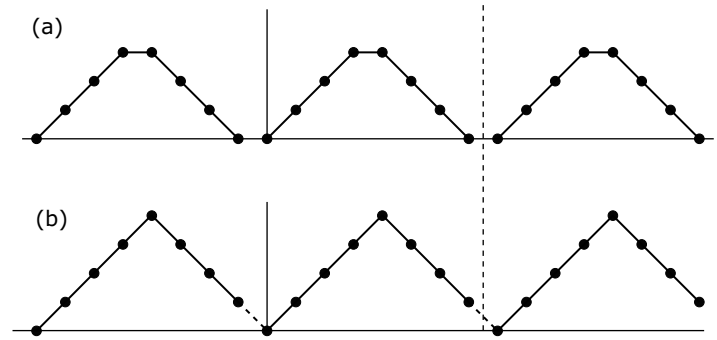


Figure 21.4 Two different configurations for an even-length window: (a) naive approach, and (b) approach that exploits the periodic extension implicit in the DFT

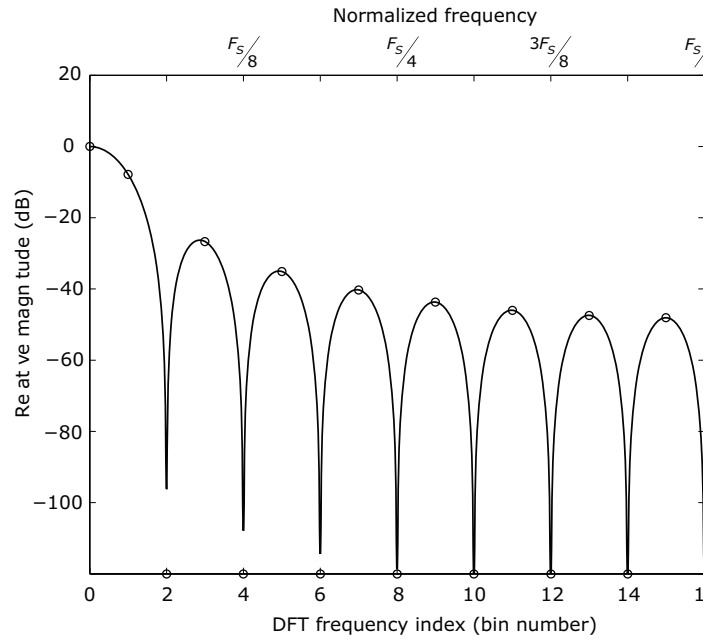


Figure 21.5 Magnitude of DTFT for 33-point triangular window (solid trace) with magnitude of DFT for coresponding 32-point truncated windows (circles)

truncating the final point of the original 33-point window. Figure 21.6 shows the DFT locations that result from the incorrect use of a 32-point symmetric window. The DFT locations are displaced from their correct locations, but they are close enough to explain why many practitioners are able to use even-length symmetric windows incorrectly and still obtain usable results.

21.3 Lobe Structure in the Frequency Response

There are four fundamental characteristics tied directly to the lobe structure of a window's DTFT response.

1. Width of the main lobe. The width can be specified as the null-to-null width or as the 3-dB or 6-dB bandwidth of the main lobe.
2. Peak level of the first side lobe.
3. "Ultimate" attenuation of side lobes far removed from the main lobe. In the case of discrete-time windows, the ultimate attenuation is the attenuation at the side lobe peak closest to $\omega = \pi$.
4. Spacing of the nulls in the DTFT response.

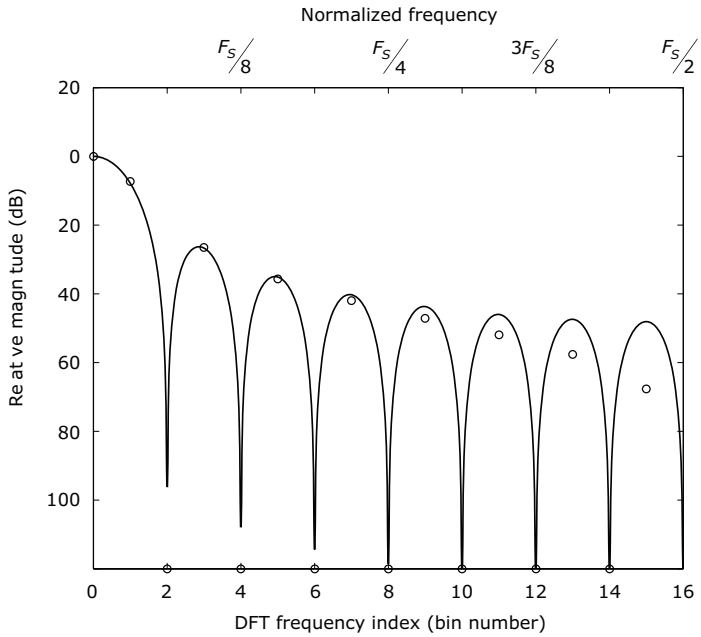


Figure 21.6 Magnitude of DTFT for 33-point triangular window (solid trace) with magnitude of DFT for symmetric 32-point window (circles)

Assessing Window Functions: Sinusoidal Analysis Techniques

When a window is employed to reduce leakage in the DFT, the window's DTFT response can be used directly in one of two ways to assess the DFT's output due to known (or approximately known) sinusoidal components in the input signal.

- The *bin-centric* approach, detailed in Recipe 22.1, determines the response of a single DFT bin to signals over the entire range of frequencies supported by the chosen sampling rate (i.e., the frequency range $-(2T)^{-1}$ to $(2T)^{-1}$).
- The *signal-centric* approach, detailed in Recipe 22.2, determines the response of every DFT bin to a signal component consisting of a single sinusoid at frequency f_c.

Recipe 22.1

Using a Window's DTFT Response to Assess the Frequency Response for a Single Bin in a Windowed DFT

This analysis determines the response of a single DFT bin, say, bin k, to signals over the entire range of frequencies from $-(2T)^{-1}$ to $(2T)^{-1}$, where T is the time-domain sampling interval.

1. Center a copy of the window's DTFT response, $W(f)$, over the frequency corresponding to bin k. This frequency-shifted copy of the response can be denoted as $W_k(f) = W(f - kF)$.
2. For an input signal consisting of an ideal sinusoid of cyclic frequency f_c, the DFT bin k response due to the positive-frequency portion of the input signal's spectrum is determined as

$$X^{+}[k] = \begin{cases} W_k(f_c) & \text{for } x[n] = \cos(2\pi f_c nT) \\ -jW_k(f_c) & \text{for } x[n] = \sin(2\pi f_c nT) \end{cases}$$

3. The DFT bin k response due to the negative-frequency portion of the input signal's spectrum is determined as

$$X^{-}[k] = \begin{cases} W_k(NF - f_c), & x[n] = \cos(2\pi f_c nT) \\ jW_k(NF - f_c), & x[n] = \sin(2\pi f_c nT) \end{cases}$$

4. The total bin k response is then determined as

$$X[k] = X^{+}[k] + X^{-}[k]$$

Example 22.4

Analyzing Triangular Window Performance Using a Bin-centric Strategy

Assume that a triangular window is applied to the input of a 32-point DFT. Use Recipe 22.1 to determine the response in a single DFT bin, say, bin 6, when the input signal is

$$x[n] = \cos\left(2\pi n \frac{10.7}{F}\right)$$

where F is the frequency interval between DFT bins. Figure 22.1 shows the real and imaginary components for $W_6(f)$. As indicated in the figure, the DFT bin 6 response due to the positive-frequency portion of the input spectrum is obtained as

$$X^+[6] = -0.149672 - j0.201809$$

and the bin 6 response due to the negative-frequency portion of the input spectrum is obtained as

$$X^-[6] = -0.0298469 - j0.040244$$

The total bin 6 response is then

$$\begin{aligned} X[6] &= X^+[k] + X^-[k] \\ &= -0.179519 - j0.161565 \end{aligned}$$

The magnitude of the response is

$$\begin{aligned} |X[6]| &= \sqrt{(-0.179519)^2 + (0.161565)^2} \\ &= 0.242517 \\ &= -36.42 \text{ dB relative to main lobe peak} \end{aligned}$$

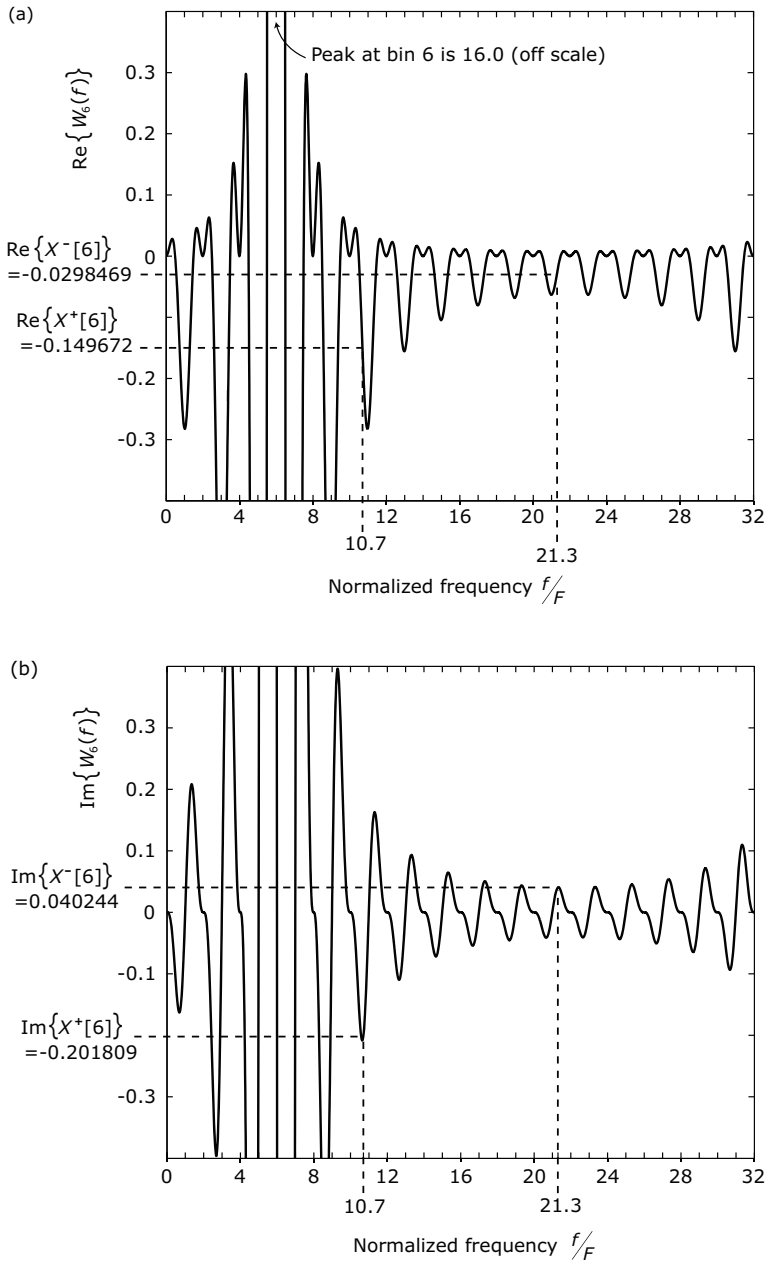


Figure 22.1 DTFT response for a triangular window after response has been shifted to align the center of the main lobe with the frequency corresponding to DFT bin 6: (a) real part, (b) imaginary part

Example 22.5

Analyzing Triangular Window Performance Using a Signal-centric Strategy

Assume that a triangular window is applied to the input of a 32-point DFT. Use Recipe 22.2 to determine the response in the DFT bins when the input signal is

$$x[n] = \cos\left(2\pi n \frac{10.7}{F}\right)$$

where F is the frequency interval between DFT bins. The real and imaginary parts of the composite continuous-frequency response, $X(f)$, obtained using Eq. (R22.1) from Recipe 22.2, are shown in Figure 22.3. The magnitude of this response is plotted with the corresponding DFT magnitude response, $|X[m]|$, in Figure 22.4. Notice that the bin 6 response matches the response obtained using the bin-centric approach in Example 22.1.

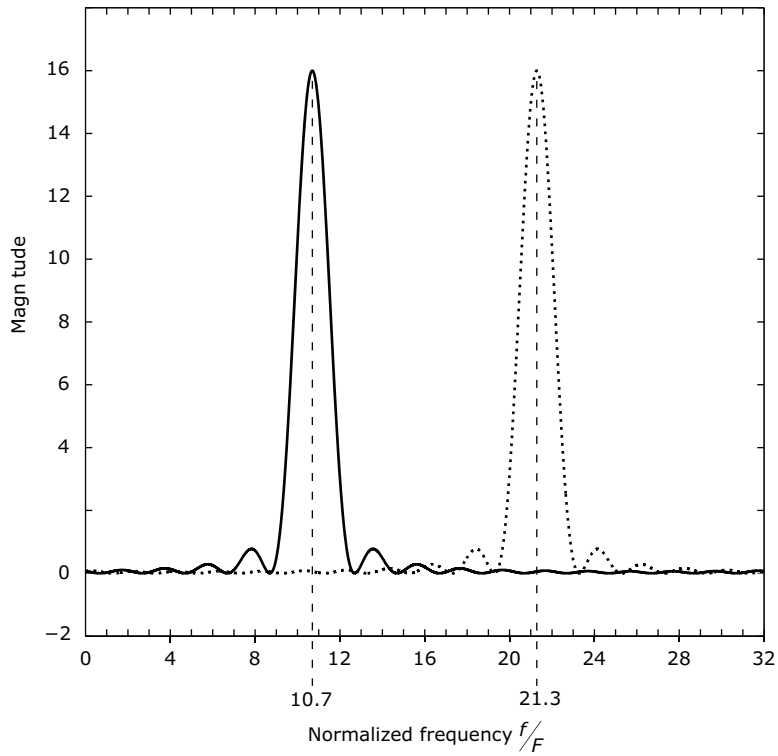


Figure 22.2 Analysis strategy for signal-centric approach. One copy of the DTFT spectrum (solid trace) is centered at frequency f_c, and a second copy (dotted trace) is centered at frequency $NF - f_c$

Recipe 22.2

Signal-Centric Approach for Assessing the Response of a Windowed DFT

This analysis determines the response of every DFT bin to a signal component comprising a single real sinusoid at frequency f_c.

1. Center a copy of the window's DTFT response, $W(f)$, at frequency f_c and a second copy at $NF - f_c$, as indicated in Figure 22.2. These frequency-shifted copies of the response can be denoted as $W(f - f_c)$ and $W(f - NF + f_c)$, respectively. It would be cumbersome to plot two copies of a complex DTFT response in a single figure, therefore Figure 22.2 shows the magnitude responses $|W(f - f_c)|$ and $|W(f - NF + f_c)|$ just to convey a sense of the frequency relationship between the two copies of the response.

2. The composite continuous-frequency response, $X(f)$, is obtained as the appropriately phased complex sum of $W(f - f_c)$ and $W(f - NF - f_c)$

$$X(f) = \begin{cases} W(f - f_c) + W(f - NF + f_c) \\ \qquad \text{for input of } x[n] = \cos(2\pi f_c nT) \\ -jW(f - f_c) + jW(f - NF + f_c) \\ \qquad \text{for input of } x[n] = \sin(2\pi f_c nT) \end{cases}$$

(R22.1)

The DFT response, $X[m]$, is obtained by sampling $X(f)$ at the DFT bin frequencies

$$X[m] = X(f)\Big|_{f=mF} \qquad m = 0, 1, \ldots, N-1$$

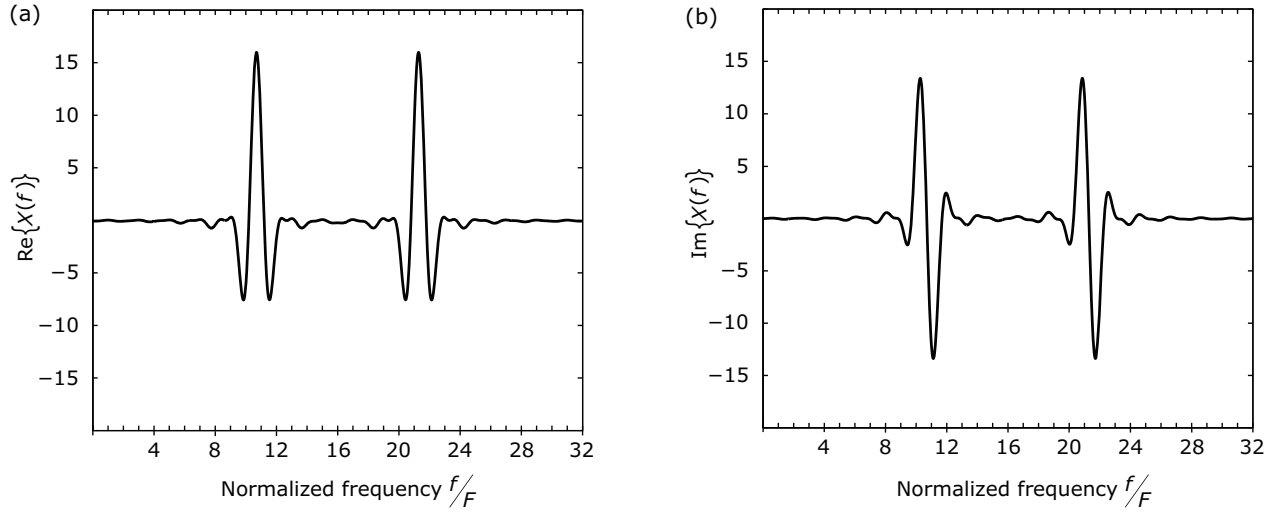


Figure 22.3 DTFT response for a triangular-windowed input signal $x[n]=\cos(2\pi f_c nT)$: (a) real part, (b) imaginary part

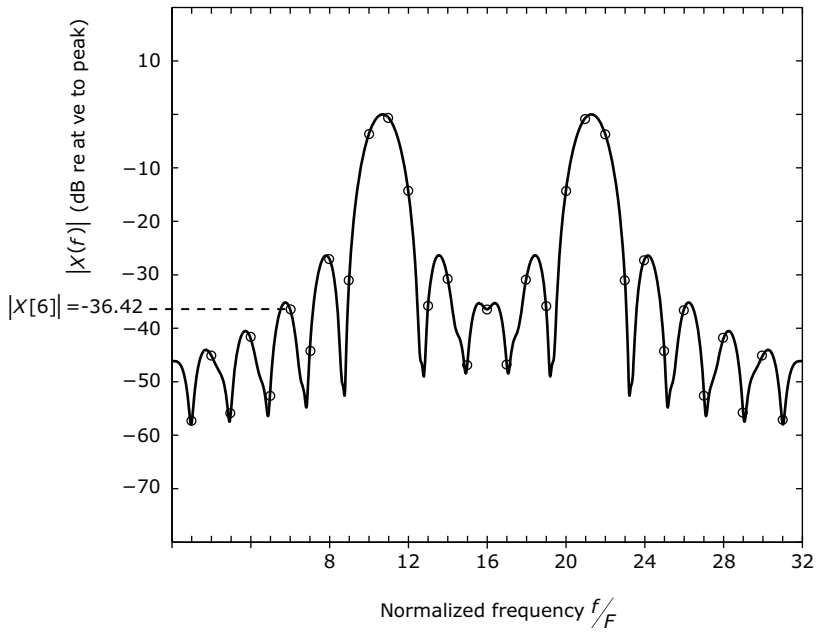


Figure 22.4 DTFT magnitude response (solid trace) and DFT magnitude response (circles) for a triangular windowed, 32-sample segment of input signal $x[n]=\cos(2\pi f_c nT)$

Window Characteristics

There are many different window functions, and selecting the best one for a particular application can be a daunting task. However there are a number of metrics that can be used to weed out the unsuitable windows and zero in on the several "best" window choices for particular requirements.

23.1 Main Lobe Width

Any window having a finite duration in time has a frequency-domain magnitude response that exhibits a lobed structure similar to the example shown in Figure 23.1. Perhaps the most obvious quantifiable characteristic for a window concerns the width of the main lobe in the magnitude response. The null-to-null measurement is a conceptually straightforward way to characterize the width of the main lobe, but it is not the most useful way.

The 6-dB bandwidth of the main lobe is a useful metric because for two equal-strength tones to be resolvable in the output of a windowed DTFT, their frequencies must differ by more than the 6-dB bandwidth of the window.

The 3-dB bandwidth of the main lobe is also useful because it can be combined with the *equivalent noise bandwidth,* discussed later, in section 23.3, to form an indicator of window performance. Specifically, in [1], harris compared more than 40 different windows, and based on DFT results for these windows, he concluded that the ratio of equivalent noise bandwidth to 3-dB bandwidth is a sensitive indicator of overall window performance. Of the windows he tested, good windows always had bandwidth ratios that satisfied

$$1.04 \le \frac{B_{\text{neq}}}{B_{\text{3dB}}} \le 1.055 \qquad \textbf{(23.1)}$$

Windows with this ratio less than 1.04 had wide main lobes that resulted in high processing loss. Windows with this ratio greater than 1.055 tended to have high side lobe structure that resulted in poor two-tone detection capability.

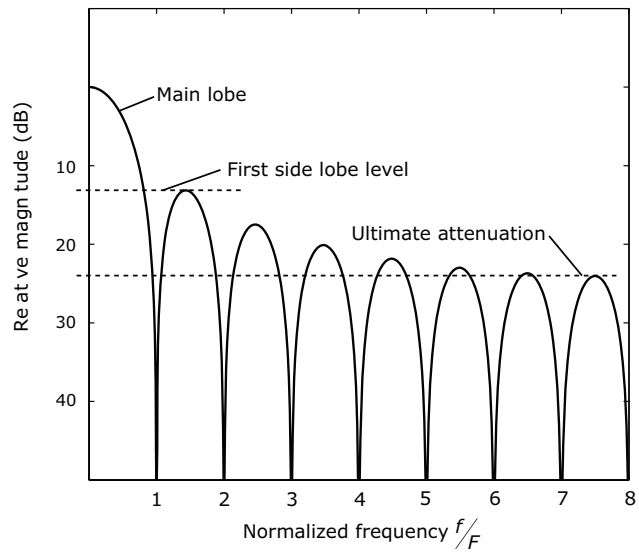


Figure 23.1 DTFT response of a rectangular window showing the features of the response's lobed structure

23.2 Side Lobe Attenuation

There are two useful measures of side lobe attenuation.

1. Peak level of the first side lobe.
2. Ultimate attenuation of side lobes far removed from the main lobe. In the case of discrete-time windows, the ultimate attenuation is the attenuation at the side lobe peak closest to $\omega = \pi$.

Poor side lobe attenuation degrades the two-tone detection capability of a window, especially in cases where one tone is much stronger than the other. Even when the tones' frequencies are separated by more than the 6-dB bandwidth of the window's main lobe, the weaker signal often is lost in the side lobes of the stronger signal's response.

Poor side lobe attenuation also causes bias in the amplitude and frequency estimates formed by the DFT. The side lobes of the negative-frequency component for a tone create bias in the positive-frequency component, and the side lobes of the positive-frequency component create bias in the negative-frequency component.

23.3 Equivalent Noise Bandwidth

The magnitude-squared response for a typical window function is shown in Figure 23.2. If the window function is applied to a signal consisting of zero-mean white noise, the total noise power in the result can be computed in the frequency domain as

$$P_W = \frac{N_0}{2} \int_{-(2T)^{-1}}^{(2T)^{-1}} \left|W(f)\right|^2 df \tag{23.2}$$

Figure 23.2 also shows a dashed-line rectangle that can be thought of as the magnitude response of an ideal lowpass filter. This rectangle has a height equal to the peak in the window's response and a width equal to $2B_{\text{neq}}$. The output noise power from the ideal filter is

$$P_F = N_0 B \left[W(0)\right]^2 \tag{23.3}$$

The value of B for which the window and the ideal filter produce the same output noise power is called

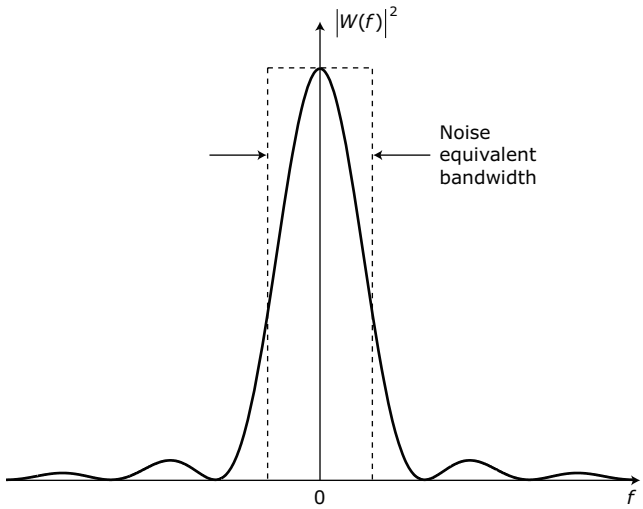


Figure 23.2 Equivalent noise bandwidth. The total area under the curve equals the area of the dashed rectangle.

the *equivalent noise bandwidth*[1], and is found by equating Eqs. (23.2) and (23.3) and solving for B to yield

$$B_{\text{neq}} = B = \frac{\int_{-(2T)^{-1}}^{(2T)^{-1}} \left|W(f)\right|^2 df}{2\left[W(0)\right]^2} \tag{23.4}$$

We can invoke Parseval's theorem to obtain time-domain expressions for the numerator and denominator of Eq. (23.4) to yield a more convenient equation for B_{neq} that is based directly on the time-domain window function

$$B_{\text{neq}} = \frac{\sum_n \left[w(nT)\right]^2}{2\left[\sum_n w(nT)\right]^2} \tag{23.5}$$

The index limits for the summations are not stated explicitly, because in some applications n will range from $-N/2$ through $N/2$ and in other applications n will range from 0 through $N-1$.

23.4 Processing Gain

The processing gain of a windowed transform can be analyzed by considering the DFT to be a bank of matched filters, with each filter matched to a sinusoid at one of the bin frequencies, ω_k. The windowed transform's processing gain, G_P, is defined as the ratio of output signal-to-noise ratio (SNR) to input SNR, when the input to the transform is a noisy sinusoid at one of the transform's bin frequencies:

$$G_P = \frac{S_{\text{out}} / N_{\text{out}}}{S_{\text{in}} / N_{\text{in}}} \tag{23.6}$$

when the input signal is defined by

$$x[n] = A\exp\left(j\omega_k nT\right) + \rho\left(nT\right) \tag{23.7}$$

where

$$\omega_k = \frac{2\pi}{NT}k \tag{23.8}$$

and $\rho(nT)$ is a white noise sequence with variance σ_ρ^2. The input signal power is simply $S_{\text{in}} = A^2$, and the input noise power is $N_{\text{in}} = \sigma_\rho^2$. That portion of the windowed transform's output that is due to signal is given by

$$\begin{aligned} X_S(\omega_k) &= \\ &\sum_{n=0}^{N-1} w(nT)A\exp(j\omega_k nT)(\exp(-j\omega_k nT) \\ &= A\sum_{n=0}^{N-1} w(nT) \end{aligned} \tag{23 9}$$

and the corresponding signal power is given by

$$S_{\text{out}} = \left|X_S(\omega_k)\right|^2 = A^2\left[\sum_{n=0}^{N-1} w(nT)\right]^2 \tag{23.10}$$

The *coherent gain* is defined as the magnitude of the output signal divided by the magnitude of the input signal:

$$G_{\text{coh}} = \frac{\left|A\sum_{n=0}^{N-1} w(nT)\right|}{\left|A\right|} = \sum_{n=0}^{N-1} w(nT) \tag{23.11}$$

The portion of the windowed transform's output due to noise is given by

$$X_N(\omega_k) = \sum_{n=0}^{N-1} w(nT)\rho(nT)\exp(-j\omega_k nT) \tag{23.12}$$

and the output noise power is the mean-square value of $X_N(\omega_k)$:

$$\begin{aligned} N_{\text{out}} &= \mathcal{E}\left\{\left|X_N(\omega_k)\right|^2\right\} \\ &= \sigma_\rho^2\sum_{n=0}^{N-1}\left[w(nT)\right]^2 \end{aligned} \tag{23.13}$$

1. Equivalent noise bandwidth is also called *noise equivalent bandwidth* or simply *noise bandwidth*. The subscript "neq" is used here because "eqn" is so often used as an abbreviation for "equation."

Thus, the processing gain can be determined as

$$G_P = \frac{S_{\text{out}} / N_{\text{out}}}{S_{\text{in}} / N_{\text{in}}}$$

$$= \frac{A^2 \left[\sum_{n=0}^{N-1} w(nT)\right]^2 / \sigma_\rho^2 \sum_{n=0}^{N-1} \left[w(nT)\right]^2}{A^2 / \sigma_\rho^2} \tag{23.14}$$

$$= \frac{\left[\sum_n w(nT)\right]^2}{\sum_n \left[w(nT)\right]^2} = \frac{1}{2B_{\text{neq}}}$$

where B_{neq} is the equivalent noise bandwidth discussed in Section 23.3. The processing loss, L_P, is the reciprocal of the processing gain

$$L_P = \frac{1}{G_P} = 2B_{\text{neq}} \tag{23.15}$$

23.5 Scalloping Loss

The processing loss computed in the previous section is based on an input signal that is a sinusoid at a frequency corresponding to one of the DFT bin frequencies. There will be additional loss for an input sinusoid at any frequency that is not one of the bin frequencies. The maximum loss will occur for an input frequency that is exactly midway between two consecutive bin frequencies. The input signal becomes

$$x[n] = A\exp\left(j\omega_{(k+1/2)}nT\right) + \rho(nT)$$

where

$$\omega_{(k+1/2)} = \frac{2\pi}{NT}(k + 1/2)$$

The portion of the windowed transform's output in bin k that is due to a noise-free sinusoidal signal of frequency $\omega_{(k+1/2)}$ is then given by

$$X_S\left(\omega_{(k+1/2)}\right) = \sum_{n=0}^{N-1} w(nT) A \exp(j\omega_{(k+1/2)}nT)\exp(-j\omega_k nT)$$

After some trigonometric manipulation, this output component can be expressed as

$$X_S\left(\omega_{(k+1/2)}\right) = A\sum_{n=0}^{N-1} w(nT)\exp(j\frac{\pi}{N}n)$$

The *scalloping loss* is defined as the ratio of the coherent gain for a signal midway between two bin frequencies to the coherent gain for a signal at a bin frequency:

$$L_S = \frac{G_{\text{coh}}\left(\omega_{(k+1/2)}\right)}{G_{\text{coh}}(\omega_k)} = \frac{\left|\sum_{n=0}^{N-1} w(nT)\exp\left(j\frac{\pi}{N}n\right)\right|}{\sum_{n=0}^{N-1} w(nT)} \tag{23.16}$$

$$= \frac{\left|W\left(\frac{\pi}{NT}\right)\right|}{W(0)}$$

23.6 Worst Case Processing Loss

The worst case processing loss is defined as the product of the processing loss from Eq. (23.15) and the scalloping loss from Eq. (23.16).

References

1. f. j. harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proc. IEEE*, vol. 66, no. 1, January 1978, pp. 51–83.

Window Choices

In this note, some of the more useful windows and some of the more commonly encountered windows are compared in terms of the characteristics discussed in Note 23. The characteristics of all windows discussed in this note are listed together in Table 24.1 for easy reference.

24.1 Rectangular Window

The rectangular window is somewhat different from all of the other window functions in that rather than being explicitly applied to a data sequence, it instead represents the situation when a long or infinite sequence is truncated to form a shorter-duration segment. Modeling the truncation process as a windowing operation does provide a convenient mechanism for analyzing DFT leakage, as demonstrated in Note 15. The rectangular window is included in this note because it serves as a baseline from which the improvements offered by other windows can be gauged.

Truncating a DFT input sequence to a length of N samples can be mathematically modeled as multiplying the sequence by a *rectangular window* sequence that has non-zero values only for sample indices 0 through $N-1$:

$$w[n] = \begin{cases} 1 & 0 \le n < N \\ 0 & \text{otherwise} \end{cases} \quad (24.1)$$

For a normalized sampling interval of $T = 1$, the DTFT of the rectangular window defined by Eq.(24.1) is given by

$$W(f) = \exp\left[-j\pi f(N-1)\right]\frac{\sin(N\pi f)}{\sin(\pi f)} \quad (24.2)$$

The exponential factor in (24.2) has a magnitude of 1 for all values of f, and represents a simple linear phase shift. This phase shift is a consequence of defining the window over the interval $0 \le n < N$. If the

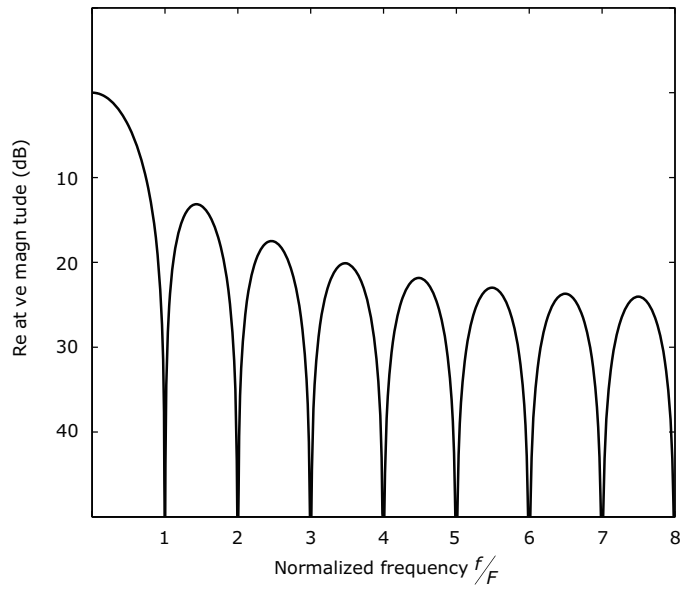


Figure 24.1 DTFT magnitude response for a 16-point rectangular window

rectangular window is defined to be symmetric about the origin, then the DTFT of the window is

$$W_{\text{sym}}(f) = \frac{\sin(N\pi f)}{\sin(\pi f)} \qquad (24.3)$$

The DTFT magnitude response for a 16-point rectangular window is shown in Figure 24.1. The null-to-null width of the main lobe for the rectangular window is 2*F*, where *F* is the DFT bin spacing. The peaks of the first side lobes are about 13 dB below the peak of the main lobe. The ultimate side lobe attenuation is about 30 dB. As shown in Figure 24.1, the rectangular window's response has nulls at all integer multiples of the frequency increment $F = (NT)^{-1}$. This spacing of the nulls means that for a sinusoidal signal with a frequency of $f = kF$, the DFT has a non-zero response only in bins k and $N - k$. However, for a sinusoidal signal with a frequency that is not an integer multiple of F, the DFT exhibits some response in *every* bin. As shown in Figure 24.2, when the main lobe of the rectangular window's response is centered on a frequency component that falls between two adjacent DFT bin frequencies, the main lobe straddles the two adjacent bins. When the signal is near the midpoint between two bins, the response in each of the two straddled bins is significant. However, as the signal is moved closer to one of the bins, the response in the other straddled bin becomes less significant.

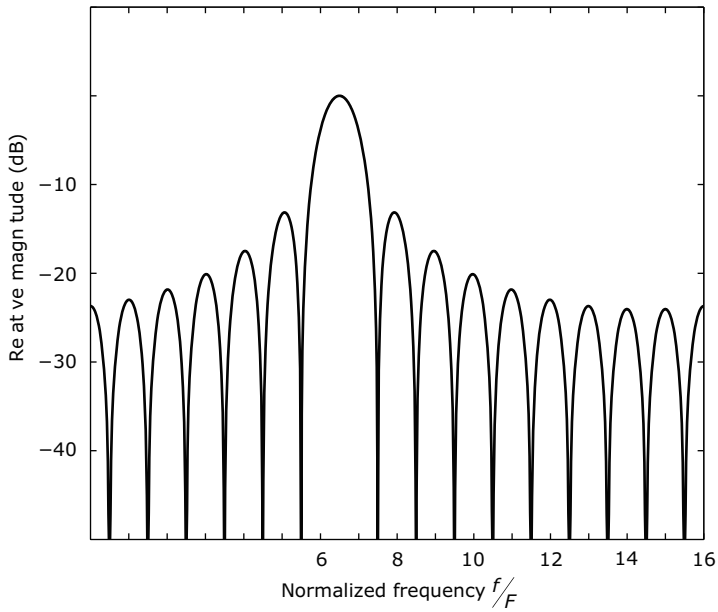


Figure 24.2 DTFT magnitude response for 16-point rectangular window, after shifting the center of the main lobe on a normalized frequency of 6.5

24.2 Triangular Window

The triangular window's response has nulls at even-valued integer multiples of the DFT frequency increment. As shown in Figure 24.3, the side lobe peaks occur at frequencies very close to odd-valued

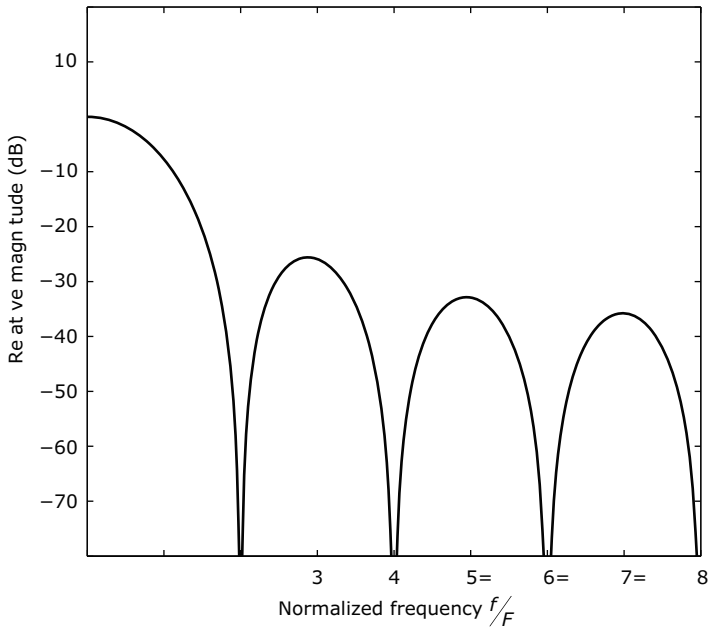


Figure 24.3 DTFT magnitude response for a 16-point triangular window

integer multiples of the frequency increment. Furthermore, the null-to-null width of the main lobe in the response spans an interval of $4F$. Therefore, when the triangular window's response is centered on bin k, the responses at bins $k-1$ and $k+1$ are down only 7.8 dB from the peak. The responses at bins $k \pm 3$ are down by about 26.7 dB from the peak at bin k. When the signal frequency is not an integer multiple of F, the main lobe straddles four consecutive DFT bins, as shown in Figure 24.4. The ultimate side lobe attenuation is about 48 dB.

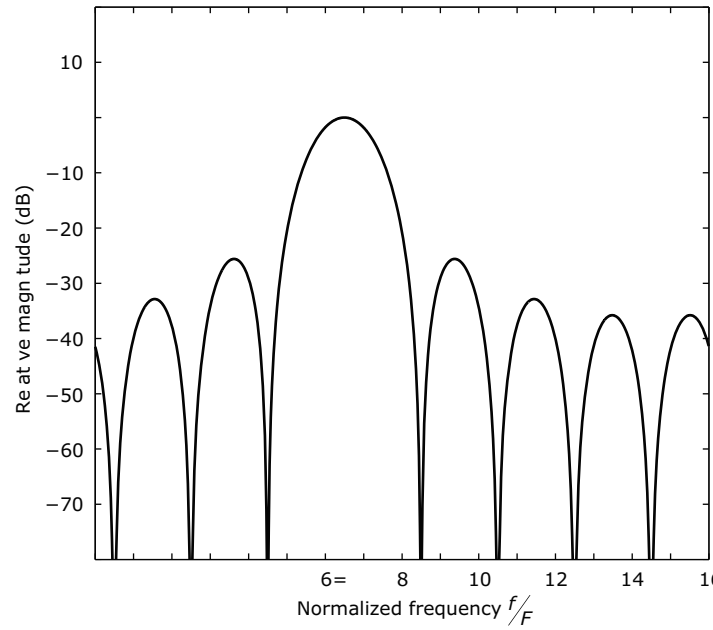


Figure 24.4 DTFT magnitude response for a 16-point triangular window after shifting to center the main lobe on a normalized frequency of 6.5

Table 24.1 Window Characteristics

Window	α	Side lobe peak (dB)	Side lobe fall-off rate (dB per octave)	3dB bandwidth, B_{3dB} (bins)	6 dB bandwidth, B_{6dB} (bins)	Equiv. noise bandwidth, B_{neq} (bins)	B_{neq}/B_{3dB}	Scalloping loss (dB)	Worst-case processing loss (dB)
Rectangular	—	−13	−6	0.89	1.21	1.00	1.1236	3.92	3.92
Triangular	—	−27	−12	1.28	1.78	1.33	1.0391	1.82	3.07
Dolph-Chebyshev	2.5	−50	0	1.33	1.85	1.39	1.0451	1.70	3.12
	3.0	−60	0	1.44	2.01	1.51	1.0486	1.44	3.23
	3.5	−70	0	1.55	2.17	1.62	1.0452	1.25	3.35
	4.0	−80	0	1.65	2.31	1.73	1.0485	1.10	3.48
Kaiser	2.0	−46	−6	1.43	1.99	1.50	1.049	1.46	3.20
	2.5	−57	−6	1.57	2.20	1.65	1.051	1.20	3.38
	3.0	−69	−6	1.71	2.39	1.80	1.0526	1.02	3.56
	3.5	−82	−6	1.83	2.57	1.93	1.0546	0.89	3.74
Hann	—	−32	−18	1.44	2.00	1.50	1.0417	1.42	3.18
Hamming	—	−43	−6	1.30	1.81	1.36	1.0462	1.78	3.10

24.3 Dolph-Chebyshev Window

The Dolph-Chebyshev window has its origins in antenna design [2] and its response has the minimum main-lobe width for a given side lobe level. As depicted in Figure 24.5, all the side lobes peak at the same level. For a side lobe level of -20α, the window's discrete-frequency response is given by

$$W(k) = (-1)^k \frac{\cos\left[N\cos^{-1}\left[\beta\cos\left(\pi\frac{k}{N}\right)\right]\right]}{\cosh\left[N\cosh^{-1}(\beta)\right]} \qquad 0 \le |k| \le N-1 \tag{24.4}$$

where

$$\beta = \cosh\left[\frac{1}{N}\cosh^{-1}\left(10^{\alpha}\right)\right]$$

Often, $\beta > 1$ and, consequently, evaluation of Eq. (24.4) may entail computing the inverse cosine of values with magnitudes greater than unity. In such cases, the following formula can be used:

$$\cos^{-1}(X) = \begin{cases} \frac{\pi}{2} - \tan^{-1}\left(\frac{x}{\sqrt{1-x^2}}\right), & |x| \le 1 \\ \ln\left(x + \sqrt{x^2-1}\right), & |x| > 1 \end{cases}$$

The time-domain window coefficients are obtained by taking the inverse DFT of Eq. (24.4).

24.4 Kaiser Window

The discrete-time Kaiser window is defined by

$$w[n] = \frac{I_0\left[\pi\alpha\sqrt{1-\left(1-\frac{2n}{N-1}\right)^2}\right]}{I_0(\pi\alpha)} \qquad 0 \le n \le N-1$$

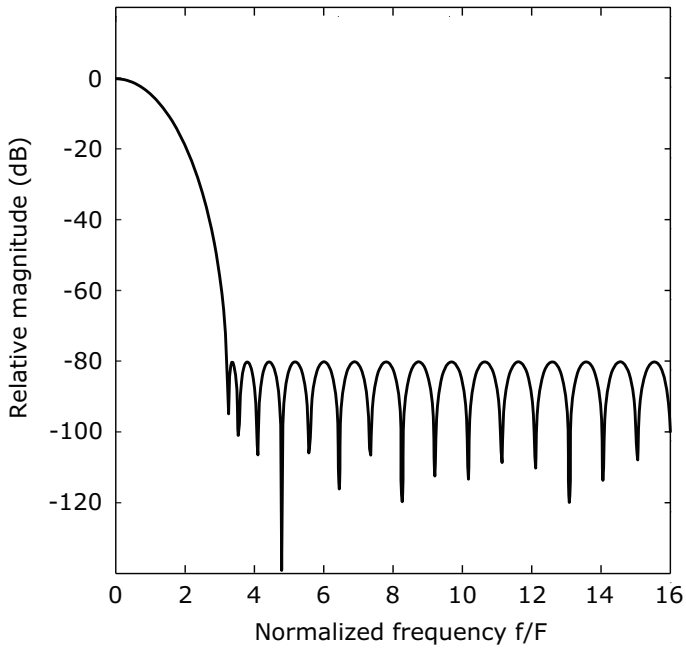


Figure 24.5 DTFT magnitude response for a 32-point Dolph-Chebyshev window

where I_0 is the zero-order modified Bessel function of the first kind, given by

$$I_0(x) = \sum_{k=0}^{\infty} \left[\frac{(x/2)^k}{k!} \right]^2$$

The Kaiser window is discussed further in Note 25, and its use in designing FIR filters is discussed in Note 36.

24.5 Hann Window

The discrete-time, odd-length Hann *lag* window is defined as

$$w[n] = \cos^2\left(\frac{n\pi}{N}\right) = \frac{1}{2}\left[1 + \cos\left(\frac{2\pi n}{N}\right)\right]$$

$$n = \frac{-(N-1)}{2}, \ldots, -1, 0, 1, \ldots, \frac{(N-1)}{2}$$

and the discrete-time (usually even-length) Hann *data* window is defined as

$$w[n] = \sin^2\left(\frac{n\pi}{N}\right) = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{N}\right)\right]$$

$$n = 0, 1, 2, \ldots, N-1$$

24.6 Hamming Window

The discrete-time, odd-length Hamming *lag* window is defined as

$$w[n] = 0.54 + 0.46\cos\left(\frac{2\pi n}{N}\right)$$

$$n = \frac{-(N-1)}{2}, \ldots, -1, 0, 1, \ldots, \frac{(N-1)}{2}$$

and the discrete-time (usually even-length) Hamming *data* window is defined as

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

$$n = 0, 1, 2, \ldots, N-1$$

References

1. f. j. harris, "On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proc. IEEE*, vol. 6, no. 1, pp. 51–83, January 1978.
2. C. L. Dolph, "A Current Distribution for Broadside Arrays Which Optimizes the Relationship Between Beam Width and Side-Lobe Level," *Proc. IRE*, vol. 35, June 1946, pp. 335–348.

Kaiser Windows

One way to structure the window optimization problem is to find the particular time-limited function having the minimum energy outside the main lobe of its frequency response. The solution to this problem, found by Slepian and Pollak [1], involves prolate spheroidal wave functions. Kaiser [2, 3] found an approximate solution that is simpler to compute than the exact solution of Slepian and Pollak. The continuous-time form of the Kaiser window and its frequency response are listed in Math Box 25.1. The specification of the continuous-time window is such that it spans a time interval of $(N-1)T$. This span is equivalent to saying that the continuous-time window begins at the time corresponding to sample $-(N-1)/2$ and ends at the time corresponding to sample $(N-1)/2$, where N is odd.

The definition and properties for the discrete-time window are summarized in Math Box 25.2. A closed-form expression has not been found for the frequency response of the discrete-time window. The response given in Eq. (MB 25.4) is simply the window function's DTFT after simplification, based on the window's even symmetry around its central sample. Equation (MB 25.5) is a closed-form approximation of the spectrum derived by Antoniou in [4], and Eq. (MB 25.6) is a closed-form approximation given by f. j. harris in the famous paper [5], in which he catalogs the properties of many different window functions. In this same paper, harris refers to the window defined by Eq. (MB 25.2) as the Kaiser-Bessel window.

Essential Facts

- A time-limited window optimized to have minimum energy outside the main lobe of its frequency response takes the form of a *prolate spheroidal wave function*.
- The Kaiser windows are an easier-to-compute family of approximations to the prolate spheroidal wave functions.
- The definition of the Kaiser window includes a parameter, β, that can be used to adjust the trade-off between peak side lobe ripple and the width of the main lobe in the window's frequency response.
- Because of the extra degree of design freedom provided by the parameter β, the Kaiser window is the one most often used in the windowing technique for designing linear phase FIR filters that is described in Note 35.

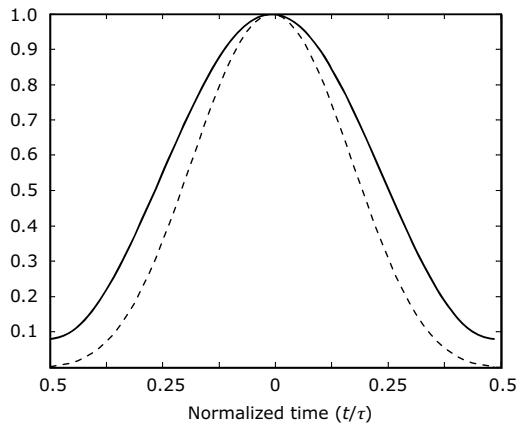


Figure 25.1 Kaiser window with $\beta = 5.4414$ (solid trace) compared to Hamming window (dotted trace)

The Kaiser window performs well in comparison to other windows. Figure 25.1 shows a Hamming window and a Kaiser window with $\beta = \pi\sqrt{3}$. As shown in Figure 25.2, the main lobe width for the Kaiser window's response matches the main lobe width for the Hamming response, but the side lobes roll off more quickly.

As shown in Figure 25.3, for $\beta = 8.5$, the main lobe of the Kaiser window's response is slightly narrower than the Blackman main lobe. The close-in side lobes are almost 10 dB lower for the Kaiser window; however, the Blackman side lobes continue to drop until they fall below the Kaiser side lobes at approximately $f = F_S/4$. The difference in the time domain is not so dramatic. As shown in Figure 25.4, the difference between the Blackman window and the Kaiser window with $\beta = 8.5$ is barely discernible. The area of greatest difference is in the tails, as shown in the zoomed area.

Math Box 25.1

Continuous-Time Kaiser Window

The continuous-time Kaiser window is defined as

$$w_c(t) = \begin{cases} \dfrac{I_0\left[\beta\sqrt{1-\left(1-\dfrac{2t}{(N-1)T}\right)^2}\right]}{I_0(\beta)} & \text{for } |t| \le \dfrac{(N-1)T}{2} \\ 0 & \text{otherwise} \end{cases} \quad \textbf{(MB 25.1)}$$

In [3], Kaiser gives the spectrum of (MB 25.1) as

$$W_c(\omega) = \frac{2\sin\left(\tau\sqrt{\omega^2 - (\beta/\tau)^2}\right)}{I_0(\beta)\sqrt{\omega^2 - (\beta/\tau)^2}}$$

where

$$\tau = \frac{(N-1)T}{2}$$

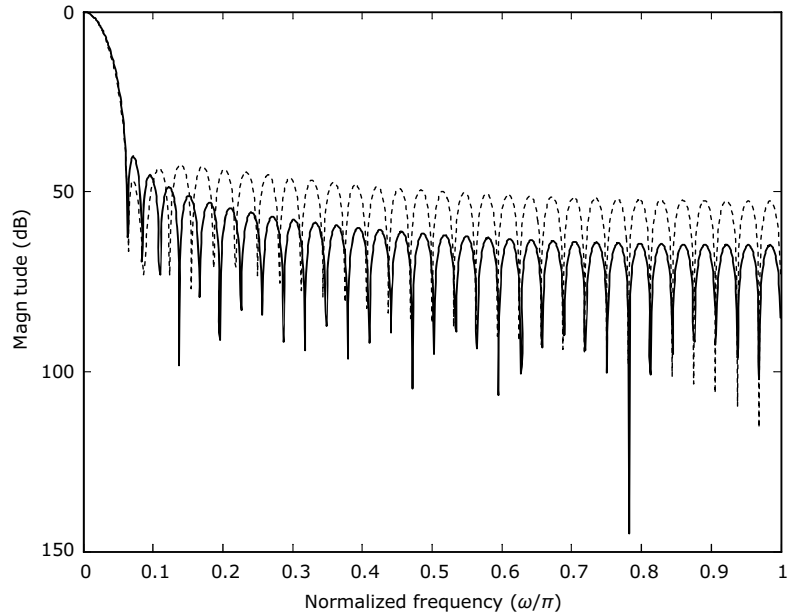


Figure 25.2 Comparison of magnitude spectra for a Hamming window (dashed trace) and for a Kaiser window with $\beta = 5.4414$ (solid trace)

Math Box 25.2

Discrete-Time Kaiser Window

The Kaiser data window is defined as

$$w[n] = \frac{I_0\left[\beta\sqrt{1-\left(1-\frac{2n}{N-1}\right)^2}\right]}{I_0(\beta)} \qquad 0 \le n \le N-1 \tag{MB 25.2}$$

where I_0 is the zero-order modified Bessel function of the first kind given by

$$I_0(x) = \sum_{k=0}^{\infty}\left[\frac{(x/2)^k}{k!}\right]^2 \tag{MB 25.3}$$

In most practical applications, an adequate approximation for $I_0(x)$ can be obtained from the first 20 terms in the summation of Eq. (MB 25.3). The spectrum of the Kaiser window is given by

$$W(\omega) = w[0] + 2\sum_{k=0}^{\infty} w[n]\cos(\omega n T) \tag{MB 25.4}$$

The spectrum can be approximated as

$$W(\omega) \approx \frac{(N-1)\sin\left[\beta\sqrt{(\omega\tau/\beta)^2-1}\right]}{\beta I_0(\beta)\sqrt{(\omega\tau/\beta)^2-1}} \tag{MB 25.5}$$

or

$$W(\omega) \approx \frac{N\sinh\left[\sqrt{\beta^2-(N\omega/2)^2}\right]}{I_0(\beta)\sqrt{\beta^2-(N\omega/2)^2}} \tag{MB 25.6}$$

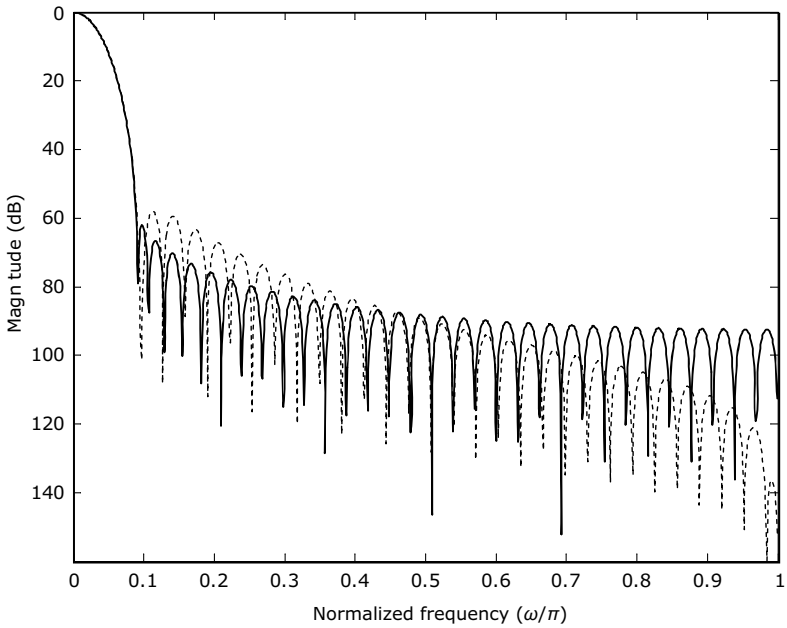


Figure 25.3 Comparison of magnitude spectra for a Blackman window (dashed trace) and for a Kaiser window with $\beta = 8.5$ (solid trace)

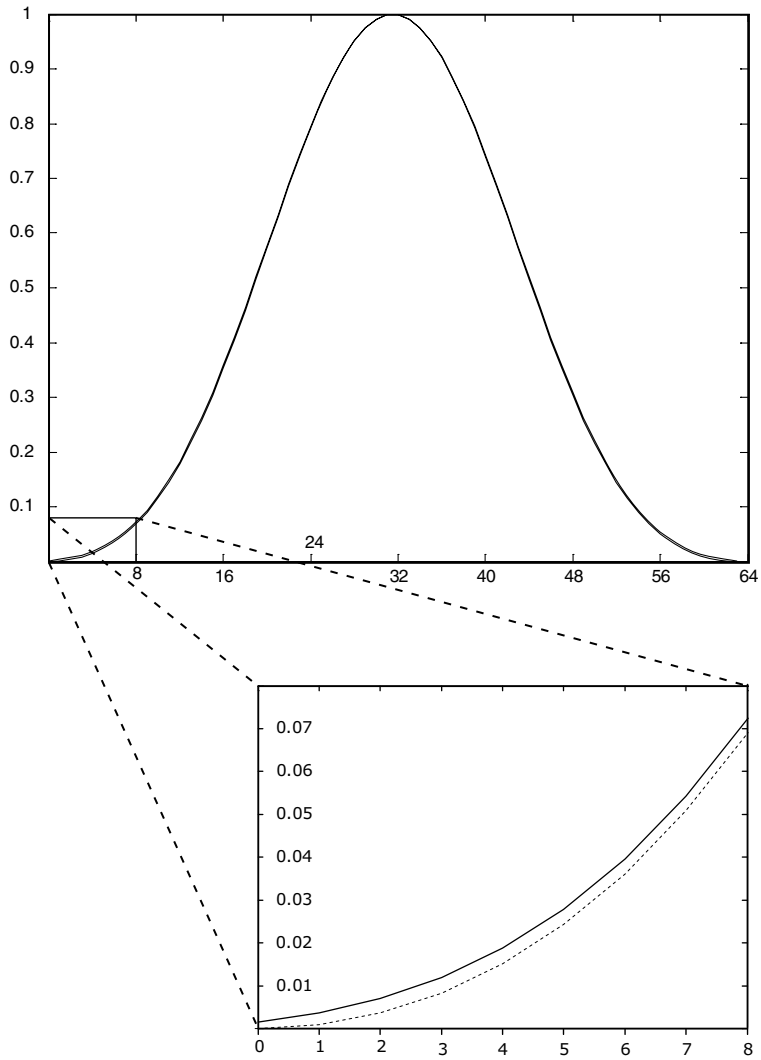


Figure 25.4 The difference between the Blackman window and the Kaiser window with $\beta = 8.5$ really can be observed only in the magnified portions of the window tails. At the point of greatest difference, the Blackman window (dotted trace) is lower than the Kaiser window (solid trace) by approximately 3×10^{-3}.

References

1. D. Slepian and H. Pollak, "Prolate-Spheroidal Wave Functions, Fourier Analysis and Uncertainty—I," *Bell Syst. Tech. J.*, vol. 40, January 1961, pp. 43–64.
2. J. F. Kaiser, "Digital Filters," Chapter 7 in *System Analysis by Digital Computer*, F. F. Kuo and J. F. Kaiser, eds., John Wiley & Sons, 1966.
3. J. F. Kaiser, "Nonrecursive Digital Filter Design Using the I_0-sinh Window Function," *Proc. 1974 IEEE Int. Symp. on Circuits and Syst.*, April 22–25, 1974, pp. 20–23.
4. A. Antoniou, *Digital Filters: Analysis and Design*, McGraw-Hill, 1979.
5. f. j. harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proc. IEEE*, vol. 66, no. 1, January 1978, pp. 51–83.

Unmodified Periodogram

This note presents the definition and basic properties of the unmodified periodogram. The unmodified periodogram is the simplest of the classical, nonparametric, DFT-based methods that are used to estimate the power spectrum of a discrete-time signal. Examples of its performance against various signal combinations and noise conditions are also presented.

For a discrete-time signal, $x[n]$, the unmodified periodogram, $S_U[m]$, is computed as the normalized, squared magnitude of the N-point DFT for an L-sample segment of the signal that has been padded with $N-L$ zeros (see Note 16: Exploring DFT Resolution for a discussion of zero-padding in the DFT):

$$S_U[m] = \frac{1}{N}\left|X[m]\right|^2 \qquad \textbf{(26.1)}$$

where $X[m]$ is the slightly modified DFT given by

$$X[m] = \sum_{n=0}^{L-1} x[n]\exp\left(\frac{-j2\pi mn}{N}\right) \qquad \textbf{(26.2)}$$

In order to generate a periodogram plot that appears to be a function of continuous frequency, it is a common practice to use a DFT length, N, that is many times larger than the number of signal samples, L. In fact, the periodogram is formally defined as a function of continuous frequency that involves the DTFT of $x[n]$. However, in practice, the periodogram is evaluated at a number of discrete frequencies using the DFT as in Eq. (26.2). It is not necessary to include the $N-L$ zero-valued padding samples explicitly in Eq. (26.2). The denominator in the exponential factor is based on an N-point DFT, but the zero-valued samples are implicitly included by having the upper summation limit of $L-1$ rather than $N-1$.

Essential Facts

- Historically, periodograms have been defined as functions of continuous frequency that are related to the corresponding discrete-time signal via the discrete-time Fourier transform (DTFT). However, in practice, the DFT or FFT is used to evaluate the DTFT at only a finite number of discrete frequencies, leading to a result that is a function of discrete frequency.
- The unmodified periodogram is the simplest of the classical Fourier-transform-based spectrum estimation techniques.
- The unmodified periodogram's response to a distinct sinusoidal component has the narrowest main lobe response available from any periodogram technique.
- The main disadvantages of the unmodified periodogram include
  - High side lobe leakage
  - No reduction in additive noise that may be present in the signal
  - High variability in the result when the input signal is inherently random (such as a communications signal that has been modulated by a random data sequence)

The unmodified periodogram is also referred to as the *sample spectrum* or simply *periodogram*. The use of the descriptor "unmodified" is primarily to emphasize the distinction between Eq. (26.1) and the modified periodogram that is discussed in Note 29.

The unmodified periodogram's advantage over other spectrum estimation techniques is its simplicity and relatively low computational burden (particularly if the periodogram is evaluated using a DFT that is sized to allow implementation via an efficient FFT algorithm). These advantages can be decisive in an application such as a front-panel spectrum display in a consumer audio device. In such an application, the performance requirements might be quite lax—the high and low frequency content of the displayed spectrum needs only to appear correlated to the audio that is playing. In addition, because the input signal is truncated with a simple rectangular window, the unmodified periodogram's response to a distinct sinusoidal component has the narrowest main-lobe response available from any periodogram technique.

References

1. S. L. Marple, *Digital Spectral Analysis with Applications,* Prentice Hall, 1987.
2. M. H. Hayes, *Statistical Digital Signal Processing and Modeling,* Wiley, 1996.

Math Box 26.1

Properties of the Unmodified Periodogram

Definition

$$S_U\left(e^{j\omega}\right)=\frac{1}{N}\left[\sum_{n=0}^{N-1}x[n]\exp\left(-jn\omega\right)\right] \qquad \textbf{(MB 26.1)}$$

Bias

$$\mathcal{E}\left\{S_U\left(e^{j\omega}\right)\right\}=\frac{1}{2\pi}S_x\left(e^{j\omega}\right)\otimes W_B\left(e^{j\omega}\right) \qquad \textbf{(MB 26 2)}$$

where $\mathcal{E}$ indicates expected value, $S_x(e^{j\omega})$ is the true power spectrum of x, and $W_B(e^{j\omega})$ is the Fourier transform of the Bartlett window. The presence of $W_B(e^{j\omega})$ in Eq. (MB 26.2) indicates that the unmodified periodogram is a *biased* estimate of $S_x(e^{j\omega})$. However, as N becomes very large, $W_B(e^{j\omega})$ approaches an impulse function, so

$$\lim_{N\to\infty}\mathcal{E}\left\{S_U\left(e^{j\omega}\right)\right\}=S_x\left(e^{j\omega}\right) \qquad \textbf{(MB 26 3)}$$

Therefore, the unmodified periodogram is said to be an asymptotically unbiased estimate of $S_x(e^{j\omega})$. The appearance of a Bartlett window in Eq. (MB 26.2) might seem odd given that the unmodified periodogram does not involve the explicit use of a window function. However, like all "unwindowed" DFTs, the periodogram does implicitly involve a rectangular window to accomplish the truncation of the input sequence to just N nonzero samples. By Parseval's theorem, the power spectrum is the Fourier transform of the autocorrelation function, and the autocorrelation of two rectangular windows is a triangular window—hence the convolution by $W_B(e^{j\omega})$ in Eq. (MB 26.2).

6-dB Bandwidth of Bin Response

$$\omega_{6\text{dB}}=0.89\frac{2\pi}{N} \qquad \textbf{(MB 26.4)}$$

Variance

$$\text{var}\left\{S_U\left(e^{j\omega}\right)\right\}\approx\left[S_x\left(e^{j\omega}\right)\right]^2 \qquad \textbf{(MB 26 5)}$$

Exploring Periodogram Performance: Sinusoids in Additive White Gaussian Noise

This note demonstrates how the unmodified periodogram's ability to resolve sinusoidal components with distinct frequencies varies with both the signal-to-noise ratio and the frequency separation of the components. In subsequent notes, similar techniques are used to assess the performance of other types of periodograms—most of which outperform the unmodified periodogram.

27.1 SNR Variations

The MATLAB code in Computer Listing 27.1 generates two cosine waves embedded in additive white Gaussian noise (AWGN) and then computes the unmodified periodogram for the composite signal. This code was run for the combinations of frequency and SNR listed in Table 27.1, with the results plotted in the figures as indicated by the table entries.

When the SNR is set to 200 dB, the noise in the signal can be considered nonexistent. For Case 1, the frequencies of the two sinusoids are symmetric about the normalized frequency 0.125, which corresponds to FFT bin 32. The frequencies selected provide a separation equal to the 6-dB bandwidth of

Computer Listing 27.1

Unmodified Periodogram for Sinusoids in AWGN

The following segment of MATLAB code computes the unmodified periodogram for a 256-sample segment of two equal amplitude cosine waves in AWGN.

```
%
% Specifics for current demonstration
%
N=256;
noise_seed = 113057;
snr = 200;
freq1 = 31.555/N;
freq2 = 32.445/N;
ti = 0:N-1;
x1 = cos(ti*2*pi*freq1);
x2 = cos(ti*2*pi*freq2);
x = x1 + x2;
y = awgn(x,snr,'measured',noise_seed);
%
% Computing the unmodified periodogram
%
S_U = periodogram(y,[],'twosided');
%
% Plotting result
%
abscissa = 0:1:N/2;
abscissa = abscissa/N;
plot(abscissa, 10*log10(S_U));
```

Table 27.1 Cases Demonstrated in Section 27.1

Case	`freq1`	`freq2`	Δf	`snr`	Figure
1	$31.555/N = 0.123262$	$32.445/N = 0.126738$	$0.89/N = 0.003477$	200	27.1
2	$31.555/N = 0.123262$	$32.445/N = 0.126738$	$0.89/N = 0.003477$	0	27.2
3	$31.555/N = 0.123262$	$32.445/N = 0.126738$	$0.89/N = 0.003477$	−10	27.3
4	$31.95/N = 0.124805$	$33.05/N = 0.129102$	$1.1/N = 0.004297$	200	27.4
5	$31.55/N = 0.123242$	$33.45/N = 0.130664$	$1.9/N = 0.007422$	200	27.5

the bin response as listed in Math Box 26.1 of Note 26. As shown in Figure 27.1, there are two distinct peaks in the periodogram. Because a DFT is used to generate the periodogram, peaks in the result can occur only at frequencies that correspond to the bin frequencies of the DFT. The peaks in Figure 27.1 fall at $\hat{f}_1 = (31/N) = 0.121094$ and at $\hat{f}_2 = (33/N) = 0.128906$, so the normalized errors in estimating f_1 and f_2 are 0.002168 and -0.002168, respectively.

Cases 2 and 3 demonstrate how the resolvability of closely spaced frequency components degrades as the SNR is decreased. Figure 27.2 shows the periodogram for the case where SNR = 0 dB; the peaks are still distinct and larger than any other frequency components. Figure 27.3 shows the periodogram for the case where SNR = −10 dB; the peaks are no longer discernible.

Increasing Δf from $0.89/N$ to $1.1/N$, as in Case 4, demonstrates that increasing the frequency separation of the sinusoids does not always improve or even maintain the resolvability exhibited in cases having smaller frequency separations. Figure 27.4 shows only a single peak in the periodogram.

Increasing Δf further, to $1.9/N$, as in Case 5, restores two distinct peaks, as shown in Figure 27.5, but the notch between the peaks is not as deep as the notch in Case 1. The shallower notch fills in quickly as the SNR is reduced.

27.2 Frequency Spacing

The periodogram sometimes exhibits counterintuitive results wherein increasing frequency separation in the input signal's frequency components leads to degraded resolvability of peaks in the periodogram. This note demonstrates this phenomenon and reveals the mechansim that causes the

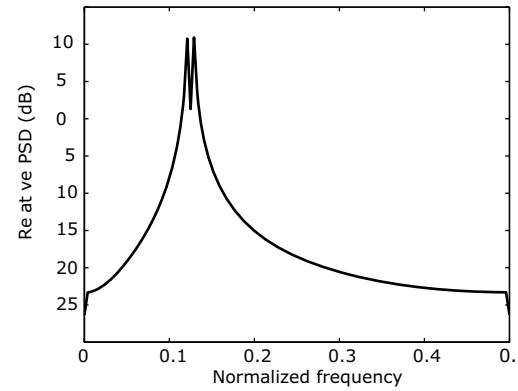


Figure 27.1 Periodogram result for Case 1, with $f_1 = 31.555/N = 0.123262$, $f_2 = 32.445/N = 0.126738$, and SNR = 200 dB

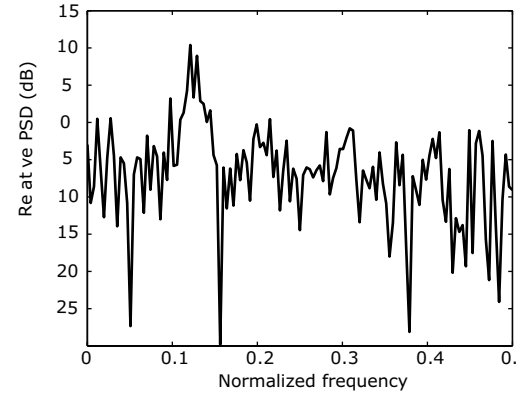


Figure 27.2 Periodogram result for Case 2, with $f_1 = 31.555/N = 0.123262$, $f_2 = 32.445/N = 0.126738$, and SNR = 0 dB

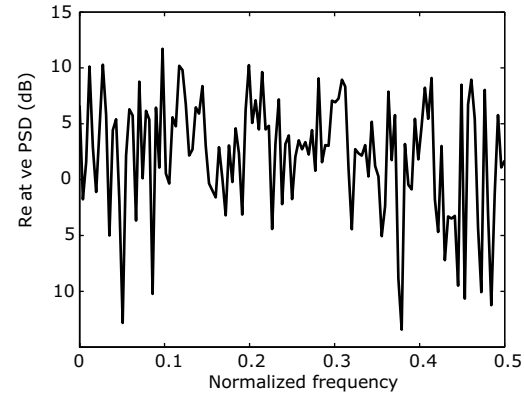


Figure 27.3 Periodogram result for Case 3, with $f_1 = 31.555/N = 0.123262$, $f_2 = 32.445/N = 0.126738$, and SNR = -10 dB

counterintuitive result. The plots generated in this demonstration have been designed to provide a zoomed-in view of the area around the peaks in the periodogram.

- The plots have been modified to include the continuous-frequency result corresponding to the discrete-frequency periodogram. The periodogram represents a sampled version of this continuous-frequency result, and the resolution behavior of the periodogram is governed by the location of the sampling instants relative to the peaks and valleys of the continuous result.
- In this demonstration, the DTFT of the two-sinusoid signal is generated using the Dirichlet kernel, as described in Note 14. The square of the Dirichlet kernel is the *Fejer kernel,* but the continuous-frequency power spectrum for a sum of sinusoids is not a sum of Fejer kernels. The Dirichlet kernels corresponding to the individual components must be summed *before* computing the squared magnitude.
- MATLAB code that generates a periodogram, along with the corresponding continuous-frequency result, is provided in Computer Listing 27.2. When this code is run for the cases listed in Table 27.2, it produces results that are plotted in the specific figures indicated in the right-most column of the table.
- Case 1 corresponds to Case 1 from Section 27.1. As shown in Figure 27.6, the notch between the two peaks falls directly on a bin frequency, and the two adjacent bin frequencies fall close enough to the peaks in the continuous-frequency result to provide good resolvability. The frequencies of the sinusoids are indicated in the plot,

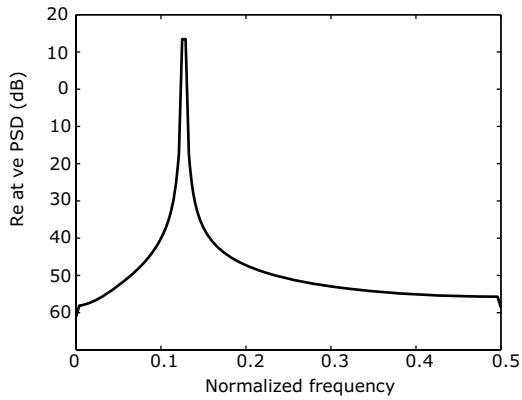


Figure 27.4 Periodogram result for Case 4, with $f_1 = 31.95/N = 0.124805$, $f_2 = 33.05/N = 0.129102$, and SNR = 200 dB

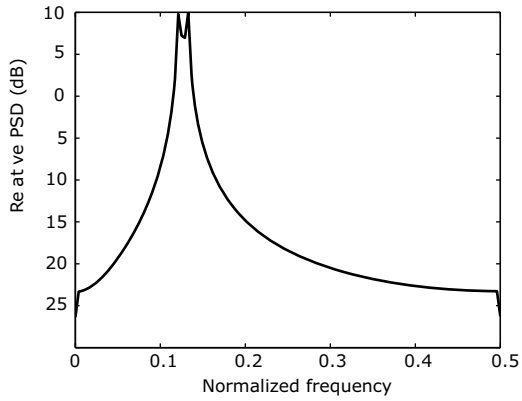


Figure 27.5 Periodogram result for Case 5, with $f_1 = 31.55/N = 0.123242$, $f_2 = 33.45/N = 0.130664$, and SNR = 200 dB

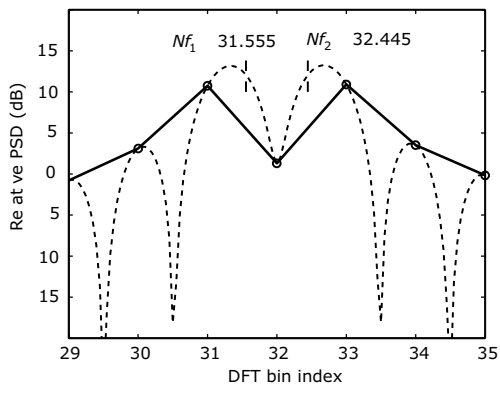


Figure 27.6 Periodogram result for Case 1. The dashed trace indicates the corresponding continuous-frequency power spectrum.

Computer Listing 27.2

Sinusoids in AWGN: Resolution Behavior Illuminated

The following *segment* of MATLAB code computes the unmodified periodogram for a noise-free 256-sample segment of two equal-amplitude cosine waves. The results for DFT bins 29 through 35 are plotted along with the continuous-frequency result corresponding to the discrete-frequency periodogram.

```
N=256;
freq1 = 31.6/N;
freq2 = 33.4/N;
ti = 0:N-1;
x1 = cos(ti*2*pi*freq1);
x2 = cos(ti*2*pi*freq2);
y = x1 + x2;

interp_fact = 10;
x      =      linspace(0,pi*(N-1)/N,1+interp_
fact*(N-1));
x_cyc = x./(2*pi);
x_1_lo = x - 2*pi*freq1;
x_1_hi = x - 2*pi*(1-freq1);
x_2_lo = x - 2*pi*freq2;
x_2_hi = x - 2*pi*(1-freq2);

losinc_1          =          exp(-j*x_1_lo*(N-
1)/2).*(N*0.5*diric(x_1_lo,N));
hisinc_1          =          exp(-j*x_1_hi*(N-
1)/2).*(N*0.5*diric(x_1_hi,N));
losinc_2          =          exp(-j*x_2_lo*(N-
1)/2).*(N*0.5*diric(x_2_lo,N));
hisinc_2          =          exp(-j*x_2_hi*(N-
1)/2).*(N*0.5*diric(x_2_hi,N));

dtft_result = losinc_1 + hisinc_1 + losinc_2
+ hisinc_2;
dtft_result = dtft_result/sqrt(pi*N);
dtft_db = 20*log10(abs(dtft_result));

abscissa = (0:1:N/2)/N;
plot(abscissa,10*log10(S_U),'ko',abscissa,10
*log10(S_U),...
'k',x_cyc,dtft_db,'r');
xlim('manual');
xlim([29/N 35/N]);
```

and these frequencies are offset somewhat from the peaks in the composite continuous-frequency result. The peaks in this composite result are shifted outward from the notch due to destructive interference between the individual Dirichlet kernels comprising the overall result.

- Case 2 has Δf identical to Case 1, but the two frequencies are shifted so that the notch in the continuous-frequency spectrum does not fall on a DFT bin frequency. As a result, the periodogram exhibits a single wide peak, as shown in Figure 27.7.

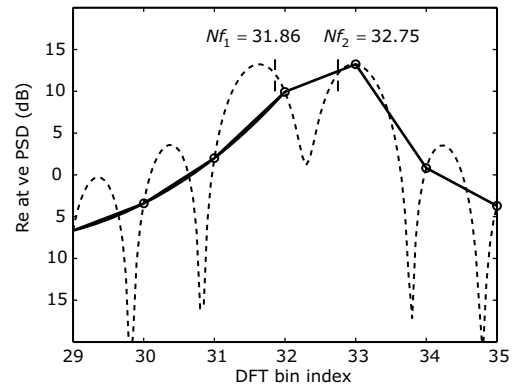


Figure 27.7 Periodogram result for Case 2. The dashed trace indicates the corresponding continuous-frequency power spectrum.

- In Case 3, the frequency f_1 is the same as in Case 1, but Δf is increased to $1.495/N = 0.00584$. The periodogram for this case exhibits a single wide peak, as shown in Figure 27.8.
- In Case 4, Δf has been increased to $1.8/N = 0.007031$, which is more than twice the 6-dB bandwidth of the bin response given in Note 26. As shown in Figure 27.9, the periodogram exhibits a single peak with a plateau that extends over four DFT bins.

Based on the results of this demonstration and the one in Section 27.1, it appears that the periodogram may not be the best analysis tool for signals involving closely spaced frequency components. Improved two-tone resolution performance can usually be obtained from one of the spectrum estimation techniques based on an autoregressive signal model. These techniques are introduced in Note 68. When one of the two tones has a much greater amplitude than the other, it can be difficult to resolve the weaker tone even when it is well separated in frequency from the stronger tone. In these cases, resolvability of the weaker tone often can be improved by the increased side lobe suppression offered by the *modified periodogram* technique described in Note 29.

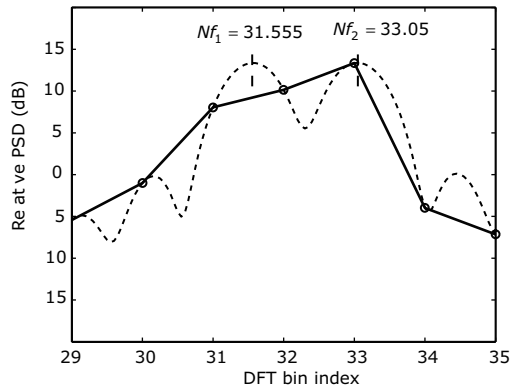


Figure 27.8 Periodogram result for Case 3. The dashed trace indicates the corresponding continuous-frequency power spectrum.

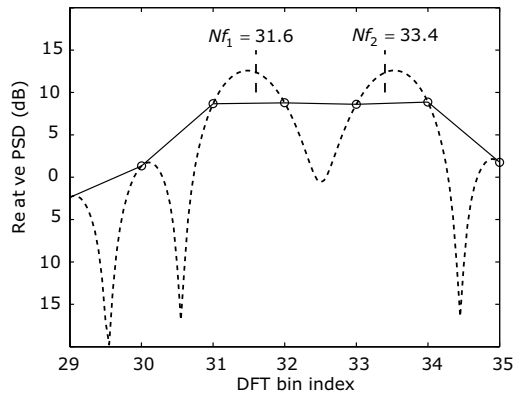


Figure 27.9 Periodogram result for Case 4. The dashed trace indicates the corresponding continuous-frequency power spectrum.

Table 27.2 Cases Demonstrated in Section 27.2

Case	`freq1`	`freq2`	Δf	`snr`	Figure
1	$31.555/N = 0.123262$	$32.445/N = 0.126738$	$0.89/N = 0.003477$	200	27.6
2	$31.86/N = 0.124453$	$32.75/N = 0.127930$	$0.89/N = 0.003477$	200	27.7
3	$31.555/N = 0.123262$	$30.5/N = 0.129102$	$1.495/N = 0.00584$	200	27.8
4	$31.6/N = 0.123438$	$33.4/N = 0.130419$	$1.8/N = 0.007031$	200	27.9

Exploring Periodogram Performance: Modulated Communications Signals

This note demonstrates the performance of the unmodified periodogram when it is used to estimate the spectrum of a *continuous-phase frequency shift keyed* (CPFSK) signal. This signal has nulls in its spectrum, but these nulls are not evident in the periodogram.

The utility of a spectrum estimation technique should not be judged solely on the technique's ability to resolve closely spaced sinsoidal components. This note applies the unmodified periodogram technique to a binary CPFSK signal. The random nature of the data signal used to modulate the CPFSK signal helps to highlight how the unmodified periodogram's high variance has a weakness in dealing with signals that are inherently random, even when there is no noise added to the signal.

The properties of CPFSK signals are summarized in Math Box 28.1. A `PracSim` simulation [1] was used to generate a 256-sample segment of a binary CPFSK signal with a symbol duration of $T_{\text{symb}} = 8$, a sampling interval of $T_{\text{samp}} = 1$, and a peak frequency deviation of $f_d = 0.04375$.

- The segment of MATLAB code in Computer Listing 28.1 computes the unmodified periodogram for the CPFSK signal that was generated by `PracSim` and written to the file `cpfsk_sig.txt`. The MATLAB code also plots the theoretical power spectral density (PSD) from Eq. (MB 28.1) that has been pre-computed and stored in the file `samspcdb.txt`. The resulting plot is shown in Figure 28.1.

Math Box 28.1

Power Spectral Density for Continuous Phase Frequency Shift Keying (CPFSK)

For a peak frequency deviation of f_d and a modulating signal consisting of rectangular pulses of width T, a CPFSK signal has a power spectral density given by [1] as

$$S(f) = \frac{T}{4M}\left[\sum_{n=1}^{M} A_n^2(f) + \frac{2}{M}\sum_{n=1}^{N}\sum_{m=1}^{M} A_n(f)A_m(f)B_{n+m}(f)\right] \quad \text{(MB 28.1)}$$

where M is the number of different symbols in the signaling alphabet, and

$$A_k(f) = \frac{\sin\{\pi T[f - f_d(2k-1-M)]\}}{\pi T[f - f_d(2k-1-M)]}$$

$$B_k(f) = \frac{\cos(2\pi fT - \alpha_k) - \beta\cos\alpha_k}{1 + \beta^2 - 2\beta\cos(2\pi fT)}$$

$$\alpha_k = 2\pi f_d T(k-1-M)$$

$$\beta = \frac{2}{M}\sum_{n=1}^{M/2}\cos[2\pi f_d T(2n-1)]$$

- The noise-like variations in the periodogram plotted in Figure 28.1 are a consequence of using a single sample function to estimate the statistics across the ensemble of all possible data sequences that could drive the CPFSK modulator. These variations can be reduced by using a modified periodogram technique, such as the Bartlett periodogram discussed in Note 30, that averages over multiple sample functions.

Computer Listing 28.1

Unmodified Periodogram for CPFSK Signal

```
M = csvread('cpfsk_sig.txt');
samp_idxs = M(:,1);
sig_vals = M(:,2);

R = csvread('samspcdb.txt');
pracsim_freq = R(:,1);
pracsim_pdgm = R(:,2);
cpfsk_ideal_psd_db = R(:,3);

N = 256;
noise_seed = 113057;
snr = 200;
y = awgn(sig_vals,snr,'measured',noise_seed);
w = hamming(N);
U = ((norm(w))^2)/N;
%
% Computing the Periodogram
%
fft_result = fft(y.*w);
S_U_2 = abs(fft_result.*conj(fft_result));
S_U = S_U_2(1:1+(N/2));
S_U = S_U./(N*U);
%
% Plotting result
%
abscissa = 0:1:N/2;
abscissa = abscissa/N;
plot(abscissa,10*log10(S_U),'ko',...
     pracsim_freq,cpfsk_ideal_psd_db,'r');
```

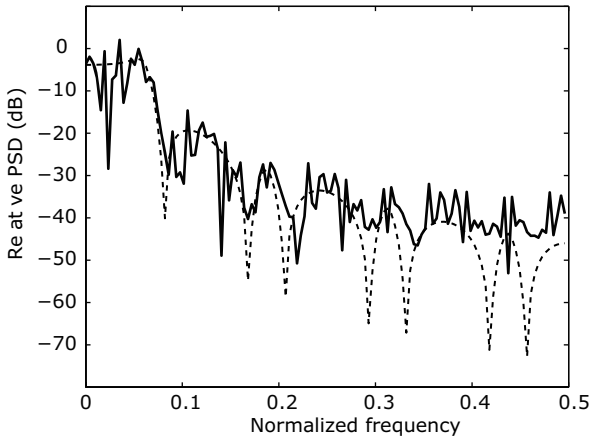


Figure 28.1 Periodogram result for CPFSK signal. The dashed trace is the theoretical PSD, and the solid trace is the corresponding unmodified periodogram.

Reference

1. C. B. Rorabaugh, *Simulating Wireless Communication Systems: Practical Models in C++*, Prentice Hall, 2004.

Modified Periodogram

The *modified periodogram* is an otherwise unmodified periodogram in which a window function is applied to the signal before the DFT is performed:

$$S_M[m] = \frac{T}{NU}\left|Y[m]\right|^2 \qquad (29.1)$$

where $Y[m]$ is the DFT of $y[n] = w[n]x_N[n]$, with $w[n]$ being a window function and $x_N[n]$ being an N-point sample extracted from $x[n]$ according to

$$x_N[n] = \begin{cases} x[n] & 0 \le n < N \\ 0 & \text{otherwise} \end{cases} \qquad (29.2)$$

The window function $w[n]$ can be any one of the windows listed in Table 24.1. The factor U that appears in Eq. (29.1) is obtained as

$$U = \frac{1}{N}\sum_{n=0}^{N-1}\left|w[n]\right|^2 \qquad (29.3)$$

This factor is needed in order for $S_M[m]$ to be an asymptotically unbiased estimate of $P_x(e^{j\omega})$.

Essential Facts

- The modified periodogram is only slightly more costly to implement than the unmodified periodogram discussed in Note 26. The additional cost is associated with applying a window function to the data prior to performing the DFT.
- The response to a distinct sinusoidal component has a wider main lobe response than for the case of the unmodified technique.
- Side lobe leakage is reduced relative to the unmodified periodogram. The amount of reduction depends upon the particular window function that is used.
- The absence of any averaging still leaves two major disadvantages:
  - No mitigation of additive noise that may be present in the signal
  - High variability in the result when the input signal is inherently random (such as a communications signal that has been modulated by a random data sequence)

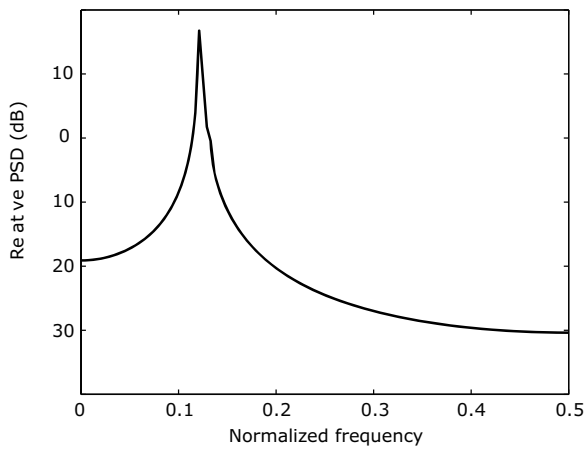


Figure 29.1 Periodogram result for Example 29.1 using a rectangular window

Example 29.1

Side Lobe Reduction in the Modified Periodogram

The MATLAB code in Computer Listing 29.1 generates a signal consisting of two cosine waves—one strong and one weak—and then computes the modified periodogram for this signal.

This code was run for several different window types: (1) rectangular window (equivalent to the unmodified periodogram), (2) Hann window, and (3) Hamming window. The amplitude of the stronger cosine was ten times larger than the amplitude of the weaker cosine. The results are shown in Figures 29.1 through 29.3. Of the three cases, the Hamming window clearly offers the best resolution of the weaker component.

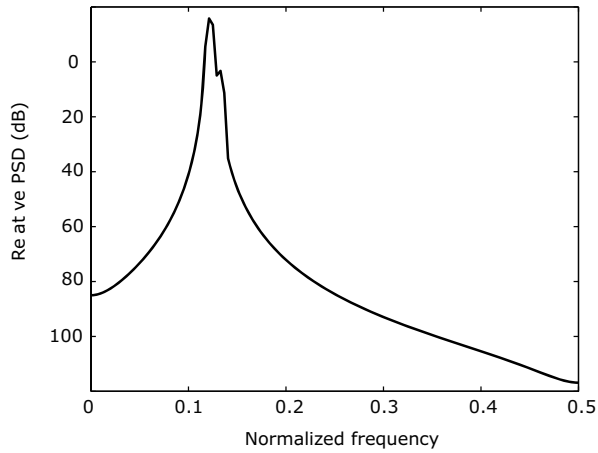


Figure 29.2 Periodogram result for Example 29.1 using a Hann window

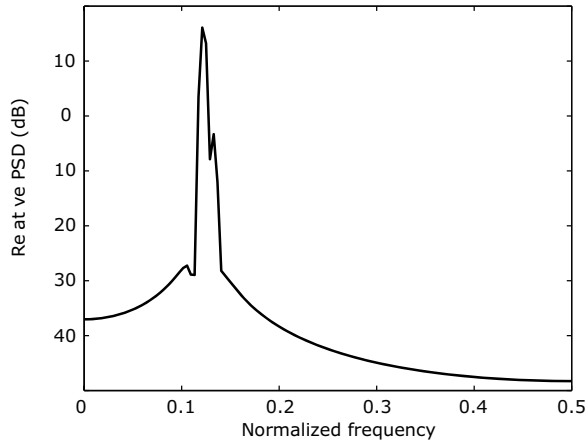


Figure 29.3 Periodogram result for Example 29.1 using the Hamming window

Example 29.2

Spectral Nulls in the Modified Periodogram

A `PracSim` simulation [4] was used to generate a 256-sample segment of a binary CPFSK signal with $T_{symb} = 8$, $T_{samp} = 1$, and $f_d = 0.04375$. The modified periodogram for this signal was then computed for the cases of: (1) rectangular window (equivalent to unmodified periodogram), (2) Hann window, and (3) Hamming window. The results are shown in Figures 29.4, 29.5, and 29.6, respectively. The use of a window improves the periodogram's fidelity in the vicinity of the spectral nulls, and it also appears to reduce the periodogram bias error at higher frequencies. However, the noise-like variations in the periodogram are relatively unaffected by windowing. The properties of CPFSK signals are summarized in Math Box 28.1 of Note 28.

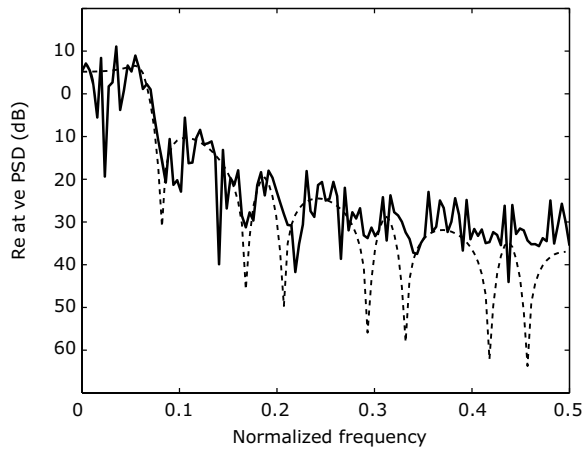


Figure 29.4 Periodogram results for Example 29.2. The dotted trace is the theoretical PSD, and the solid trace is the corresponding unmodified periodogram.

Computer Listing 29.1

Modified Periodogram for Sinusoids of Disparate Powers

The following segment of MATLAB code computes the modified periodogram for a 256-sample segment of two unequal-amplitude cosine waves.

```
%
% Specifics for this example
%
N=256;
freq1 = 31.3/N;
freq2 = 33.9/N;
a = 0.10;
ti = 0:N-1;
x1 = cos(ti*2*pi*freq1);
x2 = cos(ti*2*pi*freq2);
y = x1 + a*x2;
w = hamming(N);
U = ((norm(w))^2)/N;
%
% Computing the periodogram
%
fft_result = fft(y.*w');
S_U_2 = abs(fft_result.*...
    conj(fft_result));
S_U = S_U_2(1:1+(N/2));
S_U = S_U./(N*U);
%
% Plotting result
%
abscissa = 0:1:N/2;
abscissa = abscissa/N;
plot(abscissa, 10*log10(S_U));
```

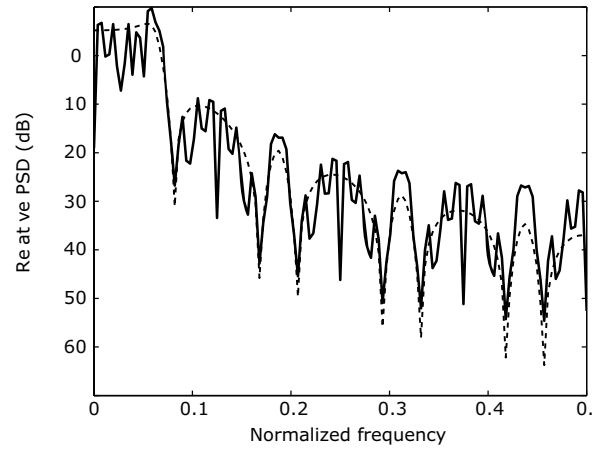


Figure 29.5 Periodogram results for Example 29.2. The dotted trace is the theoretical PSD, and the solid trace is the corresponding modified periodogram using a Hann window.

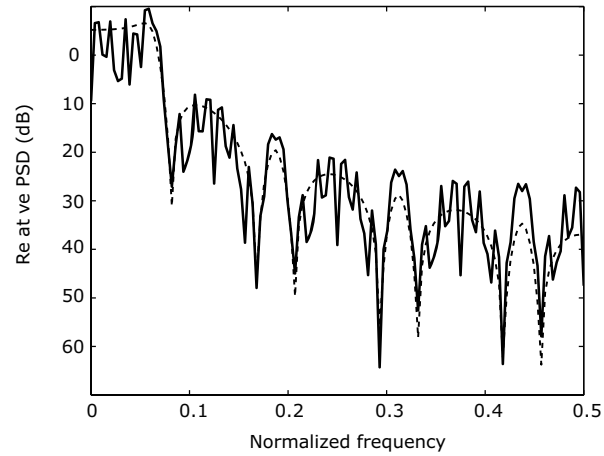


Figure 29.6 Periodogram results for Example 29.2. The dotted trace is the theoretical PSD, and the solid trace is the corresponding modified periodogram using a Hamming window.

References

1. M. H. Hayes, *Statistical Digital Signal Processing and Modeling*, Wiley, 1996.
2. M. S. Bartlett, "Periodogram Analysis and Continuous Spectra," *Biometrika 37*, 1950, pp. 1–16.
3. C. B. Rorabaugh, *DSP Primer*, McGraw-Hill, 1999.
4. C. B. Rorabaugh, *Simulating Wireless Communication Systems: Practical Models in C++*, Prentice Hall, 2004.

Bartlett's Periodogram

The Bartlett periodogram is based on the notion of creating a pseudo-ensemble of sample sequences by dividing a long sequence of samples into P non-overlapping segments of D samples each. The individual sample spectra for these segments are then averaged to form the periodogram. There is an implicit assumption that the random process under consideration is ergodic and that therefore this pseudoensemble is an adequate substitute for the unobservable "true" ensemble. The steps for computing a Bartlett periodogram are listed in Recipe 30.1.

Recipe 30.1

Computing the Bartlett Periodogram

1. Divide the available sample sequence into P non-overlapping segments of D samples each. If the original sequence is $x[k]$, the pth segment can be expressed as

$$x_p[n] = x[pD+n]$$

2. Compute the discrete-frequency sample spectrum for each of the P segments

$$S_p[m] = \frac{T}{D}\left|\sum_{n=0}^{D-1} x_p[n]\exp(-j2\pi mn/D)\right|^2$$
$$= \frac{T}{D}\left|X_p[m]\right|^2 \qquad m = 0,1,\ldots,D-1$$

where $X_p[m]$ is the DFT of the segment $x_p[n]$.

3. Compute the arithmetic average of the P different sample spectra at each frequency:

$$S_B[m] = \frac{1}{P}\sum_{p=0}^{P-1} S_p[m] \qquad m = 0,1,\ldots,D-1$$

The result, $S_B[m]$, is the Bartlett periodogram.

Example 30.1

Noise Reduction in the Bartlett Periodogram

The MATLAB code in Computer Listing 30.1 generates a pair of cosine waves in AWGN and then computes the Bartlett periodogram for different combinations of SNR and P, as listed in Table 30.1.

Case 1, with $P = 1$, is equivalent to the unmodified periodogram, and the signal is noise-free. Two distinct peaks corresponding to the two sinusoids are visible in the plotted results, shown in Figure 30.1. In Case 2, the SNR is reduced to -10 dB, and the presence of the sinusoidal signal components cannot be discerned in the periodogram results (Figure 30.2). Case 3, with $P = 8$, demonstrates the advantage provided by the Bartlett periodogram. A peak is clearly visible in the plotted result (Figure 30.3). However, it is not possible to resolve two distinct peaks corresponding to the two distinct sinusoidal frequencies.

Table 30.1 Cases for Example 30.1

Case	`freq1`	`freq2`	`snr`	D	P	Figure
1	31.555/D	32.445/D	200	256	1	30.1
2	31.555/D	32.445/D	-10	256	1	30.2
3	31.555/D	32.445/D	-10	256	8	30.3

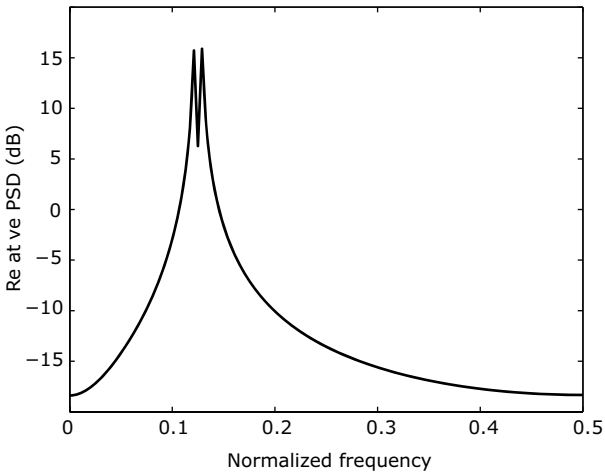


Figure 30.1 Periodogram result for Example 30.1, Case 1

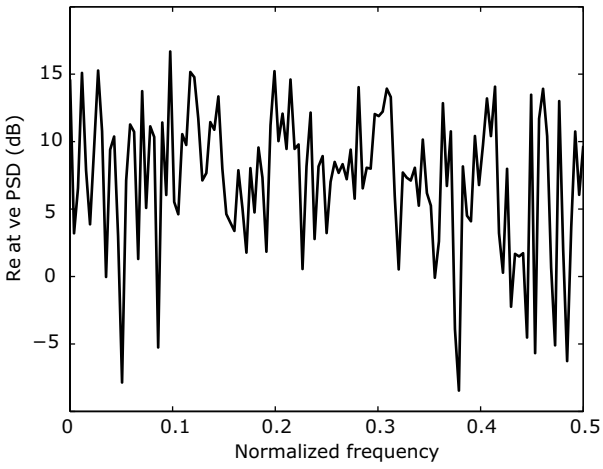


Figure 30.2 Periodogram result for Example 30.1, Case 2

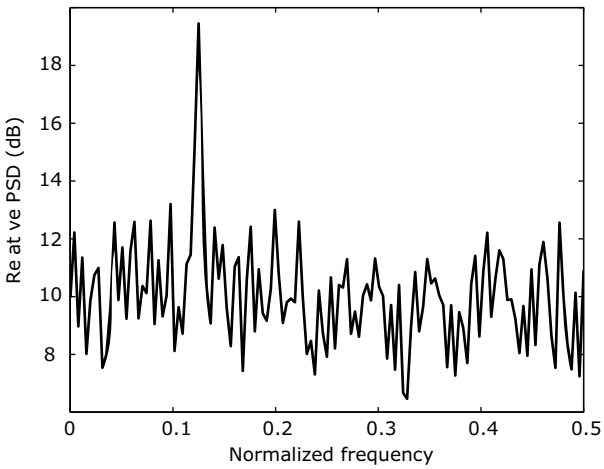


Figure 30.3 Periodogram result for Example 30.1, Case 3

Computer Listing 30.1

Bartlett Periodogram

The following segment of MATLAB code computes the Bartlett periodogram for a signal consisting of two equal-amplitude cosine waves in AWGN.

```
%
% Specifics for current experiment
%
D=256;
P=8;
N=D*P;
noise_seed = 113057;
snr = -10;
freq1=31.555/D;
freq2=32.445/D;
phi = 0*pi;
ti=0:N-1;
x1=cos(ti*2*pi*freq1);
x2=cos(ti*2*pi*freq2+phi);
x = x1 + x2;
y=awgn(x,snr,'measured',noise_seed);
%
% Computing the Bartlett Periodogram
%
S_B = 0;
pD = 1;
for p=1:P
    fft_result = fft(y(pD:pD+D-1));
    S_P_2 = abs(fft_result.*...
          conj(fft_result));
    S_P = S_P_2(1:1+(D/2));
    S_B = S_B + S_P;
    pD = pD+D;
end;
S_B = S_B./(D*P);
%
% Plotting result
%
abscissa = 0:1:D/2;
abscissa = abscissa/D;
plot(abscissa,10*log10(S_B));
```

Example 30.2

Variance Reduction in the Bartlett Periodogram

A `PracSim` simulation[3] was used to generate a 2048-sample segment of binary CPFSK signal with $T_{symb} = 8$, $T_{samp} = 1$, and $f_d = 0.04375$. The properties of CPFSK signals are summarized in Math Box 28.1 of Note 28. The Bartlett periodogram was computed for the cases of $P = 1$ (equivalent to unmodified periodogram) and $P = 8$. The results are shown in Figures 30.4 and 30.5 respectively. The averaging provided by the Bartlett technique reduces the noise-like variations in the periodogram at low frequencies, but the periodogram's bias error at higher frequencies is relatively unaffected by the averaging.

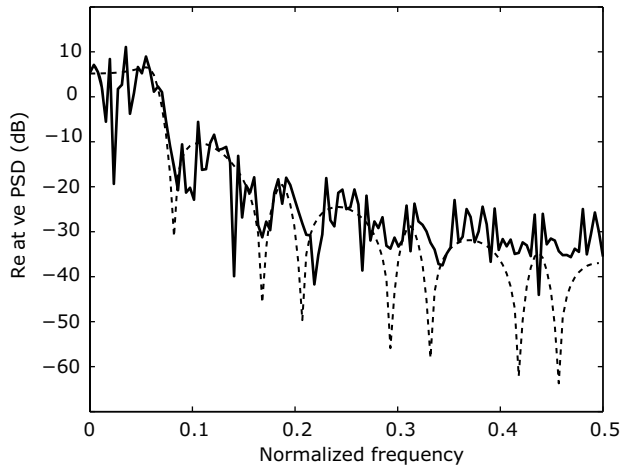


Figure 30.4 Periodogram result for Example 30.2, with $P = 1$

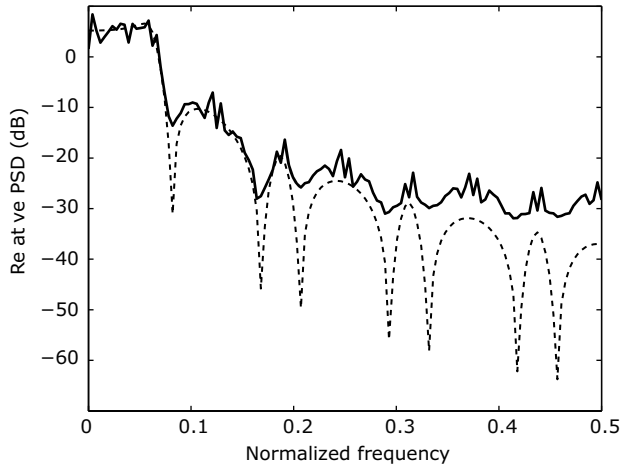


Figure 30.5 Periodogram result for Example 30.2, with $P = 8$

References

1. M. S. Bartlett, "Smoothing Periodograms from Time Series with Continuous Spectra," *Nature*, vol. 161, pp. 686–687.
2. M. S. Bartlett, "Periodogram Analysis and Continuous Spectra," *Biometrika* 37, 1950, pp. 1–16.
3. C. B. Rorabaugh, *Simulating Wireless Communication Systems: Practical Models in C++*, Prentice Hall, 2004.

Welch's Periodogram

The Welch periodogram can be viewed as a generalization of the Bartlett periodogram. As with the Bartlett, the Welch periodogram is based on the notion of creating a pseudoensemble of sample sequences by dividing a long sequence of samples into a set of shorter segments. However, these shorter segments can be allowed to overlap with their neighbors for some portion of their length. Furthermore, a data window, $w[n]$, is applied to each segment before the segment's sample spectrum is computed. The specific steps for computing a Welch periodogram are listed in Recipe 31.1.

Example 31.1

Side Lobe Reduction in the Welch Periodogram

The MATLAB code in Computer Listing 31.1 generates a signal consisting of two cosine waves, x_1 and x_2, in AWGN, and then computes the Welch periodogram for this signal. The amplitude of x_1 is four times larger than the amplitude of x_2. This code was run for the combinations of P, overlap, and window type listed in Table 31.1, with the results plotted in the figures indicated by the table entries.

Case 1, with a rectangular window and $P = 1$, is equivalent to an unmodified periodogram. Case 2, with $P = 128$, is equivalent to a Bartlett periodogram. The low side lobe levels of the Hann window used in Case 3 and of the Hamming window used in Cases 4 and 5 make it possible to distinguish the spectral peak corresponding to the weaker signal, x_2.

Recipe 31.1

Computing the Welch Periodogram

1. Divide the available sample sequence into P (possibly) overlapping segments of D samples each, with a shift of $S < D$ samples between consecutive segments. If the original sequence is $x[k]$, the pth segment can be expressed as

$$x_p[n] = x[pS + n]$$

2. Apply a data window, $w[n]$, to each segment:

$$y_p[n] = w[n]x_p[n] \qquad p = 0,1,\ldots,P-1$$

3. Compute the discrete-frequency sample spectrum for each of the P windowed segments:

$$S_p[m] = \frac{T}{UD}\left|Y_p[m]\right|^2 \qquad m = 0,1,\ldots,D-1$$

where

$$U = \sum_{n=0}^{D-1}\left|w[n]\right|^2$$

4. Compute the arithmetic average of the P different sample spectra at each frequency:

$$S_W[m] = \frac{1}{P}\sum_{p=0}^{P-1} S_p[m] \qquad m = 0,1,\ldots,D-1$$

The result, $S_W[m]$, is the Welch periodogram.

Table 31.1 Cases for Example 31.1

Case	P	Overlap (Samples)	Window	Figure
1	1	—	Rectangular	31.1
2	128	111	Rectangular	31.2
3	128	111	Hann	31.3
4	128	111	Hamming	31.4
5	128	0	Hamming	31.5

Computer Listing 31.1

Welch Periodogram for Two Unequal Sinusoids

The following segment of MATLAB code computes the Welch periodogram for a signal consisting of two unequal-amplitude cosine waves.

```
%
% D = 256;
P = 128;
overlap = 0;
a = 0.25;
S = D - overlap;
N = D*P;
noise_seed = 113057;
snr = -10;
freq1 = 31.55/D;
freq2 = 34.45/D;
phi = 0*pi;
ti = 0:N-1;
x1 = cos(ti*2*pi*freq1);
x2 = cos(ti*2*pi*freq2+phi);
x = x1 + a*x2;
y = awgn(x,snr,'measured',noise_seed);
%
% Computing the Welch Periodogram
%
S_W = 0;
pD = 1;
window=hamming(D);
U = ((norm(window))^2)/D;
for p=1:P
    fft_result = fft((window').*y(pD:pD+D-1));
    S_P_2 = abs(fft_result.*conj(fft_result));
    S_P = S_P_2(1:1+(D/2));
    S_W = S_W + S_P;
    pD = pD+S;
end;
S_W = S_W./(D*P*U);
%
% Plotting result
%
abscissa = 0:1:D/2;
abscissa = abscissa/D;
plot(abscissa,10*log10(S_W));
```

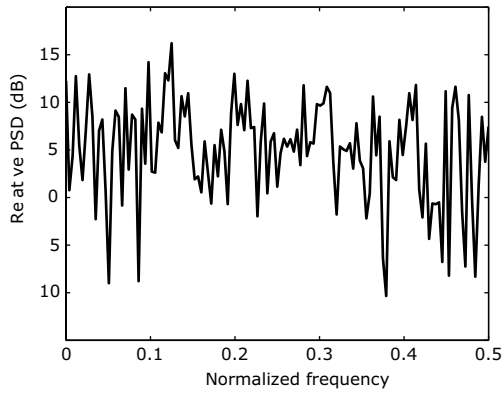


Figure 31.1 Periodogram for Example 31.1, Case 1

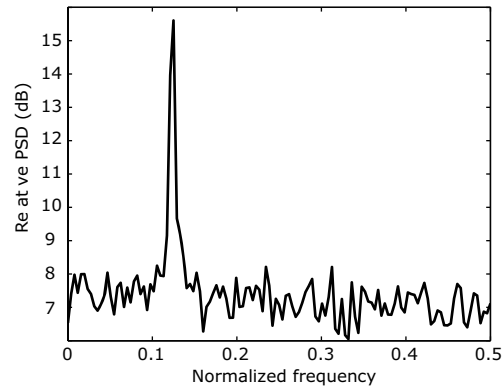


Figure 31.2 Periodogram for Example 31.1, Case 2

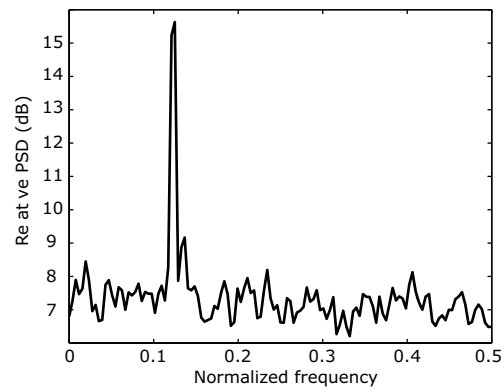


Figure 31.3 Periodogram for Example 31.1, Case 3

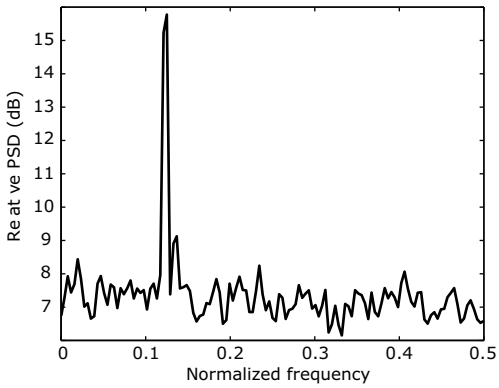


Figure 31.4 Periodogram for Example 31.1, Case 4

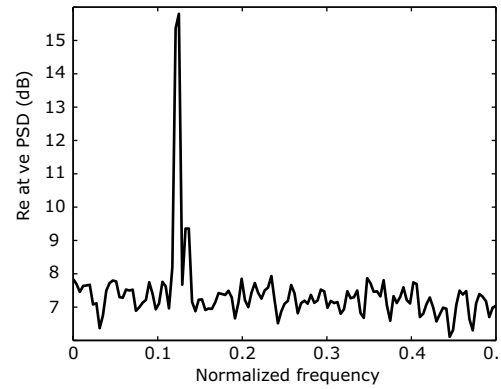


Figure 31.5 Periodogram for Example 31.1, Case 5

Example 31.2

Spectral Nulls in the Welch Periodogram

A PracSim simulation [1] was used to generate a 2048-sample segment of binary CPFSK signal with $T_{symb} = 8$, $T_{samp} = 1$, and $f_d = 0.04375$. The segment of MATLAB code in Computer Listing 31.2 computes the Welch periodogram of this signal. The code was run for the different combinations of P and the window types listed in Table 31.2, with the results plotted in the figures indicated by the table entries. In each case, there is no overlap between consecutive applications of the window. Each plot of results also includes the theoretical PSD for CPFSK that was discussed in Note 28.

Table 31.2 Cases for Example 31.2

Case	P	Window	Figure
1	1	Rectangular	31.6
2	1	Hamming	31.7
3	32	Rectangular	31.8
4	32	Hamming	31.9

Computer Listing 31.2

Welch Periodogram for CPFSK Signal

```
%
M = csvread('cpfsk_sig_big.txt');
samp_idxs = M(:,1);
sig_vals = M(:,2);

R = csvread('samspcdb.txt');
pracsim_freq = R(:,1);
pracsim_pdgm = R(:,2);
cpfsk_ideal_psd_db = R(:,3);

D=256;
P=1;
overlap = 0;
S = D - overlap;
noise_seed = 113057;
snr = 200;
y=awgn(sig_vals,snr,'measured',noise_seed);
w=hamming(D);
U = ((norm(w))^2)/D;
%
% Computing the Periodogram
%
S_W = 0;
pD = 1;
for p=1:P
    fft_result = fft(w.*sig_vals(pD:pD+D-1));
    S_P_2 = abs(fft_result.*conj(fft_result));
    S_P = S_P_2(1:1+(D/2));
    S_W = S_W + S_P;
    pD = pD+D;
end;
S_W = S_W./(D*P*U);
%
% Plotting result
%
abscissa = 0:1:D/2;
abscissa = abscissa/D;
plot(abscissa,10*log10(S_W),pracsim_freq,...
     cpfsk_ideal_psd_db,'r');
```

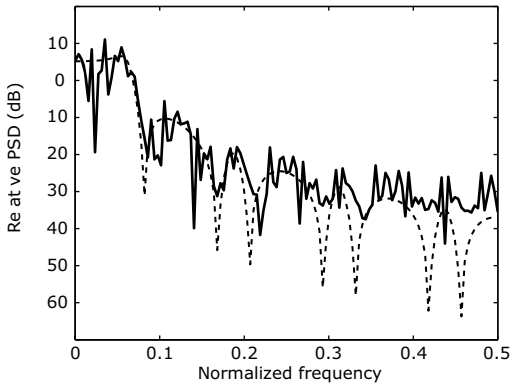


Figure 31.6 Periodogram for Example 31.2, Case 1

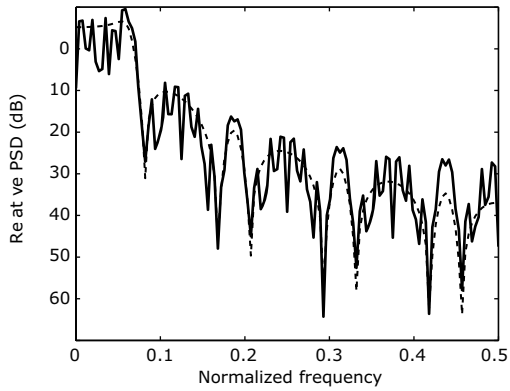


Figure 31.7 Periodogram for Example 31.2, Case 2

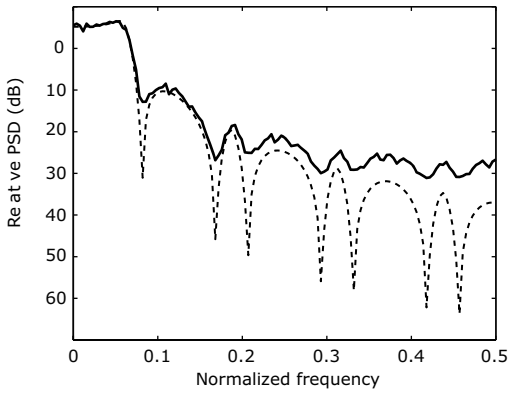


Figure 31.8 Periodogram for Example 31.2, Case 3

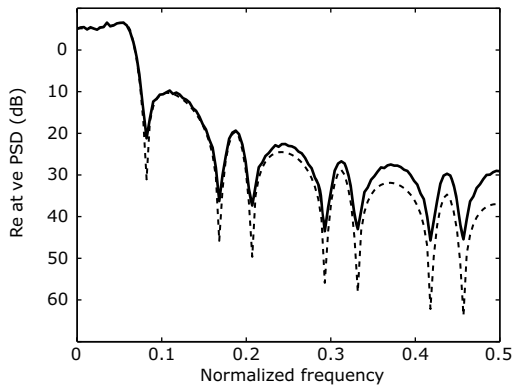


Figure 31.9 Periodogram for Example 31.2, Case 4

References

1. C. B. Rorabaugh, *Simulationg Wireless Communication Systems: Practical Models in C++*, Prentice Hall, 2004.

Designing FIR Filters: Background and Options

The block diagram in Figure 32.1 represents the canonical form for an *M*th order *finite-impulse response* (FIR) digital filter in that the diagram is a direct implementation of the filter's defining difference equation:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] \qquad (32.1)$$

Examination of the equation reveals that the filter's output at time n is simply a weighted sum of the inputs at times $n-M$ through n. The notation used in Eq. (32.1) is consistent with the notation used in the difference equation for an IIR filter, as presented in Note 49. Comparison of Eqs. (32.1) and (49.1) reveals that FIR filters can be viewed as a special case of IIR filters.

Equation (32.1) has the form of an $(M+1)$-point discrete convolution, with the coefficients b_k taking the place of the unit-sample-response sequence, $h[k]$. Therefore, for an FIR filter, the unit sample response, $h[k]$, is defined by the coefficients b_k , and Eq. (32.1) can be immediately rewritten as the convolution equation:

$$y[n] = \sum_{k=0}^{M} h[k] x[n-k]$$

The filter's system function, $H(z)$, is obtained by taking the z transform of $h[k]$:

$$H(z) = \sum_{k=0}^{M} h[k] z^{-k} \qquad (32.2)$$

This system function has zeros but no finite poles, so in some contexts, FIR filters are referred to as *all-zero filters*. (See Note 44 for a discussion of the z transform.)

Essential Facts

- It is possible to design FIR filters that have exactly linear-phase response and which therefore do not introduce delay distortion into signals passing through the filter.
- In order to achieve narrow transition bands, FIR filters often require many more coefficients than an IIR design of comparable performance.
- Linear-phase FIR designs exhibit coefficient symmetry that can be exploited to reduce the computational burden for implementing the filter.
- FIR filters are also referred to as *all-zero* filters or *moving-average* filters.

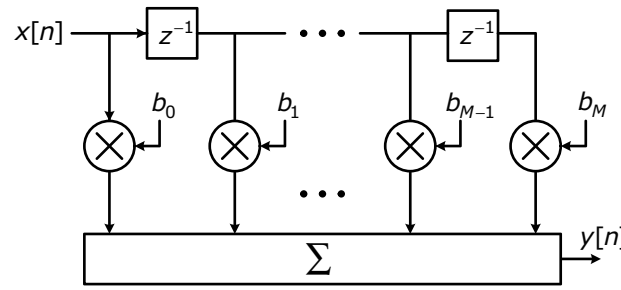


Figure 32.1 Block diagram for FIR filter

32.1 Implementation Structures

The block diagram in Figure 32.1 can be converted into the corresponding signal flow graph shown in Figure 32.2(a). The transposition theorem can be applied to this direct form structure to obtain the *transposed direct form* structure in Figure 32.2(b).

The system function from Eq. (32.2) can be expressed as a product of first- and second-order polynomials in the form

$$H(z)=\prod_{k=1}^{M_1}\left(\beta_{0,k}+\beta_{1,k}\,z^{-1}\right)\times\prod_{k=1}^{M_2}\left(b_{0,k}+b_{1,k}\,z^{-1}+b_{2,k}\,z^{-2}\right) \tag{32.3}$$

where

$$M=M_1+2M_2$$

and

- Each real root of $H(z)$ is the root of one of the first-order polynomials.
- Each second-order polynomial has as its roots a complex-conjugate pair of complex roots from $H(z)$.

Typical FIR filters usually have a single real root, and for real-input-real-output FIR filters, the complex roots of $H(z)$ always occur in conjugate pairs. Under these constraints, Eq. (32.3) can be rewritten for even M as

$$H(z)=\prod_{k=1}^{M/2}\left(b_{0,k}+b_{1,k}\,z^{-1}+b_{2,k}\,z^{-2}\right) \tag{32.4}$$

and for odd M as

$$H(z)=\left(b_{0,0}+b_{1,0}\,z^{-1}\right)\times\prod_{k=1}^{(M-1)/2}\left(b_{0,k}+b_{1,k}\,z^{-1}+b_{2,k}\,z^{-2}\right) \tag{32.5}$$

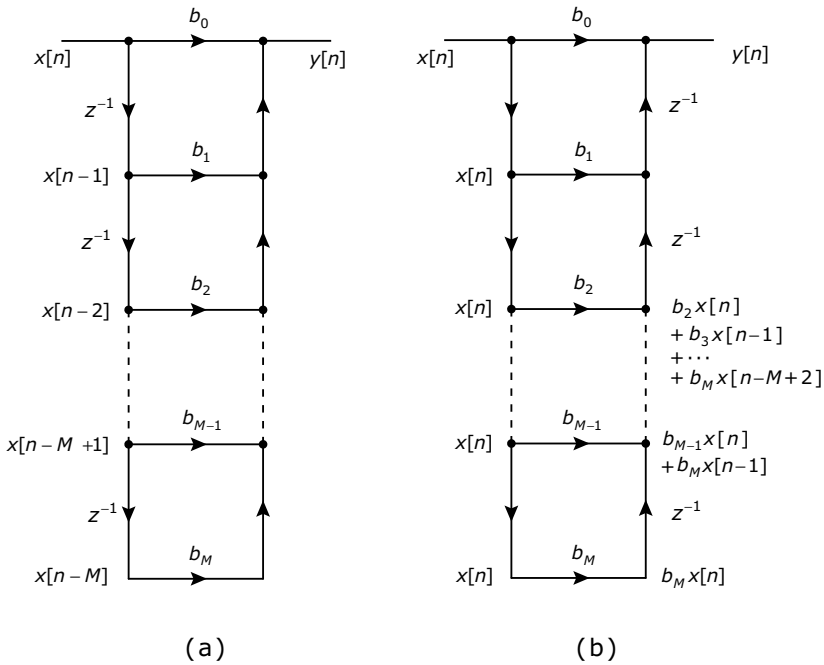


Figure 32.2 Structures for an FIR filter: (a) direct form, (b) transposed direct form

FIR filters can be implemented in a form that mimics the structure of Eq. (32.4) or (32.5). Each second-order polynomial has a complex-conjugate pair of roots and can be implemented as a second-order filter section having real-valued coefficients. The second-order section, shown in Figure 32.3 implements the polynomial

$$b_{0,k} + b_{1,k}\, z^{-1} + b_{2,k}\, z^{-2}$$

An even-order filter can be implemented as a cascade of these second-order sections, as shown in Figure 32.4. An odd-order filter can be implemented as a cascade of second-order sections plus one first-order section, as shown in Figure 32.5.

The linear-phase FIR filters discussed in Note 33 all exhibit symmetries in their unit-sample responses. These symmetries, which are listed in Table 32.1, can be exploited to design the efficient implementation structures shown in Figures 32.6 through 32.9.

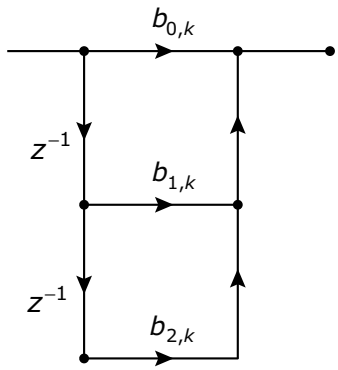


Figure 32.3 A second-order filter section

Table 32.1 Linear-phase FIR Filter Types

Type	N	$h[n]$ symmetry
1	odd	even
2	even	even
3	odd	odd
4	even	odd

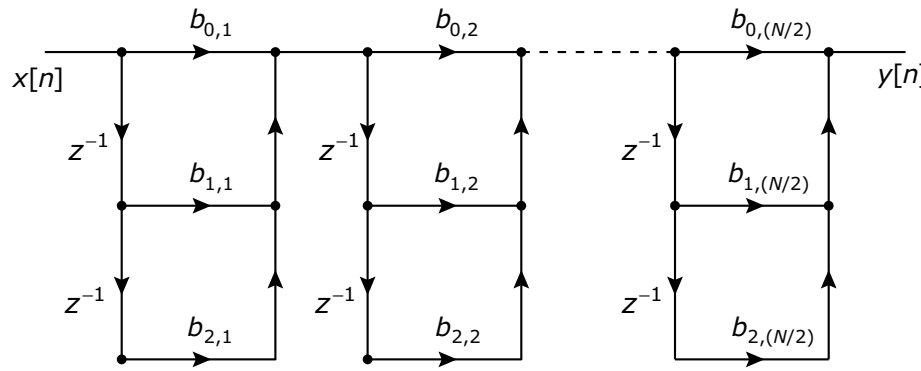


Figure 32.4 Cascade structure for an even-length FIR filter

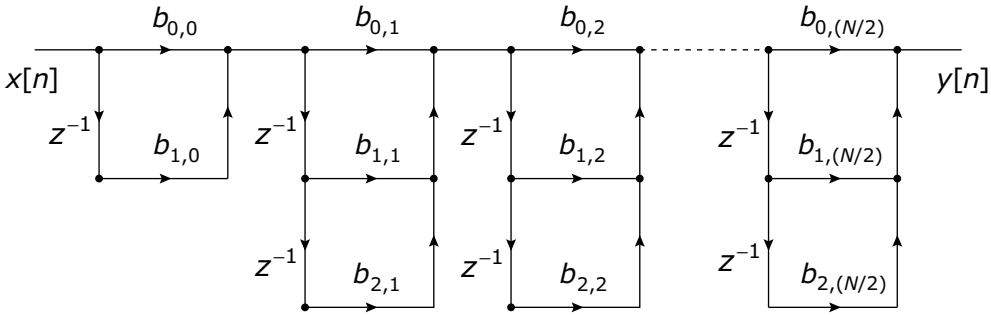


Figure 32.5 Cascade structure for an odd-length FIR filter

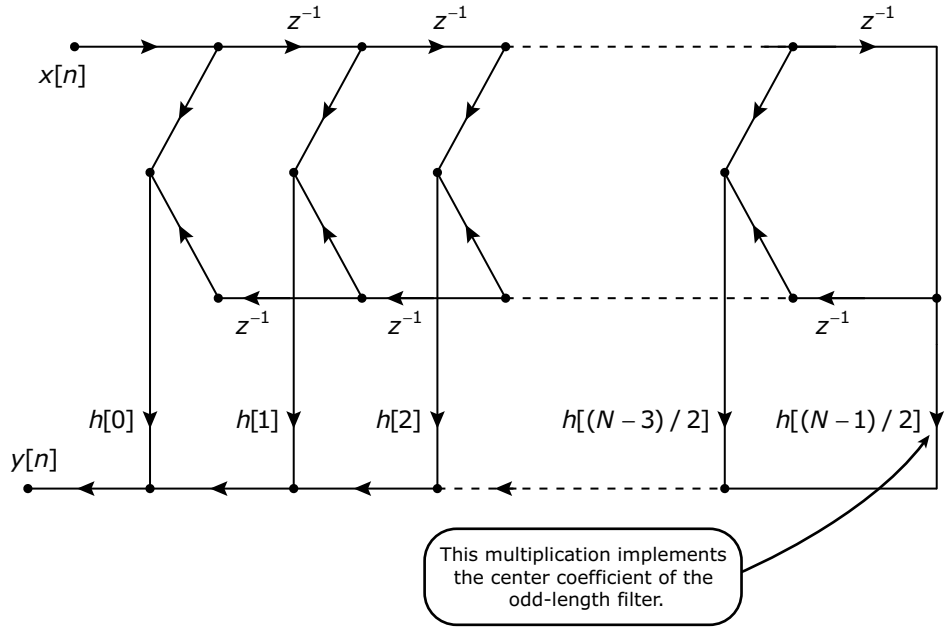


Figure 32.6 Direct-waveform structure for type 1 (odd length, even symmetry) linear-phase FIR filter

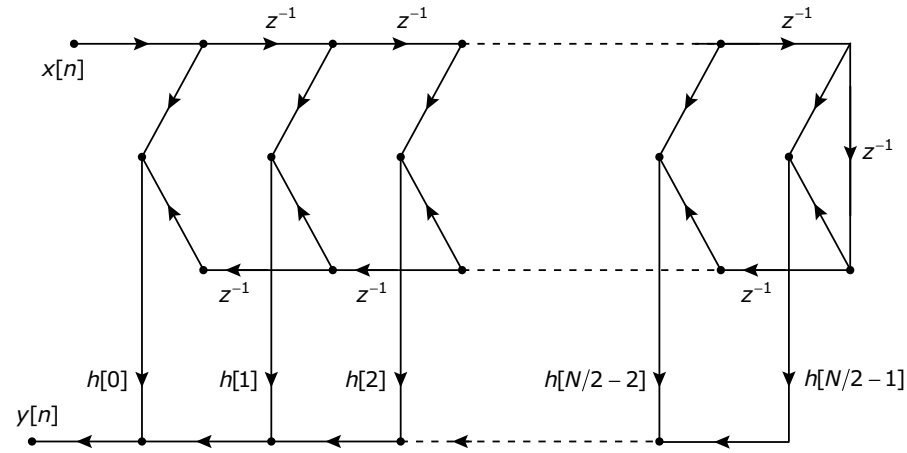


Figure 32.7 Direct-form structure for type 2 (even length, even symmetry) linear-phase FIR filter

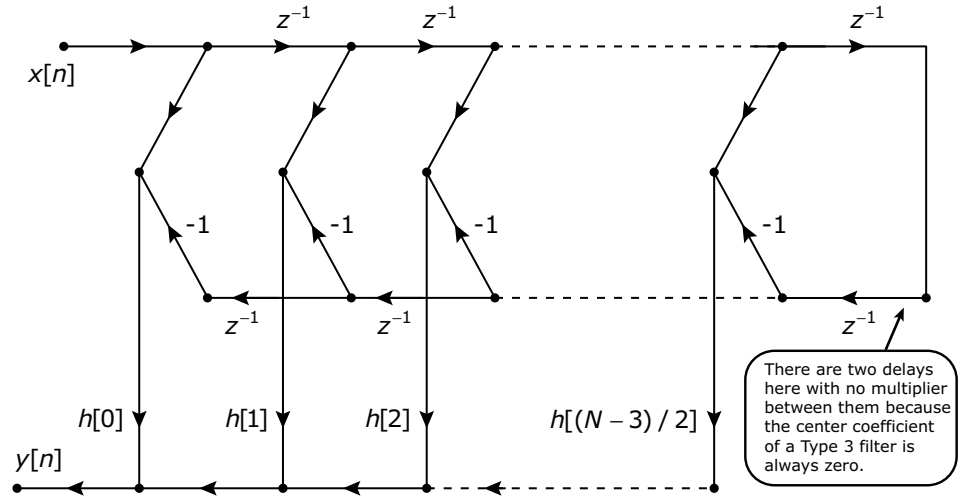


Figure 32.8 Direct-form structure for type 3 (odd length, odd symmetry) linear-phase FIR filter

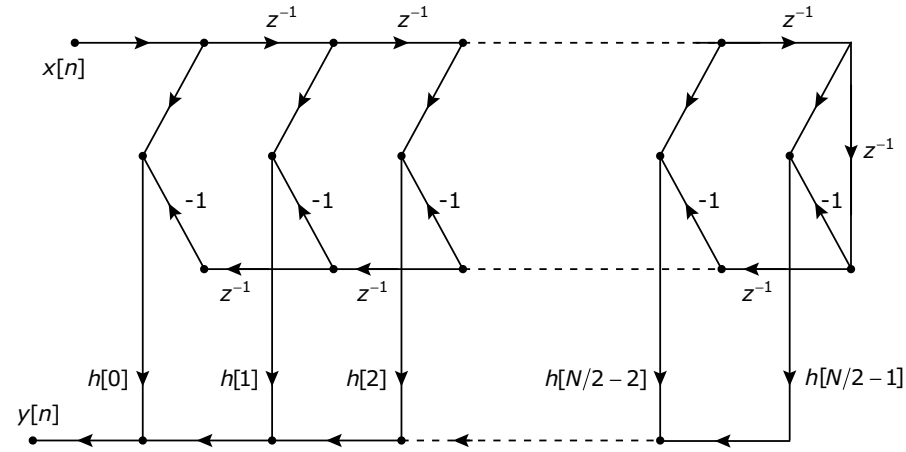


Figure 32.9 Direct-form structure for type 4 (even length, odd symmetry) linear-phase FIR filter

Linear-Phase FIR Filters

One of the advantages that FIR filters have over IIR filters is that it is relatively easy to design FIR filters that have constant group delay across all frequencies. Constant group delay is a desireable property for a filter to have because it means that a signal passing through the filter does not experience any delay distortion due to different frequency components being delayed by different amounts. Linear-phase FIR filters are conventionally separated into four types depending upon the four combinations of odd-even filter length and odd-even symmetry in the filter's impulse response. This note examines these four types of linear-phase FIR filters and their properties.

When a filter has constant *group delay* (see Note 39, section 39.2), the filter's phase response is a straight line in the phase-versus-frequency plane—hence, such filters are usually called *linear-phase* filters. However, in mathematical terms, the phase is a *linear function of frequency* only if the straight-line phase response passes through the origin of phase-versus-frequency plane and can therefore be expressed as

$$\theta(\omega) = -\alpha\omega \tag{33.1}$$

where the constant α is equal to both the filter's group delay and phase delay. A straight-line phase response that does not pass through the origin can be expressed as

$$\theta(\omega) = \beta - \alpha\omega \tag{33.2}$$

Even though Eq. (33.2) defines a straight line, the phase response is not a linear function of frequency because the mathematical definition of linearity is not satisfied, specifically

$$\theta(\omega_1) + \theta(\omega_2) = 2\beta - \alpha(\omega_1 + \omega_2)$$
$$\neq \theta(\omega_1 + \omega_2)$$

Filters that satisfy Eq. (33.2) have constant group delay, but for non-zero β, they do not have constant phase delay. Such filters are properly called *constant group delay filters*, but historically they have been lumped together with filters that satisfy Eq. (33.1) and are informally referred to as linear-phase filters. True linear-phase filters that have a phase response of the form given by Eq. (33.1) are categorized as either *Type 1*, if the length of their impulse response is odd, or as *Type 2,* if the length of their impulse response is even. The properties of

Strategic Considerations

- A *linear-phase* FIR filter has constant group delay, and therefore, delay distortion is not introduced into signals passing through the filter.
- Most of the popular FIR design techniques yield constant-group-delay (CGD) designs.
- A signal is delayed by $(N-1)/2$ sample times in passing through a CGD filter of length N. To achieve narrow transition bands, N often needs to be rather large, leading to large signal delays. In applications where such delays are unacceptable, designs other than linear-phase FIRs should be considered. The best alternative in these cases is usually a *minimum-phase* FIR design.
- A CGD filter can be classified as one of four types based on whether the filter's impulse-response length is odd or even and whether the symmetry of this impulse response is odd or even. Type 1 has odd length and even symmetry. Type 2 has even length and even symmetry. Type 3 has odd length and odd symmetry. Type 4 has even length and odd symmetry.
- Because of the symmetry conditions that must be satisfied, Types 2 and 3 are not suitable for highpass or bandstop filters, while Types 3 and 4 are not suitable for lowpass or bandstop filters. Because of a constant 90-degree phase shift in their phase responses, Types 3 and 4 are useful for implementing FIR differentiators and Hilbert transformers.

Type 1 and Type 2 filters are summarized in List 33.1. Constant-group-delay filters that have a phase response of the form given by Eq. (33.2) are categorized as either *Type 3,* if the length of their impulse response is odd, or as *Type 4,* if the length of their impulse response is even. The properties of Type 3 and Type 4 filters are summarized in List 33.2.

Table 33.1 summarizes the frequency response properties for the four different types of linear-phase FIR filters. Each filter's frequency response, $H(\omega)$, is expressed as a product of a complex exponential phase term and a purely real-valued amplitude response, $A(\omega)$. The symmetries that the various $A(\omega)$ exhibit with respect to $\omega = 0$ and $\omega = \pi$ limit the applications for which each filter type can be used. Because Types 3 and 4 must have $A(0) = 0$, they are not suitable for use as lowpass or bandstop filters where a nonzero response is needed at $\omega = 0$. Similarly, because they must have $A(\pi) = 0$, Types 2 and 3 are not suitable for use as highpass or bandstop filters.

The period of 4π that Table 33.1 gives for the normalized-frequency amplitude response of filter types 2 and 4 is often the source of some consternation. The very definition of "normalized frequency" was set up so that the baseband image of a digital filter's response spans a normalized frequency interval of ±0.5 cyc/samp or $\pm\pi$ rad/samp. The filter's response is completely specified by its behavior in a frequency interval of 2π rad/sec, so how can it not be periodic in 2π radians? This issue is discussed at length in Note 34.

Table 33.1 Frequency Responses for Linear-Phase and Constant-Group-Delay FIR Filters

Type	$\boldsymbol{H(\omega)}$	$\boldsymbol{A(\omega)}$
1	$\exp\left(-j\omega T\dfrac{N-1}{2}\right)A(\omega)$	$\displaystyle\sum_{k=0}^{(N-1)/2} a_k \cos\omega kT$
2	$\exp\left(-j\omega T\dfrac{N-1}{2}\right)A(\omega)$	$\displaystyle\sum_{k=1}^{N/2} b_k \cos\left[\omega\left(k-\tfrac{1}{2}\right)T\right]$
3	$\exp\left(-j\left[\omega T\dfrac{N-1}{2}-\dfrac{\pi}{2}\right]\right)A(\omega)$	$\displaystyle\sum_{k=1}^{(N-1)/2} a_k \sin\omega kT$
4	$\exp\left(-j\left[\omega T\dfrac{N-1}{2}-\dfrac{\pi}{2}\right]\right)A(\omega)$	$\displaystyle\sum_{k=1}^{N/2} b_k \sin\left[\omega\left(k-\tfrac{1}{2}\right)T\right]$

where $a_0 = h\left[\dfrac{N-1}{2}\right]$ $\quad a_k = 2h\left[\dfrac{N-1}{2}-k\right]$ $\quad b_k = 2h\left[\dfrac{N}{2}-k\right]$ for $k \neq 0$

List 33.1

Properties of True Linear-Phase FIR Filters

- The phase response, $\theta(\omega)$, is a linear function of frequency:

$$\theta(\omega) = -\alpha\omega \qquad -\pi \le \omega \le \pi \tag{33.1}$$

  where the constant α is both the phase delay and the group delay of the filter in units of sample intervals.

- The phase response is a straight line, with a slope of α, passing through the origin of the phase-versus-frequency coordinate plane.

- For an FIR filter having N coefficients, there is a unique value of α for which a linear phase is produced. This value is given by

$$\alpha = \frac{N-1}{2} \tag{33.2}$$

- In order to achieve a linear-phase response, the impulse response of the filter must exhibit even symmetry:

$$h[n] = h[N-1-n] \qquad 0 \le n \le N-1 \tag{33.3}$$

- For a Type 1 filter, the number of taps, N, is odd, and the center of symmetry coincides with the sample at $n = (N-1)/2$, as shown in Figure 33.1(a).

- For a Type 2 filter, the number of taps, N, is even, and the center of symmetry falls midway between the sample at $n = (N-2)/2$ and the sample at $n = N/2$, as shown in Figure 33.1(b).

List 33.2

Properties of Constant-Group-Delay FIR Filters

- The phase response, $\theta(\omega)$, has constant slope but is not constrained to pass through the origin of the phase-versus-frequency coordinate plane:

$$\theta(\omega) = \beta - \alpha\omega \tag{33.1}$$

- For a filter having N coefficients, there are unique values of α and β for which this phase response can be produced, and these values are given by

$$\alpha = \frac{N-1}{2} \qquad \beta = \pm\frac{\pi}{2} \tag{33.2}$$

- In order to achieve the phase response of (33.1), the impulse response must be odd symmetric or antisymmetric, that is, $h[n]$ must satisfy

$$h[n] = -h[N-1-n] \qquad 0 \le n \le N-1 \tag{33.3}$$

- For a Type 3 filter, the number of taps, N, is odd and the center of antisymmetry coincides with the sample at $n = (N-1)/2$, as shown in Figure 33.2(a). When $n = (N-1)/2$ with N odd, Eq. (33.3) yields

$$h\left[\frac{N-1}{2}\right] = -h\left[N-1-\frac{N-1}{2}\right] = -h\left[\frac{N-1}{2}\right]$$

  Therefore, $h[(N-1)/2]$ must always equal zero in Type 3 filters.

- For a Type 4 filter, the number of taps, N, is even, and the center of antisymmetry falls midway between the sample at $n = (N-2)/2$ and the sample at $n = N/2$, as shown in Figure 33.2(b).

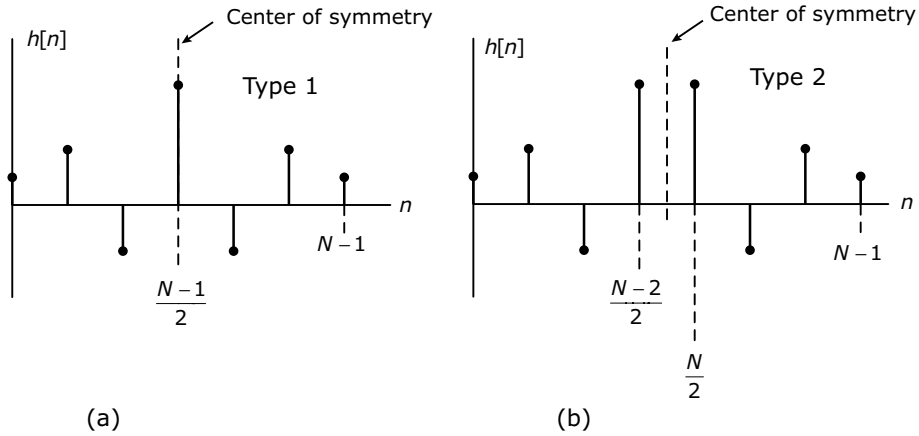


Figure 33.1 Impulse responses for linear-phase FIR filters: (a) the coefficients for a Type 1 filter exhibit even symmetry about $n = (N-1)/2$, and (b) the coefficients for a Type 2 filter exhibit even symmetry about the abscissa, midway between $n = (N-2)/2$ and $n = \mathrm{N}/2$.

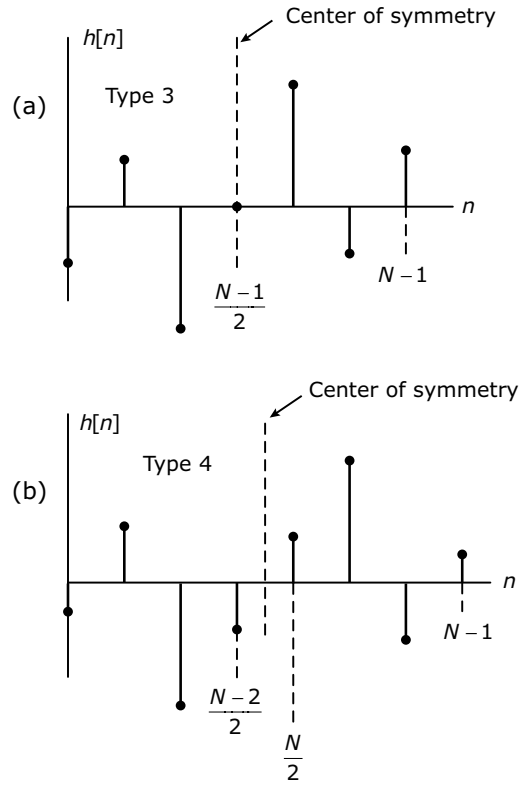


Figure 33.2 Impulse responses for constant-group-delay FIR filters: (a) Type 3 filter has odd symmetry about $n = (N-1)/2$; (b) Type 4 has odd symmetry about the abscissa, midway between $n = (N-2)/2$ and $n = N/2$.

Periodicities in Linear-Phase FIR Responses

Most filter design tools generate a filter's frequency response in *magnitude-phase* form. When working with linear-phase filters, it is sometimes convenient to view the filters' responses in *amplitude-phase form*. This note explores some of the issues that arise in converting a magnitude-phase form into an amplitude-phase form.

The magnitude-phase response can be converted into the amplitude-phase response using Recipe 34.1. However, when this is done for Type 2 or Type 4 linear-phase FIR filters, the resulting normalized-frequency amplitude response is periodic in 4π. This result appears to contradict the fact that the normalized-frequency response for any digital filter should be periodic in 2π. This note explores this apparent contradiction.

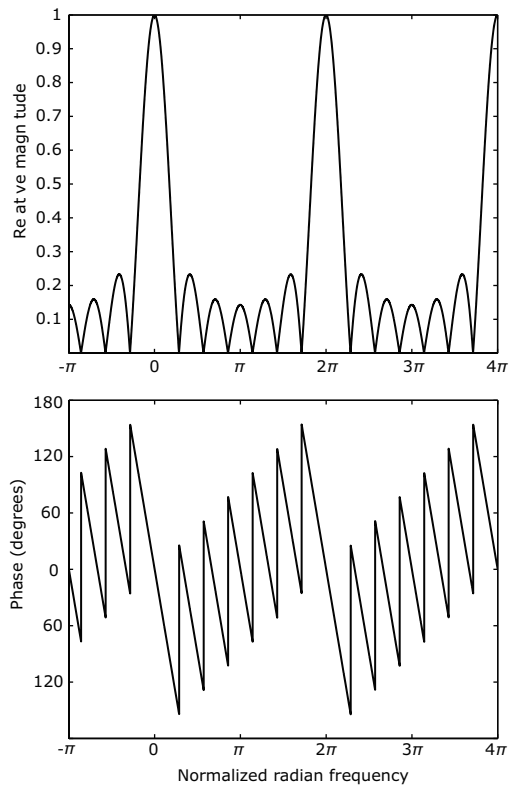


Figure 34.1 Magnitude and phase responses for a 7-tap "boxcar" FIR averaging filter

Example 34.1

Responses for Type 1 Filter

Figure 34.1 shows the magnitude and phase responses for a 7-tap FIR "boxcar" averaging filter, which happens to be a Type 1 linear-phase filter. The impulse response of this filter is rectangular, having equal weights for all coefficients.

$$y[n] = \sum_{k=0}^{6} \frac{1}{7} x[n-k]$$

The nickname "boxcar" comes from the appearance of the rectangular impulse response as it slides along the time axis. The nulls in the magnitude response occur at the same frequencies at which the phase response exhibits jump discontinuities of 180 degrees, or π radians.

Using Recipe 34.1, the magnitude and phase responses of Figure 34.1 can be converted into the amplitude and phase responses of Figure 34.2. The zero crossings in the amplitude response correspond to the nulls in the magnitude response of Figure 34.1.

The phase response shown in Figure 34.3 has phase jumps of $3 \times 360 = 1080$ degrees removed at frequencies corresponding to odd multiples of π radians per second. The phase is presented in this way to show that the phase is a ramp of constant slope across the full 2π of the baseband and each of its images. The phase decreases by 1080 degrees across each image, and we're showing a 1080-degree jump increase at the end of each image. Thus the phase starts at the same value of 540 degrees at the lefthand edge of each image. If we remove all jumps that are integer multiples of 360 degrees, the phase response is equivalent to the continuous ramp shown in Figure 34.3. Notice that the slope of this ramp agrees with Eq. (33.2) from Note 33.

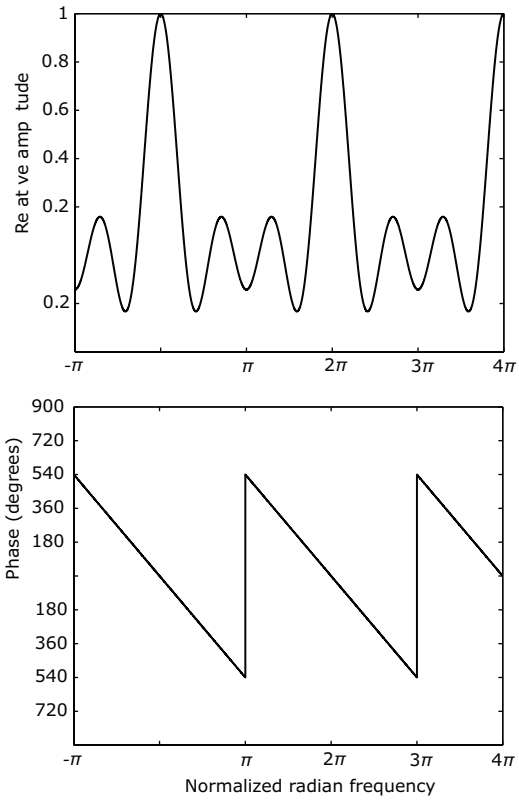


Figure 34.2 Amplitude and phase responses for a 7-tap "boxcar" FIR averaging filter

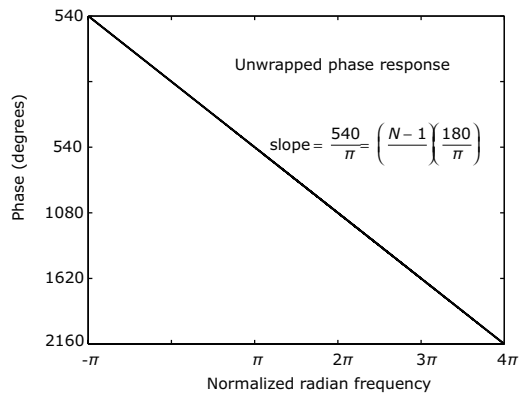


Figure 34.3 Phase response equivalent to Figure 34.2 with all phase jumps of 360 degrees removed

Recipe 34.1

Converting Magnitude-Phase Response to Amplitude-Phase Response

1. Initially, set the amplitude response equal to the magnitude response.
2. Start at $\omega = 0$ and begin moving left along the phase response, looking for a 180-degree jump decrease in phase.*
3. When a 180-degree jump is encountered at some frequency, ω_j, do the following:
   (a) Modify the amplitude response by inverting the sign of the amplitude at frequencies to the left of the phase jump.
   (b) Remove the phase jump by adding 180 degrees to the phase response at all frequencies to the left of the jump.†
4. Continue moving left along the phase response and performing step 3 for each encountered jump until the left edge of the phase response is reached.
5. Start at $\omega = 0$ and begin moving right along the phase response, looking for a 180-degree jump increase in phase.
6. When a 180-degree jump is encountered at some frequency, ω_j, do the following:
   (a) Modify the amplitude response by inverting the sign of the amplitude at all frequencies to the right of the phase jump.
   (b) Remove the phase jump by subtracting 180 degrees from the phase response at all frequencies to the right of the phase jump.
7. Continue moving to the right along the frequency response and perform step 6 for each encountered jump until the right edge of the phase response is reached.

* Due to the quantized and sampled nature of the numerically computed phase response, the jump discontinuities are rarely exactly 180 degrees. Any jump that is within a reasonable tolerance of 180 degrees should be considered a 180-degree jump. It should be noted that the phase response of a linear filter should not include any discontinuities other than 180-degree phase jumps.

† It is tempting to refer to removal of these phase jumps as "unwrapping" the phase. However, unwrapping is used to describe the process of removing jumps of 2π from phase responses. Such removals can be performed on any phase response. Removing jumps of π, as discussed in this procedure, is appropriate only when adapting the "usual" phase response to be used in conjunction with the amplitude response rather than with the magnitude response.

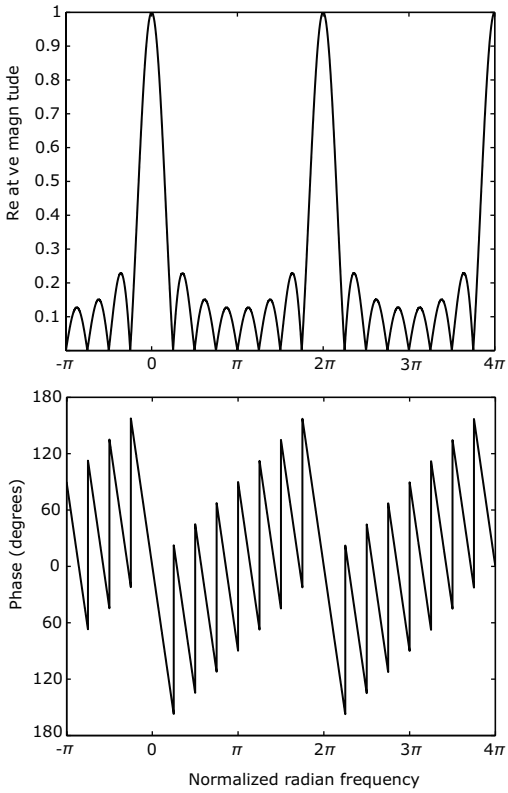


Figure 34.4 Amplitude and phase responses for an 8-tap "boxcar" FIR averaging filter.

Example 34.2

Responses for Type 2 Filter

Figure 34.4 shows the magnitude and phase responses for an 8-tap FIR "boxcar" averaging filter, which happens to be a Type 2 linear-phase filter. The nulls in the magnitude response occur at the same frequencies at which the phase response exhibits jump discontinuities of 180 degrees, or π radians.

Using Recipe 34.1, the magnitude and phase responses of Figure 34.4 can be converted into the amplitude and phase responses of Figure 34.5. The zero crossings in the amplitude response correspond to the nulls in the magnitude response of Figure 34.4. As shown, the amplitude response is periodic in 4π, with even images and odd images having opposite signs.

The phase decreases by 1260 degrees across each image, but because 1260 is not an integer multiple of 360 degrees, we cannot simply insert jumps of $360n$ degrees at the end of each image to make the phase response periodic in 2π. The phase response can be made periodic in 4π by alternating phase jumps of 1080 degrees and 1440 degrees, as shown in Figure 34.6.

Another possibility that might come to mind would be to reverse the sign of alternate images to yield the amplitude response shown in Figure 34.7, which is periodic in 2π. The phase response must be changed to compensate for the sign changes in the amplitude response. We can add 180 degrees to the 1080-degree jump and subtract 180 degrees from the 1440-degree jump to obtain a phase response that is also periodic in 2π, as shown. Unfortunately, this phase response is equivalent to the unwrapped response shown in Figure 34.8, which cannot be made into a continuous ramp.

The bottom line is that the normalized-frequency response for a digital filter can always be made periodic in 2π, but there may be discontinuities at the inter-image boundaries. In some cases, to obtain a response without discontinuities, it is necessary to cast the response in terms of components that are periodic in 4π rather than in the expected 2π.

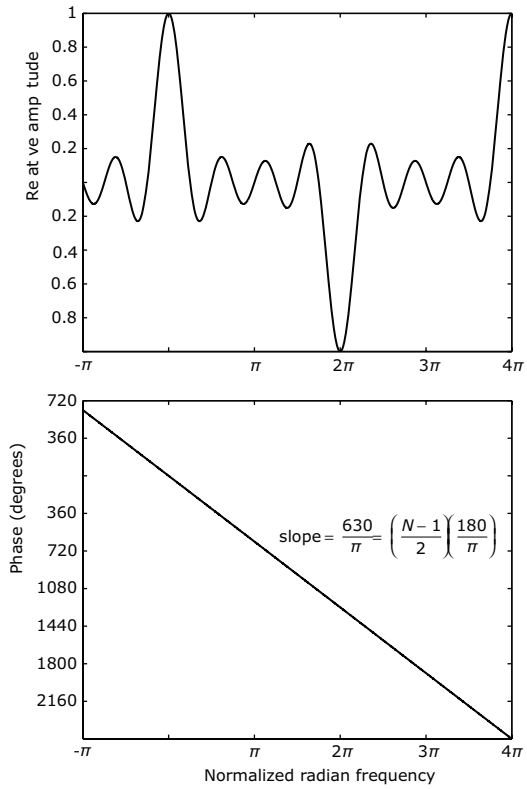


Figure 34.5 Amplitude and unwrapped phase responses for an 8-tap "boxcar" FIR averaging filter

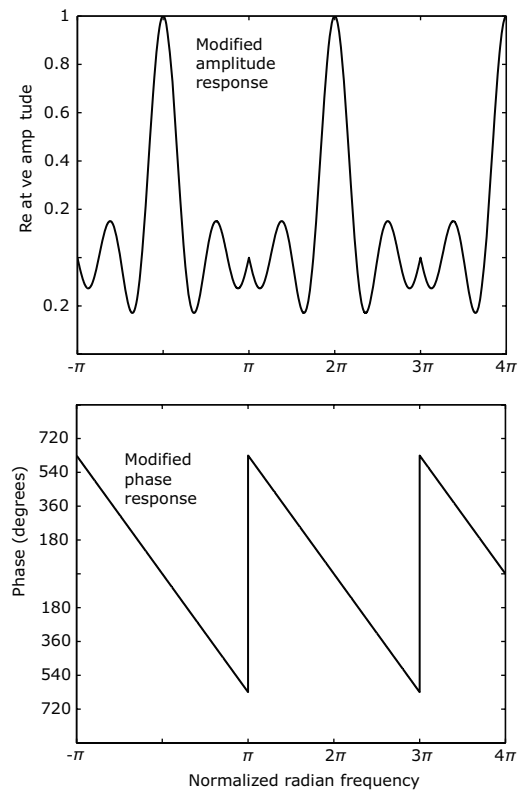


Figure 34.7 Type 2 amplitude and phase responses forced to be periodic in 2π

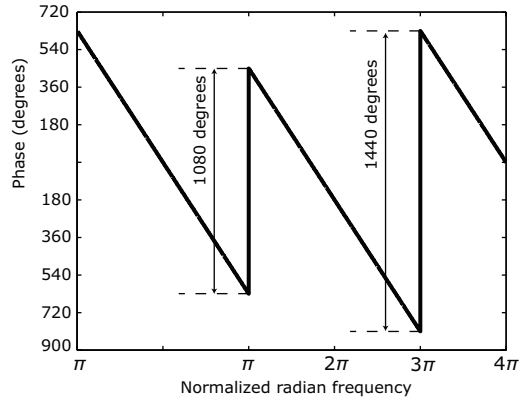


Figure 34.6 Wrapped version of phase response from Figure 34.5 showing alternating phase jumps of 1080 and 1440 degrees

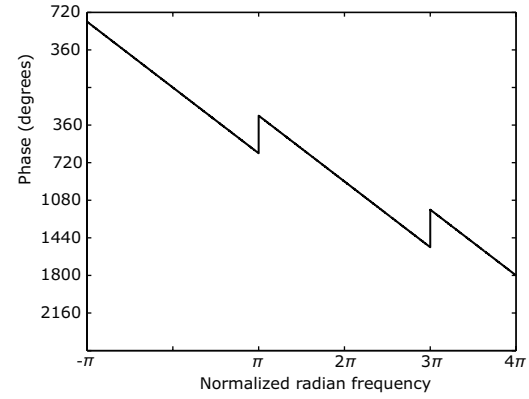


Figure 34.8 Unwrapped version of phase response from Figure 34.7 showing discontinuities that cannot be removed unless the amplitude response is allowed to be periodic in 4π

Designing FIR Filters: Basic Window Method

The window method of FIR design is based on the fact that the frequency response of a digital filter is periodic, and therefore can be represented as a Fourier series. A template for the desired frequency response is selected and expanded as a Fourier series. This expansion is then truncated to finite-number terms by multiplying the sequence of Fourier series coefficients with a sequence of samples obtained from a time-limited window function. The resulting finite sequence of terms is then used as the coefficients for an FIR filter. This filter has a frequency response that approximates the original desired response. When a rectangular window is used to truncate the coefficient sequence, the window method is called the *Fourier series method*.

Recipe 35.1

Fourier Series Method for FIR Filter Design

The following steps detail the basic strategy for using the Fourier series method. This strategy is specialized to specific band configurations in Recipes 35.2 through 35.5.

1. Specify a desired response, $H_d(\lambda)$, as a function of normalized continuous radian frequency λ for $-\pi \le \lambda \le \pi$.
2. Specify the desired number of filter taps, N.
3. Compute the filter coefficients $h[n]$ for $n = 0, 1, 2, \ldots, N-1$ using

$$h[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(\lambda)\left[\cos(m\lambda) + j\sin(m\lambda)\right]d\lambda$$

where $m = n-(N-1)/2$. When $H_d(\lambda)$ is even symmetric, computation of the coefficients simplifies to

$$h[n] = \frac{1}{\pi}\int_{0}^{\pi} H_d(\lambda)\cos(m\lambda)\,d\lambda$$

4. Using the DTFT, compute the frequency response of the filter defined by $h[n]$. If the performance is not adequate, modify N and/or $H_d(\lambda)$, and repeat step 3.

Recipe 35.2

Approximating the Ideal Lowpass Filter

Perform the following steps to approximate the ideal lowpass filter response depicted in Figure 35.1(a).

1. Specify the desired cutoff frequency, λ_U.
2. Determine the desired number of filter taps, N.
3. For $n = 0, 1, 2, \ldots, \left\lceil \frac{N-1}{2} \right\rceil - 1$, compute $h[n]$ and $h[N-1-n]$ as

$$h[n] = h[N-1-n] = \frac{\sin(m\lambda_U)}{m\pi}$$

where $m = n-(N-1)/2$. The notation $\lceil x \rceil$ indicates the *ceiling* function, which has a value equal to the largest integer, less than or equal to x.

4. If N is odd, compute $h[(N-1)/2]$ as

$$h\left[\frac{N-1}{2}\right] = \frac{\lambda_U}{\pi}$$

Recipe 35.3

Approximating the Ideal Highpass Filter

Perform the following steps to approximate the ideal lowpass filter response depicted in Figure 35.1(b).

1. Specify the desired cutoff frequency, λ_L.
2. Determine the desired number of filter taps, N.
3. For $n = 0, 1, 2, \ldots, \left\lceil \frac{N-1}{2} \right\rceil - 1$, compute $h[n]$ and $h[N-1-n]$ as

$$h[n] = h[N-1-n] = \frac{-\sin(m\lambda_L)}{m\pi}$$

   where $m = n - (N-1)/2$.
4. If N is odd, compute $h[(N-1)/2]$ as

$$h\left[\frac{N-1}{2}\right] = 1 - \frac{\lambda_L}{\pi}$$

Recipe 35.5

Approximating the Ideal Bandstop Filter

Perform the following steps to approximate the ideal bandpass filter response depicted in Figure 35.1(d).

1. Specify the desired lower cutoff frequency, λ_L, and the upper cutoff frequency, λ_U.
2. Determine the desired number of filter taps, N.
3. For $n = 0, 1, 2, \ldots, \left\lceil \frac{N-1}{2} \right\rceil - 1$, compute $h[n]$ and $h[N-1-n]$ as

$$h[n] = h[N-1-n] = \frac{1}{m\pi}\left[\sin(m\lambda_L) - \sin(m\lambda_U)\right]$$

   where $m = n - (N-1)/2$.
4. If N is odd, compute $h[(N-1)/2]$ as

$$h\left[\frac{N-1}{2}\right] = 1 - \frac{\lambda_U - \lambda_L}{\pi}$$

Recipe 35.4

Approximating the Ideal Bandpass Filter

Perform the following steps to approximate the ideal bandpass filter response depicted in Figure 35.1(c).

1. Specify the desired lower cutoff frequency, λ_L, and the upper cutoff frequency, λ_U.
2. Determine the desired number of filter taps, N.
3. For $n = 0, 1, 2, \ldots, \left\lceil \frac{N-1}{2} \right\rceil - 1$, compute $h[n]$ and $h[N-1-n]$ as

$$h[n] = h[N-1-n] = \frac{1}{m\pi}\left[\sin(m\lambda_U) - \sin(m\lambda_L)\right]$$

   where $m = n - (N-1)/2$.
4. If N is odd, compute $h[(N-1)/2]$ as

$$h\left[\frac{N-1}{2}\right] = \frac{\lambda_U - \lambda_L}{\pi}$$

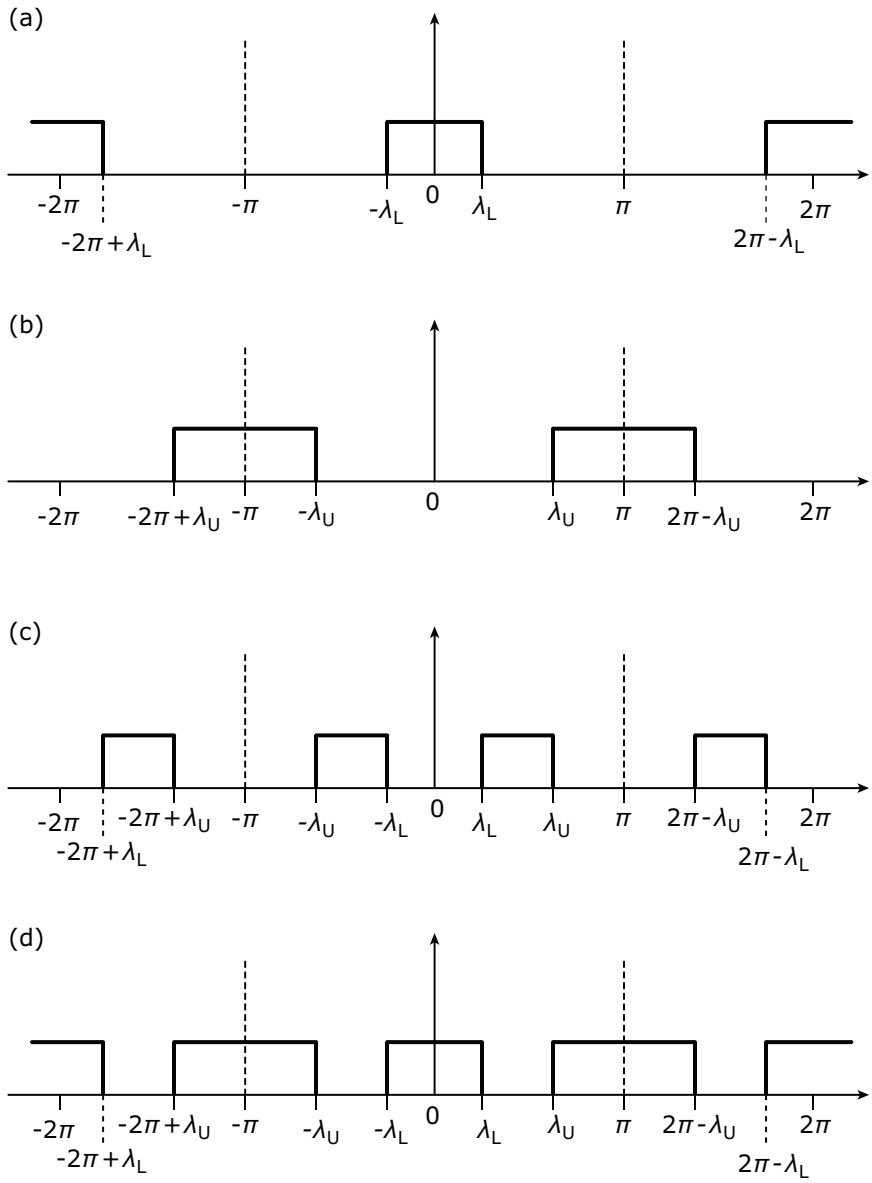


Figure 35.1 Frequency response for ideal digital filters: (a) lowpass, (b) highpass, (c) bandpass, and (d) bandstop

References

1. D. Slepian and H. Pollak, "Prolate-Spheroidal Wave Functions, Fourier Analysis and Uncertainty—I," *Bell Syst. Tech. J.*, vol. 40, January 1961, pp. 43–64.
2. J. F. Kaiser, "Digital Filters," Chapter 7 in *System Analysis by Digital Computer*, F. F. Kuo and J. F. Kaiser eds., John Wiley & Sons, 1966.
3. J. F. Kaiser, "Nonrecursive Digital Filter Design Using the I_0–sinh Window Function," *Proc. 1974 IEEE Int. Symp. on Circuits and Syst.*, April 22–25, 1974, pp. 20–23.
4. A. Antoniou, *Digital Filters: Analysis and Design*, McGraw-Hill, 1979.
5. f. j. harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proc. IEEE*, vol. 66, no. 1, January 1978, pp. 51–83.

Designing FIR Filters: Kaiser Window Method

The Kaiser window is the window most often used in the windowing technique for designing linear-phase FIR filters that is described in Note 35. In order to exploit the adjustability of the Kaiser window most effectively, the filter design techniques presented in Note 35 must be modified, as shown for the lowpass case in Recipe 36.1. Comparing Recipe 35.1 and Recipe 36.1 reveals that Recipe 36.1 can be made generic with respect to band configuration if changes are made to steps 2, 8, and 9 to obtain the result shown in Recipe 36.2.

Recipe 36.1

Designing Lowpass FIR Filters Using the Kaiser Window

Perform the following steps to approximate the ideal lowpass filter response.

1. Based on the intended application, determine the following:
   - the normalized passband edge frequency, λ_P
   - the normalized stopband edge frequency, λ_S
   - the maximum tolerable passband ripple, δ_P
   - the maximum tolerable stopband ripple, δ_S
2. Determine the transition width, $\Delta\lambda$, as
$$\Delta\lambda = \lambda_S - \lambda_P$$
3. Set δ equal to the smaller of δ_P or δ_S, and compute A as
$$A = -20\log_{10}\delta$$
4. Use the value of A computed in step 3 to determine the value for β^* as
$$\beta = \begin{cases} 0.1102(A-8.7) & A > 50 \\ 0.5842(A-21)^{0.4} + 0.07886(A-21) & 21 \le A \le 50 \\ 0 & A < 21 \end{cases}$$

* This formula for β was determined empirically by Kaiser and reported in [3].

5. Determine the required number of taps, N, as
$$N = 1 + \frac{A-8}{2.285\Delta\lambda}$$
6. Compute the window coefficients as
$$w[n] = \frac{I_0\left[\beta\sqrt{1-\left(1-\frac{2n}{N-1}\right)^2}\right]}{I_0(\beta)} \qquad n = 0,1,\ldots,\left\lceil\frac{N-1}{2}\right\rceil$$
7. For $n = 0, 1, \ldots, \left\lceil\frac{N-1}{2}\right\rceil - 1$, compute $h[n]$ and $h[N-1-n]$ as
$$h[n] = h[N-1-n] = \frac{\sin\left[m(\lambda_p+\lambda_S)/2\right]}{m\pi}w[n]$$
where $m = n - (N-1)/2$.
8. If N is odd, compute $h[(N-1)/2]$ as
$$h\left[\frac{N-1}{2}\right] = \frac{\lambda_C}{\pi}w\left[\frac{N-1}{2}\right]$$
9. Check the filter response against the conditions specified in step 1. If the filter meets the specified conditions, the design is complete. If the filter response does not meet the specifications, increase N by 1, and return to step 6.

Recipe 36.2

Designing FIR Filters Using the Kaiser Window

Perform the following steps to approximate the ideal filter response.

1. Based on the intended application, determine the following specifications:
   (a) The maximum tolerable passband ripple, δ_P, and maximum tolerable stopband ripple, δ_S. Bandpass filters have two stopbands. If the tolerable ripple is different for these two stopbands, set δ_S to the to the more stringent of the two. Bandstop filters have two passbands; if the tolerable ripple is different for these two passbands, set δ_P to the more stringent of the two.
   (b) Critical frequencies, as depicted in Figure 35.1
   - the normalized passband edge frequencies: λ_P for LP, HP; λ_{PL} and λ_{PU} for BP, BS
   - the normalized passband edge frequencies: λ_S for LP, HP; λ_{SL} and λ_{SU} for BP, BS

2. Determine the transition width, $\Delta\lambda$, as

$$\Delta\lambda = \begin{cases} \lambda_S - \lambda_P & \text{lowpass} \\ \lambda_P - \lambda_S & \text{highpass} \\ \min(\lambda_{PL} - \lambda_{SL}, \lambda_{SU} - \lambda_{PU}) & \text{bandpass} \\ \min(\lambda_{SL} - \lambda_{PL}, \lambda_{PU} - \lambda_{SU}) & \text{bandstop} \end{cases}$$

3. Set δ equal to the smaller of δ_p or δ_s, and compute A as

$$A = -20\log_{10}\delta$$

4. Use the value of A computed in step 3 to determine the value for β as

$$\beta = \begin{cases} 0.1102(A - 8.7) & A > 50 \\ 0.5842(A-21)^{0.4} + 0.07886(A-21) & 21 \le A \le 50 \\ 0 & A < 21 \end{cases}$$

5. Determine the required number of taps, N, as

$$N = 1 + \frac{A - 8}{2.285\Delta\lambda}$$

6. Compute the window coefficients as

$$w[n] = \frac{I_0\left[\beta\sqrt{1-\left(1-\frac{2n}{N-1}\right)^2}\right]}{I_0(\beta)} \qquad n = 0, 1, \ldots, \left\lceil \frac{N-1}{2} \right\rceil$$

7. For $n = 0, 1, \ldots,$, compute $h[n]$ and $h[N-1-n]$ as

$$h[n] = h[N-1-n] = \begin{cases} w[n]\dfrac{\sin\left[m(\lambda_P + \lambda_S)/2\right]}{m\pi} & \text{lowpass} \\ w[n]\dfrac{-\sin\left[m(\lambda_P + \lambda_S)/2\right]}{m\pi} & \text{highpass} \\ w[n]\dfrac{\sin\left[m(\lambda_{PU} + \lambda_{SU})/2\right] - \sin\left[m(\lambda_{PL} + \lambda_{SL})/2\right]}{m\pi} & \text{bandpass} \\ w[n]\dfrac{\sin\left[m(\lambda_{PL} + \lambda_{SL})/2\right] - \sin\left[m(\lambda_{PU} + \lambda_{SU})/2\right]}{m\pi} & \text{bandstop} \end{cases}$$

where $m = n - (N-1)/2$.

8. If N is odd, compute $h[(N-1)/2]$ as

$$h\left[\frac{N-1}{2}\right] = \begin{cases} w\left[\dfrac{N-1}{2}\right]\dfrac{\lambda_P + \lambda_S}{2\pi} & \text{lowpass} \\ w\left[\dfrac{N-1}{2}\right]\left(1 - \dfrac{\lambda_P + \lambda_S}{2\pi}\right) & \text{highpass} \\ w\left[\dfrac{N-1}{2}\right]\dfrac{\lambda_{PU} + \lambda_{SU} - \lambda_{PL} - \lambda_{SL}}{2\pi} & \text{bandpass} \\ w\left[\dfrac{N-1}{2}\right]\left(1 + \dfrac{\lambda_{PU} + \lambda_{SU} - \lambda_{PL} - \lambda_{SL}}{2\pi}\right) & \text{bandstop} \end{cases}$$

9. Check the filter response against the conditions specified in step 1. If the filter meets the specified conditions, the design is complete. If the filter response does not meet the specifications, increase N by 1, and return to step 6.

References

1. D. Slepian and H. Pollak, "Prolate-Spheroidal Wave Functions, Fourier Analysis and Uncertainty—I," *Bell Syst. Tech. J.*, vol. 40, January 1961, pp. 43–64.
2. J. F. Kaiser, "Digital Filters," Chapter 7 in *System Analysis by Digital Computer*, F. F. Kuo and J. F. Kaiser eds., John Wiley & Sons, 1966.
3. J. F. Kaiser, "Nonrecursive Digital Filter Design Using the I_0–sinh Window Function," *Proc. 1974 IEEE Int. Symp. on Circuits and Syst.*, April 22–25, 1974, pp. 20–23.
4. A. Antoniou, *Digital Filters: Analysis and Design*, McGraw-Hill, 1979.
5. f. j. harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proc. IEEE*, vol. 66, no. 1, January 1978, pp. 51–83.

Designing FIR Filters: Parks-McClellan Algorithm

The *Parks-McClellan algorithm* is by far the most widely used technique for designing FIR filters. For many years, this algorithm was called the *Remez algorithm* or the *Remez exchange*. However, the recent trend has been to name the approach in honor of Thomas Parks and John McClellan, the two individuals who first publicized the utility of using the Remez exchange for designing FIR filters.

An FIR approximation to some ideal desired response typically exhibits error ripples around the ideal response, as shown in Figure 37.1(a). The Parks-McClellan (PM) algorithm is based on the fact that, for a given filter length, the worst-case error is minimized when all of the error extrema are equal in magnitude, as shown in Figure 37.1(b). Hence, a filter resulting from the PM algorithm is often referred to as an *equiripple* filter.

The filter produced by the PM algorithm exhibits the following characteristics.

- The passband has ripples that deviate from unity by $\pm\delta_P$.
- The stopband has ripples that deviate from zero by $\pm\delta_S$.
- The passband edge frequency, ω_P, and stopband edge frequency, ω_S, match the specified values.
- The maximum approximation error is minimized.

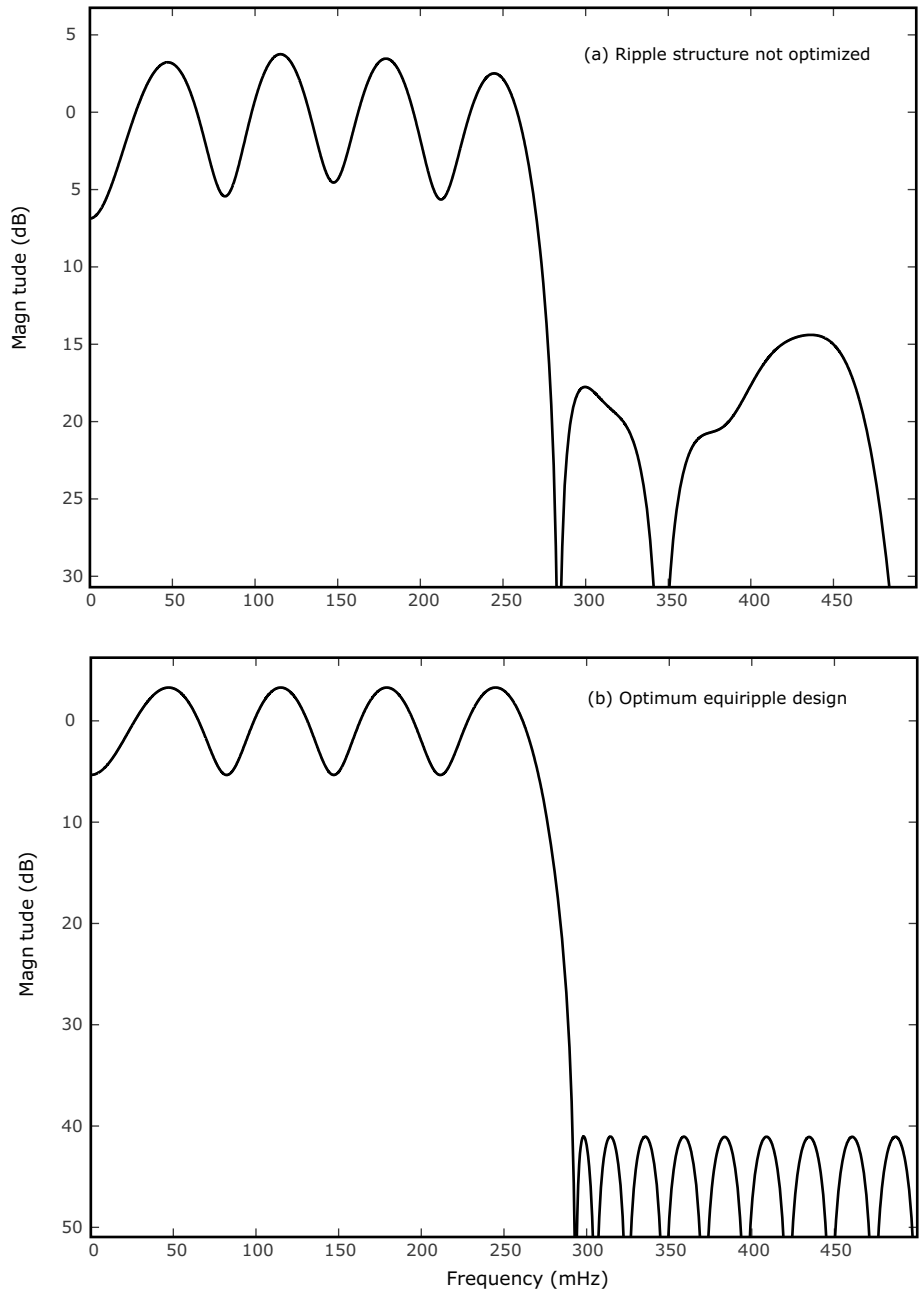


Figure 37.1 Comparison of magnitude responses for two 36-tap FIR filter designs: (a) suboptimal design showing excessive ripple, (b) optimized equiripple design

Laplace Transform

The Laplace transform is a mathematical tool that is used primarily for analyzing linear analog systems such as filters. This transform is of interest in digital signal processing because analog filters having transfer functions defined in terms of Laplace transforms are often used as the starting point in the design of digital IIR filters. The characterization of analog filters is discussed in Note 39, and commonly used analog filter families are discussed in Notes 40 through 43.

The Laplace transform for a continuous-time function, $x(t)$, is usually denoted as $X(s)$ or $\mathcal{L}[x(t)]$, and is defined by Eq. (MB 38.1). The complex variable, s, is usually referred to as *complex frequency*, and can be put into the form $\sigma+j\omega$, where σ and ω are real variables, sometimes referred to as *neper frequency* and *radian frequency,* respectively.

Early interest in the Laplace transform was driven by the fact that if we take the Laplace transform of both sides of a differential equation in continuous time, t, we obtain an algebraic equation in complex frequency, s, that can be more easily solved for the desired quantity. The behavior of a linear analog filter is described by a differential equation in t, and consequently, the Laplace transform plays a big role in the analysis and characterization of analog filters.

The inverse Laplace transform is defined by Eq. (MB 38.2). Evaluating the integrals in Eq. (MB 38.1) and, especially, Eq. (MB 38.2) can be a major chore. However, in practice, direct evaluation of these integrals usually can be avoided by using some well-known transform pairs selected from Table 38.1, along with a number of transform properties selected from Table 38.2.

Math Box 38.1

Laplace Transform

$$X(s)=\mathcal{L}\{x(t)\}=\int_{-\infty}^{\infty}x(t)e^{-st}dt \qquad \text{(MB 38.1)}$$

Inverse Transform

$$x(t)=\mathcal{L}^{-1}\{X(s)\}=\frac{1}{2\pi j}\int_{C}X(s)e^{st}ds \qquad \text{(MB 38.2)}$$

where C is a closed contour of integration chosen to include all singularities of $X(s)$

Table 38.1 Laplace Transform Pairs

#	$x(t)$	$X(s)$
1	1	$\frac{1}{s}$
2	$u_1(t)$	$\frac{1}{s}$
3	$\delta(t)$	1
4	t	$\frac{1}{s^2}$
5	t^n	$\frac{n!}{s^{n+1}}$
6	$\sin \omega t$	$\frac{\omega}{s^2+\omega^2}$
7	$\cos \omega t$	$\frac{s}{s^2+\omega^2}$
8	e^{-at}	$\frac{1}{s+a}$
9	$e^{-at}\sin \omega t$	$\frac{\omega}{(s+a)^2+\omega^2}$
10	$e^{-at}\cos \omega t$	$\frac{s+a}{(s+a)^2+\omega^2}$

Table 38.2 Laplace Transform Properties

#	Property	Time Function	Transform
1	Homogeneity	$af(t)$	$aF(s)$
2	Additivity	$f(t)+g(t)$	$F(s)+G(s)$
3	Linearity	$af(t)+bg(t)$	$aF(s)+bG(s)$
4	First derivative	$\frac{d}{dt}f(t)$	$sF(s)-f(0)$
5	Second derivative	$\frac{d^2}{dt^2}f(t)$	$s^2F(s)-sf(0)-\frac{d}{dt}f(0)$
6	kth derivative	$\frac{d^{(k)}}{dt^k}f(t)$	$s^kF(s)-\sum_{n=0}^{k-1}s^{k-1-n}f^{(n)}(0)$
7	Integration	$\int_{-\infty}^{t}f(\tau)d\tau$	$\frac{F(s)}{s}+\frac{1}{s}\left(\int_{-\infty}^{t}f(\tau)d\tau\right)_{t=0}$
		$\int_{0}^{t}f(\tau)d\tau$	$\frac{F(s)}{s}$
8	Frequency shift	$e^{-at}f(t)$	$X(s+a)$
9	Time shift right	$u_1(t-\tau)f(t-\tau)$	$e^{-\tau s}F(s)\ \ a>0$
10	Time shift left	$f(t+\tau),\quad f(t)=0 \text{ for } 0<t<\tau$	$e^{\tau s}F(s)$
11	Convolution	$y(t)=\int_0^t h(t-\tau)x(\tau)d\tau$	$Y(s)=H(s)X(s)$

Characterizing Analog Filters

Several of the most popular techniques for designing IIR filters are based on transformations or mappings of analog prototype filters into digital filters. Often these prototype filters are classic filter designs such as *Butterworth, Chebyshev,* or *elliptic.* These classical filters are usually presented in the literature as normalized lowpass filters that must be denormalized and possibly transformed into other band configurations before they can be used as prototypes for IIR filters. This note covers techniques that are used to characterize and frequency-scale classical analog filters.

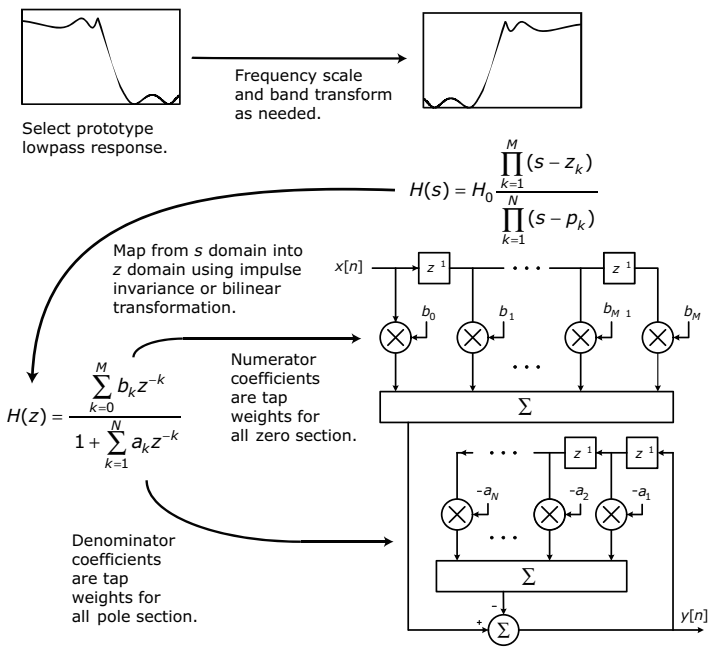
Neglecting imperfections in the components used to implement them, classical analog filters are considered to be time-invariant, lumped-parameter linear systems. The fundamental properties of linear systems are defined in Math Box 39.1.

A linear system can be characterized by a differential equation, step response, impulse response, complex-frequency-domain system function, or transfer function: There are a number of useful and well known relationships between these various characterizations.

- The time-domain input signal, $x(t)$, and output signal, $y(t)$, are related by a time-domain differential equation.
- The complex-frequency-domain system function is obtained by performing the Laplace transform on the time-domain differential equation.
- The *impulse response*, $h(t)$, can be obtained by solving the differential equation for $x(t)$ set equal to the unit impulse, $\delta(t)$.

Key Concept 39.3

Using Analog Prototypes for IIR Filters



- The *step response*, $a(t)$, can be obtained by solving the differential equation for $x(t)$ set equal to the unit step, $u(t)$.
- The *impulse response*, $h(t)$, can be obtained by differentiating the *step response*, $a(t)$, with respect to time.
- The *step response*, $a(t)$, can be obtained by integrating the *impulse response*, $h(t)$, with respect to time.
- The *transfer function* can be obtained by solving the complex-frequency-domain system function for $H(s) = Y(s)/X(s)$, where $Y(s)$ and $X(s)$ are, respectively, the Laplace transforms of $y(t)$ and $x(t)$.
- The transfer function, $H(s)$, can be obtained as the Laplace transform of the impulse response, $h(t)$.

39.1 Transfer Functions

The transfer function, $H(s)$, of a system is equal to the Laplace transform of the output signal divided by the Laplace transform of the corresponding input signal:

$$H(s) = \frac{\mathcal{L}\{y(t)\}}{\mathcal{L}\{x(t)\}}$$

The transfer function can be put into the form

$$H(s) = H_0 \frac{P(s)}{Q(s)}$$

where $P(s)$ and $Q(s)$ are polynomials in s, and H_0 is the gain of the filter at zero frequency. These polynomials can be expressed in sum-of-powers form to yield

$$H(s) = H_0 \frac{\sum_{k=0}^{M} c_k s^k}{\sum_{k=0}^{N} d_k s^k}$$

Alternatively, the polynomials $P(s)$ and $Q(s)$ can be expressed in factored form to yield

$$H(s) = H_0 \frac{\prod_{k=1}^{M} (s - z_k)}{\prod_{k=1}^{N} (s - p_k)}$$

Math Box 39.1

Properties of Linear Systems

- A system, H, operating on an input signal, $x(t)$, to produce an output signal, $y(t)$, can be represented in mathematical notation as

$$y(t) = H\left[x(t)\right]$$

- A system is *homogenous* if multiplying the input by a constant gain, a, is equivalent to multiplying the output by the same constant gain:

$$H \text{ is homogeneous} \Leftrightarrow H\left[ax(t)\right] = aH\left[x(t)\right]$$

- A system is *additive* if the output produced for the sum of two input signals is equal to the sum of the outputs produced for each input individually:

$$H \text{ is additive} \Leftrightarrow H\left[x_1(t) + x_2(t)\right] = H\left[x_1(t)\right] + H\left[x_2(t)\right]$$

- A system that is both homogenous and additive is called a *linear system*.
- A system is said to be *relaxed* if it is not still responding to any previously applied input.
- The characteristics of a *time-invariant* system do not change over time. A time-invariant system is also called a *fixed* system or a *stationary* system.
- In a *causal* system, the output at time t can depend upon the input only at times t and prior.

The roots $z_1, z_2, \ldots, z_M$ of $P(s)$ are called *zeros of the transfer function*, and the roots $p_1, p_2, \ldots, p_N$ of $Q(s)$ are called *poles of the transfer function*. For the system represented by $H(s)$ to be stable and realizable in the form of a lumped parameter network, the conditions listed in Math Box 39.2 must be satisfied.

Math Box 39.2

Conditions for $H(s)$ to be Stable and Realizable as a Lumped-Parameter Network

1. The coefficients, c_k, in $P(s)$ must be real.
2. The coefficients, d_k, in $Q(s)$ must be real and positive.
3. The polynomial, $Q(s)$, must have a non-zero term for each degree of s from highest to lowest unless all even-degree terms or all odd-degree terms are missing.
4. If $H(s)$ is the voltage ratio or current ratio, the maximum degree of s in $P(s)$ cannot exceed the maximum degree of s in $Q(s)$.
5. If $H(s)$ is a transfer impedance or a transfer admittance, then the maximum degree of s in $P(s)$ can exceed the maximum degree of s in $Q(s)$ by no more than 1.

39.2 Magnitude, Phase, and Delay Responses

The steady-state frequency response of a linear system can be determined by evaluating the transfer function, $H(s)$, at $s = j\omega$:

$$H(j\omega) = H(s)\big|_{s=j\omega} = |H(j\omega)|e^{j\theta(\omega)}$$

where $\theta(\omega)$ is the phase response given by

$$\theta(\omega) = \tan^{-1}\left\{\frac{\mathrm{Im}[H(j\omega)]}{\mathrm{Re}[H(j\omega)]}\right\}$$

and $|H(j\omega)|$ is the magnitude response given by

$$|H(j\omega)| = [H(s)H(-s)]\big|_{s=j\omega} = \left(\{\mathrm{Re}[H(j\omega)]\}^2 + \{\mathrm{Im}[H(j\omega)]\}^2\right)^{1/2}$$

The *group delay*, $\tau_g(\omega)$, is defined as

$$\tau_g(\omega) = \frac{-\mathrm{d}}{\mathrm{d}\omega}\theta(\omega)$$

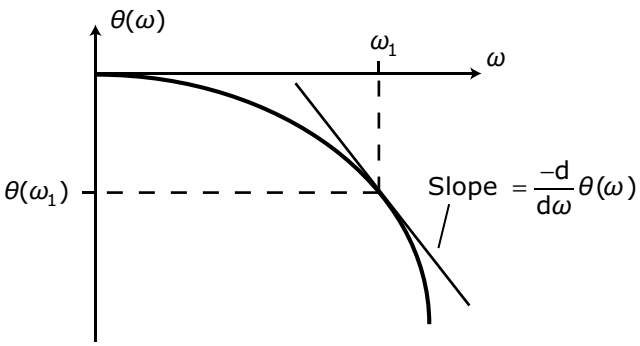


Figure 39.1 Group delay response

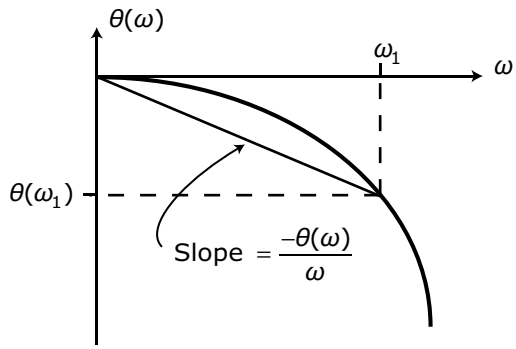


Figure 39.2 Phase delay response

where $\theta(\omega)$ is the phase response. As shown in Figure 39.1, the group delay at a frequency ω_1 is equal to the negative slope of a tangent to the phase response curve at the point corresponding to ω_1. Group delay is also called *envelope delay* because when a modulated carrier is passed through the system, the modulated envelope is delayed by τ_g. If the group delay is not consistent over the entire bandwidth of the signal, the envelope is distorted.

The *phase delay*, $\tau_p(\omega)$, of a linear system is defined as

$$\tau_p(\omega) = \frac{-\theta(\omega)}{\omega}$$

As depicted in Figure 39.2, the phase delay at a frequency ω_1 is equal to the negative slope of a secant drawn from the origin to the phase response curve at the point corresponding to ω_1. Phase delay is also called *carrier delay* because an unmodulated carrier at a frequency ω_1 experiences a delay of $\tau_p(\omega)$ when passing through the system.

39.3 Features of the Lowpass Response

The magnitude response for an analog lowpass filter will have one of the four general shapes shown in Figures 39.3 through 39.6. In each case, the filter response can be divided into three segments—the *passband*, the *transition band*, and the *stopband*. In some cases, the boundaries between these segments are defined by distinct features in the filter's magnitude response, and sometimes the boundaries are based on an arbitrary line drawn on the magnitude response.

- The monotonic magnitude response shown in Figure 39.3 is an example of where the inter-band boundaries are defined by the frequencies at which the response crosses arbitrary levels. In most cases, the passband is defined to end when the magnitude response is 3 dB below the response level at zero frequency. Sometimes a different attenuation level such as 1 dB or 6 dB is used to define the end of the passband. In monotonic responses, there is almost no agreement regarding the level that sets the boundary between the transition band and the

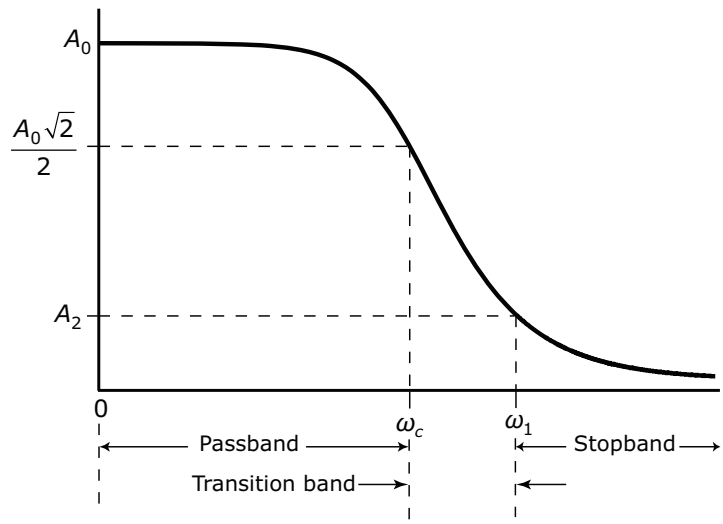


Figure 39.3 Monotonic magnitude response of a lowpass filter

stopband. A Butterworth filter has a monotonic magnitude response.

- The response shown in Figure 39.4 has ripples in the passband, and the troughs of the ripples define the level that defines the passband. The transition band begins when the response drops below the bottoms of the ripple troughs. If there is less than 3 dB of ripple in the passband, the 3 dB attenuation point is occasionally used to define the end of the passband. As with the monotonic response, there is no general agreement as to the level that should be used to define the beginning of the stopband. A Chebyshev filter has a response with ripples in the passband.
- The response shown in Figure 39.5 has ripples in the stopband, and the crests of the ripple define the level that defines the stopband. The transition band ends and the stopband begins when the response first drops below the level corresponding to the ripple crests. The passband edge is defined as it is for the monotonic response. A Chebyshev Type 2 filter has a response with ripples in the stopband.
- The response shown in Figure 39.6 has ripples in both the passband and the stopband. The troughs in the passband ripple set the level that defines the end of the passband, and the crests in the stopband ripple set the level that defines the beginning of the stopband. An elliptic filter has a response with ripples in both the passband and the stopband.

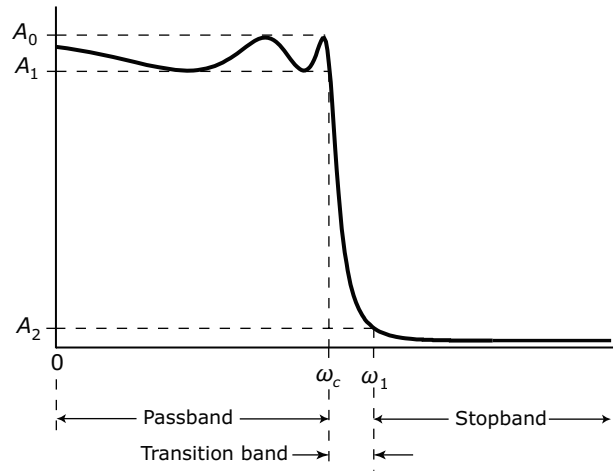


Figure 39.4 Magnitude response of a lowpass filter with ripples in the passband

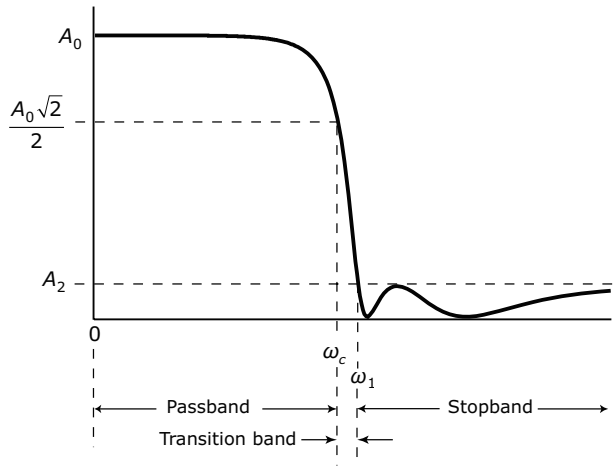


Figure 39.5 Magnitude response of a lowpass filter with ripples in the stopband

39.4 Passband Transformations

When working with the classical filter families presented in Notes 40 through 43, it is common practice to first select a lowpass prototype filter having the appropriate characteristics and then transform this prototype into a bandpass or bandstop configuration as required. This note describes transformations that can be used to convert lowpass into bandpass or lowpass into bandstop. Similar transformations exist for converting lowpass into

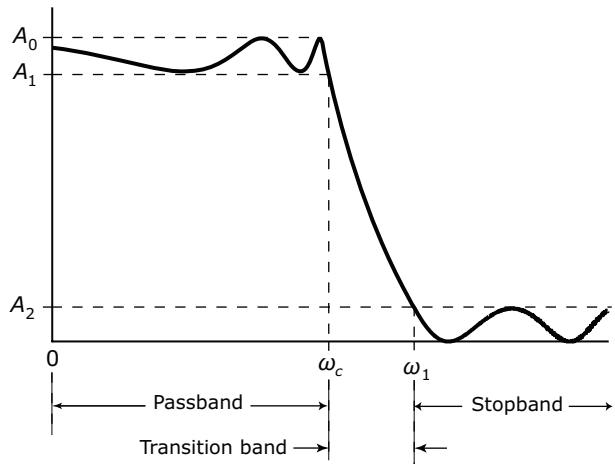


Figure 39.6 Magnitude response of a lowpass filter with ripples in both the passband and the stopband.

highpass, but highpass filters are rarely used in DSP applications.

Bandpass Transformation

Consider a lowpass filter normalized for 3-dB attenuation at frequency ω_0 and having a transfer function, $H(S)$. There are a number of different transformations that can be used to map this prototype into a corresponding bandpass filter. The "conventional" transformation is based on evaluating $H(S)$ at

$$S = \frac{\omega_0}{\omega_2 - \omega_1}\left(\frac{s}{\omega_0} + \frac{\omega_0}{s}\right) \tag{39.1}$$

to obtain a filter with center frequency ω_0, lower 3-dB frequency ω_1, and upper 3-dB frequency ω_2. The properties of this transformation are examined in detail in [1]. If the lowpass prototype has zeros on the imaginary axis at $S_i = \pm j\beta$, each such pair transforms to two conjugate pairs of bandpass zeros on the imaginary axis:

$$s = \pm j\left[\frac{\beta(\omega_2 - \omega_1)}{2} \pm \omega_0\sqrt{1 + \left(\frac{\gamma\beta}{2}\right)^2}\right] \tag{39.2}$$

where γ is the relative bandwidth given by

$$\gamma = \frac{\omega_2 - \omega_1}{\omega_0}$$

In order to use a lowpass protoype filter having a real-valued pole at $S_i = -\alpha$, where $\alpha > 0$, the frequencies of the bandpass filter must satisfy

$$|\alpha| < \frac{2\omega_0}{\omega_2 - \omega_1} \tag{39 3}$$

If Eq. (39.3) is satisfied, the real lowpass pole transforms to the complex-conjugate pair of bandpass poles:

$$s = \frac{\alpha(\omega_1 - \omega_2)}{2} \pm j\omega_0\sqrt{1 - \left(\frac{\gamma\alpha}{2}\right)^2} \tag{39.4}$$

Each complex-conjugate pair of lowpass poles, $-\alpha \pm j\beta$, transforms to two complex-conjugate pairs of bandpass poles given by

$$s_1 = \omega_0\left[\frac{-\gamma\alpha}{2} + b \pm j\left(\frac{\gamma\beta}{2} - a\right)\right]$$
$$s_2 = \omega_0\left[\frac{-\gamma\alpha}{2} - b \pm j\left(\frac{\gamma\beta}{2} + a\right)\right] \tag{39 5}$$

where

$$a = \sqrt{\frac{A + \sqrt{A^2 + B^2}}{2}}$$

$$b = \sqrt{\frac{-A + \sqrt{A^2 + B^2}}{2}}$$

$$A = 1 - \frac{\gamma^2(\alpha^2 - \beta^2)}{4} \qquad B = \frac{-\alpha\beta\gamma^2}{2}$$

Bandstop Transformation

Consider a lowpass filter normalized for 3-dB attenuation at frequency ω_0 and having a transfer function, $H(S)$. For mapping such a prototype into the corresponding bandstop filter, we can use a transformation that is based upon evaluating $H(S)$ at

$$S = \frac{\gamma s(\omega_2 - \omega_1)}{s^2\omega_0^2}$$

to obtain a bandstop filter with center frequency ω_0, lower 3-dB frequency ω_1, and upper 3-dB frequency ω_2. If the lowpass prototype has zeros on the imaginary axis at $S_i = \pm j\beta$, each such pair transforms to two conjugate pairs of bandstop zeros on the imaginary axis:

$$s = \pm j\left[\frac{\omega_2 - \omega_1}{2\beta} \pm \omega_0\sqrt{1+\left(\frac{\gamma}{2\beta}\right)^2}\right] \qquad (39.6)$$

In order to use a lowpass prototype filter having a real-valued pole at $S_i = -\alpha$, where $\alpha > 0$, the frequencies of the bandpass filter must satisfy

$$|\alpha| > \frac{\omega_2 - \omega_1}{2\omega_0} \qquad (39.7)$$

If Eq.(39.7) is satisfied, this pole transforms to the complex-conjugate pair of bandstop poles:

$$s = \frac{\omega_1 - \omega_2}{2\alpha} \pm j\omega_0\sqrt{1-\left(\frac{\gamma}{2\alpha}\right)^2} \qquad (39.8)$$

Each complex-conjugate pair of lowpass poles $-\alpha \pm j\beta$ transforms to two complex-conjugate pairs of bandstop poles given by

$$s_1 = \omega_0\left[\frac{-\gamma\alpha}{2\left(\alpha^2+\beta^2\right)} + b \pm j\left(\frac{\gamma\beta}{2\left(\alpha^2+\beta^2\right)} - a\right)\right]$$

$$s_2 = \omega_0\left[\frac{-\gamma\alpha}{2\left(\alpha^2+\beta^2\right)} - b \pm j\left(\frac{\gamma\beta}{2\left(\alpha^2+\beta^2\right)} + a\right)\right]$$

where

$$A = 1 - \frac{\gamma^2}{4\left(\alpha^2+\beta^2\right)} \qquad B = \frac{-\alpha\beta\gamma}{2\left(\alpha^2+\beta^2\right)}$$

$$a = \sqrt{\frac{A+\sqrt{A^2+B^2}}{2}} \qquad b = \sqrt{\frac{-A+\sqrt{A^2+B^2}}{2}}$$

References

1. H. J. Blinchikoff and A. I. Zverev, *Filtering in the Time and Frequency Domains*, Wiley-Interscience, New York, 1976.

Butterworth Filters

Lowpass Butterworth filters are designed to have a magnitude response that is as flat as possible at low frequencies, and that decreases monotonically with increasing frequency, as shown in Figure 40.1. A lowpass Butterworth filter has a transfer function of the form shown in Math Box 40.1. Examination of this transfer function reveals that a Butterworth lowpass filter is an all-pole design that is completely specified by just two parameters—the number of poles, n, and the 3-dB cutoff frequency, ω_{3dB}. The number of poles is usually set to acheive a desired amount of attenuation, A_{SB}, at and above some specified stopband-edge frequency, ω_{SB}. This frequency and the amount of stopband attenuation are not direct specifications on the filter—they are only indirect specifications that are used to determine the minimum required number of poles using Eq. (MB 40.2).

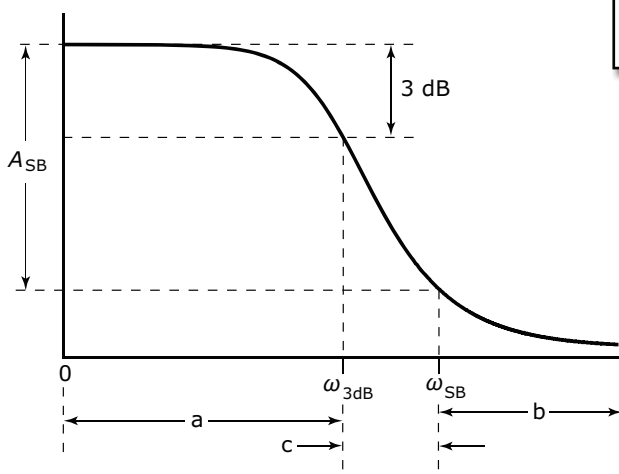


Figure 40.1 Magnitude response showing critical features and parameters used to specify a Butterworth filter: (a) passband, (b) stopband, (c) transition band

Math Box 40.1

Characteristics of Lowpass Butterworth Filters

Transfer Function

$$H(s) = \frac{\omega_{3dB}^n}{\prod_{k=1}^{n}(s - \omega_{3db} p_k)} \qquad \text{(MB 40.1)}$$

where

$$p_k = \exp\left(j\pi \frac{2k+n-1}{2n}\right)$$
$$= \cos\left(\pi \frac{2k+n-1}{2n}\right) + j\sin\left(\pi \frac{2k+n-1}{2n}\right)$$

Minimum Order

$$n = \frac{\log\left(10^{-A_{SB}/10} - 1\right)}{2\log\left(\omega_{SB} / \omega_{3dB}\right)} \qquad \text{(MB 40 2)}$$

where

A_{SB} = minimum stopband attenuation

ω_{3dB} = frequency at which passband response is 3 dB below peak

ω_{SB} = frequency that defines the start of the stopband

The poles for a Butterworth LPF always lie at equally spaced points on the left half of the unit circle in the s plane, as shown in Figure 40.2. Odd-order filters have one real pole at $s=-1$, and all remaining poles occur in complex-conjugate pairs. The poles for an even-order filter all occur in complex-conjugate pairs. If the denominator factors corresponding to complex-conjugate pairs in Eq. (MB 40.1) are multiplied together, the transfer function's denominator can be expressed as a product of quadratic terms having all real coefficients:

$$H(s)=\begin{cases}\dfrac{\omega_{3\text{dB}}^{n}}{\prod_{k=1}^{n/2}(s^2+\omega_{3\text{db}}b_k s+\omega_{3\text{db}}^2)} & n \text{ even}\\[2ex] \dfrac{\omega_{3\text{dB}}^{n}}{(s+1)\prod_{k=1}^{(n-1)/2}(s^2+\omega_{3\text{db}}b_k s+\omega_{3\text{db}}^2)} & n \text{ odd}\end{cases} \tag{40.1}$$

where

$$b_k=-2\sigma_k=-2\text{Re}\{p_k\}$$

40.1 Frequency Response

Figure 40.3 shows the magnitude, phase, and group delay responses for lowpass Butterworth filters of orders 1 through 6.

40.2 Prototype Considerations

Bilinear transformation (see Note 51) distorts, or "warps," the frequency axis in mapping an analog filter into the corrresponding IIR digital filter. Typically, critical frequencies used to specify the filter's performance are "pre-warped" to compensate for the warping that occurs in the bilinear transformation. As indicated in Design Procedure 40.1, if a Butterworth filter is to be used as a prototype for the bilinear transformation, it is important to do the prewarping of $\omega_{3\text{dB}}$ and ω_{SB} *before* these frequencies are used to determine the necessary filter order.

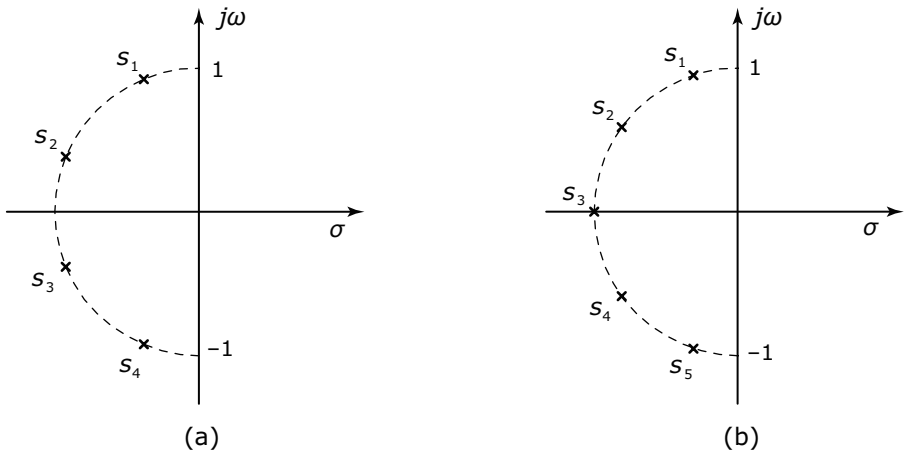


Figure 40.2 Pole locations for lowpass Butterworth filters: (a) fourth-order and (b) fifth-order

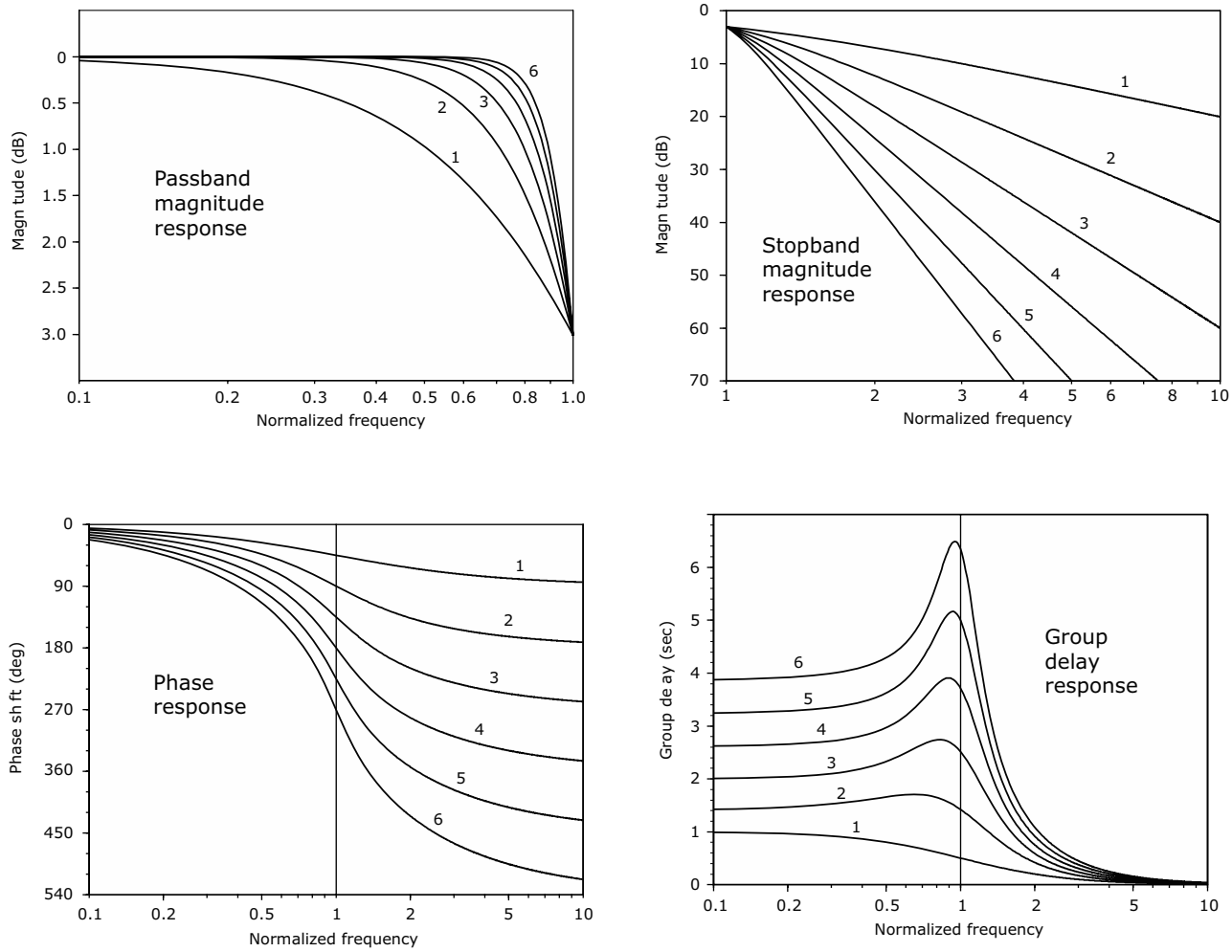


Figure 40.3 Frequency response plots for lowpass Butterworth filters of orders 1 through 6

Example 40.1

Butterworth Prototype

Use Design Procedure 40.1 to design a Butterworth prototype filter that will be used in the bilinear transformation to design an IIR filter that meets the following specifications:

- The 3-dB cutoff frequency is 3 kHz.
- The sampling rate is 30,000 samples per second.
- At least 30 dB of attenuation is at frequencies at or above 6 kHz.

The prewarped cutoff frequency for the prototype is obtained as

$$f_{3\text{dB}} = \frac{3\times10^4}{\pi}\tan\frac{3\times10^3\pi}{3\times10^4} = 3102.75 \text{ Hz}$$

$$\omega_{3\text{dB}} = 2\pi f_{3\text{dB}} = 19,495.18 \text{ rad/sec.}$$

The prewarped stopband frequency is obtained as

$$\omega_{\text{SB}} = \frac{2}{T}\tan\frac{\omega'_{\text{SB}}}{2} = \frac{2}{(30,000)^{-1}}\tan\frac{12,000\pi}{60,000} = 43592.55 \text{ rad/sec.}$$

The minimum number of poles is obtained as

$$n = \left\lceil \frac{\log\left(10^{-A_{\text{SB}}/10}-1\right)}{2\log\left(\omega_{\text{SB}}/\omega_{3\text{dB}}\right)} \right\rceil = \left\lceil \frac{\log\left(10^{30/10}-1\right)}{2\log\left(\frac{43,592.55}{19,495.18}\right)} \right\rceil = 5$$

Finally, the protoype's transfer function is given by

$$H_a(s) = \frac{\omega_{3\text{dB}}^5}{\prod_{k=1}^{5}\left(s-\omega_{3\text{dB}}p_k\right)}$$

where

$$p_1 = \cos\frac{3\pi}{5} + j\sin\frac{3\pi}{5} = -0.309017 + 0.951057j$$

$$p_2 = \cos\frac{4\pi}{5} + j\sin\frac{4\pi}{5} = -0.809017 + 0.587785j$$

$$p_3 = \cos\pi + j\sin\pi = -1$$

$$p_4 = \cos\frac{6\pi}{5} + j\sin\frac{6\pi}{5} = \cos\frac{4\pi}{5} - j\sin\frac{4\pi}{5} = -0.809017 - 0.587785j$$

$$p_5 = \cos\frac{7\pi}{5} + j\sin\frac{7\pi}{5} = \cos\frac{3\pi}{5} - j\sin\frac{3\pi}{5} = -0.309017 - 0.951057j$$

Design Procedure 40.1

Designing a Butterworth Prototype

1. Based on the requirements of the intended application, determine desired values for the following critical performance parameters to be exhibited by the final IIR design:

   ω'_{3dB} = 3-dB cutoff frequency

   ω'_{SB} = stopband-edge frequency

   A_{SB} = maximum stopband level in dB

2. Determine the prewarped frequencies, ω_{3dB}, and ω_{SB}, corresponding to the desired final frequencies, ω'_{3dB} and ω'_{SB}. If the prototype is being designed for use in the bilinear transformation, the prewarped frequencies are given by

$$\omega_{3dB} = \frac{2}{T}\tan\frac{\omega'_{3dB}T}{2}$$

$$\omega_{SB} = \frac{2}{T}\tan\frac{\omega'_{SB}T}{2}$$

   where

$$\omega_{3dB} = \omega'_{3dB}$$

$$\omega_{SB} = \omega'_{SB}$$

3. Determine the minimum filter order as

$$n = \left\lceil \frac{\log\left(10^{-A_{SB}/10} - 1\right)}{2\log\left(\dfrac{\omega_{SB}}{\omega_{3dB}}\right)} \right\rceil$$

4. Generate the transfer function for the analog prototype filter as

$$H(s) = \frac{\omega_{3db}^{n}}{\prod_{k=1}^{n}\left(s - \omega_{3db}p_k\right)}$$

   where

$$p_k = \exp\left(j\pi\frac{2k+n-1}{2n}\right)$$

$$= \cos\left(\pi\frac{2k+n-1}{2n}\right) + j\sin\left(\pi\frac{2k+n-1}{2n}\right)$$

Chebyshev Filters

Chebyshev filters comprise one of the analog filter families that are commonly used as prototypes for IIR digital filters designed using either the *impulse invariance method* (Note 50) or the *bilinear transformation* (Note 51).

A lowpass Chebyshev filter is obtained as an equiripple approximation to the response of an ideal lowpass filter. This approximation yields a filter having a squared magnitude response given by

$$\left|H(j\omega)\right|^2 = \frac{1}{1+\epsilon^2 T_n^2(\omega)}$$

where

$\epsilon^2 = 10^{r/20} - 1$

r = passband ripple

$T_n(\omega)$ = Chebyshev polynomial of order n

The first eleven Chebyshev polynomials are listed in Table 41.1. The table can be extended using the recurrence relation:

$$T_{n+1}(\omega) = 2\omega T_n(\omega) - T_{n-1}(\omega)$$

Table 41.1 Chebyshev Polynomials

n	$T_n(\omega)$
0	1
1	ω
2	$2\omega^2 - 1$
3	$4\omega^3 - 3\omega$
4	$8\omega^4 - 8\omega^2 + 1$
5	$16\omega^5 - 20\omega^3 + 5\omega$
6	$32\omega^6 - 48\omega^4 + 18\omega^2 - 1$
7	$64\omega^7 - 112\omega^5 + 56\omega^3 - 7\omega$
8	$128\omega^8 - 256\omega^6 + 160\omega^4 - 32\omega^2 + 1$
9	$256\omega^9 - 576\omega^7 + 432\omega^5 - 120\omega^3 + 9\omega$
10	$512\omega^{10} - 128\omega^8 + 1120\omega^6 - 400\omega^4 + 50\omega^2 - 1$

Math Box 41.1

Chebyshev Transfer Function

The transfer function for an nth order Chebyshev LPF normalized for a ripple bandwidth equal to 1 is given by

$$H(s) = \frac{H_0}{\prod_{k=1}^{n}(s - p_k)} \qquad \text{(MB 41.1)}$$

where

$$H_0 = \begin{cases} \prod_{k=1}^{n}(-p_k) & n \text{ odd} \\ 10^{r/20}\prod_{k=1}^{n}(-p_k) & n \text{ even} \end{cases}$$

$$p_k = \sigma_k + j\,\omega_k$$

$$\sigma_k = \left[\frac{(1/\gamma) - \gamma}{2}\right]\sin\frac{(2k-1)\pi}{2n}$$

$$\omega_k = \left[\frac{(1/\gamma) - \gamma}{2}\right]\cos\frac{(2k-1)\pi}{2n}$$

$$\gamma = \left(\frac{1+\sqrt{1+\epsilon^2}}{\epsilon}\right)^{1/n}$$

$$\epsilon = \sqrt{10^{r/10} - 1}$$

Figure 41.1 shows the important features in the magnitude response for a Chebyshev lowpass filter. The response is most often presented in a form where the response is normalized to have a ripple bandwidth, ω_r, equal to 1, because this form involves simpler calculations. However, comparison with other filter families is often easier when the magnitude response is normalized to have the 3-dB frequency ω_{3dB} equal to 1. Design Procedure 41.1 can be used to renormalize a filter design from $\omega_r = 1$ to $\omega_{3dB} = 1$. The transfer function for an nth order Chebyshev LPF normalized for $\omega_r = 1$ is given in Math Box 41.1.

There are several different equations that can be used to estimate the minimum number of poles that are required for a Chebyshev lowpass filter to achieve a desired set of specifications. Two of the most commonly cited equations are from Rabiner and Gold [1]:

$$n = \frac{\log\left(g + \sqrt{g^2 - 1}\right)}{\log\left(\omega_{SB} + \sqrt{\omega_{SB} - 1}\right)} \qquad (41.1)$$

Design Procedure 41.1

Renormalizing Chebyshev LPF Transfer Functions

Assuming that values of ε, H_0, and the pole values p_k have been obtained for a Chebyshev filter having a ripple bandwidth normalized to 1, this filter can be renormalized for a 3-dB bandwidth equal to 1 by performing the following steps.

1. Compute A using

$$A = \frac{\cosh^{-1}(1/\varepsilon)}{n} = \frac{1}{n}\ln\left(\frac{1+\sqrt{1-\varepsilon^2}}{\varepsilon}\right)$$

2. Using the values of A obtained in step 1, compute R as

$$R = \cosh A = \frac{e^A + e^{-A}}{2}$$

3. Use R to compute $H_{3dB}(s)$ as

$$H_{3dB}(s) = \frac{H_0/R}{\prod_{k=1}^{n}(s - p_k/R)}$$

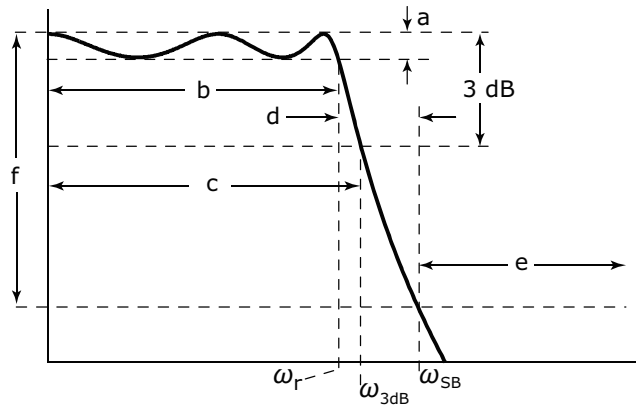


Figure 41.1 Magnitude response of a fifth-order Chebyshev filter. Features are: (a) ripple limits, (b) ripple bandwidth, (c) 3-dB bandwidth, (d) transition band, (e) stopband, and (f) stopband attenuation.

where

$$g = \sqrt{\frac{A^2 - 1}{\varepsilon^2}}$$

$$A = 10^{A_{SB}/20}$$

and from Van Valkenberg [2]:

$$n = \left\lceil \frac{\cosh^{-1}\left[\sqrt{10^{A_{SB}/10} - 1} \Big/ \sqrt{10^{A_{PB}/10} - 1}\right]}{\cosh^{-1}(\omega_{SB}/\omega_{PB})} \right\rceil \quad (41.2)$$

These two equations provide similar, but not identical, results. It's worth noting that the MATLAB help file for the function `cheb1ord()` cites Rabiner and Gold, but Eq. (41.2) is used for the actual implementation contained in `cheb1ord.m`.

Figure 41.2 shows the magnitude and phase responses for lowpass Chebyshev filters of orders 1 through 6.

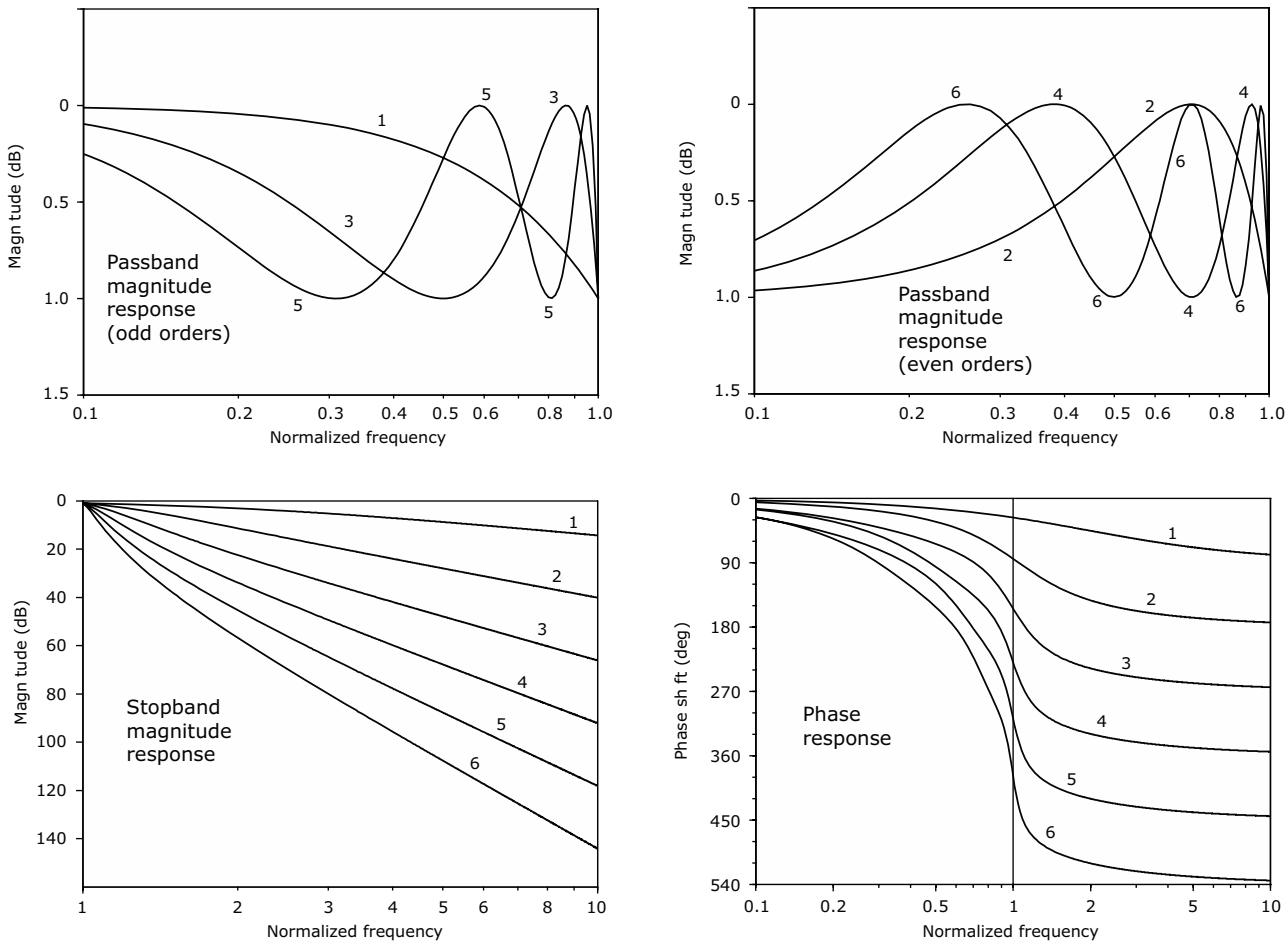


Figure 41.2 Frequency response plots for lowpass Chebyshev filters of orders 1 through 6

Design Procedure 41.2

Designing a Chebyshev Prototype

1. Based on the requirements of the intended application, determine desired values for the following critical performance parameters to be exhibited by the final IIR design:

   ω'_{PB} = passband edge frequency based on ripple bandwidth

   ω'_{SB} = stopband edge frequency

   A_{SB} = minimum stopband attenuation

   ε = passband ripple parameter

2. Determine the prewarped frequencies, ω_{3dB} and ω_{SB}, corresponding to the desired final frequencies, ω'_{3dB} and ω'_{SB}. If the prototype is being designed for use in the bilinear transformation, the prewarped frequencies are given by

$$\omega_{PB} = \frac{2}{T}\tan\frac{\omega'_{PB}T}{2}$$

$$\omega_{SB} = \frac{2}{T}\tan\frac{\omega'_{SB}T}{2}$$

   where T is the sampling interval for the IIR filter. If the prototype is being designed for use with the impulse invariance technique, there is no prewarping, so

$$\omega_{PB} = \omega'_{PB}$$

$$\omega_{SB} = \omega'_{SB}$$

3. Determine the minimum filter order using either Eq. (41.1) or Eq. (41.2).

4. Insert the value for ε from step 1, and the value for n from step 3 into Eq. (MB 41.1) to obtain the desired transfer function.

References

1. L. R. Rabiner and B. Gold, *Theory and Applications of Digital Signal Processing*, Prentice Hall, 1975.
2. M. E. Van Valkenberg, *Analog Filter Design*, Oxford University Press, 1982.
3. A. Antoniou, *Digital Filters: Analysis and Design*, McGraw-Hill, 1979.

Elliptic Filters

Elliptic filters have ripples in both the passband and the stopband, and generally have better selectivity than Chebyshev filters of comparable orders. Determining the transfer function is somewhat more complicated for an elliptical filter than it is for Butterworth or Chebyshev filters. The process begins with the specification of four parameters, which are depicted in Figure 42.1.

A_p = maximum passband loss, dB
A_s = minimum stopband loss, dB
ω_p = passband cutoff frequency
ω_s = stopband cutoff frequency

The minimum filter order needed to achieve the performance specified by these parameters is determined using Design Procedure 42.1. Once the order, n, is determined, the actual minimum stopband loss can be computed as

$$A_s = 10\log\left(1+\frac{10^{A_p/10}-1}{16q^n}\right) \qquad (42.1)$$

where q is the modular constant computed in step 4 of Design Procedure 42.1.

As with other filter types, the design of an elliptical filter starts with a frequency-normalized prototype. However, because the passband edge frequency and stopband edge frequency can be specified independently, the normalization for an elliptical filter must be based on something other than 3-dB bandwidth. Specifically, the prototype is normalized such that

$$\sqrt{\Omega_p\Omega_s} = 1 \qquad (42.2)$$

where Ω_p and Ω_s are the normalized passband edge frequency and the normalized stopband edge frequency, respectively. These frequencies are related to the corresponding desired frequencies, ω_p and ω_s, by a frequency-scaling factor, α, such that

$$\Omega_p = \frac{\omega_p}{\alpha} \qquad \Omega_s = \frac{\omega_s}{\alpha} \qquad (42.3)$$

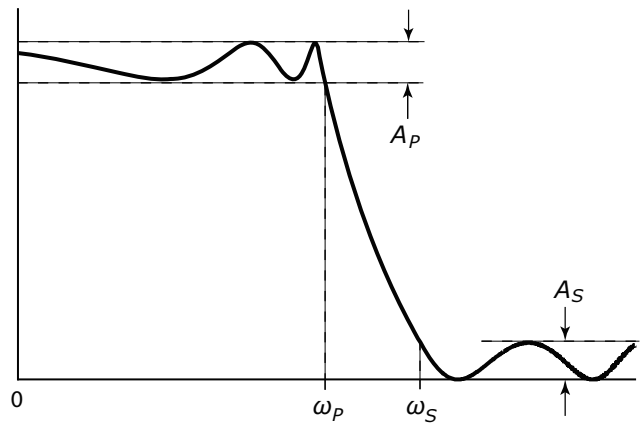


Figure 42.1 Magnitude response showing parameters used to specify an elliptical filter.

Design Procedure 42.1

Determining the Minimum Required Order for Elliptic Filters

1. Based on the requirements of the intended application, determine desired values for the following critical performance parameters to be exhibited by the final IIR design:

   ω'_P = passband edge frequency

   ω'_S = stopband edge frequency

   A_S = minimum stopband loss, dB

   A_P = maximum passband loss, dB

2. Determine the prewarped frequencies, ω_P and ω_S, corresponding to the desired final frequencies, ω'_P and ω'_S. If the prototype is being designed for use in the bilinear transformation, the prewarped frequencies are given by

$$\omega_P = \frac{2}{T}\tan\frac{\omega'_P T}{2}$$
$$\omega_S = \frac{2}{T}\tan\frac{\omega'_S T}{2}$$

   where T is the sampling interval for the IIR filter. If the prototype is being designed for use with the impulse invariance technique, there is no prewarping, so

$$\omega_P = \omega'_P$$
$$\omega_S = \omega'_S$$

3. Using ω_P and ω_S, compute the selectivity factor, k, as

$$k = \omega_P / \omega_S$$

4. Using the selectivity factor computed in step 3, compute the modular constant, q, using

$$q = u + 2u^5 + 15u^9 + 150u^{13}$$

   where

$$u = \frac{1 - \sqrt[4]{1-k^2}}{2\left(1 + \sqrt[4]{1-k^2}\right)}$$

5. Using the values of A_P and A_S determined in step 1, compute the discrimination factor, D, as

$$D = \frac{10^{A_S/10} - 1}{10^{A_P/10} - 1}$$

6. Using the value of D from step 5 and the value of q from step 4, compute the minimum required order, n, as

$$n = \left\lceil \frac{\log(16D)}{\log(1/q)} \right\rceil$$

We can solve for the required value of α by substituting Eq. (42.3) into Eq. (42.2) to obtain

$$\sqrt{\frac{\omega_p \omega_s}{\alpha^2}} = 1$$
$$\alpha = \sqrt{\omega_p \omega_s}$$

After the parameters k, q, and u are determined using Design Procedure 42.1, the corresponding transfer function can be determined using Design Procedure 42.2. If a specification in terms of pole and zero locations is desired, the poles can be obtained by solving for the roots of

$$s^2 + b_i s + c_i = 0 \qquad (42.4)$$

and the zeros can be obtained by solving for the roots of

$$s^2 + a_i = 0$$

Design Procedure 42.2

Designing an Elliptic Prototype Filter

1. Use Design Procedure 42.1 to determine a viable combination of values for A_p, A_s, ω_p, ω_s, and n.
2. Using the values of A_p and n from step 1, compute V as
$$V = \frac{1}{2n}\ln\left(\frac{10^{A_p/20}+1}{10^{A_p/20}-1}\right)$$
3. Using the value of V from step 2, above, and the value of q from step 4 of Design Procedure 42.1, compute p_0 as
$$p_0 = \left|\frac{q^{1/4}\sum_{m=0}^{6}(-1)^m q^{m(m+1)}\sinh[(2m+1)V]}{0.5+\sum_{m=0}^{6}(-1)^m q^{m^2}\cosh(2mV)}\right|$$
4. Using the value of $k=\omega_p/\omega_s$ and the value of p_0 from step 3, compute W as
$$W = \left[\left(1+\frac{p_0^2}{k}\right)\left(1+kp_0^2\right)\right]^{1/2}$$
5. Determine the number of quadratic sections in the filter as
$$r = \begin{cases} n/2 & n \text{ even} \\ (n-1)/2 & n \text{ odd} \end{cases}$$
6. For $i=1,2,\ldots,r$, compute X_i as
$$X_i = \frac{2q^{1/4}\sum_{m=0}^{6}(-1)^m q^{m(m+1)}\sin[(2m+1)\mu\pi/n]}{1+2\sum_{m=0}^{6}(-1)^m q^{m^2}\cos(2m\mu\pi/n)}$$
where
$$\mu = \begin{cases} i & n \text{ odd} \\ i-\frac{1}{2} & n \text{ even} \end{cases}$$
7. For $i=1,2,\ldots,r$, compute Y_i as
$$Y_i = \left[\left(1-\frac{X_i^2}{k}\right)\left(1-kX_i^2\right)\right]^{1/2}$$
8. For $i=1,2,\ldots,r$, use the W, X_i, and Y_i from steps 4, 6, and 7 to compute the coefficients a_i, b_i, and c_i as
$$a_i = \frac{1}{X_i^2}$$
$$b_i = \frac{2p_0Y_i}{1+(p_0X_i)^2}$$
$$c_i = \frac{(p_0Y_i)^2+(X_iW)^2}{\left[1+(p_0X_i)^2\right]^2}$$
9. Using a_i and c_i, compute H_0 as
$$H_0 = \begin{cases} p_0\prod_{i=1}^{r}\frac{c_i}{a_i} & n \text{ odd} \\ 10^{-A_p/20}\prod_{i=1}^{r}\frac{c_i}{a_i} & n \text{ even} \end{cases}$$
10. Compute the normalized transfer function as
$$H_N(s) = \frac{H_0}{d}\prod_{i=1}^{r}\frac{s^2+a_i}{s^2+b_is+c_i}$$
where
$$d = \begin{cases} s+p_0 & n \text{ odd} \\ 1 & n \text{ even} \end{cases}$$

References

1. M. E. Van Valkenberg, *Analog Filter Design*, Oxford University Press, 1982.
2. A. Antoniou, *Digital Filters: Analysis and Design*, McGraw-Hill, 1979.

Bessel Filters

Bessel filters are designed to have maximally flat group-delay characteristics. Consequently, there are no oscillations in the step response, thus making Bessel filters attractive for applications that call for low levels of pulse distortion. The transfer function of an nth-order lowpass Bessel filter is given by

$$H(s) = \frac{b_0}{q_n(s)} \tag{43.1}$$

where $q_n(s)$ is the polynomial of degree n given by

$$q_n(s) = \sum_{k=1}^{n} b_k s^k$$

$$b_k = \frac{(2n-k)!}{2^{n-k}k!(n-k)!}$$

The polynomial $q_n(s)$ can be obtained from $q_{n-1}(s)$ and $q_{n-2}(s)$ using the recursion

$$q_n(s) = (2n-1)q_{n-1} + s^2 q_{n-2}$$

Table 43.1 lists $q_n(s)$ for $n = 2$ through $n = 8$.

Equation (43.1) does not provide an explicit expression for the poles of a Bessel filter. The pole locations are found by using numerical methods to solve for the roots of $q_n(s) = 0$.

The filters defined by Eq. (43.1) are normalized to have unit delay at $\omega = 1$. The poles, p_k, and denominator coefficients, b_k, can be renormalized for a 3-dB frequency of $\omega = 1$ using

$$p_k' = Ap_k \qquad b_k' = A^{n-k}b_k$$

where the value of A is obtained from Table 43.2. Odd-order filters have one real pole, and all remaining poles occur in complex-conjugate pairs. The poles for an even-order filter all occur in complex-conjugate pairs. If the denominator factors corresponding to complex-conjugate pairs are multiplied together, the denominator can be expressed as a product of quadratic terms having all real coefficients.

43.1 Frequency Response

Figure 43.1 shows the magnitude, phase, and group-delay responses for lowpass Bessel filters of orders 2 through 6.

Table 43.1 Denominator Polynomials for Transfer Functions of Bessel Filters Normalized to Have Unit Delay at $\omega = 0$

n	$q_n(s)$
2	$s^2 + 3s + 3$
3	$s^3 + 6s^2 + 15s + 15$
4	$s^4 + 10s^3 + 45s^2 + 105s + 105$
5	$s^5 + 15s^4 + 105s^3 + 420s^2 + 945s + 945$
6	$s^6 + 21s^5 + 210s^4 + 1260s^3 + 4725s^2 + 10{,}395s + 10{,}395$
7	$s^7 + 28s^6 + 378s^5 + 3150s^4 + 17{,}325s^3 + 62{,}370s^2 + 135{,}135s + 135{,}135$
8	$s^8 + 36s^7 + 630s^6 + 6930s^5 + 9450s^4 + 270{,}270s^3 + 945{,}945s^2 + 2{,}027{,}025s + 2{,}027{,}025$

Table 43.2 Factors for Renormalizing Bessel Filter Poles from Unit Delay at $\omega = 0$ to 3-dB Attenuation at $\omega = 1$

n	A
2	1.35994
3	1.74993
4	2.13011
5	2.42003
6	2.69996
7	2.95000
8	3.17002

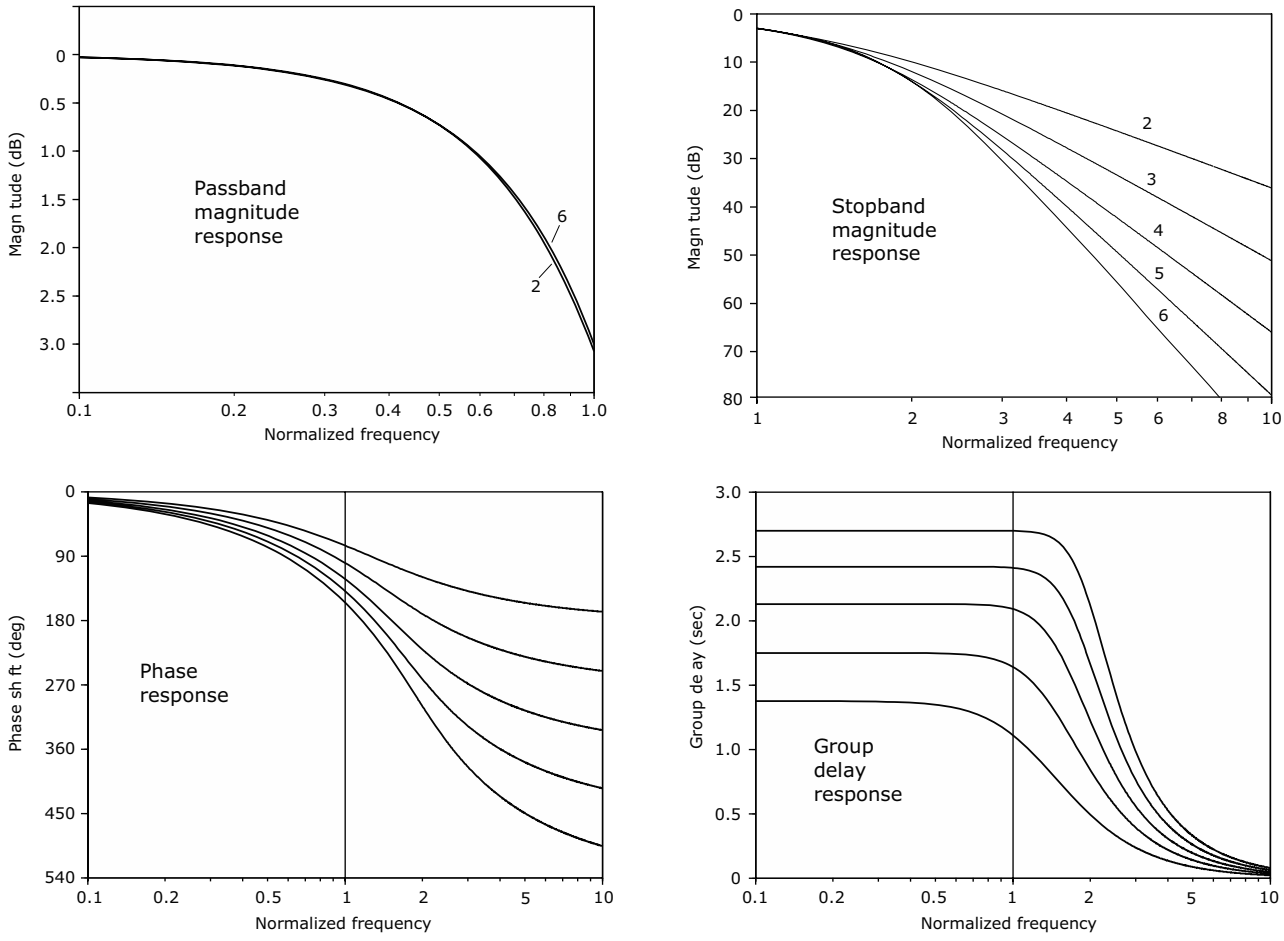


Figure 43.1 Frequency response plots for lowpass Bessel filters of orders 2 through 6

The *z* Transform

The *z transform* is a mathematical tool that plays a role in the analysis of discrete-time systems similar to the role played by the Laplace transform in the analysis of lumped-parameter continuous-time systems. It has a place in the theoretical exploration of all linear shift-invariant discrete-time systems, but in practice, the *z* transform (and its relationship to the Laplace transform) is most commonly used in the design of IIR filters. In fact, some authors, such as Lyons [1], discuss the *z* transform only in conjunction with IIR filter design.

Sometimes, pole and zero locations (which are features obtained from the *z* transform) are used as visualization aids to provide insight into the behavior of adaptive filters as their coefficients evolve over time or converge to different values as a function of SNR. An example of such use is provided by Kay in [2].

The *z* transform can be defined in either a one-sided (unilateral) or two-sided (bilateral) form:

$$X(z) = \sum_{n=0}^{\infty} x[n] z^{-n} \quad \text{(one-sided)} \tag{44.1}$$

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n} \quad \text{(two-sided)} \tag{44.2}$$

where z is a continuous complex-valued variable. Different authors follow different conventions with respect to which form they consider to be the default when "one-sided" or "two-sided" is not specifically called out. The two-sided transform is the form most often used in DSP applications. When $x[n]$ is a causal sequence, the two forms are equivalent.

44.1 Region of Convergence

The series represented by Eq. (44.2) does not always converge for every possible value of z. The portion of the z plane for which the series does converge

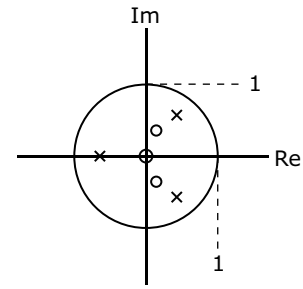
Key Concept 44.4

Role of the *z* Transform in Digital Filter Analysis

- The z transform of a digital filter's unit sample response is called the system function and is denoted $H(z)$. In general, a linear shift-invariant digital filter has a system function that is a ratio of polynomials in z:

$$H(z) = P(z)/Q(z)$$

- Roots of these polynomials are often plotted in the complex z plane to help visualize various aspects of the filter's behavior. Roots of the numerator are called *zeros*, and are typically plotted using small circles (○). Roots of the denominator are called *poles*, and are typically plotted using small multiplication signs (×).



is called the *region of convergence* (ROC). Whether or not the series converges depends upon the magnitude of z rather than on the specific complex value of z. Therefore, ROC boundaries are always defined by circles centered at the origin of the z plane. There are four major configurations for the ROC, as depicted in Figure 44.1. Depending upon whether $z=0$ and $|z|=\infty$ are included in the ROC, there are a total of nine specific variations on the four basic configurations:

- entire z plane
- entire z plane except for $z=0$
- entire z plane except for $|z|=\infty$
- entire z plane except for $z=0$ and $|z|=\infty$
- interior of circle
- interior of circle except for $z=0$
- exterior of circle
- exterior of circle except for $|z|=\infty$
- annulus

As summarized in Table 44.1, the ROC for any particular z transform depends upon the characteristics of the sequence $x[n]$.

Table 44.1 ROC configurations corresponding to various constraints placed on the sequence $x[n]$

$x[n]$	ROC for $X(z)$
Single sample at $n=0$	Entire z plane
Finite-duration, causal, $x[n]=0$ for all $n<0$, $x[n]\neq 0$ for some $n>0$	z plane except for $z=0$
Finite-duration, $x[n]\neq 0$ for some $n<0$, $x[n]=0$ for all $n>0$	z plane except for $\|z\|=\infty$
Finite-duration, $x[n]\neq 0$ for some $n<0$, $x[n]\neq 0$ for some $n>0$	z plane except for $z=0$ and $\|z\|=\infty$
Right-sided, $x[n]=0$ for all $n<0$	Outward from outermost pole
Right-sided, $x[n]\neq 0$ for some $n<0$	Outward from outermost pole, $\|z\|=\infty$ is excluded
Left-sided, $x[n]=0$ for all n >0	Inward from innermost pole
Left-sided, $x[n]\neq 0$ for some n >0	Inward from innermost pole, $z=0$ is excluded
Two-sided	Annulus

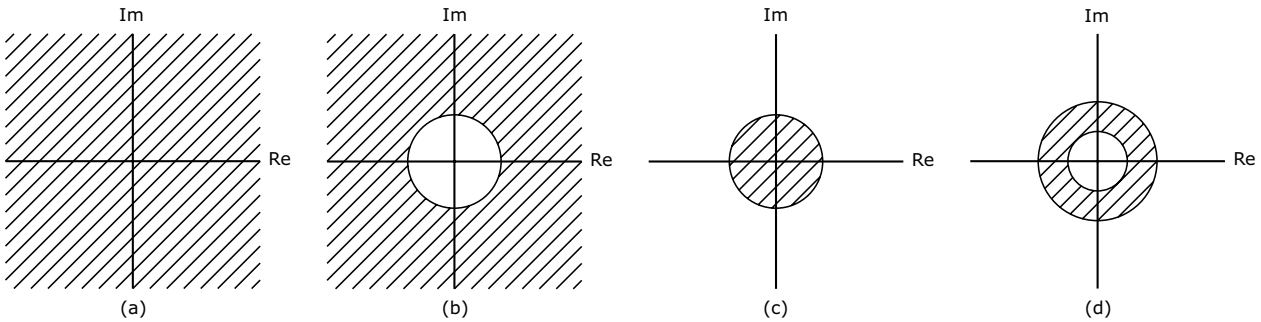


Figure 44.1 Possible configurations for region of covergence: (a) the entire z-plane, (b) exterior of a circle, (c) interior of a circle, and (d) an annulus

Table 44.2 Bilateral *z*-Transform Pairs

$x[n]$	$X(z)$	ROC
$\delta[n]$	1	All z
$\delta[n-m],\ m>0$	z^{-m}	$z\neq 0$
$\delta[n-m],\ m<0$	z^{-m}	$\lvert z\rvert\neq\infty$
$u[n]$	$\frac{z}{z-1}$	$\lvert z\rvert>1$
$-u[-n-1]$	$\frac{z}{z-1}$	$\lvert z\rvert<1$
$-a^n u[-n-1]$	$\frac{z}{z-a}$	$\lvert z\rvert>\lvert a\rvert$
$-na^n u[-n-1]$	$\frac{az}{(z-a)^2}$	$\lvert z\rvert<\lvert a\rvert$

Table 44.3 Unilateral *z*-Transform Pairs

$x[n]$	$X(z)$	ROC
1	$\frac{z}{z-1}$	$\lvert z\rvert>1$
$u[n]$	$\frac{z}{z-1}$	$\lvert z\rvert>1$
$\delta[n]$	1	Entire z plane
a^n	$\frac{z}{z-a}$	$\lvert z\rvert>\lvert a\rvert$
na^n	$\frac{az}{(z-a)^2}$	$\lvert z\rvert>\lvert a\rvert$
n^2a^n	$\frac{az(z+a)}{(z-a)^3}$	$\lvert z\rvert>\lvert a\rvert$
n^3a^n	$\frac{az(z^2+4az+a^2)}{(z-a)^4}$	$\lvert z\rvert>\lvert a\rvert$
$(n+1)a^n$	$\frac{z^2}{(z-a)^2}$	$\lvert z\rvert>\lvert a\rvert$
$\frac{(n+1)(n+2)}{2!}a^n$	$\frac{z^3}{(z-a)^3}$	$\lvert z\rvert>\lvert a\rvert$
$\frac{(n+1)(n+2)(n+3)}{3!}a^n$	$\frac{z^4}{(z-a)^4}$	$\lvert z\rvert>\lvert a\rvert$
$\frac{(n+1)(n+2)(n+3)(n+4)}{4!}a^n$	$\frac{z^5}{(z-a)^5}$	$\lvert z\rvert>\lvert a\rvert$
nT	$\frac{Tz}{(z-1)^2}$	$\lvert z\rvert>1$
$(nT)^2$	$\frac{T^2z(z+1)}{(z-1)^3}$	$\lvert z\rvert>1$
$(nT)^3$	$\frac{T^3z(z^2+4z+1)}{(z-1)^4}$	$\lvert z\rvert>1$
$\frac{a^n}{n!}$	$e^{a/z}$	0
e^{-anT}	$\frac{z}{z-e^{-aT}}$	$\lvert z\rvert>\lvert e^{-aT}\rvert$
$e^{-anT}\sin\omega_0 nT$	$\frac{ze^{-aT}\sin\omega_0 T}{z^2-2ze^{-aT}\cos\omega_0 T+e^{-2aT}}$	$\lvert z\rvert>\lvert e^{-aT}\rvert$
$e^{-anT}\cos\omega_0 nT$	$\frac{z^2-ze^{-aT}\cos\omega_0 T}{z^2-2ze^{-aT}\cos\omega_0 T+e^{-2aT}}$	$\lvert z\rvert>\lvert e^{-aT}\rvert$

Table 44.4 Properties of the z Transform

Property	Time Sequence	Transform	ROC
	$x[n]$	$X(z)$	R_x
	$y[n]$	$Y(z)$	R_y
Homogeneity	$ax[n]$	$aX(z)$	R_x
Additivity	$x[n] + y[n]$	$X(z) + Y(z)$	Contains $r_x \cap r_y$, but may be larger
Linearity	$ax[n] + by[n]$	$aX(z) + bY(z)$	Contains $r_x \cap r_y$, but may be larger
Time shift	$x[n-m]$	$z^{-m}X(z)$	R_x
Time reversal	$x[-n]$	$X(z^{-1})$	$1/R_x$
Frequency shift	$a^n x[n]$	$X\left(\frac{z}{a}\right)$	$\lvert a\rvert R_x$
Convolution	$x[n] \otimes y[n]$	$X(z)Y(z)$	Contains $R_x \cap R_y$, but may be larger
Conjugation	$x^*[n]$	$X^*(z^*)$	R_x
Differentiation of transform	$nx[n]$	$-z\frac{d}{dz}X(z)$	R_x

References

1. R. Lyons, *Understanding Digital Signal Processing*, Prentice Hall, 1996.
2. S. M. Kay, "The Effects of Noise on the Autoregressive Spectral Estimator," *IEEE Trans. Acous. Speech and Signal Proc. ASSP*, vol. 27, no. 5, Oct. 1979, pp. 478–485.

Computing the Inverse *z* Transform Using the Partial Fraction Expansion

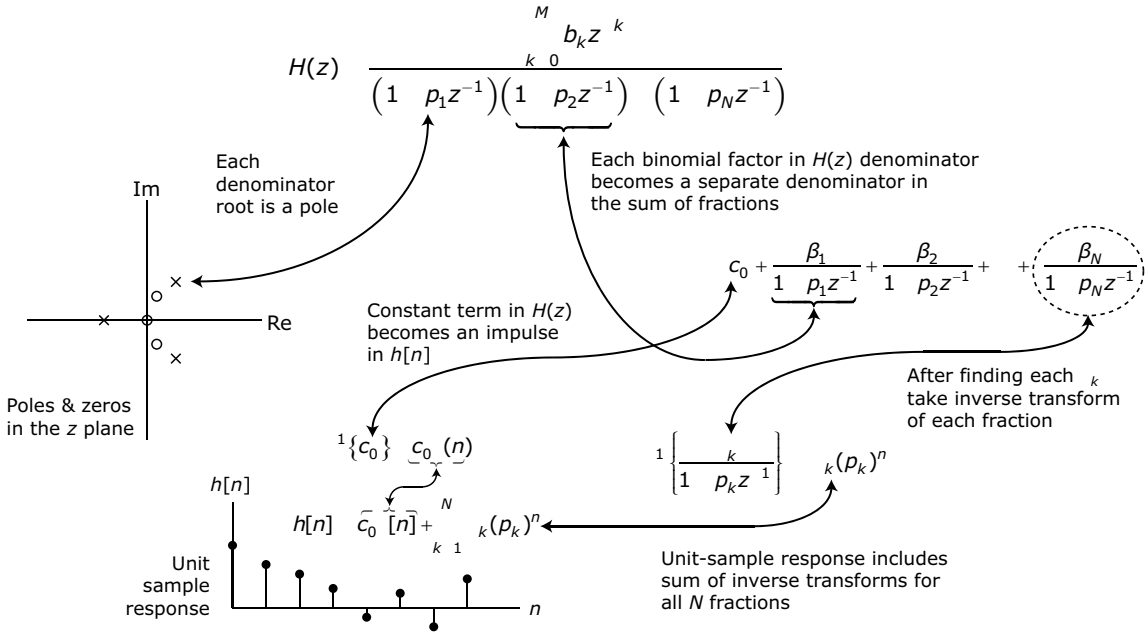
Often IIR filters are designed in the *z* domain, and in order to obtain the filter's unit sample response, it is necessary to compute the inverse *z* transform of the filter's system function. This note provides an overview of the *partial fraction expansion* (PFE) technique that can be used to compute the inverse *z* transform. Notes 46 through 48 cover the specifics of how the PFE is applied to system functions exhibiting various properties, as detailed in the "Related Notes" sidebar on the next page.

There are a handful of fundamentally different approaches for inverting the *z* transform, and each of these approaches can have several different minor variations. Most of the popular approaches are based on the idea that a function of interest in the *z* domain can be broken down into a sum of simpler constituent functions that can be easily inverted by using a table of common *z*

Key Concept 45.1

Using the Inverse *z* Transform to Compute the IIR Sample Response

Many IIR filters are designed directly in the *z* domain, and the inverse *z* transform must be used to obtain the system's unit sample response.



The sum-of-fractions expansion shown above for $H(z)$ assumes that the numerator's degree, M, equals the denominator's degree, N. When $M < N$, the c_0 term vanishes, and when $M > N$, there are additional non-fractional terms of the form $c_k z^{-k}$ for $k = 1$ through $k = M - N$.

transform pairs. Because the transform and its inverse are linear, the inverses of the constituent functions can be summed to obtain the inverse transform for the original function of interest.

For a linear shift-invariant discrete-time system, the system's z domain *system function* is the z transform of the system's unit sample response. Many IIR filters are designed in the z domain, and in order to obtain the system's unit sample response, it is necessary to compute the inverse z transform of the filter's system function. Consider the system function given by

$$H(z) = \frac{B(z)}{A(z)} \tag{45.1}$$

where $A(z)$ and $B(z)$ are polynomials in powers of z^{-1}

$$B(z) = \sum_{k=0}^{M} b_k z^{-k} \tag{45.2}$$

$$A(z) = \sum_{k=0}^{N} a_k z^{-k} \tag{45.3}$$

In discussing IIR filters, many texts show the IIR system function as

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 - \sum_{k=1}^{N} a_k z^{-k}} \tag{45.4}$$

As shown in Note 61, this is the natural form of the system function when it is derived from the filter's difference equation. However, in discussions about computing the inverse z transform, it is notationally convenient to use the denominator form defined by Eqs. (45.1) and (45.3). Despite the change, the constant term, a_0, should always equal unity for an IIR filter. As shown in Note 61, the constant term corresponds to the filter's "current" output in the difference equation where a value other than unity makes no sense. However, be aware that some texts confuse matters by offering the advice that if the constant term is some value, $a_0 \neq 1$, then all of the coefficients in the denominator should be divided by a_0 so that the constant term becomes $a_0/a_0 = 1$.

The denominator polynomial from Eq. (45.1) can be recast as a product of binomials:

$$A(z) = \prod_{k=1}^{N} \left(1 - p_k z^{-1}\right) \tag{45.5}$$

where the values p_k are known as *poles* of the system function. If none of the N pole values are repeated, then the system function is said

Related Notes

Note 46 Demonstrates use of the PFE for the case in which all poles are distinct and the system function, $H(z)$, is a proper rational function.

Note 47 Demonstrates use of the PFE for the case in which all poles are distinct and the system function, $H(z)$, is an improper rational function. Polynomial division is used to convert $H(z)$ into the sum of a polynomial, $C(z)$, and a proper rational function, $H_R(z)$.

Note 48 Demonstrates use of the PFE for the case in which all poles are distinct, and the system function, $H(z)$, is an improper rational function. The approach used is based on the assumption that $H(z)$ can be expressed as the sum of a polynomial, $C(z)$, and a proper rational function, $H_R(z)$. The form of the partial fraction expansion is modified to include the coefficients needed to expand $C(z)$.

to have simple poles, and $H(z)$ can be expressed as a sum of less complicated ratios as

$$H(z) \quad \frac{\beta_1}{1 \quad p_1 z^{\ 1}} + \frac{\beta_2}{1 \quad p_2 z^{\ 1}} + \cdots + \frac{\beta_N}{1 \quad p_N z^{\ 1}} \tag{45.6}$$

Equation (45.6) is known as the *partial fraction expansion* (PFE) for $H(z)$. The inverse z transforms for each term on the right-hand side of this PFE can be computed using

$$\mathcal{Z}^{\ 1}\left\{\frac{\beta_k}{1 \quad p_k z^{\ 1}}\right\} \quad \beta_k (p_k)^n \tag{45 7}$$

Because of a relationship between the PFE and Cauchy's residue theorem, the coefficients β_k are sometimes referred to as *residues*.

Inverse *z* Transform via Partial Fraction Expansion:

Case 1: All Poles Distinct with $M < N$ in System Function

This note covers the form of the partial fraction expansion that can be used to compute the inverse *z* transform of system functions that have no repeated poles and in which the degree of the denominator polynomial exceeds the degree of the numerator polynomial. The method is based on the basic strategy discussed in Note 45. Alternative approaches for use on system functions that do not meet these constraints are covered in Notes 47 through 49.

The form of the partial fraction expansion discussed in this note is suitable for use on system functions of the form

$$H(z)=\frac{b_0+b_1z^{-1}+b_2z^{-2}+\quad+b_Mz^{-M}}{\left(1-p_1z^{-1}\right)\left(1-p_2z^{-1}\right)\quad\left(1-p_Nz^{-1}\right)}$$

$$p_1\neq p_2\neq\quad\neq p_N \tag{46.1}$$

where $M < N$, thus making $H(z)$ a *proper rational function*.

The expansion can be formed directly by using each binomial factor from the denominator of Eq. (46.1) as a denominator for one of the fractional terms in the expansion:

$$H(z)=\frac{\beta_1}{1-p_1z^{-1}}+\frac{\beta_2}{1-p_2z^{-1}}+\quad+\frac{\beta_N}{1-p_Nz^{-1}} \tag{46.2}$$

When a common denominator is generated to perform the additions indicated in Eq. (46.2), the resulting denominator always equals the denominator in Eq. (46.1). Therefore, we can simplify the notation by just retaining the numerator that results after the common denominator has been

Design Procedure 46.1

Solving for PFE Coefficients: Coefficient Matching Approach

This procedure is appropriate for solving a partial fraction expansion of the form

$$\frac{b_0+b_1z^{-1}+b_2z^{-2}+\cdots+b_Mz^{-M}}{(z-p_1)(z-p_2)\cdots(z-p_{N-1})(z-p_N)}=\frac{\beta_1}{1-p_1z^{-1}}+\frac{\beta_2}{1-p_2z^{-1}}+\cdots+\frac{\beta_N}{1-p_Nz^{-1}} \tag{DP 46.1}$$

1. Explicitly construct the RHS of Eq. (46.3) using the known values for the coefficients, b_k, and poles, p_k.
2. Perform the necessary algebraic manipulations so that the coefficient for each power of z^{-1} on the right-hand side is obtained as a function of the unknown coefficients, β_k

$$B(z)=f_0(\beta_0,\beta_1,\ldots,\beta_N)+z^{-1}f_1(\beta_0,\beta_1,\ldots,\beta_N)+z^{-2}f_2(\beta_0,\beta_1,\ldots,\beta_N)+\cdots+z^{-M}f_M(\beta_0,\beta_1,\ldots,\beta_N) \tag{DP 46.2}$$

3. Set the coefficients from the LHS of Eq. (46.3) equal to the corresponding coefficients from the RHS of Eq. (DP 46.2):

$$\begin{aligned}b_0&=f_0(\beta_0,\beta_1,\ldots,\beta_N)\\b_1&=f_1(\beta_0,\beta_1,\ldots,\beta_N)\\&\vdots\\b_M&=f_M(\beta_0,\beta_1,\ldots,\beta_N)\end{aligned} \tag{DP 46.3}$$

4. Solve the system of linear equations (DP 46.3) for the coefficients β_k.

implemented to accomplish the additions. This "numerators-only" equation has the form

$$\begin{aligned} B(z) &= b_0 + b_1 z^{-1} + b_2 z^{-1} + \cdots + b_M z^{-M} \\ &= \beta_1 \prod_{k=2}^{N} \left(1 - p_k z^{-1}\right) + \beta_2 \prod_{\substack{k=1 \\ k \neq 2}}^{N} \left(1 - p_k z^{-1}\right) \\ &\quad + \quad + \beta_n \prod_{\substack{k=1 \\ k \neq n}}^{N} \left(1 - p_k z^{-1}\right) \\ &\quad + \quad + \beta_N \prod_{k=1}^{N-1} \left(1 - p_k z^{-1}\right) \end{aligned} \tag{46.3}$$

Typically, the values for all the poles, p_k, are known, so straightforward algebraic manipulations can be used to put the right-hand side of Eq. (46.3) into a form in which the coefficient for each power of z^{-1} is obtained as a function of the unknown coefficients, β_{nk}:

$$\begin{aligned} B(z) &= f_0\left(\beta_0, \beta_1, \ldots, \beta_N\right) + z^{-1} f_1\left(\beta_0, \beta_1, \ldots, \beta_N\right) \\ &\quad + z^{-2} f_2\left(\beta_0, \beta_1, \ldots, \beta_N\right) + \cdots \\ &\quad + z^{-M} f_M\left(\beta_0, \beta_1, \ldots, \beta_N\right) \end{aligned} \tag{46.4}$$

Coefficients for like powers of z must have the same value in the LHS of Eq. (46.3) and in the RHS of Eq.(46.4) so we can construct the system of equations

$$\begin{aligned} b_0 &= f_0\left(\beta_0, \beta_1, \ldots, \beta_N\right) \\ b_1 &= f_1\left(\beta_0, \beta_1, \ldots, \beta_N\right) \\ &\vdots \\ b_M &= f_M\left(\beta_0, \beta_1, \ldots, \beta_N\right) \end{aligned} \tag{46.5}$$

From the form of the RHS of Eq. (46.3), we know that each function, $f_m(\beta_0, \beta_1, \ldots, \beta_N)$, is a linear function of the coefficients, β_n, so it is a straightforward process to solve this system of equations for the unknown values of β_n.

Once the values for the β_n are found, the filter's unit sample response can be written as

$$h[n] = \beta_1 (p_1)^n + \beta_2 (p_2)^n + \cdots + \beta_N (p_N)^n \tag{46.6}$$

This approach for computing the inverse z transform is summarized in Design Procedure 46.1, and is demonstrated in Example 46.1.

Example 46.1

Coefficient Matching Approach When $H(z)$ Is a Proper Rational Function

Consider the system function given by

$$H(z) = \frac{5 + \left(2 - 4\sqrt{2}\right) z^{-1} + \left(3 - \sqrt{2}\right) z^{-2}}{1 - \left(1 - \sqrt{2}\right) z^{-1} + \left(1 - \sqrt{2}\right) z^{-2} + z^{-3}} \tag{46.7}$$

The denominator can be factored to yield

$$H(z) = \frac{5 + \left(2 - 4\sqrt{2}\right) z^{-1} + \left(3 - \sqrt{2}\right) z^{-2}}{\left(1 - \lambda z^{-1}\right) + \left(1 - \lambda^* z^{-1}\right) + \left(1 + z^{-1}\right)} \tag{46.8}$$

where

$$\lambda = \frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2} \qquad \lambda^* = \frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} \tag{46.9}$$

In terms of the notation used in Design Procedure 46.1, we have

$$b_0 = 5 \qquad b_1 = 2 - 4\sqrt{2} \qquad b_2 = 3 - \sqrt{2}$$
$$p_1 = \lambda \qquad p_2 = \lambda^* \qquad p_3 = 1 \tag{46.10}$$

Expanding Eq. (46.8) in the form of Eq. (46.2) yields

$$H(z) = \frac{\beta_1}{\left(1 - \lambda z^{-1}\right)} + \frac{\beta_2}{\left(1 - \lambda^* z^{-1}\right)} + \frac{\beta_3}{\left(1 - z^{-1}\right)} \tag{46.11}$$

Finding a common denominator to add the right-hand side yields a numerator of

$$\begin{aligned} B(z) &= \left(\beta_1 + \beta_2 + \beta_3\right) \\ &\quad + z^{-1}\left[\beta_1\left(1 - \lambda^*\right) + \beta_2\left(1 - \lambda\right) + \beta_3\left(\lambda - \lambda^*\right)\right] \\ &\quad + z^{-2}\left(-\beta_1 \lambda^* - \beta_2 \lambda + \beta_3\right) \end{aligned} \tag{46.12}$$

continues

Example 46.1 *(continued)*

Setting the coefficients from Eq. (46.12) equal to the numerator coefficients from Eq. (46.8) for like powers of z yields the system of equations

$$\begin{aligned} \beta_1 + \beta_2 + \beta_3 &= 5 \\ \beta_1\left(1-\lambda^*\right)+\beta_2\left(1-\lambda\right)+\beta_3\left(\lambda-\lambda^*\right) &= \left(2-4\sqrt{2}\right) \\ -\beta_1\lambda^* - \beta_2\lambda + \beta_3 &= \left(3-\sqrt{2}\right) \end{aligned} \tag{46.13}$$

Substituting the values for λ and λ^*, and then solving for β_1, β_2, and β_3 yields

$$\beta_1 = \beta_2 = 1 \qquad \beta_3 = 3 \tag{46.14}$$

Thus, the unit sample response is given by

$$h[n] = \lambda^n + (\lambda^*)^n + 3(-1)^n \tag{46.15}$$

Inverse *z* Transform via Partial Fraction Expansion

Case 2: All Poles Distinct with $M \geq N$ in System Function (Explicit Approach)

This note presents a procedure for computing the inverse *z* transform for system functions in which all poles are distinct and the degree, *M*, of the numerator equals or exceeds the degree, *N*, of the denominator. The procedure uses polynomial division to convert the improper rational function, *H*(*z*), into the sum of a polynomial, *C*(*z*), and a proper rational function, $H_R(z)$.

In order for a rational function to be expanded as a sum of partial-fraction terms, the degree of the numerator must be less than the degree of the denominator. In cases where $M \geq N$, the system function must be restructured as the sum of a polynomial, $C(z)$, and a proper rational function, $H_R(z)$:

$$H(z) = C(z) + H_R(z) \tag{47.1}$$

where

$$H_R(z) = \frac{R(z)}{A(z)}$$

$$C(z) = \sum_{k=0}^{M-N} c_k z^{-k}$$

This restructuring can be implemented by solving directly for the values of the coefficients in $C(z)$ and $R(z)$ using polynomial division, and then forming a partial fraction expansion for just $H_R(z)$ instead of for $H(z)$. This approach is detailed in Design Procedure 47.1 and is demonstrated in Example 47.1. Polynomial division is detailed in Math Box 47.1 on the next page.

Design Procedure 47.1

Partial Fraction Expansion: Simple Pole Case with $M \geq N$ Solving Explicitly for $C(z)$ and $R(z)$

The system function is an improper rational function if the order, *M*, of the numerator polynomial equals or exceeds the order, *N*, of the denominator polynomial. This procedure is based on explicitly restructuring the system function as the sum of a polynomial, $C(z)$, and a proper rational function, $H_R(z) = R(z)/A(z)$:

$$H(z) = C(z) + \frac{R(z)}{A(z)} \tag{DP 47.1}$$

1. Use polynomial division, as described in Math Box (47.1), to determine the coefficients for $C(z)$ and $R(z)$. The quotient produced by the division is $C(z)$, and the remainder is $R(z)$.
2. Construct the partial fraction expansion for $H_R(z)$:

$$H_R(z) = \frac{R(z)}{\left(1-p_1 z^{-1}\right)\left(1-p_2 z^{-1}\right)\cdots\left(1-p_N z^{-1}\right)} = \frac{\beta_1}{1-p_1 z^{-1}} + \frac{\beta_2}{1-p_2 z^{-1}} + \cdots + \frac{\beta_N}{1-p_N z^{-1}} \tag{DP 47.2}$$

3. Use Design Procedure 46.1 from Note 46 to solve for the *N* values of β_k.
4. Using values for c_k obtained in step 1, and the values for β_k obtained in step 3, the filter's unit sample response, $h[n]$, can be written as

$$h[n] = c_0\delta(n) + c_1\delta(n-1) + \cdots + c_{M-N}\delta(n-M+N) + \beta_1(p_1)^n + \beta_2(p_2)^n + \cdots + \beta_N(p_N)^n \tag{DP 47.3}$$

Example 47.2

Explicitly Solving for a New Numerator When *H*(*z*) Is Improper

Consider the system function given by

$$H(z)=\frac{1+\frac{33}{2}z^{-1}+\frac{1}{2}z^{-2}-21z^{-3}-12z^{-4}}{1+3z^{-1}-6z^{-2}-8z^{-3}} \quad (47.2)$$

$$=\frac{1+\frac{33}{2}z^{-1}+\frac{1}{2}z^{-2}-21z^{-3}-12z^{-4}}{\left(1+4z^{-1}\right)\left(1-2z^{-1}\right)\left(1+z^{-1}\right)}$$

The corresponding unit sample response can be determined using Design Procedure 47.1. Polynomial division can be used, as in Math Box 47.1, to solve for $C(z)$ and $R(z)$ explicitly:

$$C(z)=\tfrac{3}{2}+\tfrac{3}{2}z^{-1}$$

$$R(z)=-\tfrac{1}{2}+\tfrac{21}{2}z^{-1}+5z^{-2}$$

Then we only need to expand the proper rational function that is based on the remainder of the division operation:

$$H_R(z)=\frac{R(z)}{A(z)}=\frac{-\frac{1}{2}+\frac{21}{2}z^{-1}+5z^{-2}}{1+3z^{-1}-6z^{-2}-8z^{-3}} \quad (47.3)$$

$$=\frac{\beta_1}{1+4z^{-1}}+\frac{\beta_2}{1-2z^{-1}}+\frac{\beta_3}{1+z^{-1}}$$

Comparing Eq. (47.2) to Eq. (DP 46.1) from Note 46 reveals that

$$b_0=-\frac{1}{2} \qquad b_1=\frac{21}{2} \qquad b_2=5$$

$$p_1=-4 \qquad p_2=2 \qquad p_3=-1$$

Adding the right-hand side of Eq. (47.2) yields a numerator of

$$R(z)=\beta_1\left(1-2z^{-1}\right)\left(1+z^{-1}\right)+\beta_2\left(1+4z^{-1}\right)\left(1+z^{-1}\right)+\beta_3\left(1+4z^{-1}\right)\left(1-2z^{-1}\right) \quad (47.4)$$

Setting coefficients from Eq. (47.3) equal to coefficients from Eq. (47.2) for like powers of z yields the system of equations

$$b_0=-\tfrac{1}{2}=\beta_1+\beta_2+\beta_3$$

$$b_1=\tfrac{21}{2}=-\beta_1+5\beta_2+2\beta_3$$

$$b_2=5=-2\beta_1+4\beta_2-8\beta_3$$

Math Box 47.1

Polynomial Division

Consider the system function given by

$$H(z)=\frac{1+\frac{33}{2}z^{-1}+\frac{1}{2}z^{-2}-21z^{-3}-12z^{-4}}{1+3z^{-1}-6z^{-2}-8z^{-3}}$$

For purposes of computing the inverse z transform, the goal of polynomial division is to eliminate from the numerator terms involving z^{-k} for $k \geq N$, where N is the most negative degree of z in the denominator. This goal is best accomplished by reversing the order of the polynomials before beginning the division:

$$-8z^{-3}-6z^{-2}+3z^{-1}+1 \;\overline{)\; -12z^{-4}-21z^{-3}+\tfrac{1}{2}z^{-2}+\tfrac{33}{2}z^{-1}+1}$$

Begin by dividing the dividend's first term, $-12z^{-4}$, by the divisor's first term, $-8z^{-3}$. The result is $\frac{3}{2}z^{-1}$, so we multiply the divisor by $\frac{3}{2}z^{-1}$ and subtract the result from the dividend:

$$\begin{array}{r} \tfrac{3}{2}z^{-1} \\ -8z^{-3}-6z^{-2}+3z^{-1}+1 \;\overline{)\; -12z^{-4}-21z^{-3}+\tfrac{1}{2}z^{-2}+\tfrac{33}{2}z^{-1}+1} \\ \underline{-12z^{-4}\;\; -9z^{-3}\;\; +\tfrac{9}{2}z^{-2}+\tfrac{3}{2}z^{-1}} \\ -12z^{-3}\;\; -4z^{-2}+15z^{-1}+1 \end{array}$$

We can call the result of this subtraction the *working remainder*. The terms appearing above the division symbol comprise the *quotient*. Next, divide the first term ($-12z^{-3}$) of the working remainder by the first term ($-8z^{-3}$) of the divisor. The result is ³⁄₂. Therefore, we multiply the divisor by ³⁄₂ and subtract the result from the working remainder to obtain a new working remainder:

$$\begin{array}{r} \tfrac{3}{2}z^{-1}\;\; +\tfrac{3}{2} \\ -8z^{-3}-6z^{-2}+3z^{-1}+1 \;\overline{)\; -12z^{-4}-21z^{-3}+\tfrac{1}{2}z^{-2}+\tfrac{33}{2}z^{-1}+1} \\ \underline{-12z^{-4}\;\; -9z^{-3}\;\; +\tfrac{9}{2}z^{-2}+\tfrac{3}{2}z^{-1}} \\ -12z^{-3}\;\; -4z^{-2}+15z^{-1}+1 \\ \underline{-12z^{-3}\;\; -9z^{-2}\;\; +\tfrac{9}{2}z^{-1}+\tfrac{3}{2}} \\ 5z^{-2}+\tfrac{21}{2}z^{-1}-\tfrac{1}{2} \end{array}$$

The process stops when the highest power of z^{-1} in the divisor is greater than the highest power of z^{-1} in the working remainder. The working remainder when the process stops becomes the new numerator in the fractional part of the system function. The quotient becomes the new nonfractional $C(z)$, and we can rewrite the system function as

$$H(z)=\tfrac{3}{2}+\tfrac{3}{2}z^{-1}+\frac{5z^{-2}+\frac{21}{2}z^{-1}-\frac{1}{2}}{-8z^{-3}-6z^{-2}+3z^{-1}+1}$$

Inverse *z* Transform via Partial Fraction Expansion

Case 3: All Poles Distinct with $M \geq N$ in System Function (Implicit Approach)

This note presents a procedure for computing the inverse *z* transform for system functions in which all poles are distinct, and the degree, *M*, of the numerator equals or exceeds the degree, *N*, of the denominator. The approach presented herein is an alternative to the approach presented in Note 47.

In order for a rational function to be expanded as a sum of partial-fraction terms, the degree of the numerator must be less than the degree of the denominator. In cases where $M \geq N$, the system function must be restructured as the sum of a polynomial, $C(z)$, and a proper rational function, $H_R(z)$:

$$H(z) = C(z) + H_R(z) \qquad (48.1)$$

where

$$H_R(z) = \frac{R(z)}{A(z)}$$

$$C(z) = \sum_{k=0}^{M-N} c_k z^{-k}$$

This restructuring can be exploited by writing an expression for $H(z)$ using symbolic coefficients for c_k and β_k and then using a coefficient-matching technique to solve for c_k and β_k values without ever explicitly determining the coefficients for $R(z)$. This approach is detailed in Design Procedure 48.1 and is demonstrated in Example 48.1.

Design Procedure 48.1

Partial Fraction Expansion: Simple Pole Case with $M \geq N$ (Implicit Approach)

The system function is an improper rational function if the order, M, of the numerator polynomial equals or exceeds the order, N, of the denominator polynomial. This procedure is based on implicitly restructuring the system function as the sum of a polynomial, $C(z)$, and a proper rational function, $H_R(z) = R(z)/A(z)$:

$$H(z) = C(z) + \frac{R(z)}{A(z)} \qquad (DP\ 48.1)$$

However, the coefficients for $R(z)$ are never explicitly determined.

1. The number of fractional terms in the PFE depends only upon the degree of the denominator. Because $H(z)$ and $H_R(z)$ have the same denominator, we can immediately write

$$H(z) = C(z) + \frac{\beta_1}{1 - p_1 z^{-1}} + \frac{\beta_2}{1 - p_2 z^{-1}} + \cdots + \frac{\beta_N}{1 - p_N z^{-1}} \qquad (DP\ 48.2)$$

2. The polynomial portion of the PFE has at most $M - N + 1$ terms, so we can expand $C(z)$ to yield

$$H(z) = c_0 + c_1 z^{-1} + \cdots + c_{M-N} z^{-(M-N)} + \frac{\beta_1}{1 - p_1 z^{-1}} + \frac{\beta_2}{1 - p_2 z^{-1}} + \cdots + \frac{\beta_N}{1 - p_N z^{-1}} \qquad (DP\ 48.3)$$

3. Use Design Procedure 48.2 to solve for the N values of β_k and the $M - N + 1$ values of c_k.

4. Once the values for c_k and β_k are determined, the filter's unit sample response, $h[n]$, can be written as

$$h[n] = c_0 \delta(n) + c_1 \delta(n-1) + \cdots + c_{M-N} \delta(n - M + N) + \beta_1 (p_1)^n + \beta_2 (p_2)^n + \cdots + \beta_N (p_N)^n \qquad (DP\ 48.4)$$

Design Procedure 48.2

Solving for PFE Coefficients: Coefficient-Matching Approach for Expansion with Both Rational and Non-Rational Polynomial Terms

This procedure is appropriate for solving a PFE of the form

$$\frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_M z^{-M}}{(z - p_1)(z - p_2)\cdots(z - p_{N-1})(z - p_N)} = c_0 + c_1 z^{-1} + \cdots + c_{M-N} z^{-(M-N)} + \frac{\beta_1}{1 - p_1 z^{-1}} + \frac{\beta_2}{1 - p_2 z^{-1}} + \cdots + \frac{\beta_N}{1 - p_N z^{-1}} \tag{DP 48.5}$$

When a common denominator is generated to perform the indicated additions on the right-hand side of this equation, the resulting denominator always equals the denominator on the left-hand side. Therefore, we can simplify the notation by just retaining the numerators that result after the common denominator has been implemented:

$$\begin{aligned} b_0 + b_1 z^{-1} + b_2 z^{-1} + \cdots + b_M z^{-M} &= c_0 \prod_{k=1}^{N} \left(1 - p_k z^{-1}\right) \\ &+ c_1 z^{-1} \prod_{k=1}^{N} \left(1 - p_k z^{-1}\right) + \cdots + c_{M-N} z^{-(M-N)} \prod_{k=1}^{N} \left(1 - p_k z^{-1}\right) + \beta_1 \prod_{k=2}^{N} \left(1 - p_k z^{-1}\right) \\ &+ \beta_2 \prod_{\substack{k=1 \\ k \neq 2}}^{N} \left(1 - p_k z^{-1}\right) + \cdots + \beta_n \prod_{\substack{k=1 \\ k \neq n}}^{N} \left(1 - p_k z^{-1}\right) + \cdots + \beta_N \prod_{k=1}^{N-1} \left(1 - p_k z^{-1}\right) \end{aligned} \tag{DP 48.6}$$

1. Explicitly construct Eq. (DP 48.5) using the known values for the coefficients, b_k, and the poles, p_k.
2. Perform the necessary algebraic manipulations so that the coefficient for each power of z^{-1} on the right-hand side is obtained as a function of the unknown residues, β_k, and unknown coefficients, c_k:

$$\begin{aligned} B(z) &= f_0\left(\beta_0, \ldots, \beta_N; c_0, \ldots, c_{M-N}\right) + z^{-1} f_1\left(\beta_0, \ldots, \beta_N; c_0, \ldots, c_{M-N}\right) \\ &+ z^{-2} f_2\left(\beta_0, \ldots, \beta_N; c_0, \ldots, c_{M-N}\right) + \cdots + z^{-M} f_M\left(\beta_0, \ldots, \beta_N; c_0, \ldots, c_{M-N}\right) \end{aligned} \tag{DP 48.7}$$

3. Set the coefficients from the left-hand side of Eq. (DP 48.6) equal to the corresponding coefficients from the right-hand side of Eq. (DP 48.7):

$$\begin{aligned} b_0 &= f_0\left(\beta_0, \ldots, \beta_N; c_0, \ldots, c_{M-N}\right) \\ b_1 &= f_1\left(\beta_0, \ldots, \beta_N; c_0, \ldots, c_{M-N}\right) \\ &\vdots \\ b_M &= f_M\left(\beta_0, \ldots, \beta_N; c_0, \ldots, c_{M-N}\right) \end{aligned} \tag{DP 48.8}$$

4. Solve the system of linear equations (DP 48.8) for the residues, β_k, and coefficients, c_k.

Example 48.3

Implicitly Solving for a New Numerator When *H*(*z*) Is Improper

Consider the system function given by

$$H(z)=\frac{1+\frac{33}{2}z^{-1}+\frac{1}{2}z^{-2}-21z^{-3}-12z^{-4}}{1+3z^{-1}-6z^{-2}-8z^{-3}}$$
$$=\frac{1+\frac{33}{2}z^{-1}+\frac{1}{2}z^{-2}-21z^{-3}-12z^{-4}}{\left(1+4z^{-1}\right)\left(1-2z^{-1}\right)\left(1+z^{-1}\right)} \quad (48.2)$$

The corresponding unit sample response can be determined using Design Procedures 48.1 and 48.2.

Comparing Eq. (48.2) to Eq. (DP 48.5) reveals that

$$b_0=1 \quad b_1=\tfrac{33}{2} \quad b_2=\tfrac{1}{2} \quad b_3=-21 \quad b_4=-12$$
$$p_1=-4 \quad p_2=2 \quad p_3=-1$$

The value for $M-N=1$, so $C(z)$ must include terms for z_0 and z_{-1}. Following the form of Eq. (DP 48.3), we can write

$$H(z)=c_0+c_1z^{-1}+\frac{\beta_1}{1+4z^{-1}}+\frac{\beta_2}{1-2z^{-1}}+\frac{\beta_3}{1+z^{-1}} \quad (48.3)$$

If the right-hand side of Eq. (48.2) is added over a common denominator of

$$A(z)=1+3z^{-1}-6z^{-2}-8z^{-3}$$

the numerator of the result is given by

$$\begin{aligned}B(z)&=c_0\left(1+3z^{-1}-6z^{-2}-8z^{-3}\right)\\&+c_1\left(z^{-1}+3z^{-2}-6z^{-3}-8z^{-4}\right)\\&+\beta_1\left(1-2z^{-1}\right)\left(1+z^{-1}\right)\\&+\beta_2\left(1+4z^{-1}\right)\left(1+z^{-1}\right)\\&+\beta_3\left(1+4z^{-1}\right)\left(1-2z^{-1}\right)\end{aligned} \quad (48.4)$$

Setting coefficients from Eq. (48.4) equal to coefficients from Eq. (48.2) for like powers of *z* yields the system of equations

$$\begin{aligned}b_0&=1=c_0+\beta_1+\beta_2+\beta_3\\b_1&=\tfrac{33}{2}=3c_0+c_1-\beta_1+5\beta_2+2\beta_3\\b_2&=\tfrac{1}{2}=-6c_0+3c_1-2\beta_1+4\beta_2-8\beta_3\\b_3&=-21=-8c_0-6c_1\\b_4&=-12=-8c_1\end{aligned} \quad (48.5)$$

Solving this system of equations yields

$$c_0=c_1=\tfrac{3}{2} \quad \beta_1=\tfrac{-5}{2} \quad \beta_2=\tfrac{4}{3} \quad \beta_3=\tfrac{2}{3} \quad (48.6)$$

Designing IIR Filters: Background and Options

The block diagram in Figure 49.1 represents the direct form implementation for an *infinite impulse response* (IIR) digital filter. The filter consists of an all-zero section followed by an all-pole section. At time n, the all-zero section computes a weighted sum of the inputs from times $n-M$ through n. To this result, the all-pole section adds a weighted sum of the previous outputs from times $n-N$ through $n-1$. The usual mathematical characterizations for an IIR filter are summarized in Math Boxes 49.1 and 49.2.

The difference equation for the filter's output, $y[n]$, can be written by inspecting the block diagram:

$$y[n]=\sum_{k=1}^{N}a_k y[n-k]+\sum_{k=0}^{M}b_k x[n-k] \quad (49.1)$$

The system function, $H(z)$, can be obtained by taking the z transform of each term in the difference equation, factoring out $X(z)$ and $Y(z)$, and then forming $H(z)$ as the ratio $Y(z)/X(z)$:

$$Y(z)=\sum_{k=1}^{N}a_k Y(z)z^{-k}+\sum_{k=0}^{M}b_k X(z)z^{-k} \quad (49.2)$$

$$Y(z)\left[1-\sum_{k=1}^{N}a_k z^{-k}\right]=X(z)\sum_{k=0}^{M}b_k z^{-k} \quad (49.3)$$

$$H(z)=\frac{Y(z)}{X(z)}=\frac{\sum_{k=0}^{M}b_k z^{-k}}{1-\sum_{k=1}^{N}a_k z^{-k}} \quad (49.4)$$

The unit sample response, $h[n]$, can then be obtained as the inverse z transform of $H(z)$. The *partial fraction expansion* described in

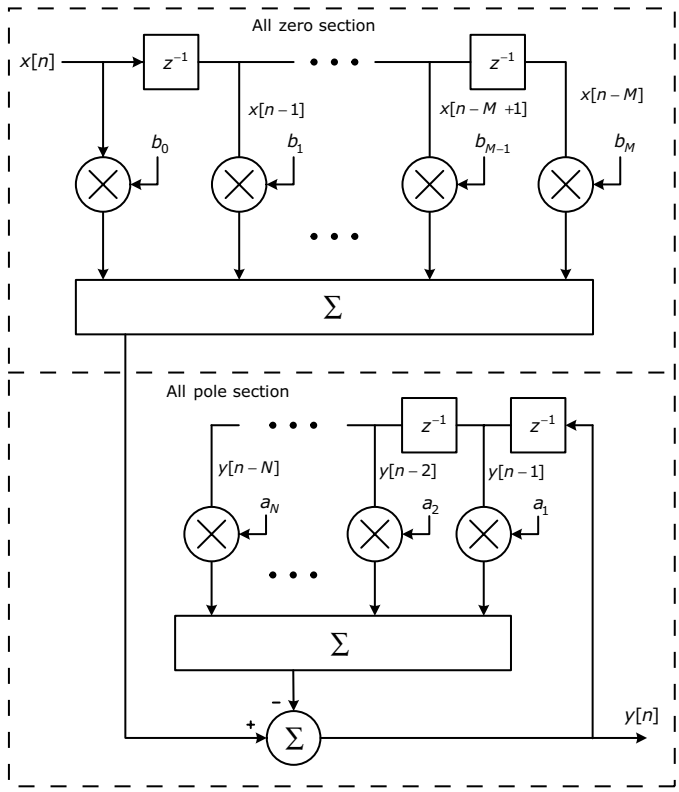


Figure 49.1 Block diagram for IIR filter

Note 46 is the usual choice for computing this inverse transform.

The difference equation for an IIR filter often appears in the alternate form

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k] \qquad (49.5)$$

which often proves to be more convenient for subsequent mathematical manipulation. Because Eq. (49.1) is explicitly solved for $y[n]$, it would be tempting to refer to Eq. (49.1) as the *explicit* form and to Eq. (49.5) as the *implicit* form for the IIR difference equation. *Explicit* versus *implicit* distinctions are widespread in the study of differential equations, but as used there, it is always possible to convert between explicit form and implicit form through the use of differentiation, integration, and algebraic manipulation. While Eqs. (49.1) and (49.5) are equivalent in their ability to represent an IIR filter adequately, it is not possible to convert from one to the other without redefining the sense of a_k. Solving Eq. (49.5) for $y[n]$ yields

$$y[n] = \frac{1}{a_0}\left[-\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k]\right]$$

To make $y[n]$ from this equation equal to the $y[n]$ from Eq. (49.1), we would need to set $a_0 = 1$ and reverse the sign of a_k for $k = 1, 2, \ldots, N$. Therefore, the use of *explicit* and *implicit* to distinguish between Eqs. (49.1) and (49.5) is not appropriate. In this book, the form of Eq. (49.1) is described as following the *systems convention*, and the form of Eq. (49.5) is described as following the *mathematical convention*.

Math Box 49.1

IIR Characterizations under the "Systems Convention"

Difference Equation

$$y[n] = \sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k]$$

System Function

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 - \sum_{k=1}^{N} a_k z^{-k}}$$

Unit Sample Function

$$h[n] = Z^{-1}\{H(z)\}$$

Frequency Response

$$H(\omega) = \frac{\sum_{k=0}^{M} b_k \exp(-j\omega k)}{1 - \sum_{k=1}^{N} a_k \exp(-j\omega k)}$$

49.1 Implementation Structures

The block diagram of Figure 49.1 can be converted into the corresponding signal flow graph shown in Figure 49.2. As already mentioned, the filter consists of an all-zero section followed by an all-pole section. Because the sections are linear and time-invariant, we can commute the order of the sections to produce the structure shown in Figure 49.3. This structure can then be simplified by noticing that the two delay chains running down the center of the figure are delaying the exact same signal. These two delay chains can be merged to yield the *direct form II* structure shown in Figure 49.4.

Math Box 49.2

IIR Characterizations under the "Mathematical Convention"

Difference Equation

$$\sum_{k=0}^{N} a_k\, y[n-k] = \sum_{k=0}^{M} b_k x[n-k]$$

System Function

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}}$$

Unit Sample Function

$$h[n] = Z^{-1}\{H(z)\}$$

Frequency Response

$$H(w) = \frac{\sum_{k=0}^{M} b_k \exp(-jwk)}{\sum_{k=0}^{N} a_k \exp(-jwk)}$$

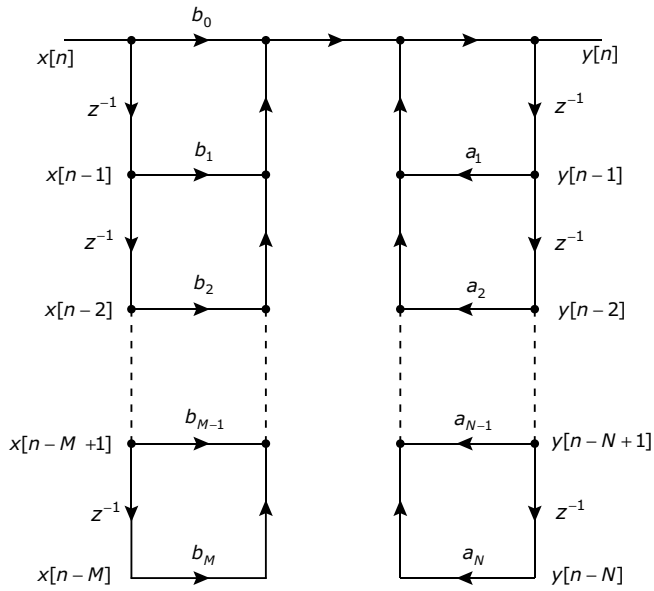


Figure 49.2 Direct form I structure for implementing an IIR filter

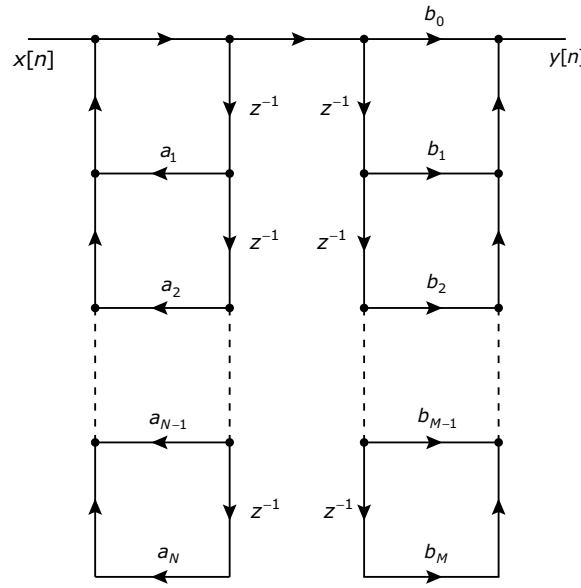


Figure 49.3 Modified direct form I structure with sequence of all-pole and all-zero sections transposed

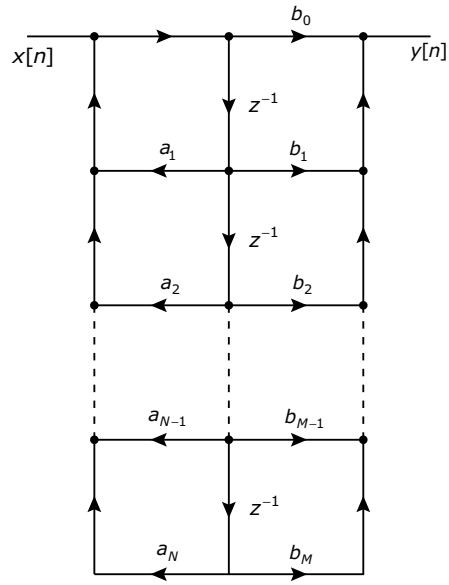


Figure 49.4 Direct form II structure for implementing anIIR filter

Designing IIR Filters: Impulse Invariance Method

The impulse invariance technique is based on the simple idea of obtaining an impulse response for a digital filter by sampling an analog filter's impulse response. Recipe 50.1 provides the details for a direct implementation of this idea. However, the actual design is often carried out in the frequency domain without ever explicitly determining the impulse response of the analog prototype. The frequency-domain approach is detailed in Recipe 50.2. The mathematical justification for the frequency-domain approach can be found in [1–2].

The details of how the impulse invariance technique maps *s*-plane features of the analog prototype filter into *z*-plane features of the targeted IIR filter are provided in Section 50.2.

50.1 Aliasing

Just as sampling an analog signal causes the signal's spectrum to be replicated along the frequency axis at intervals equal to the sampling rate, sampling a system's impulse response causes the system's frequency response to be replicated along the frequency axis at intervals equal to the sampling rate. The digital filter's response, $H(e^{j\omega})$, can be expressed in terms of the analog signal's response, $H_a(e^{j\omega})$ as

$$H(e^{j\omega}) = \frac{1}{T}\sum_{k=-\infty}^{\infty} H_a\left[j\left(\frac{\omega}{T} + \frac{2\pi}{T}k\right)\right] \qquad (50.1)$$

Strategic Considerations

The impulse invariance method is based on sampling an analog filter's impulse response to obtain the impulse response for an IIR digital filter.

- Advantages
  - If the analog filter is stable, the digital filter designed from it via impulse invariance will also be stable.
  - Impulse invariance preserves the magnitude characteristics of the analog filter.
- Disadvantages
  - Impulse invariance does not preserve the step response characteristics of the analog filter.
  - Due to aliasing of the magnitude response, impulse invariance is limited to use on lowpass or bandpass filters.
  - Sampling rate, f_s, must be high enough to ensure that $f_s/2$ falls in the stopband (upper stopband for bandpass filter).

If the analog signal's response is bandlimited such that

$$H_a\left(\frac{\omega}{T}\right)=0 \text{ for } |\omega|>\pi \qquad (50.2)$$

then there is no overlap between the shifted copies of $H_a(\cdot)$ in the summation of Eq. (50.2). In this case, the baseband image of the digital filter's response equals an amplitude-scaled version of the analog signal's response:

$$H\left(e^{j\omega}\right)=\frac{1}{T}H_a\left(j\frac{\omega}{T}\right) \text{ for } |\omega|\le\pi$$

If the constraint in Eq. (50.3) is not satisfied, then there will be aliasing among the images of the response. Practical filter responses are never perfectly bandlimited, but the impulse invariance technique can be used successfully only if the analog filters' response for $|\omega|>\pi$ is negligible. This constraint effectively restricts the use of impulse invariance to lowpass and bandpass filters.

Recipe 50.1

Impulse Invariance: Direct Method

1. Select the desired analog filter.
2. Determine the filter's impulse response, $h_a(t)$.
3. Substitute nT for t in $h_a(t)$.
4. Compute $H(z)$ as the z transform of $h_a(nT)$:
$$H(z)=\mathcal{Z}\{h_a(nT)\}$$
5. Perform the necessary algebraic manipulations to express $H(z)$ as a ratio of polynomials:
$$H(z)=\frac{\sum_{k=0}^{M}b_k z^{-k}}{1+\sum_{k=1}^{N}a_k z^{-k}}$$
6. Use the coefficients a_k and b_k to implement the IIR as one of the structures shown in Note 49.

50.2 Feature Mapping

The sampled impulse response can be viewed as the impulse response for an analog filter that has an s-domain transfer function, $\tilde{H}(s)$. In theory, this transfer function could be obtained directly as the Laplace transform of the sampled impulse response:

$$H(s)=\mathcal{L}\{h_a(nT)\}$$

and it would be different from the original prototype filter's transfer function, $H(s)$, which is the Laplace transform of the unsampled impulse response:

$$H(s)=\mathcal{L}\{h_a(t)\}$$

It can be shown (see [2]) that an alternative way to obtain the transfer function, $\tilde{H}(s)$,

involves evaluating the digital filter's system function for $z = e^{sT}$:

$$H(s) = H(z)\Big|_{z=e^{sT}}$$

It then follows that s-plane to z-plane mapping produced by the impulse invariance method can be evaluated simply by substituting the appropriate values of s into the equation $z = e^{sT}$. Some of the more useful mappings are listed below.

1. The s-plane origin maps to $z = 1$.
2. The point $s = j\pi/T$ maps to $z = -1$.
3. The segment of the $j\omega$ axis extending from 0 to π/T in the s-plane maps into the upper half of the unit circle in the z-plane.
4. The segment of the $j\omega$ axis extending from 0 to $-\pi/T$ in the s-plane maps into the lower half of the unit circle in the z-plane.
5. Any two s-plane points, s_1 and s_2, related by $s_2 = s_1 + j2\pi k/T$ (for any integer k) maps into a single point in the z-plane.
6. As a consequence of mappings 3 and 5, the segments of the $j\omega$ axis extending from $2k\pi/T$ to $(2k+1)\pi/T$ for integer k, all map into the upper half of the unit circle in the z-plane.
7. As a consequence of mappings 4 and 5, the segments of the $j\omega$ axis extending from $2k\pi/T$ to $(2k-1)\pi/T$ for integer k, all map into the lower half of the unit circle in the z-plane.
8. The left half of the s-plane maps into the interior of the unit circle in the z-plane.
9. The right half of the s-plane maps to the exterior of the unit circle in the z-plane.
10. The origin in the z-plane corresponds to $s = -\infty$ in the z-plane.

Recipe 50.2

Impulse Invariance: Frequency Domain Method

1. Select the desired analog filter.
2. Determine the filter's transfer function, $H_a(s)$.
3. Using Recipe 46.1, compute the partial fraction expansion for $H_a(s)$:

$$H_a(s) = \sum_{k=1}^{N} \frac{\beta_k}{s - s_k}$$

4. Using the values for β_k and s_k from step 3, form the partial fraction expansion for $H(z)$ as

$$H(z) = \sum_{k=1}^{N} \frac{\beta_k}{1 - e^{s_k T} z^{-1}}$$

5. Form a common denominator and add the terms in the summation from step 4 to obtain $H(z)$ as a ratio of polynomials:

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}$$

6. Use the coefficients a_k and b_k to implement the IIR filter as one of the structures shown in Note 49.

11. A vertical line in the *s*-plane located at $s = \sigma_1$ maps into a circle of radius $\exp(\sigma_1)$ centered at the origin in the *z*-plane.

The feature mapping described in this section is sometimes presented as the "standard" *s*-plane to *z*-plane mapping, but it really is a consequence of the impulse invariance method for mapping filters from the *s* domain into the *z* domain. Other filter mapping techniques (such as the bilinear transformation covered in Note 51) result in different mappings of *s*-plane transfer function features into *z*-plane system function features.

References

1. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing,* Prentice Hall, 1975.
2. A. Antoniou, *Digital Filters: Analysis and Design*, McGraw-Hill, 1979.

Designing IIR Filters: Bilinear Transformation

The bilinear transformation is the most commonly used approach for designing IIR filters. As discussed in Note 50, the impulse invariance technique uses a many-to-one frequency mapping that maps multiple intervals from the $j\omega$ axis in the s-plane repeatedly onto the same points on the unit circle in the z-plane. This mapping can introduce aliasing of the filter response, making the impulse invariance technique unsuitable for use in designing highpass or bandstop filters. The bilinear transformation uses a non-linear frequency mapping such that the entire $j\omega$ axis of the s-plane maps one-to-one onto points of the unit circle of the z-plane. The mapping avoids aliasing, but it does "warp" critical frequencies of the analog prototype into different frequencies for the corresponding features in the digital filter's response.

Major Points

- The bilinear transformation is the most commonly used technique for designing IIR filters that are based on analog prototype filter designs.
- The frequency mapping that is used in the bilinear transformation avoids the aliasing that can occur when the impulse invariance method is used. (See Key Concept 51.1.)
- The frequency mapping used in the bilinear transformation tends to compress well-separated higher-frequency features from the analog filter's response into a narrow frequency range in the digital filter's response. (See Key Concept 51.2.)

51.1 Prewarping

The s-plane to z-plane mapping that results from the bilinear transformation is depicted in Key Concept 51.2. This "warped" frequency relationship becomes increasingly compressed at higher frequencies, resulting in a need to prewarp the critical frequencies in such a way that these frequencies warp into the desired locations in the final IIR design. The usual strategy for mitigating the effect of frequency warping involves prewarping the critical frequencies using the relationship

$$\omega_a = \frac{2}{T}\tan\frac{\omega_d T}{2} \qquad (51.1)$$

where T is the sampling interval, ω_d is the desired frequency in the final IIR design, and ω_a is the prewarped frequency to be used in the design of the analog prototype.

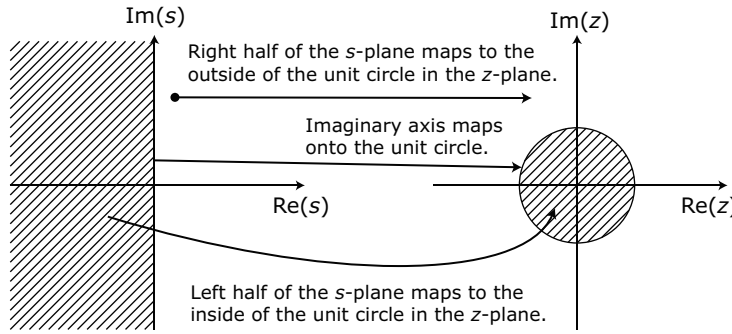
The details of the bilinear transformation (with prewarping included) are provided in Recipe 51.1.

Example 51.1 demonstrates a manual application of this procedure. Several MATLAB-based approaches for the bilinear transformation are demonstrated in Example 51.2.

Key Concept 51.1

Bilinear Transformation's Frequency Mapping Avoids Aliasing

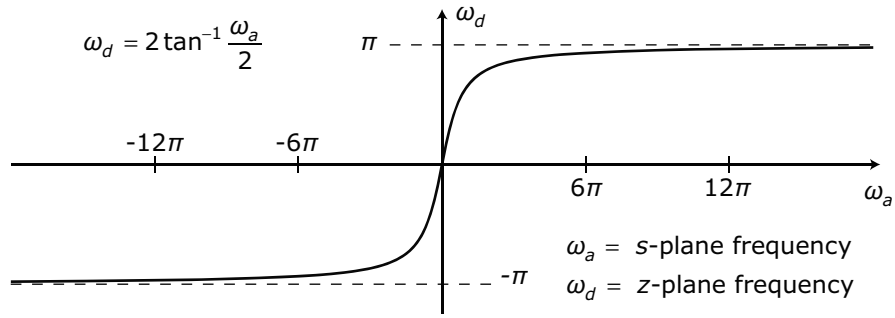
In the frequency conversion used by the bilinear transformation, the analog frequency's domian of ±∞ maps one-to-one into a digital frequency range of ±π. This mapping avoids aliasing that often occurs in the impulse invariant maping that was presented in Note 50.



Key Concept 51.2

Bilinear Frequency Mapping Warps Critical Frequencies

The mapping from the *s*-plane to the *z*-plane creates the relationship between *s*-plane frequencies and *z*-plane frequencies that is depicted below. This "warped" frequency relationship becomes increasingly compressed at higher frequencies, resulting in a need to prewarp the critical frequencies in the analog prototype so that these frequencies warp into the desired locations in the final IIR design.



Recipe 51.1

Bilinear Transformation

1. Select the basic family (Butterworth, Chebyshev, and so on) for the analog prototype filter using information from Notes 40 through 43.

2. For each critical frequency, ω_d, in the targeted IIR filter, determine the corresponding "prewarped" frequency to use in designing the analog prototype as

$$\omega_a = \frac{2}{T}\tan\frac{\omega_d T}{2} \qquad \text{(R 51.1)}$$

or

$$f_a = \frac{f_s}{\pi}\tan\frac{f_d \pi}{f_s} \qquad \text{(R 51.2)}$$

3. Using design formulas from Notes 40 through 43, along with the prewarped frequencies from step 2, obtain the transfer function, $H_a(s)$, for the desired analog prototype filter.

4. Transform $H_a(s)$ into the system function, $H(z)$, by making the substitution

$$s = \frac{2(1-z^{-1})}{T(1+z^{-1})} \qquad \text{(R 51.3)}$$

where T is the sampling interval for the digital filter.

5. Perform the necessary algebraic manipulations to put $H(z)$ into the form of a ratio of polynomials in z^{-1}:

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-1}}{1-\sum_{k=1}^{N} a_k z^{-1}} \qquad \text{(R 51.4)}$$

6. Use the a_k and b_k values to implement the IIR filter in one of the structures presented in Note 49.

Example 51.1

Manual Application of the Bilinear Transformation

Use Recipe 51.1 to obtain an IIR filter for a second-order Butterworth analog prototype. The desired 3-dB cutoff frequency is 3 kHz, and the sampling rate is 30,000 samples per second.

The prewarped cutoff frequency for the prototype is obtained as

$$f_a = \frac{3\times10^4}{\pi}\tan\frac{3\times10^3\,\pi}{3\times10^4}$$
$$= 3102.75 \text{ Hz}$$
$$\omega_a = 2\pi f_a = 19{,}495.18 \text{ rad/sec}$$

The transfer function for a second order Butterworth filter is given by

$$H_a(s) = \frac{\omega_c^2}{s^2 + \sqrt{2}\omega_c s + \omega_c^2}$$

Substitution of Eq. (R 51.3) for s yields

$$H(z) = \frac{\omega_c^2}{\left(\frac{2}{T}\right)^2\left(\frac{1-z^{-1}}{1+z^{-1}}\right)^2 + \omega_c\sqrt{2}\left(\frac{2}{T}\right)\left(\frac{1-z^{-1}}{1+z^{-1}}\right) + \omega_c^2}$$

Setting ωc = ωa = 19,495.18 and T = $(30{,}000)^{-1}$ then yields

$$H(z) = \frac{0.067455 + 0.13491z^{-1} + 0.067455z^{-2}}{1 - 1.142981z^{-1} + 0.412802z^{-2}} \qquad (51.2)$$

The magnitude and phase response shown in Figure 51.1 can be obtained using the MATLAB function `freqz(b,a,1024)` with

```
a=[1 -1.142981 0.412802]
```

and

```
b=[0.067455 0.13491 0.067455].
```

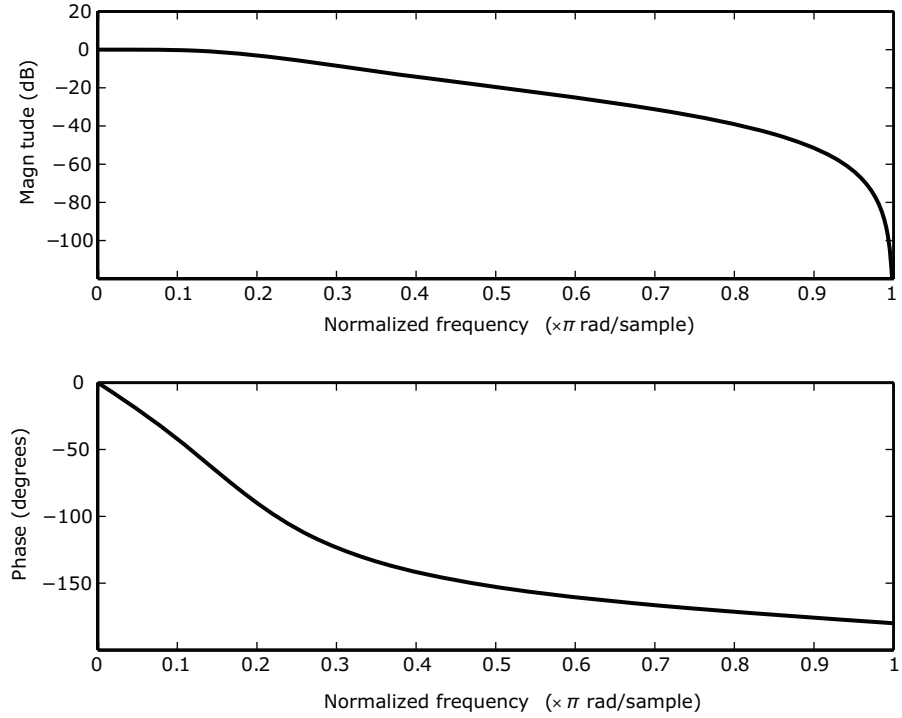


Figure 51.1 Frequency response for IIR filter from Example 51.1

Example 51.2

Bilinear Transformation Using MATLAB

Manual application of the bilinear transformation quickly becomes tedious and error-prone as the filter order is increased. As with many other design techniques for digital filters, some sort of computer-aided design tool is essential for practical applications. In this example we use MATLAB to design an IIR filter based on a Butterworth prototype. The desired 3-dB cutoff frequency is 3 kHz, and the sampling rate is 30,000 samples per second. MATLAB is rich with signal processing tools and there are a number of different ways these tools can be used to obtain our desired filter. The MATLAB function `butter` can be used to design either analog or digital IIR Butterworth filters. The call

```
[zd, pd, kd] = butter(n,Wn);
```

designs an IIR Butterworth lowpass filter of order `n` with a normalized 3-dB cutoff frequency of `Wn`. The cutoff frequency is normalized based on half the sampling rate equaling 1 rather than the more common practice of normalizing based on a sampling rate equal to 1. In this example, the normalized cutoff frequency for `butter` would be `Wn=0.2`. The first output, `zd`, is a column vector containing zeros of the filter's system function. The second output, `pd`, is a column vector containing poles of the filter's system function. The third output, `kd`, is a constant gain factor that has been removed from the filter's zeros so that the constant term in the system function's numerator polynomial equals 1. If we use `butter` to design the second order filter from Example 51.1, the results are

```
zd =
-1
-1

pd =
0.5714902512699 + 0.29359920095191
0.5714902512699 - 0.29359920095191

kd =
0.067455273889072
```

The numerator and denominator polynomials for `H(z)` can be obtained using the MATLAB poly function

```
>>bd=poly(zd)
bd =
 1   2   1

>>ad = poly(pd)
ad =
 1.00000000000000  -1.14298050253990
0.41280159809619
```

This result corresponds to

$$H(z) = (0.067455)\frac{1+2z^{-1}+z^{-2}}{1-1.14298z^{-1}+0.4128016z^{-2}}$$

and agrees with Eq. (51.2) to within a small amount of numerical error.

A second MATLAB-based approach explicitly generates the poles and zeros for the transfer function of the analog prototype and then applies the bilinear transformation as a separate step. The call

```
[z, p, k] = butter(n, Wn,'s');
```

designs an analog Butterworth lowpass filter of order `n` with a 3-dB cutoff ffequency of `Wn` radians per second. The call

```
[zd,pd,kd] = bilinear(z,p,k,fs);
```

uses the bilinear transformation to convert the analog transfer function defined by `[z,p,k]` into the discrete-time system function defined by `[zd,pd,kd]` with a sampling rate of `fs`. Thus the desired IIR filter for this example can be generated using

```
>> wd = 6000*pi;
>> T = 1/30000;
>> wa = (2/T)*tan(wd*T/2)

>> [z,p,k] = butter(2,wa,'s');
>> [zd,pd,kd] = bilinear(z,p,k,30000);

>> ad=poly(pd)
ad =
1.000000000000000  -1.142980502539901
0.412801598096189

>> bd=poly(zd)
bd =
  1     2     1

>> kd
kd =
0.067455273889072
```

Once again, the results for `ad`, `bd`, and `kd` agree with Eq. (51.2) to within a small amount of numerical error.

MATLAB provides an alternate form of `bilinear` that automatically performs prewarping for a single frequency. The calling sequence for this form is

```
[zd,pd,kd] = bilinear(z,p,k,fs,fd);
```

where `fd` is the critical frequency upon which the prewarping is to be based. Using this form of `bilinear`, the desired IIR filter can be designed using

```
>> [z,p,k] = butter(2,6000*pi,'s');
>> [zd,pd,kd] = bilinear(z,p,k,30000,3000);
```

Decimation: The Fundamentals

This note introduces the fundamental concepts involved in *decimation*, which is the name given to the process of decreasing a discrete-time signal's sampling rate while leaving all other characteristics of the signal unchanged to the maximum extent possible.

Decimation is based on the notion that some discrete-time signal of interest exists in a form that is significantly oversampled. There may be some noise and interference throughout the entire bandwidth of $\pm F_S/2$ that is supported by the sample rate, F_S, but the signal of interest has a spectrum that is confined to a bandwidth much smaller than $\pm F_S/2$.

Consider a discrete-time signal having the periodic spectrum shown in Figure 52.1(a). The spectral images are centered on integer multiples of the sampling rate, F_S, and there may be significant amounts of interference between the passband images as shown. The goal of the decimation process is to decrease the signal's sample rate by some factor, say, M, in such a way that the new signal's spectrum is similar to the one depicted in Figure 52.1(c).

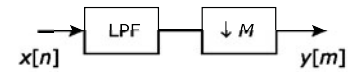
The spectral images in the new spectrum correspond to the portions of interest contained in the images in the original spectrum, but they are centered on integer multiples of the new sampling rate, F_S/M. The interference bands from the original spectrum must be attenuated, as shown in Figure 52.1(b), so that they alias in to the new passband images at acceptably low levels. The basic process for decimation is provided in Recipe 52.1.

Designing the anti-aliasing filter for a decimator can sometimes be a difficult task,

Recipe 52.1

Signal Decimation

As depicted in the block diagram, the canonical form of decimation consists of just two steps:



1. Pass the signal through a digital lowpass filter that is designed to pass the signal of interest (SOI) while attenuating any noise or interference that may exist outside of the bandwidth occupied by the SOI. This filter is just another application of the familiar anti-aliasing filter. Figure 52.1(a) shows the spectrum of a typical signal, and Figure 52.1(b) shows the spectrum after the signal has been filtered to (1) reduce the inter-image interference levels, and (2) make the transition bands slightly narrower than in the original spectrum.
2. Retain every Mth sample from the signal while discarding all others. This step is referred to as either *downsampling* or *sample rate reduction*. Figure 52.1(c) shows the signal's spectrum after downsampling by a factor of 3.

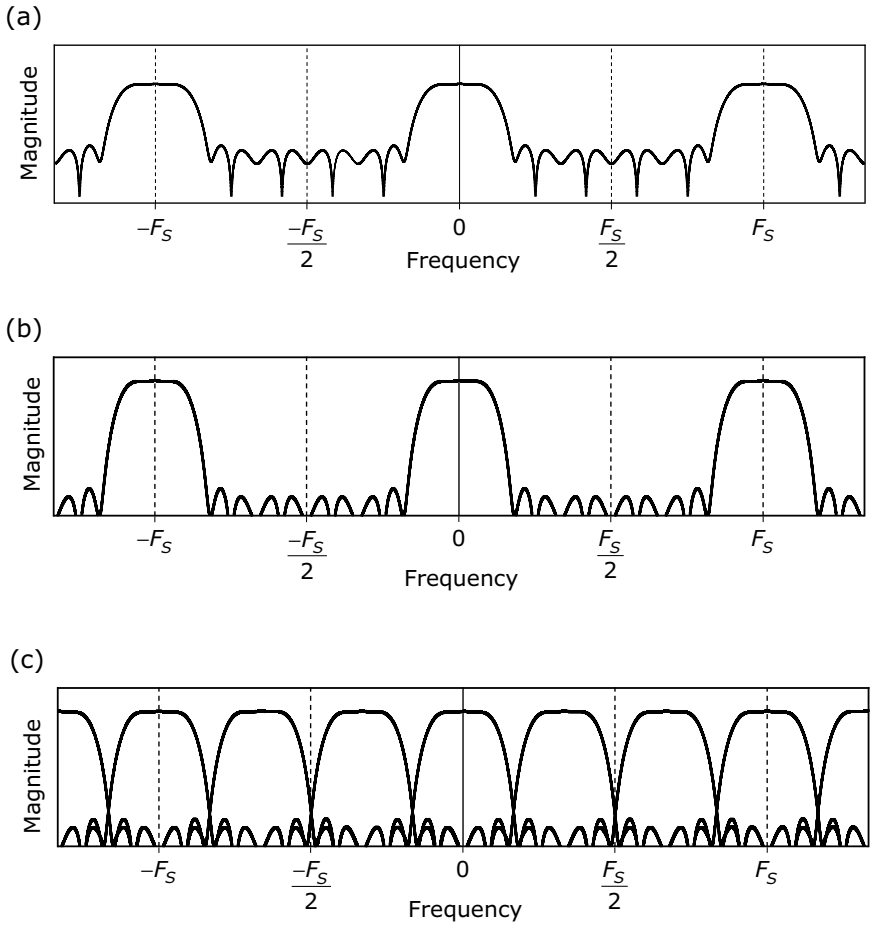


Figure 52.1 Spectrum of a discrete-time signal at key points in the decimation process: (a) original spectrum, (b) spectrum after lowpass filtering, and (c) spectrum after downsampling by a factor of 3.

especially when the value of M is large. The filter's passband edge must be high enough to pass the complete portion of interest from the baseband image, while the stopband edge must be low enough to severely attenuate all noise and interference at frequencies beyond the new (that is, post-downsampling), folding frequencies of $F_S/(2M)$.

52.1 Efficient FIR Decimators

The direct form for implementing a decimator is shown in Figure 52.2. Part (a) of the figure shows the usual block diagram representation for a decimator, while part (b) uses a signal flow graph to show the direct-form-1 implementation for the anti-aliasing filter. The filter must perform N multiplications each time a new input sample is clocked in. If the input sample rate is F_S, the multiplication burden imposed by the filter is NF_S multiplications per second.

Any form—FIR or IIR, direct or transposed—of lowpass filter can be used to implement a decimator, but the FIR direct-form-1 lends itself to further simplification by allowing the downsampler to be moved inside of the filter. The downsampler can be replaced by N downsamplers, with one placed at the output of each multiplier, as shown in Figure 52.3(a). Whether the downsampler discards a sample before or after it is multiplied by h_n makes no difference in the result, so the order of multiplication and downsampling can be commuted to obtain the structure shown in Figure 52.3(b). To generate each output sample, this structure must perform N multiplications. The output sample rate is F_S/M, so the multiplication burden imposed by the filtering is only NF_S/M multiplications per second. The concept of moving the downsampling to occur prior to the coefficient multiplication

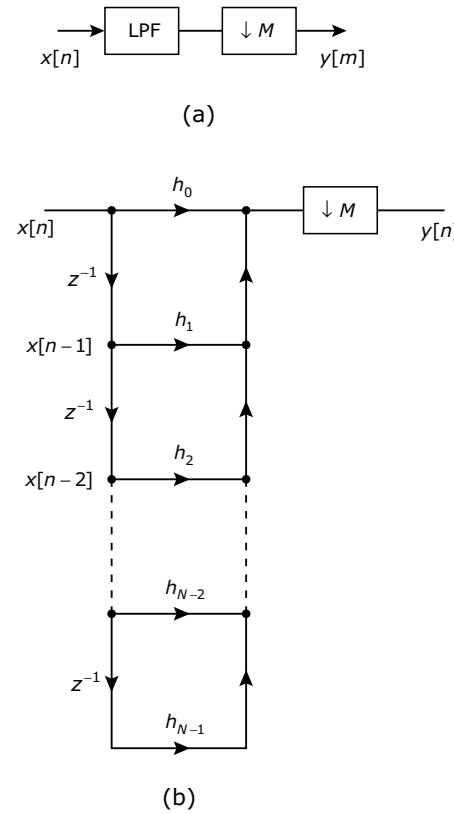


Figure 52.2 Direct form decimator structures: (a) block diagram and (b) with signal flow graph showing detail for direct form filter implementation

can be extended to the structures that are presented in Note 32 for the special case of symmetric responses for linear-phase FIR filters. Figure 52.4 shows the locations of the downsamplers that must be added to contruct a decimator built upon the FIR structure given in Figure 32.7 of Note 32.

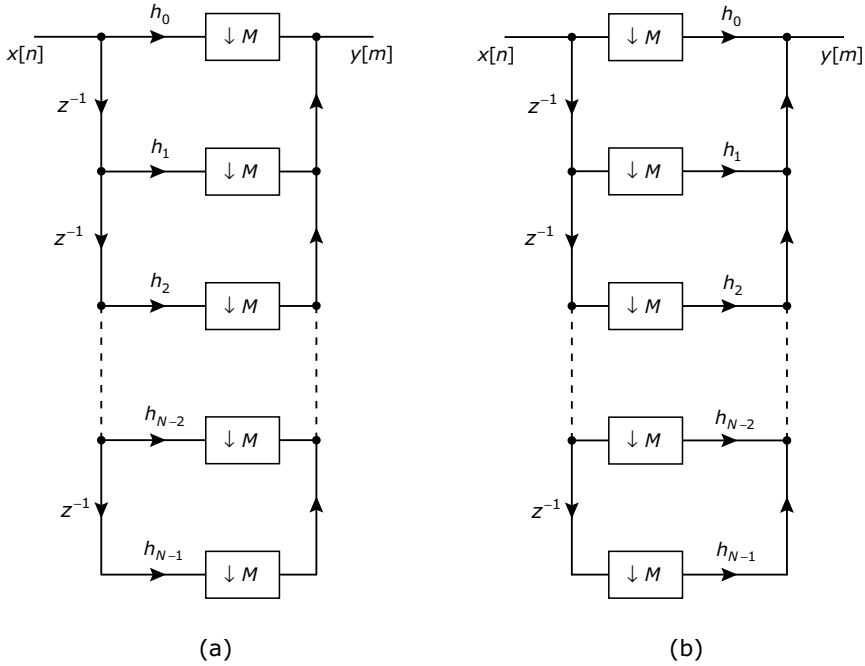


Figure 52.3 Decimator structures: (a) direct-form-1 FIR filter with downsampling moved into each multiplier branch; (b) efficient structure with downsampling and multiplication transposed in each branch

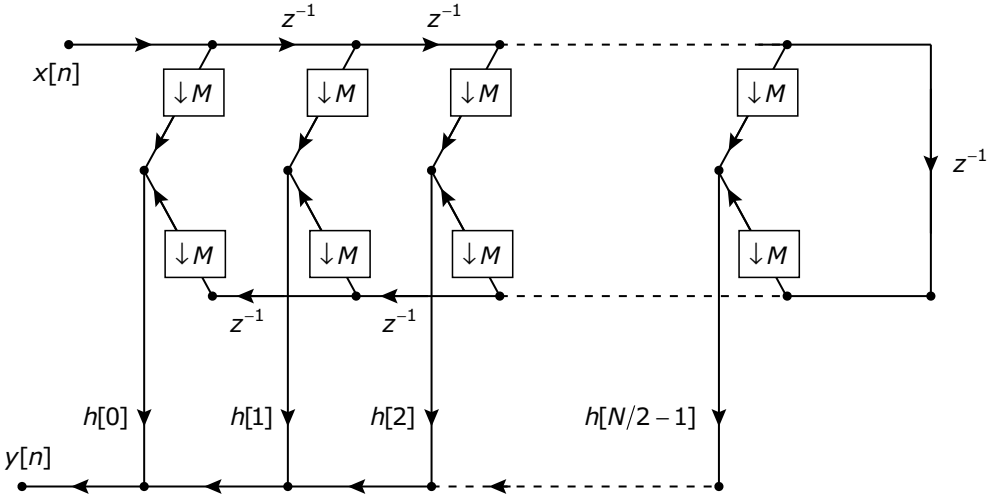


Figure 52.4 Efficient decimator structure that incorporates a Type 2 (even length, even symmetry) linear-phase FIR filter

Multistage Decimators

When large reductions in sampling rates are required, the design of adequate decimation filters becomes more difficult, usually resulting in filters with a prohibitively large number of taps. This note introduces multistage decimators that accomplish large reductions in sampling rate as a sequence of several smaller reductions. In a properly designed multistage approach, the filters for all the stages have a total tap count that is significantly lower than the filter tap count needed to obtain similar performance from a single-stage implementation.

When the desired decimation factor, M, can be expressed as a product of I positive integer factors

$$M = M_1 M_2 \cdots M_I$$

the desired decimation can be realized as a cascade of I decimators, each decimating by one of the factors, M_i. Compared to a single-stage decimator design, a design that adopts a multistage approach usually results in a reduced total computational burden, reduced memory usage, simpler filter designs, and reduced sensitivity to finite word-length effects in the implementation of the filters.

In lowpass FIR filter designs, the number of required taps, N, is approximately inversely proportional to the normalized transition bandwidth

$$N^{-1} \propto \frac{(f_s - f_p)}{F_S}$$
$$\text{or} \quad N \propto \frac{F_S}{(f_s - f_p)} \qquad (53.1)$$

where f_p is the passband edge fequency in Hz, f_s is the stopband edge frequency in Hz, and F_S is the sampling rate in samples per second. To reduce N, we need either to decrease F_S or to increase the difference between f_s and f_p. In a single-stage decimator we don't have the freedom to make either change. However, in a two-stage decimator, there is some flexibility in the choice of the first-stage stopband edge frequency, f_{S1}.

Figure 53.1 illustrates the important frequency relationships in a two-stage decimator design. The folding frequency of the first stage of the decimator is equal to half the sample rate, F_1, at the first-stage output.

As shown in the figure, all frequency content above $F_1/2$ remaining in the signal after the first-stage filter is aliased into frequencies below $F_1/2$ once downsampling by M_1 is performed. Frequencies from 0 to f_{S2} (which comprise the ultimate output spectrum) must be protected from aliasing. However, we do not care about aliasing in frequencies from f_{S2} up to $F_1/2$, because all these frequencies will be removed by the second-stage filter. Therefore, to avoid aliasing of first-stage transition-band frequencies into the output of the second-stage decimator, the stopband edge for the first-stage filter, f_{S1}, must be set to a value that does not exceed $F_1 - f_{S2}$. Figure 53.2 shows the limiting case in which $f_{P1} = f_{P2}$ and $f_{S1} = F_1 - f_{S2}$.

The width of the transition band for the first-stage filter can be as wide as $\Delta F_1 = F_1 - f_{S2} - f_{P2}$, which, in most cases, is significantly larger than the transition width of $\Delta F = f_S - f_P$ for a single-stage implementation. From Eq. (53.1), it is clear that a larger transition width results in a lower number of required taps.

For the second stage of a two-stage implementation, the edges of the transition

band are the same as they would be for a single-stage design:

$$f_{P2} = f_P \quad \text{and} \quad f_{S2} = f_S$$

Because the sample rate going into the second-stage filter has already been reduced by a factor of M_1, that is, $F_2 = F_1/M_1$, we can conclude from Eq. (53.1) that, except for changes to ripple specifications that may be needed due to multiple stages, the number of taps needed for the second stage can be approximated by

$$N_2 \cong \frac{N}{M_1}$$

In most practical cases, $N_1 + N_2$ is significantly smaller than N.

Because a total of N multiplications would be needed in the single-stage design to generate each decimator output sample, the average required multiplication rate can be approximated as

$$R = \frac{NF_S}{2M} \tag{53.2}$$

In comparison, the average required multiplication rate for the two-stage design would be

$$R = \frac{N_1 F_S}{2M_1} + \frac{N_2 F_S}{2M_1 M_2} \tag{53.3}$$

A specific case of two-stage decimator design is provided in Example 53.1.

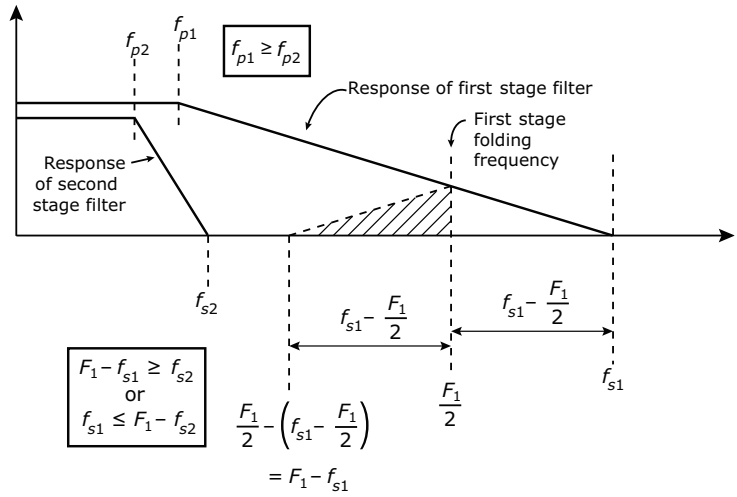


Figure 53.1 Crtical frequency relationships for a two-stage decimator

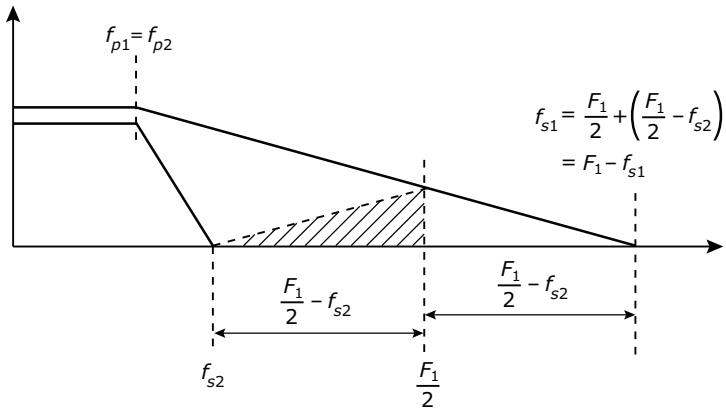


Figure 53.2 Limiting case for critical frequency relationships for a two-stage decimator

Example 53.1

Two-Stage Decimator

Consider the case of a DAT-quality digitizer that operates at 48,000 samples per second, used in a system that digitizes both voice and music. When the system is used to digitize voice, the digitized signal must be decimated from the sampling rate of 48 kHz to a more economical rate of 8 kHz for subsequent input to a digital telephone system. After downsampling the digitizer output by a factor of 6, the folding frequency is 4 kHz, so if we assume that the signal of interest occupies frequencies up to 3.5 kHz, the anti-aliasing filter must be designed for a transition band extending from 3.5 kHz to 4 kHz. Assume that we require less than 1/2 dB of passband ripple and 100 dB of stopband attenuation. The following MATLAB fragment uses the function `firpmord` to estimate the required order for an equiripple FIR filter that meets these requirements:

```
>> f = [3.5 4];
>> fs = 48;
>> a = [1 0];
>> d = 10^(0.5/20);
>> dp = (d-1)/(d+1);
>> ds = 10^(-100/20);
>> dev = [dp ds];
>> [n,fo,ao,w] = ...
>>     firpmord(f,a,dev,fs);
>> n
n =
322
```

Using the structure from Figure 52.3(b), the multiplication burden for 322 taps at an output sample rate of 8 kHz is 2.576 × 106 multiplications per second. This burden is not prohibitively large, but we can obtain a more economical design by breaking the decimation by a factor of 6 into two stages: (1) a decimation with M1 = 3 followed by (2) a decimation with M2 = 2. The first-stage filter has a transition band extending from 3.5 kHz to 8 kHz rather than from 3.5 kHZ to 4 kHz, as it is for the single-stage design. The required filter order estimated by `firpmord` is n1 = 35. The transition band for the second-stage filter extends from 3.5 kHz to 4 kHz, and the sampling rate is 16 kHz. The second-stage filter order estimated by `firpmord` is n2 = 107. The computational burden for the two-stage approach is

$$(35)(16{,}000)+(107)(8{,}000)$$

$$= 1.416 \times 106 \text{ multiplications per second.}$$

The two-stage approach has a computational burden that is reduced by a factor of 1.8 relative to the single-stage design. The original ripple specifications were somewhat conservative for a telephone application, so a practical specification would make some compromises to achieve an even more economical design.

Polyphase Decimators

An M-to-1 decimator can be implemented using the polyphase structure shown in Figure 54.1. If $h[n]$ is the response for the decimation filter when implemented in one of the conventional single-stage structures presented in Note 32, then each individual $p_\rho[n]$ in Figure 54.1 is a different downsampled version of $h[n]$ obtained as

$$p_\rho[n] = h[nM + \rho] \tag{54.1}$$

for

$$n = 0,1,2,\ldots,(N/M-1)$$
$$r = 0,1,2,\ldots,(M-1)$$

54.1 Why It Works

Consider a conventional decimator structure consisting of a lowpass filter followed by a downsampler. Out of every N output samples generated by the filter, $N(M-1)/M$ samples are discarded by the downsampler.

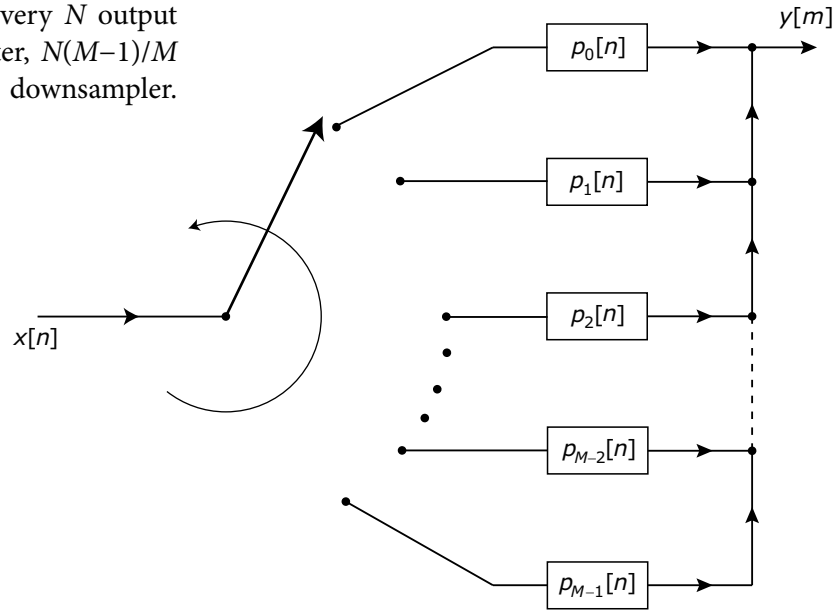


Figure 54.1 An M-to-1 decimator implemented as a polyphase structure with a commutator

The polyphase approach is motivated by a desire to avoid generating those filter-output samples that are destined to be discarded by the downsampler.

For a signal that will be downsampled by a factor of M, there are M different possible "phasing" alignments between the filter output samples and the filter coefficients. As depicted in Figure 54.2, for the specific case of $M = 3$ and $N = 7$, the possible alignments consist of:

- "Keeper" outputs align with filter's coefficients, $h[n]$, for which $n \equiv 0$ (modulo M).
- "Keeper" outputs align with filter's coefficients, $h[n]$, for which $n \equiv 1$ (modulo M).
- "Keeper" outputs align with filter's coefficients, $h[n]$, for which $n \equiv M-1$ (modulo M).

One way to eliminate the generation of "non-keeper" samples involves splitting the decimator into M parallel branches, as shown in Figure 54.3. Each branch corresponds to one of the possible alignments between the filter output sequence and the filter coefficients. The filter for branch ρ is defined by the sequence of coefficients from $h[n]$ for which $n \equiv \rho$ (modulo M). Each branch only computes results that will contribute to "keeper" outputs. The non-keeper samples have been eliminated from filter processing by moving the downsampling operation from after the filtering to before the filtering. The delays along the left-hand side of the structure cause the M subsequences to be extracted in the right order needed to create the ultimate output $y[m]$. For any given value of m, only one of the branches contributes a non-zero sample to the summation that creates $y[m]$.

The delays and downsamplers on the left side of the diagram are performing a branch demultiplexing operation that corresponds to the commutator shown in Figure 54.1.

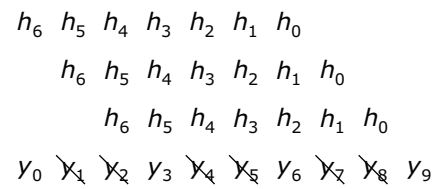


Figure 54.2 Possible phasing relationships between the decimation filter's output samples and coefficients for the case of $M = 3$ and $N = 7$

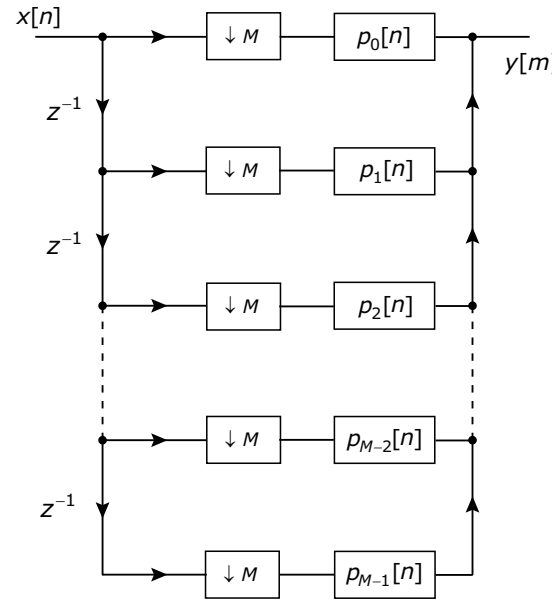


Figure 54.3 An M-to-1 decimator implemented as a polyphase structure

Interpolation Fundamentals

Interpolation is the name given to the process of increasing a discrete-time signal's sampling rate, while leaving all other characteristics of the signal unchanged to the maximum extent possible. Normally, the goal in designing a DSP system is to reduce the computational burden in the system by using the lowest possible sampling rate that provides a signal quality adequate for the targeted application. Given this goal, why concern ourselves with techniques that are used to increase the sampling rate of a signal that has already been sampled at an adequate rate? There are several situations in which interpolation of a signal may be required:

1. The signal will be subjected to some nonlinear processing that will increase its bandwidth. Before the bandwidth is increased, interpolation typically is used to increase the sampling rate to a level that is appropriate for the new, higher-signal bandwidth.
2. The signal will be added to another discrete-time signal that has a higher sampling rate.
3. There will be occasions when a signal's sample rate must be changed by a noninteger factor. The sample rate can be changed by a rational factor, L/M, by first interpolating by a factor of L and then decimating by a factor of M.

The basic process for interpolation is provided in Processing Sequence 55.1. The upsampling step is straightforward, but proper specification of the filter depends upon understanding how the upsampling changes the signal's spectrum.

Processing Sequence 55.1

Signal Interpolation

Consider the discrete-time signal depicted in Figure 55.1(a), having the periodic spectrum shown in Figure 55.1(b). The spectral images are centered on integer multiples of the sampling rate, F_s. The goal of the interpolation process is to increase the signal's sample rate by some factor, say, L, in such a way that the new signal's spectrum will be as depicted in Figure 55.2(b). The spectral images in the new spectrum are the same as the images in the original spectrum, but they are centered on integer multiples of the new sampling rate, LF_s. The desired result can be obtained by performing the following steps:

1. Insert $L-1$ zero-valued samples between each sample of the original signal, as depicted in Figure 55.3. This step is often referred to as either *upsampling* or *sample-rate expansion*.
2. Pass the upsampled signal through a digital lowpass filter that is designed to pass the baseband image and all of the images that are centered on integer multiples of LF_s, while rejecting all of the other images in the signal's spectrum. This filter is sometimes referred to as the *anti-imaging* filter.

55.1 Spectral Impacts of Upsampling

Some texts incorrectly claim that the upsampling process *compresses* the signal spectrum. This idea likely spawned from the common practice of presenting spectra that are normalized for $T = 1$. Here are the facts:

- Interpolation decreases the actual value for T, but this fact can be easily masked by the usual normalization. If the signal's normalized spectrum before upsampling is compared to the signal's spectrum after upsampling, with this "after" spectrum being normalized for $T_{\text{new}} = 1$, then it does indeed *appear* that the signal's spectrum has been compressed.
- If upsampling really did compress the original signal's spectrum, it would not be possible to recover the original signal via lowpass filtering. A lowpass filter only attenuates signal components outside of the passband; it would not be able to decompress a compressed spectrum.
- What really happens in interpolation is that the fundamental frequency range of the signal is increased from $\pm F_s/2$ to $\pm LF_s/2$.
- In the original signal's spectrum, only the baseband image fits within the fundamental frequency range, as depicted in Figure 55.4.
- In the upsampled signal's spectrum, the fundamental frequency range has been widened to the point that it contains a total of L images, including the baseband image as shown in Figure 55.5. The spectrum has not been compressed; it just appears compressed because the "window of visibility" has been made wider.

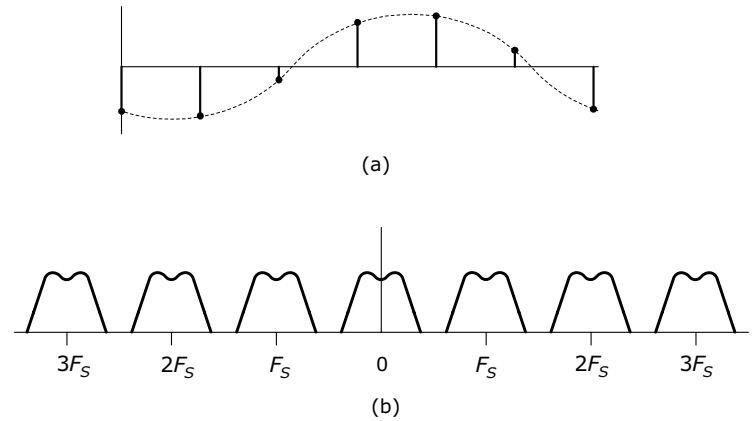


Figure 55.1 Discrete-time signal (a) and its spectrum (b)

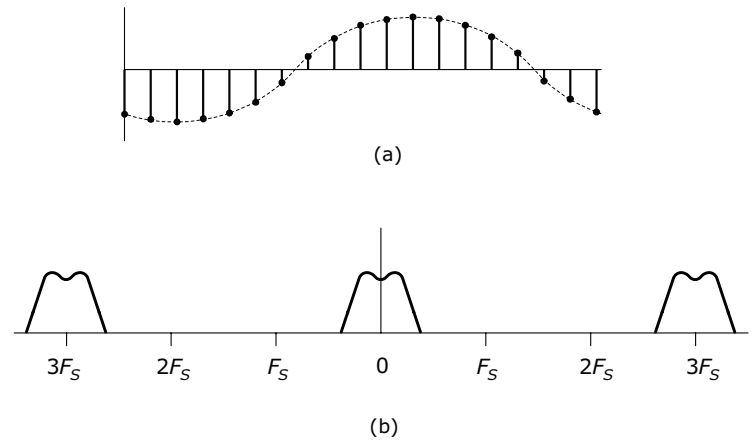


Figure 55.2 Interpolated signal (a) and its spectrum (b)

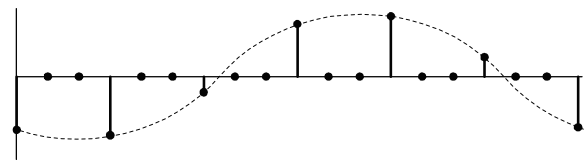


Figure 55.3 Discrete-time signal after upsampling

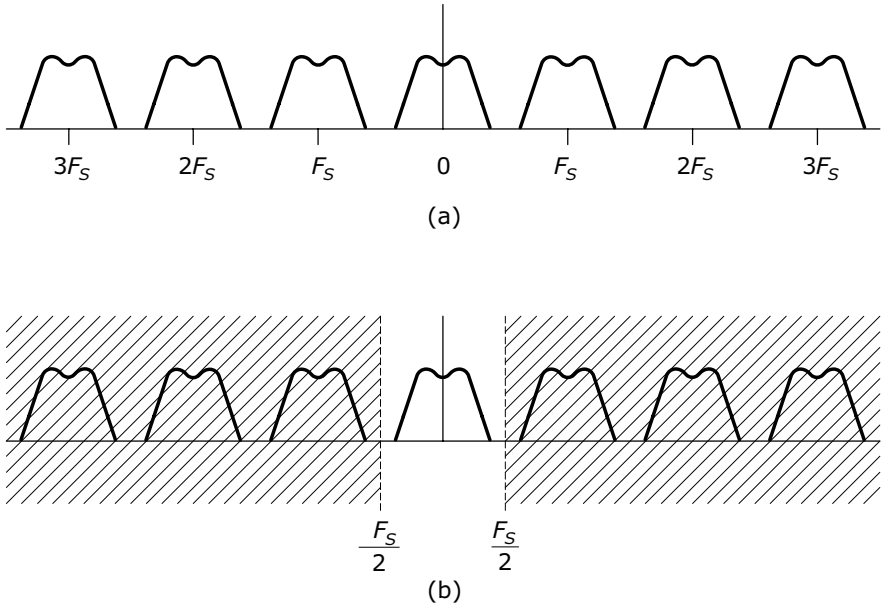


Figure 55.4 Spectrum of a discrete-time signal showing (a) spacing of its periodic images, and (b) relationship between the images and the fundamental frequency range $\pm F_s/2$

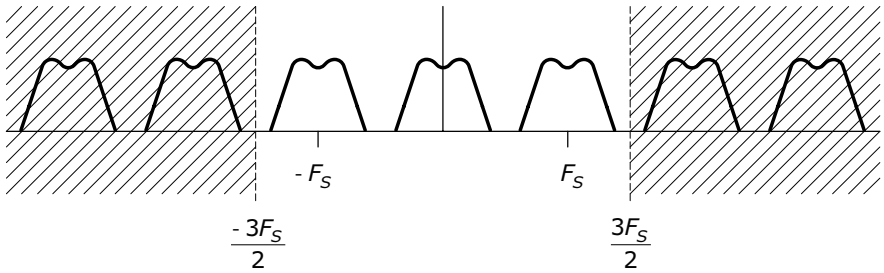


Figure 55.5 Spectrum of a discrete-time signal after upsampling by a factor of $L = 3$, showing the relationship between the spectral images and the new, fundamental frequency range $\pm 3F_s/2$

55.2 Designing the Anti-Imaging Filter

The key challenge in designing an interpolator is orchestrating the combination of sampling rates and signal bandwidths so that a reasonably efficient filter can be designed that will pass the baseband image without distortion while severely attenuating the unwanted spectral images.

Designing the anti-imaging filter for an interpolator can sometimes be a difficult task, especially when the value of L is large. The filter's passband edge must be high enough to pass the complete baseband image, while the stopband edge must be low enough to severely attenuate the image immediately adjacent to the baseband image.

In terms of absolute frequencies, the band edges for the anti-imaging filter would be similar to the band edges for an analog anti-alaising filter that might have been used to obtain the baseband image by sampling an analog signal at a rate of F_s. However, the implementation of the anti-imaging filter is more costly than we might otherwise expect because the anti-imaging filter must operate at the new sample rate of LF_s.

55.3 Efficient Interpolator Structures

The direct form for implementing an interpolator is shown in Figure 55.6. Part (a) of the figure shows the usual block diagram representation for an interpolator, while part (b) uses a signal flow graph to show the transposed direct form implementation for the anti-imaging filter. The filter must perform N multiplications to produce each output sample. If the input sample rate is F_s, the output sample rate is LF_s, and the multiplication burden imposed by the filter is NLF_s multiplications per second.

Design Strategy 55.2

Anti-Imaging Filter Parameters

1. Select a passband edge frequency high enough to pass all frequencies in the desired baseband image.
2. Select a stopband edge frequency equal to or less than the original folding frequency, $F_s/2$.
3. Select a stopband attenuation level that is adequate for the intended application.

The structure shown in Figure 55.6(b) lends itself to further simplification by allowing the upsampler to be moved inside the filter. The upsampler can be replaced by N upsamplers, with one placed at the input of each multiplier, as shown in Figure 55.7(a). The zero-valued samples inserted by the upsampler will still be zero-valued at the output of the multiplier, so the order of upsampling and multiplication can be commuted to obtain the structure shown in Figure 55.7(b). The filter in this structure must perform N multiplications each time a new input sample is clocked in, thus, the multiplication burden imposed by the filter is NF_s multiplications per second.

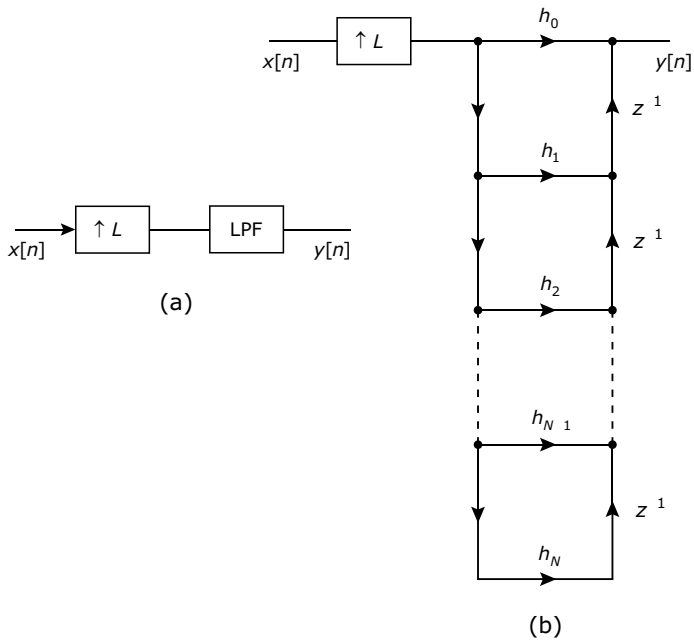


Figure 55.6 Diagrams of interpolator structure: (a) basic block diagram (b) filter represented as transposed direct-form structure

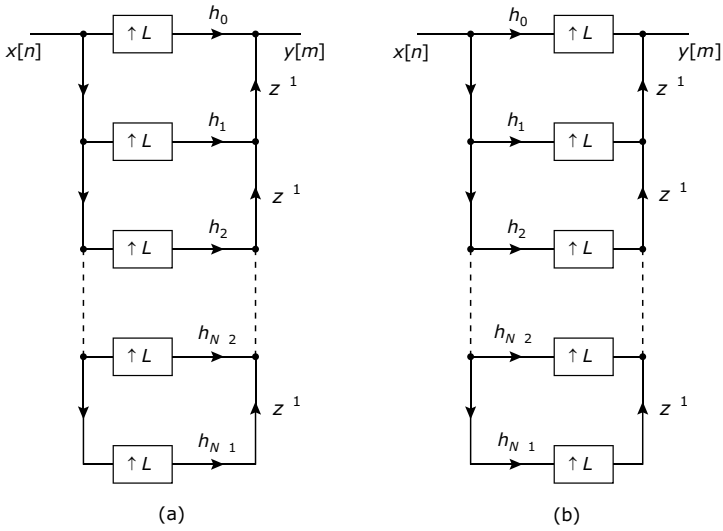


Figure 55.7 FIR interpolator structures: (a) direct structure with upsampling moved inside the filter, (b) efficient structure with order of upsampling and multiplication commuted

References

1. R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Prentice Hall, 1983.
2. R. A. Roberts and C. T. Mullis, *Digital Signal Processing*, Addison-Wesley, 1987.
3. C. B. Rorabaugh, *DSP Primer*, McGraw-Hill, 1999.

Multistage Interpolation

When the desired interpolation factor, L, can be expressed as a product of I positive integer factors, as

$$L = L_1 L_1 \cdots L_I$$

the desired interpolation can be realized as a cascade of I interpolators, each interpolating by one of the factors, L_i. Compared to a single-stage interpolator design, a design that adopts a multistage approach usually results in a reduced total computational burden, reduced memory usage, simpler filter designs, and reduced sensitivity to finite word-length effects in the implementation of the filters.

In lowpass FIR filter designs, the number of required taps, N, is approximately inversely proportional to the normalized transition bandwidth:

$$N^{-1} \propto \frac{(f_s - f_p)}{F_S} \quad \text{or} \quad N \propto \frac{F_S}{(f_s - f_p)} \tag{56.1}$$

where f_p is the passband edge fequency in Hz, f_s is the stopband edge frequency in Hz, and F_s is the sampling rate (after upsampling) in samples per second. To reduce N, we need either to decrease F_s or increase the difference (f_s–f_p). In a single-stage interpolator, we don't have the freedom to make either change. However, in a two-stage interpolator, there is some flexibility.

56.1 Designing the Anti-Imaging Filters

Figure 56.1 illustrates the important frequency relationships in the first stage of a multistage interpolator design. The figure depicts the signal spectrum after the signal has been upsampled by a factor of 2. The signal's fundamental frequency range has been widened to the point where it includes two half-images in addition to the baseband. As shown in the figure, the stopband edge frequency is placed at the lower edge of the first image.

The frequencies for passband edge and stopband edge are the same as they would be for a single-stage interpolator. However, because the filter must operate at a rate that is only twice the original sample rate, the number of required taps is much lower than for a similar filter operating at the ultimate interpolated rate. As depicted in Figure 56.1(b), after filtering, only the baseband remains in the frequency interval that is supported by the new sampling rate.

Figure 56.2 depicts the critical frequencies for the design of the second-stage filter. The figure shows the spectrum after the second stage of upsampling has been completed. The upsampling rate is not specified, but the figure shows a supported bandwidth of at least $\pm(3F_{s_1}/2)$, so F_{s_2} must be greater than $3F_{s_1} = 6\,F_s$.

Example 56.1 on the next page uses the requirements for a CD-to-DAT converter to illustrate the savings that can be realized with multistage interpolator designs.

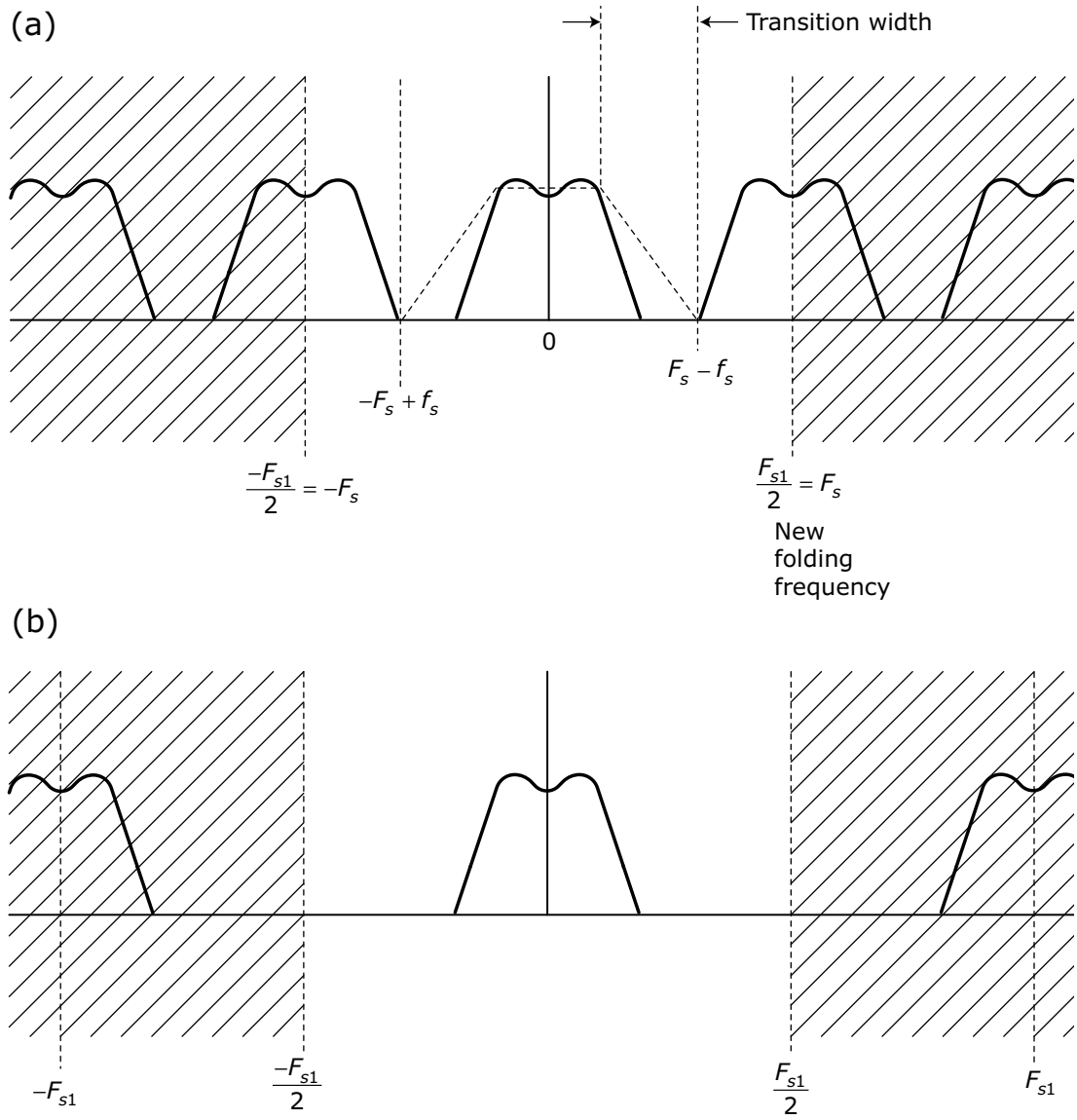


Figure 56.1 Frequency relationships involved in designing the first-stage filter for a multistage interpolator: (a) After upsampling by a factor of 2, the new sampling rate supports a wider frequency range (unshaded interval) that now includes two half-images in addition to the baseband spectrum. (b) After the first anti-imaging filter, only the baseband spectrum remains within the frequency range supported by the sampling rate.

Example 56.1

Interpolator Filters Can Impose Large Computational Burdens

Consider the classic example of converting a compact disc (CD) signal into a digital audio tape (DAT) signal. The first step in this conversion involves interpolation by a factor of 160 to convert the CD sample rate of 44.1 kHz into an intermediate rate of 7056 kHz. The folding frequency is 22.05 kHz, so if we assume that the signal of interest occupies frequencies up to 20 kHz, the anti-imaging filter must be designed for a transition band that extends from 20 kHz to 22.05 kHz. The following MATLAB fragment uses the function `firpmord` to estimate the required order for an equiripple FIR filter that meets these requirements:

```
>> f = [20 22.05];
>> f = f/7056;
>> a = [1 0];
>> dev = [0.01 0.001];
>> [n,fo,ao,w] = firpmord(f,a,dev);
>> n
n =
17494
```

Implementing an FIR filter with 17,494 taps is not a good idea. Using the structure from Figure 55.7(b), the multiplication burden for 17,494 taps at an input sample rate of 44.1 kHz is 7.715×10^8 multiplications per second. In addition to this huge computational burden, the numerical effects in a filter this large would make it nearly impossible to achieve the desired response. The large number of taps is required to achieve such a narrow transition band at such a high sample rate.

The filter design becomes much more reasonable if we break this interpolation for $L = 160$ into two stages: (1) an interpolation with $L_1 = 2$ followed by (2) an interpolation with $L_2 = 80$. The first-stage filter requirements are similar to the single-stage filter requirements, with one important exception: The output sampling rate is only 88.2 kHz instead of 7056 kHz. The filter order estimated by `firpmord` is $n_1 = 219$. The output of the first-stage filter has a spectrum as depicted in Figure 56.3. The transition band for the second-stage filter extends from 20 kHz to 66.15 kHz. The second-stage filter order estimated by `firpmord` is $n_2 = 778$. The multiplication burden for the two-stage approach is then computed as

$$(219)(44{,}100)+(778)(88{,}200) = 7.83 \times 10^7 \text{ mult/sec.}$$

This two-stage implementation is still not a trivial undertaking, but the computational burden has been decreased by a factor of 9.85. Further improvements can be obtained by going to a three-stage design with $L_1 = 2$, $L_2 = 8$, and $L_3 = 10$. The first stage remains the same as for the two-stage design. The second-stage filter has a sample rate of 705.6 kHz and a transition band extending from 20 kHz to 66.15 kHz. The estimated filter order is $n_2 = 78$. The third-stage filter has a sample rate of 7056 kHz and a transition band extending from 20 kHz to 683.55 kHz. The estimated filter order is $n_3 = 54$. The multiplication burden for the three-stage approach is

$$(219)(44{,}100) + (78)(88{,}200) + (54)(705{,}600) = 5.464 \times 10^7 \text{ mult/sec.}$$

The three-stage approach has a multiplication burden that is reduced by a factor of 14 relative to the single-stage design. The original ripple specifications were somewhat conservative for an audio application, so a practical specification would make some compromises to achieve an even more economical design. Furthermore, the direct form interpolation implementation of Figure 55.7 was assumed for computing the computational burdens. Actual interpolators would be constructed using more efficient implementation structures (such as the ones shown in Figures 32.6 through 32.9), but multistage implementations would still be more economical than the corresponding single-stage implementations.

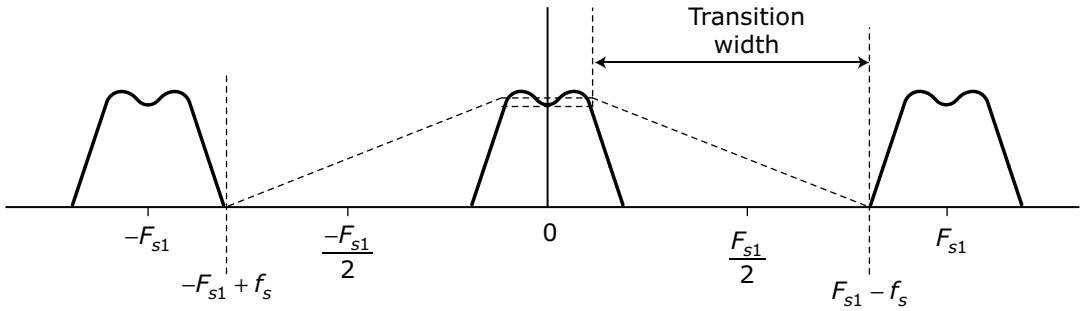


Figure 56.2 Frequency relationships involved in designing the second-stage filter for a multistage interpolator

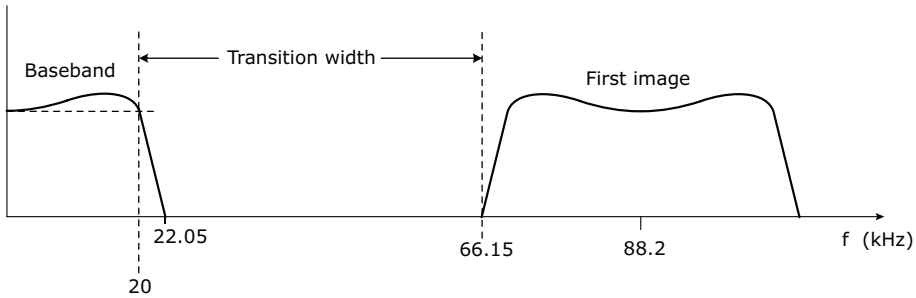


Figure 56.3 Spectrum of first-stage output from Example 56.1. Transition width shown is for second-stage anti-imaging filter.

References

1. R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Prentice Hall, 1983.
2. R. A. Roberts and C. T. Mullis, *Digital Signal Processing*, Addison-Wesley, 1987.
3. C. B. Rorabaugh, *DSP Primer*, McGraw-Hill, 1999.

Polyphase Interpolators

A 1-to-L interpolator can be implemented using the polyphase structure shown in Figure 57.1. If $h[n]$ is the response for the interpolation filter when implemented in one of the conventional single-stage structures presented in Note 32, then each individual $p_\rho[n]$ in Figure 57.1 is a different downsampled version of $h[n]$ obtained as

$$p_\rho[n] = h[nL + \rho] \qquad (57.1)$$

for

$$n = 0,1,2,\ldots,(N/L-1)$$
$$\rho = 0,1,2,\ldots,(L-1)$$

57.1 Why It Works

Consider a conventional interpolator structure consisting of an upsampler followed by a filter. As a result of the upsampler, the input signal to the filter consists of a sequence of single, non-zero samples separated by $L-1$ interspersed zero-valued samples. Of the N multiplications needed to generate each filter output sample, $N(L-1)/L$ are multiplications by zero. The polyphase approach is motivated by the goal of eliminating these multiplications by zero.

For a signal that has been upsampled by a factor of L, there are L different possible "phasing" alignments between the input samples and the filter coefficients. As depicted in Figure 57.2 for the specific case of $L = 3$ and $N = 7$, the possible alignments consist of the following.

- Non-zero inputs align with the filter's coefficients, $h[n,]$ for which $n \equiv 0$ (modulo L).
- Non-zero inputs align with the filter's coefficients, $h[n]$, for which $n \equiv 1$ (modulo L).

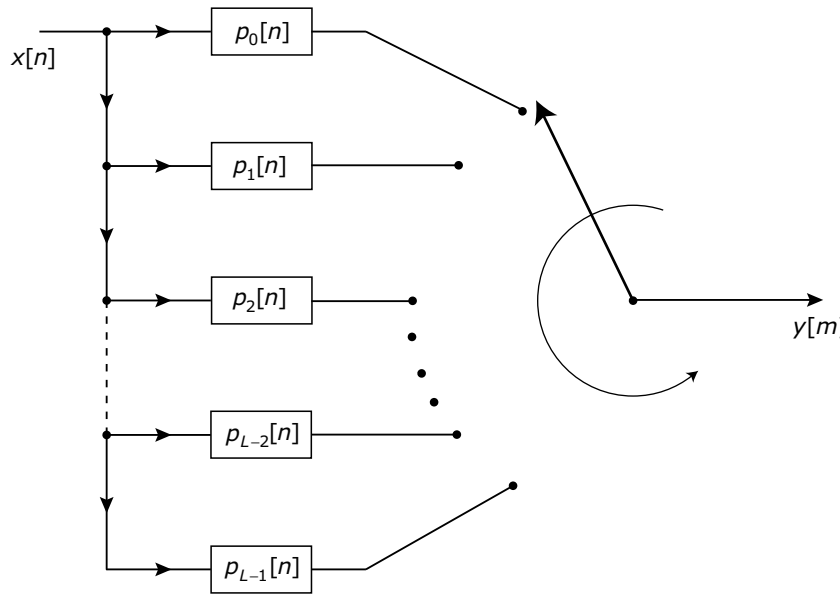


Figure 57.1 A 1-to-L interpolator implemented using a polyphase structure with a commutator

- Non-zero inputs align with the filter's coefficients, $h[n]$, for which $n \equiv L-1$ (modulo L).

One way to eliminate the multiplications by zero involves splitting the interpolator into L parallel branches, as shown in Figure 57.3. Each branch corresponds to one of the possible alignments between the input sequence and the filter coefficients. The filter for branch ρ is defined by the sequence of coefficients from $h[n]$ for which $n \equiv \rho$ (modulo L). The zero-valued samples have been eliminated from the filter inputs by moving the upsampling operation from before the filtering to after the filtering. The individual upsamplers in each branch effectively insert the zero-valued results that would have been created had the multiplications by zero not been eliminated. The delays along the right-hand side of the signal flow graph cause the L subsequences to dovetail in the right order needed to create the output $y[m]$. For any given value of m, only one of the branches contributes a non-zero sample to the summation that creates $y[m]$; the other $L-1$ branches contribute zero-valued samples that were injected by their upsamplers.

The interpolator structure of Figure 57.3 eliminates multiplication by zero, but a number of additions involving zero-valued terms still remain. These additions can be eliminated by recognizing that because only one branch output is non-zero for each output time, the additions and delays along the right side of the SFG are performing a branch-selection operation that corresponds to the commutator shown in Figure 57.1.

$$\begin{array}{ccccccccccc}
x_0 & 0 & 0 & x_1 & 0 & 0 & x_2 & 0 & 0 & x_3 & \\
h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & & & & \\
 & h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & & & \\
 & & h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & &
\end{array}$$

Figure 57.2 Possible phasing realtionships between the interpolation filter's input samples and coefficients for the case of $M=3$ with $N=7$

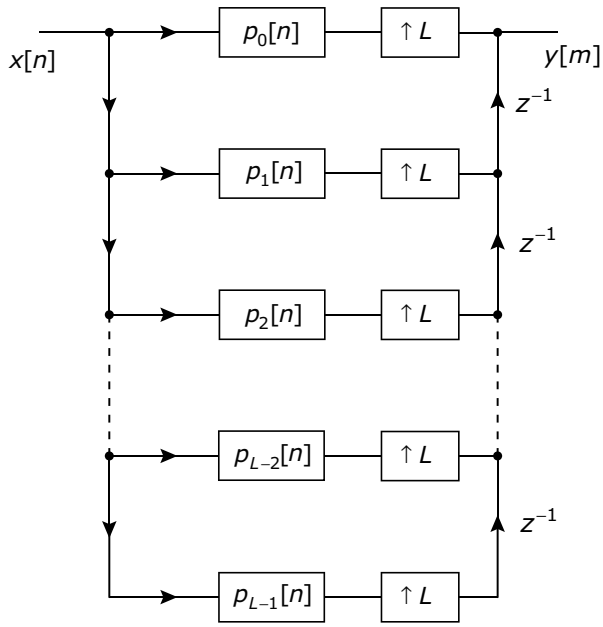


Figure 57.3 A 1-to-L interpolator implemented as a polyphase structure

References

1. M. G. Bellanger, G. Bonnerot, and M. Coudreuse, "Digital Filtering by Polyphase Network: Application to Sample-Rate Alteration and Filter Banks," *IEEE Trans. ASSP*, vol. 24, no. 2, April 1976, pp. 109–114.
2. R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Prentice Hall, 1983.
3. P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Prentice Hall, 1993.

Sampling Bandpass Signals

Design Rule 58.1

Bandpass Sampling Rates

When sampling a bandpass signal having an upper frequency of f_H and a bandwidth of B, the sampling rate, f_s, must satisfy

$$\frac{2f_H}{m+1} \le f_s \le \frac{2(f_H - B)}{m}$$

where m is an integer that satisfies

$$m \le \frac{(f_H - B)}{B}$$

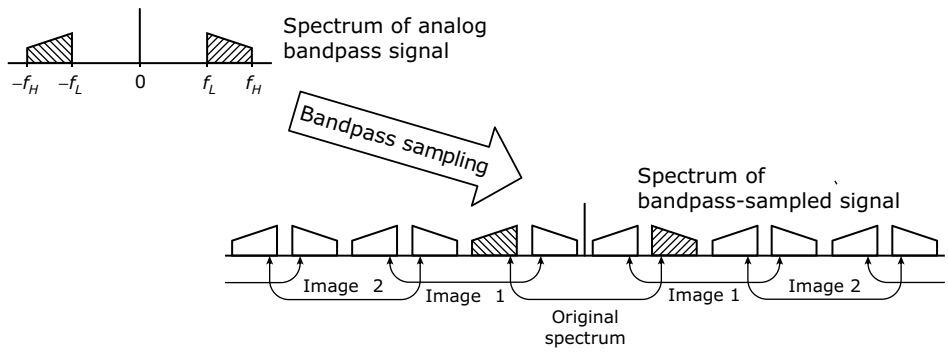
This note develops a set of rules for sampling bandpass signals at rates that are substantially lower than the rates developed of lowpass signals in Note 3.

Consider the bandpass signal depicted in Figure 58.1(a). The baseband sampling criteria developed in Note 3 would lead us to sample such a signal at some rate, f_s, selected such that $f_s > f_H$. The spectrum of the sampled signal would then be as shown in Figure 58.1(b), with the original $x = \pi$ spectrum replicated at intervals of f_s along the frequency axis. This approach works, but it uses an unnecessarily high

Key Concept 58.1

Uniform Bandpass Sampling

- Sampling a signal in the time domain creates periodic images of the original signal's spectrum in the frequency domain. These images are spaced along the frequency axis at intervals equal to the sampling rate. Therefore, as discussed in Note 3, a baseband signal must be sampled at a rate that is at least twice the highest frequency contained in the signal's spectrum to keep the images from overlaping.
- Uniform bandpass sampling avoids unnecessarily high sampling rates by exploiting the "empty space" between the two non-contiguous passbands that comprise a bandpass signal's spectrum. By samplng at a carefully selected rate, some of the sampling-created images of the passbands will be located in this empty space, thereby allowing alias-free sampling at rates significantly lower than what would be required using baseband sampling criteria.



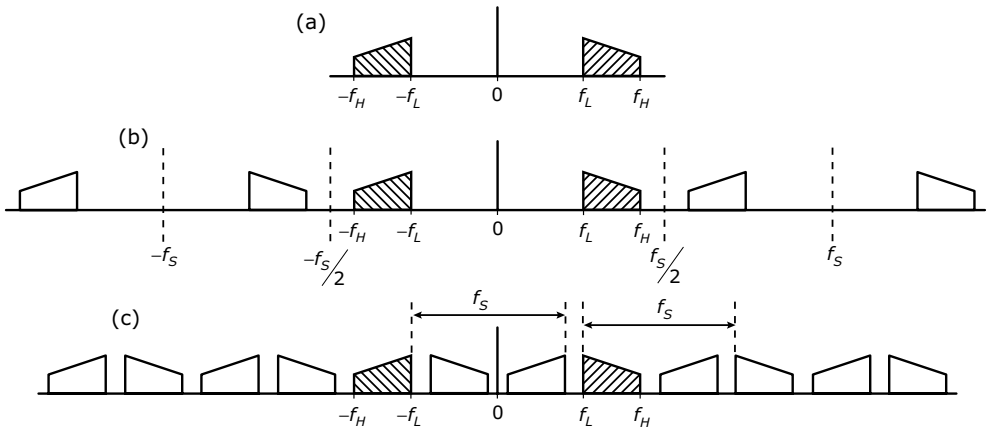


Figure 58.1 Comparison of spectra produced by baseband and bandpass sampling: (a) spectrum of original bandpass signal, (b) spectrum resulting from conventional baseband sampling, (c) spectrum produced by bandpass sampling

sampling rate that can impose high costs for A/D conversion and downstream processing—especially if the ratio between f_H and B is high. In most real-world bandpass signals, the ratio between the highest frequency, f_H, and the bandwidth, B, is much larger than depicted in the figure. In fact, it is usually impractical to draw a scale diagram that shows zero frequency, the upper frequency, and the passband of the signal in a single figure. A typical HF radio signal might have a bandwidth of 3 kHz and an upper frequency of 15 MHz; the span from zero frequency to 15 MHz would be 5000 times larger than the width of the passband. With all of the signal's useful information concentrated in a mere 3-kHz band, it seems unreasonable to sample such a signal at a rate that exceeds 3×10^7 samples per second. Alternative approaches have been developed for more efficient sampling of a bandpass signal.

Bandpass sampling schemes are based on the idea of sampling at a rate significantly lower than $2f_H$, with the rate carefully selected so that the passbands of the sampling-induced spectral images "interleave" with each other without overlapping, as shown in Figure 58.1(c).

58.1 Constraining the Sampling Rate

The empty space available for accommodating images of the original passbands extends from $-f_L$ to $-f_H$. After sampling, this space contains images of the original negative-frequency band and images of the original positive-frequency band. The number of negative- and positive-frequency images inside the interval $(-f_L, f_L)$ is always equal, and this number is obviously an integer.

Consider the negative-frequency band labeled A in Figure 58.2. The empty space begins at the right-hand edge of this band, and the mth image (labeled B in Figure 58.2) of this band will have its right-hand edge at a frequency of $-f_L+mf_s$. If m images are to fit within the available empty space, the right-hand edge of the mth image must not extend beyond the left-hand edge of the original positive-frequency band (labeled C in Figure

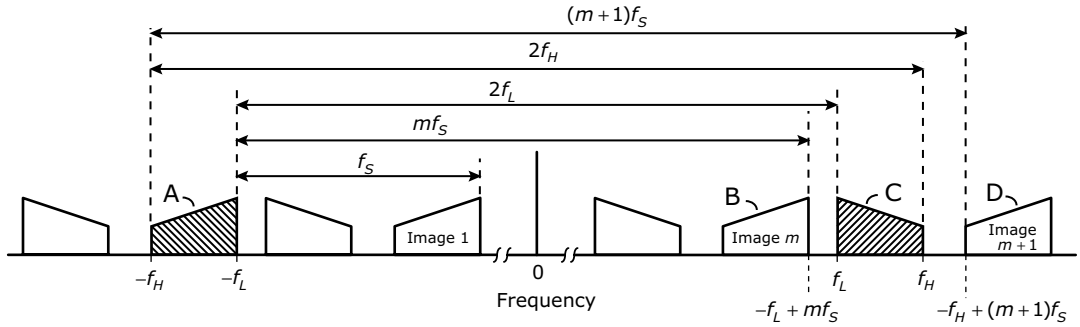


Figure 58.2 Frequency relationships between the passbands of an original bandpass signal (shaded) and the images (unshaded) created by uniform bandpass sampling

58.2). Expressed mathematically, this constraint becomes

$$-f_L + mf_s \le f_L \quad \text{or} \quad f_s \le \frac{2f_L}{m} \tag{58.1}$$

We must also ensure that the left-hand edge of the $(m+1)$-th image (labeled D in Figure 58.2) of the original negative-frequency band does not overlap with the right-hand edge of the original positive-frequency band (labeled C in Figure 58.2). The left-hand edge of the original negative-frequency band falls at $-f_L$ and the left-hand edge of the $(m+1)$-th image falls at $-f_H + (m+1)/f_s$. Thus, we can write

$$-f_H + (m+1)f_s \ge f_H$$
$$\text{or} \qquad f_s \ge \frac{2f_H}{m+1} \tag{58.2}$$

The constraints in Eqs. (58.1) and (58.2) were developed using only the orignial signal's two passbands and two images of the original negative-frequency passband. Nevertheless, because the sampled signal's spectrum is periodic and symmetric, these contraints assure that none of the negative-frequency passband's images overlap with any of the positive-frequency passband's images.

We can determine acceptable values for m by recognizing that $2m$ distinct passband images—m of these images corresponding to the negative-frequency passband and m corresponding to the positive-frequency passband—must fit within a frequency interval of $2f_L$. Because each of these images has a width of $B = f_H - f_L$, we can write

$$2mB \le 2f_L \quad \text{or} \quad m \le \frac{f_L}{B} \tag{58 3}$$

In cases where $f_L < B$, the value of m is zero, and uniform bandpass sampling reduces to the conventional baseband sampling process described in Note 3.

Bandpass Sampling: Wedge Diagrams

From Note 58, we know that in bandpass sampling, the sampling rate, f_s, must satisfy

$$\frac{2f_H}{m+1} \le f_s \le \frac{2(f_H - B)}{m} \qquad (59.1)$$

where m is an integer that satisfies

$$m \le \frac{(f_H - B)}{B} \qquad (59.2)$$

Many of the older discussions of bandpass sampling (such as [1]) include a diagram similar to Figure 59.1, which depicts only the lower bound imposed by Eq. (59.1). This diagram can be misleading to practitioners whose experience is limited to baseband sampling, where oversampling is often viewed as a good thing. It would be easy to select a sampling rate that satisfies the lower bound depicted in Figure 59.1, but that violates the upper bound imposed by Eq. (59.1). Vaughan, Scott, and White [2] appear to be the first authors to include a diagram similar to Figure 59.2, which depicts both the lower and upper bounds on the sampling rate given by Eq. (59.1).

In theory, we are free to choose any sampling rate that meets the constraints imposed by Eqs. (59.1) and (59.2). The largest choice of m allowed by Eq. (59.2) leads to the lowest sampling rates from Eq.(59.1), but choosing the largest m often can lead to a fragile sampling design. Section 59.2 explores how the choice of m can be best exploited for sampling designs that remain robust despite sample-clock inaccuracies and low-performance

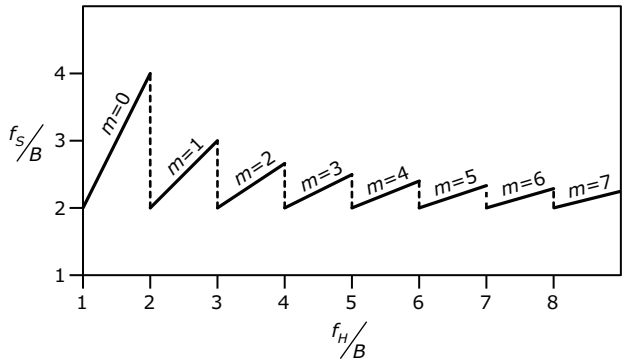


Figure 59.1 Minimum normalized sampling rate for bandpass signals versus normalized band position

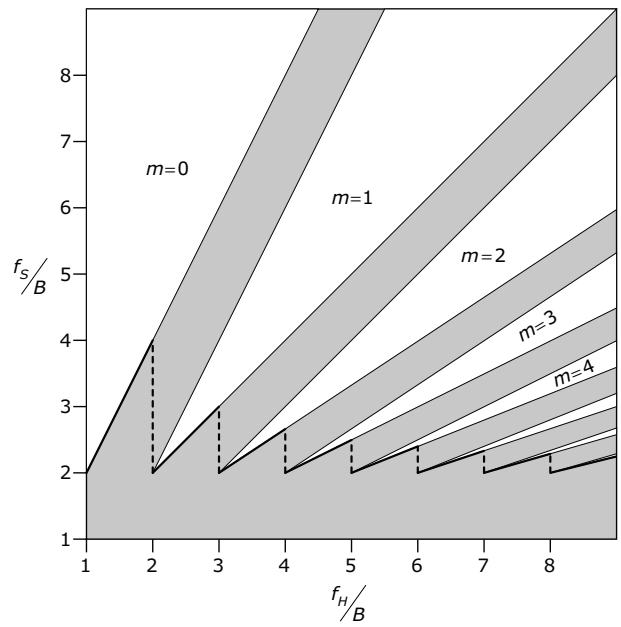


Figure 59.2 Plot of normalized sampling rate for bandpass signals versus normalized band position. The white areas indicate allowable combinations of sampling rate and band position. The shaded areas indicate combinations that exhibit aliasing.

anti-aliasing filters often encountered in cost-conscious hardware designs.

59.1 Interpreting the Wedge Diagram

Figure 59.3 is a vertically extended copy of the wedge diagram from Figure 59.2 that has been annotated to illustrate the points raised in the following discussion regarding interpretation of the wedge diagram.

- Determine the band location, f_H, and bandwidth, B, for the signal that is to be sampled, and then draw a vertical line at the value corresponding to the ratio f_H/B. The portions of this line that pass through the various wedges define the range of sampling rates that can be used for sampling the signal. For example, if we draw a vertical line at $f_H/B = 6$, as shown in Figure 59.3, the line passes through the tip of the wedge for $m = 5$, and the corresponding ordinate is $f_S = 2B$. The line also passes through the interior of the wedges corresponding to all values of $m < 5$.
- Draw horizontal lines from the points where the vertical line intersects with the sloped lines that form the wedge boundaries. These horizontal lines intersect with the vertical axis at the values of f_S/B that bound each interval of legal sampling rates.
- As shown in the figure, a signal with $f_H/B = 6$ can be sampled at normalized rates, f_S/B, that fall within any of the intervals (2.4, 2.5), (3, 3.33), (4, 5), or (6, 10). The case for $m = 0$ corresponds to the baseband sample rate of $f_S = 2f_H$, or $f_S/B = 12$.
- If the actual value for the ratio f_H/B differs from the nominal value of 6, the resulting impact on the sampled signal would be as though the vertical line were shifted either

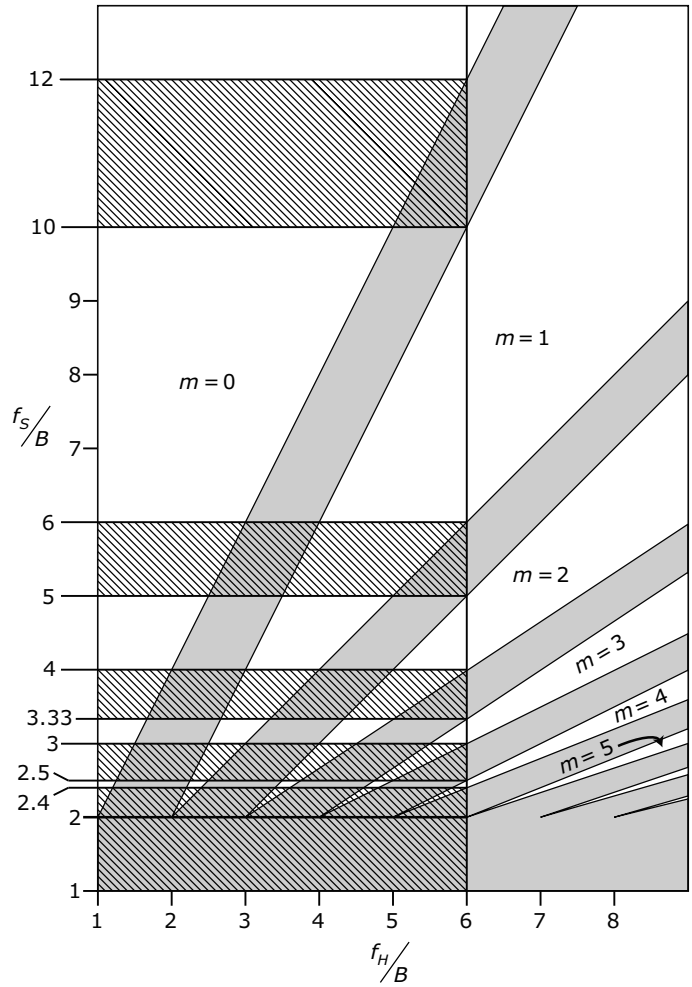


Figure 59.3 Plot of normalized sampling rate for bandpass signals versus normalized band position. The white areas indicate allowable combinations of sampling rate and band position. The shaded areas indicate combinations that exhibit aliasing. The vertical line at $f_H/B = 6$ passes through white areas for f_S/B in the intervals (2.4, 2.5), (3, 3.33), (4, 5), and (6, 10).

left (if actual ratio is lower than 6) or right (if the actual ratio is higher than 6). This shifting might cause our selected operating point to move into the shaded area where aliasing occurs. The effects of errors in f_H/B are quantified in Section 59.2.

- Similarly, if the actual value for the normalized sampling rate, f_s/B, differs from the value corresponding to our selected operating point, the actual operating point can move up or down along the vertical line and possibly move into a shaded area where aliasing occurs.

59.2 Sensitivity to Timing Errors

The top of Figure 59.4 shows an enlarged portion of a single wedge taken from a diagram like the one shown in Figure 59.2. As discussed in the previous section, for given values of f_H and B, the operating point is placed inside one of the unshaded wedges somewhere on the vertical line that passes through the appropriate value of f_H/B.

As shown in Figure 59.4, the operating point's location within the wedge defines four quantities: f_{SL}, f_{SH}, B_{GL}, and B_{GH}. The operating point corresponds to some nominal sampling rate, f_s. The value of f_{SL} indicates how much the sample clock can fall below this nominal rate before the operating point moves into the lower shaded region and aliasing occurs. Similarly, the value of f_{SH} indicates how much the sample clock can increase above the nominal rate before the operating point moves into the upper shaded region and aliasing occurs.

The bottom of Figure 59.4 shows the positive-frequency portion of the spectrum from Figure 58.2 in Note 58. The spacing between the various images in this spectrum is related to the quantities, B_{GL} and B_{GH}, that

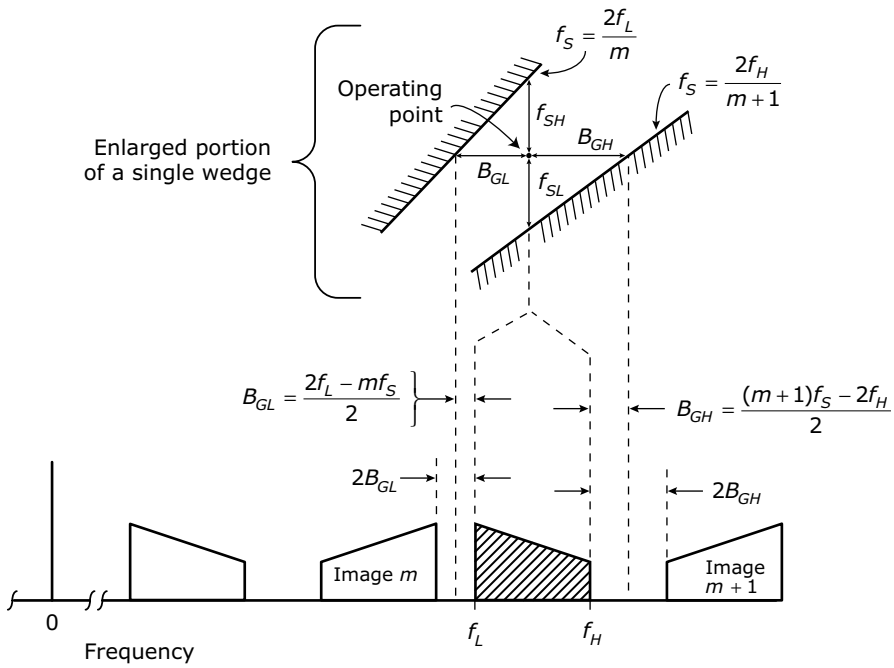


Figure 59.4 Frequency relationships between the passbands of an original bandpass signal (shaded) and the images (unshaded) created by uniform bandpass sampling

are defined by the operating point's horizontal distance from the wedge boundaries. The shaded band at the bottom of Figure 59.4 is the original signal's positive-frequency band. The gap between this band and the mth image of the original negative-frequency band is $2B_{GL}$, where B_{GL} is the horizontal distance from the operating point to the upper-left boundary of the wedge.

$$B_{GL} = \frac{2f_L - mf_s}{2} \tag{59.3}$$

The gap between the shaded band and the $(m + 1)$-th image of the original negative-frequency band is $2B_{GH}$, where B_{GH} is the horizontal distance from the operating point to the lower-right boundary of the wedge.

$$B_{GH} = \frac{(m+1)f_s - 2f_H}{2} \tag{59.4}$$

The empty intervals of length B_{GL} and B_{GH} surrounding each image band are sometimes referred to as *guard bands*.

References

1. R. S. Simpson and R. C. Houts, *Fundamentals of Analog and Digital Communication Systems*, Allyn and Bacon, 1971.
2. R. G. Vaughan, N. L. Scott, and D. L. White, "The Theory of Bandpass Sampling," *IEEE Trans. Signal Processing*, vol. 39, no. 9, September 1991, pp. 1973–1984.

Complex and Analytic Signals

An understanding of complex and analytic signals is an essential foundation for the development of many advanced signal processing techniques.

A real-valued signal always has a magnitude spectrum that is symmetric about $f = 0$, as depicted in Figure 60.1(a). Many different digital receiver architectures depend upon the abilities to (1) eliminate either all of the positive-frequency content or all of the negative-frequency content in such a signal's spectrum, and then (2) shift the surviving spectral band so that it is concentrated around zero frequency, as shown in Figure 60.1(c). A continuous-time signal having a spectrum that is zero-valued for all negative frequencies, as in Figure 60.1(b), is called an *analytic signal* because it can be represented mathematically as an analytic function [1].

This note first explores how the concepts of analytic signals can be extended to discrete-time signals and then it provides

Key Terminology

- An *analytic signal* has a spectrum that is zero-valued for all negative frequencies.
- The complex conjugate of an analytic signal has a spectrum that is zero-valued for all positive frequencies. Such a signal is sometimes described as *conjugate-analytic*.
- Continually describing some function, $x_a(t)$, as "the analytic signal corresponding to the real-valued signal $x(t)$" can become awkward. Boashash [2] coined the term *analytic associate*, which enables the simpler description "$x_a(t)$ is the analytic associate of $x(t)$."
- A discrete-time signal always exhibits periodicity in its spectrum and can therefore never be made truly analytic. Sometimes the term *analytic-like* is used to describe a discrete-time signal that is analytic over a single period of the spectrum from $f = -(2T)^{-1}$ to $f = (2T)^{-1}$, where T is the sample interval.

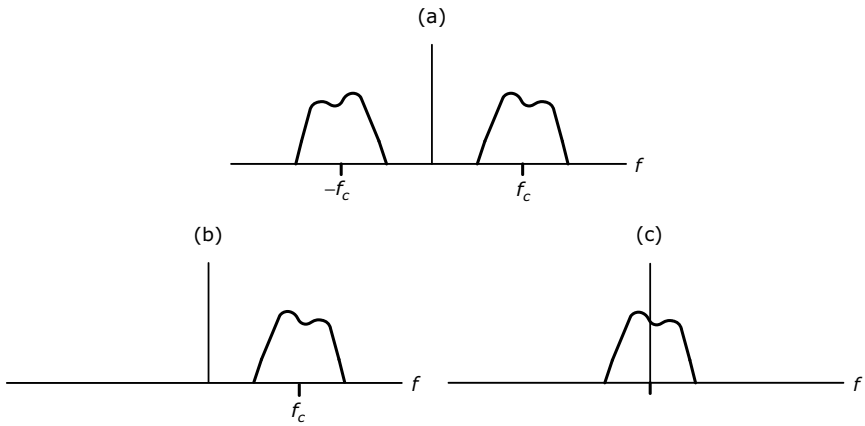


Figure 60.1 Spectra at different points in a digital receiver architecture: (a) spectrum for real-valued bandpass *intermediate-frequency* (IF) signal, (b) spectrum for the IF signal's analytic associate, (c) spectrum for complex-valued baseband signal

an overview of the various techniques for generating discrete-time *analytic-like*[1] signals. Each of these techniques is explored in depth in subsequent notes.

For any real-valued signal, $x(t)$, the corresponding analytic signal, $x_a(t)$, can be formed as

$$x_a(t) = \frac{1}{2}\Big[x(t) + j\mathcal{H}\{x(t)\}\Big] \qquad (60.1)$$

where $\mathcal{H}\{\cdot\}$ denotes the Hilbert transform[2]. Multiplication by j is equivalent to a phase shift of 90 degrees, so the net combined effect of $j\mathcal{H}\{\cdot\}$ in Eq. (60.1) is to create a phase shift of zero for positive frequencies in the spectrum of $x(t)$, and a phase shift of 180 degrees for negative frequencies in the spectrum of $x(t)$. When $j\mathcal{H}\{x(t)\}$ is added to $x(t)$, the negative-frequency components cancel and the positive-frequency components are doubled. If an analytic signal, $x_a(t)$, is formed using Eq. (60.1), it is possible to recover the original signal, $x(t)$, by simply taking the real part of $x_a(t)$

$$x(t) = \mathrm{Re}\{x_a(t)\}$$

A real-valued signal exhibits conjugate symmetry in its Fourier spectrum. Therefore, any signal not exhibiting conjugate symmetry in its spectrum must be complex-valued in time. This means that analytic signals and conjugate-analytic signals are always complex-valued in time. However, there can be signals that are complex-valued in time but that are neither analytic nor conjugate-analytic. A handy example is the complex-valued baseband signal having the spectrum depicted in Figure 60.1(c). The spectrum is not zero for either positive or negative frequencies, so the signal is neither analytic nor conjugate-analytic. However, the magnitude spectrum is not symmetric about $f = 0$, so the signal is complex-valued.

60.1 Discrete-Time Analytic Signals

The concept of an analytic signal cannot be extended directly from the continuous-time domain to the discrete-time domain. The DTFT spectrum of a discrete-time signal is always periodic, so it is not possible for the spectrum to be non-zero for positive frequencies and zero for all negative frequencies simultaneously. For discrete-time signals, the concept of analyticity is based on a DTFT spectrum like the one shown in Figure 60.2, where the spectrum is non-zero in all odd-numbered positive Nyquist zones[3] as well as in all even-numbered negative Nyquist zones. A discrete-time signal with such a spectrum is conventionally called an analytic signal, even though it cannot be represented as an analytic function. As noted previously, we follow Marple's practice and distinguish the discrete-time case as being *analytic-like*. Furthermore, we use *positive-like* to indicate frequencies in odd-numbered positive Nyquist zones and in even-numbered negative Nyquist zones. Similarly, we use *negative-like* to indicate frequencies in

1. Because the concept of analyticity is different for continuous-time and discrete-time signals, some authors distinguish the discrete-time case by using special notation or terminology. In [1], Marple initially uses the term *analytic-like*, and then transitions to using quotation marks around "analytic."

2. The usual convention for denoting the Hilbert transform is a script uppercase $\mathcal{H}$, despite the fact that this notation can be easily confused with the italic uppercase H typically used to denote the transfer function of a filter.

3. In discussing the figure, it is convenient to borrow some terminology from the A/D converter community. In discussions of A/D converters, the frequency band from 0 to $f_s/2$ is called the *first Nyquist zone*. The band from $f_s/2$ to f_s is called the *second Nyquist zone,* and so on. However, negative frequencies are not considered. We can extend the terminology by calling the band from 0 to $f_s/2$ the *first positive Nyquist zone* and calling the band from 0 to $-f_s/2$ the *first negative Nyquist zone*.

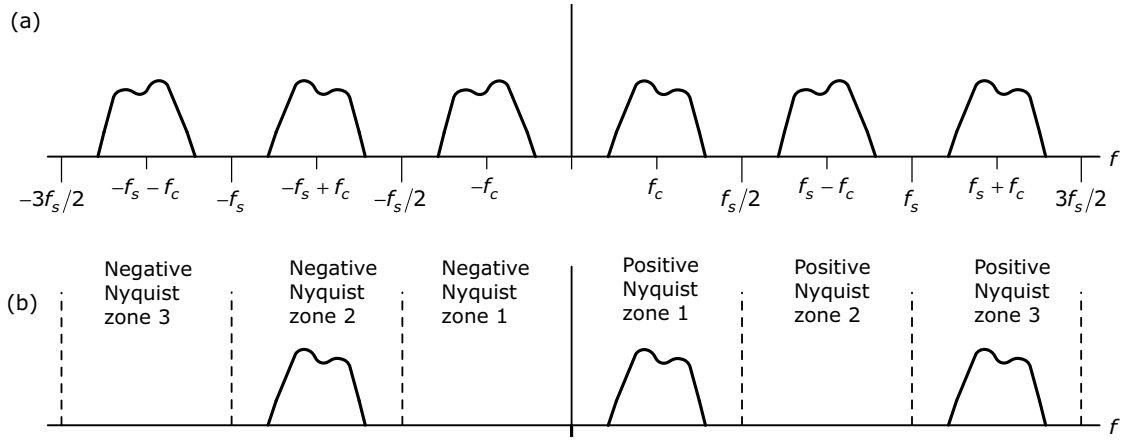


Figure 60.2 DTFT spectra for: (a) the original discrete-time, real-valued bandpass signal, and (b) the complex-valued analytic associate of the original signal

odd-numbered negative Nyquist zones and in even-numbered positive Nyquist zones.

60.2 Generating Analytic Signals

There are a number of different techniques that can be used to generate an analytic signal. The following sections introduce five of the most useful approaches.

Spectrum Tailoring Approach

The spectrum tailoring approach can be used to generate the analytic-like associate by forcing the spectrum to zero for negative-like frequencies. Recipe 60.1 lists the strategy for spectrum tailoring. However, this is just a conceptual strategy; it should not be implemented directly. A practical algorithm is a bit more complicated.

Any frequency-domain operation that is performed on a signal's DFT spectrum corresponds to some sort of time-domain filtering operation that could, in theory, be applied directly to the time-domain signal. In most cases, this filtering operation has a

Recipe 60.1

Spectrum Tailoring Strategy

1. Compute the DFT, $X[m]$, of the original signal, $x[n]$.
2. Form the transform of the analytic-like associate as follows:

$$X_a[m] = \begin{cases} X[0], & \text{for } m = 0 \\ 2X[m], & \text{for } 1 \le m \le \frac{N}{2} - 1 \\ X\left[\frac{N}{2}\right], & \text{for } m = \frac{N}{2} \\ 0, & \text{for } \frac{N}{2} - 1 \le m \le N - 1 \end{cases}$$

3. Compute the analytic-like signal, $x_a[n]$, as the IDFT of $X_a[m]$.

unit sample response that is longer than one sample in duration. Even in cases where the filter is defined directly in the frequency domain, and we do not know or need the corresponding unit sample response, we must make allowances for the length of this response by using the fast convolution technique described in Note 20. If we were to apply the three-step strategy listed above directly, the resulting analytic-like signal would, in most cases, exhibit time-domain aliasing.

Recipe 60.2 is the result of modifying Recipe 20.1 to implement spectrum tailoring. There are a few nuances in step 3 that might need further explanation. The point at $m = N/2$ sits on the transition from positive-like frequency to negative-like frequency and it is not immediately obvious why $X_a[N/2]$ is not set to either 0 or $2X[N/2]$ rather than to $X[N/2]$. However, in [1], Marple shows that setting $X_a[N/2] = X[N/2]$ is necessary to satisfy properties 1 and 2 from Math Box 60.1.

Math Box 60.1

Desired Properties for Discrete-Time "Analytic" Signals

1. Ideally, the real part of the "analytic" associate should equal the original real-valued signal:

$$z_R[n] = x[n]$$

   where $z[n] = z_R[n] + jz_I[n]$ is the analytic associate of $x[n]$.

2. The real and imaginary components of $z[n]$ should be orthogonal over the finite duration of the signal

$$\sum_{n=0}^{N-1} z_R[n] z_I[n] = 0$$

Recipe 60.2

Analytic Signal via Spectrum Tailoring

The following steps assume that values for the FFT length N, filter's memory length N_M, and input/output block length N_B have been selected in accordance with the criteria discussed in Note 20.

1. Load the first N_B samples from the signal $x[n]$ into the first N_B locations in the FFT input buffer. Set the remaining $N - N_B$ locations in the FFT input buffer equal to zero.
2. Compute the DFT result, $X[m]$.
3. Using the indicated values from $X[m]$, form $X_a[m]$, the transform of the analytic-like associate, as follows:

$$X_a[m] = \begin{cases} X[0], & \text{for } m = 0 \\ 2X[m], & \text{for } 1 \le m \le \frac{N}{2} - 1 \\ X\left[\frac{N}{2}\right], & \text{for } m = \frac{N}{2} \\ 0, & \text{for } \frac{N}{2} - 1 \le m \le N - 1 \end{cases}$$

4. Compute the IDFT of $X_a[m]$.
5. Take the first N_B samples of the IFFT output as the next N_B samples of the analytic-like signal $x_a[n]$.
6. Move the contents of locations $(N_B - N_M)$ through $(N_B - 1)$ from the FFT input buffer into locations $(N - N_M)$ through $(N - 1)$ of the same buffer.
7. Load the next N_B new samples from the signal $x[n]$ into locations 0 through $(N_B - 1)$ of the FFT input buffer.

Repeat steps 2 through 7 until all samples from the signal $x[n]$ have been exhausted.

The MATLAB function `hilbert` uses an approach described in [3] that is equivalent to the approach detailed in Recipe 60.1, but that does not include the anti-aliasing measures embodied in Recipe 60.2. When using MATLAB, these measures must be implemented via vector manipulations external to the `hilbert` function. This function might have been more aptly named `analytic`, because `y=hilbert(x)` returns a complex vector that contains the analytic-like associate of x; the Hilbert transform of x must be obtained by taking the imaginary part of y.

Hilbert Transform Approach

It is possible to generate an analytic sequence using a "discretized" version of Eq. (60.1). For any real-valued signal sequence, $x[n]$, the corresponding "analytic" sequence, $x_a[n]$, can be formed as

$$x_a[n] = \tfrac{1}{2}\left(x[n] + j\,\mathcal{H}\left[x[n]\right]\right) \qquad (60.2)$$

where $\mathcal{H}[\cdot]$ denotes the discrete Hilbert transform. An ideal discrete Hilbert transformer is a discrete-time filter having an impulse response, $h[n]$, and transfer function, $H(e^{j\omega})$, given by

$$h[n] = \begin{cases} \dfrac{2\sin^2(\pi n/2)}{\pi n}, & n \neq 0 \\ 0, & n = 0 \end{cases} = \begin{cases} 0, & \text{for } n \text{ even} \\ \dfrac{2}{\pi n} & \text{for } n \text{ odd} \end{cases} \qquad (60.3)$$

$$H(e^{j\omega}) = \begin{cases} -j & 0 \le \omega < \pi \\ j & \pi \le \omega < 2\pi \end{cases} \qquad (60.4)$$

An ideal Hilbert transformer is not realizable, so practical implementations for generating analytic-like discrete-time signals must be built around approximations to the ideal Hilbert transformer. These approximations insert delay in $\mathrm{Im}\{x_a[n]\}$, so $\mathrm{Re}\{x_a[n]\}$ must be delayed by a corresponding amount. Design of FIR Hilbert transformers is covered in Note 61.

Frequency-Shifted Lowpass Filters

A useful technique for designing FIR analytic signal generating filters is presented by Reilly et. al. in [4]. The basic idea is to design a "standard" lowpass filter having real-valued coefficients, and then phase-shifts the impulse response coefficients such that the filter's passband is shifted to cover only positive frequencies and the stopband is shifted to cover only negative frequencies. This approach is explored further in Note 62.

Paired-Filter Approach

The key that makes Eq. (60.1) work is that $x[n]$ and $j\mathcal{H}[x[n]]$ have spectra that are inphase for positive-like frequencies and 180 degrees out of phase for negative-like frequencies. A similar phasing relationship can be exploited to generate an analytic-like signal using an alternative approach that does not require a Hilbert transformer. Specifically, the analytic-like signal is formed as

$$x_a[n] = \frac{1}{2}\left[\mathcal{P}\{x[n]\} + j\mathcal{Q}\{x[n]\}\right] \qquad (60.5)$$

where, ideally, $\mathcal{P}$ and $\mathcal{Q}$ are all-pass filters that are designed to have phase responses such that the phase of $\mathcal{Q}$ lags the phase of $\mathcal{P}$ by 90 degrees for all positive-like frequencies, and the phase of $\mathcal{Q}$ leads the phase of $\mathcal{P}$ by 90 degrees for all negative-like frequencies. Then when multiplication by j advances the phase of $\mathcal{Q}\{x[n]\}$ by 90 degrees, the phases of $\mathcal{P}\{x[n]\}$ and $j\mathcal{Q}\{x[n]\}$ are inphase for

positive-like frequencies and 180 degrees out of phase for negative-like frequencies. Sometimes, the network comprising the two filters $\mathcal{P}$ and $\mathcal{Q}$ is referred to as an analytic-signal generating filter, or ASG filter. Design of IIR filter pairs for $\mathcal{P}$ and $\mathcal{Q}$, as used in Eq. (60.4), is discussed and demonstrated in Note 63.

Complex Filter Approach

It is possible to design a single complex-valued filter that generates the analytic associate of the signal applied to the filter's input. One way to design such a filter involves the complex extension of the Parks-McClellan algorithm. Design of complex-valued ASG filters using this extended algorithm is discussed in Note 64.

References

1. S. L. Marple, "Computing the Discrete-Time 'Analytic' Signal via FFT," *IEEE Trans. Signal Proc.*, vol. 47, no. 9, September 1999, pp. 2600–2603.
2. B. Boashash, *Time Frequency Signal Analysis and Processing: A Comprehensive Reference*, Elsevier, 2003.
3. `hilbert` help article from MATLAB help system.
4. A. Reilly, G. Frazer, and B. Boashash, "Analytic Signal Generation—Tips and Traps," *IEEE Trans. Signal Proc.*, vol. 42, no. 11, Nov. 1994, pp. 3241–3245.

Generating Analytic Signals with FIR Hilbert Transformers

As discussed in Note 60, many DSP applications require the creation of the analytic-like signal that corresponds to some real-valued signal of interest. This note describes just one of several approaches that can be used to generate an analytic-like signal. For an overview of other possible approaches, refer to Note 60.

For a real-valued sequence, $x[n]$, the corresponding analytic-like sequence, $x_a[n]$, can be formed as

$$x_a[n] = x[n] + j\hat{x}[n] \tag{61.1}$$

where

$$\hat{x}[n] = \mathcal{H}\{x[n]\}$$

and $\mathcal{H}\{\cdot\}$ denotes the discrete Hilbert transform. An ideal lowpass filter is unrealizable, and so is an ideal Hilbert transformer. The challenge a DSP engineer faces is to design an approximation to the desired response that provides performance that is adequate for the particular application of interest. FIR approximations to the ideal Hilbert transformer are subject to the constraints given in List 61.1. However, since most applications that require Hilbert transformers involve signals that are already bandpass in nature, meeting these constraints is usually not difficult. Equation (61.1) can be implemented using an *analytic signal generating* (ASG) filter like the one depicted in Figure 61.1. The delay element is matched to the delay of the Hilbert transformer.

Example 61.1 attempts to implement a broadband, or "almost-all-pass," ASG filter

List 61.1

Constraints on FIR Hilbert Transformers

- The ideal Hilbert transformer's frequency response is antisymmetric, so any linear-phase FIR approximation to the ideal HT needs to be either Type 3 or Type 4. (See Note 35 for a discussion of linear-phase FIR types.)
- Note that an antisymmetric linear-phase FIR filter has a purely imaginary response, as does the ideal Hilbert transformer response.
- Type 3 and Type 4 filters have a frequency response of zero at $\omega = 0$, and Type 3 filters have a response of zero at $\omega = \pi$.
  - These constraints mean that neither a Type 3 nor a Type 4 FIR design can be used to implement a filter that is all-pass for positive frequencies.
  - A Type 4 frequency response is in a transition band at frequencies just above $\omega = 0$, so a Type 4 Hilbert transformer must have a highpass response for positive frequencies.
  - A Type 3 frequency response is in a transition band at frequencies just above $\omega = 0$ and in a second distinct transition band at frequencies just below $\omega = \pi$, so a Type 3 Hilbert transformer must have a bandpass response for positive frequencies.

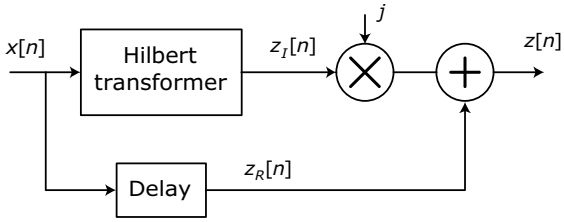


Figure 61.1 Analytic signal generating filter

that incorporates a Hilbert transformer designed using the Parks-McClellan algorithm.

Example 61.2 implements a more pronouncedly bandpass design that is only concerned with cancelling negtive frequencies in the band from $-3f_s/8$ to $-f_s/8$. The more generous transition bands in this second case result in an ASG filter with much higher stopband attenuation.

Notice that in both examples, the bandpass nature of the Hilbert transformer results in a bandstop ASG filter. The ASG filter "defaults" to passband, having its stopband at only those frequencies corresponding to the Hilbert transformer's passband. For some applications, it might be desirable for the ASG filter to have a narrowly defined passband with a "default" stopband everywhere else.

Example 61.1

Hilbert Transformer for "Almost-All-Pass" ASG Filter

The coefficients for a 31-tap FIR Hilbert transformer with a passband extending from $0.05f_s$ to $0.95f_s$ can be obtained using the following segment of MATLAB code:

```
b=firpm(30,[0.05 0.95],[1 1],'h');
b=0.5*j*b;
b(16)=b(16)+0.5;
```

The response of the Hilbert transformer is shown in Figure 61.2, and the response of the corresponding ASG filter is shown in Figure 61.3. There are ripples in the Hilbert transformer's passband that measure over 0.7 dB peak to peak. The stopband attenuation in the ASG filter is only about 34 dB.

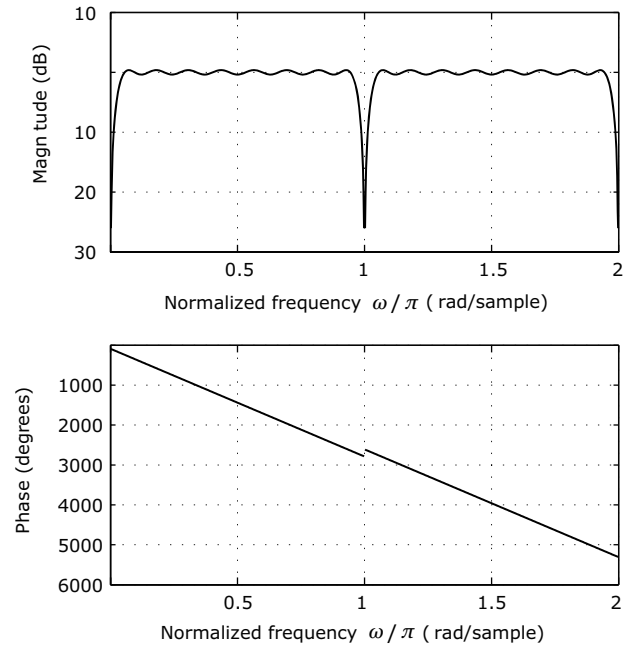


Figure 61.2 Frequency response for Hilbert transformer from Example 61.1

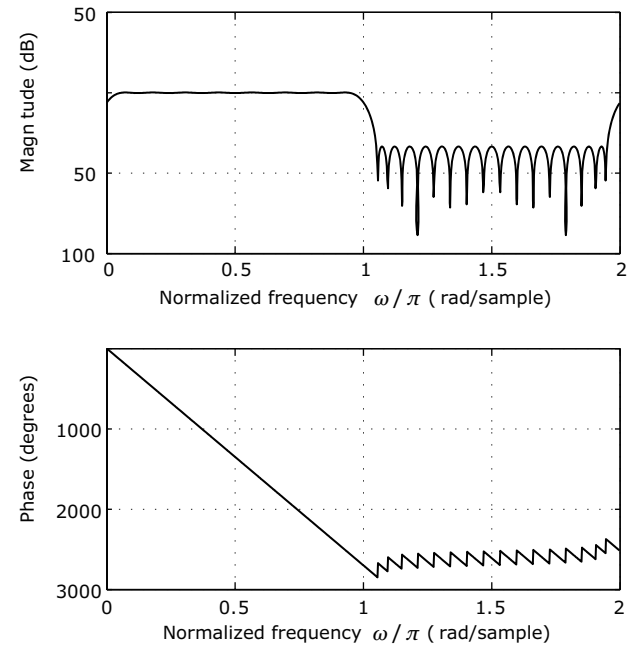


Figure 61.3 Frequency response for ASG filter from Example 61.1

Example 61.2

Hilbert Transformer for Bandpass ASG Filter

A digital I/Q scheme described in Note 65 involves a real-valued bandpass signal that has been frequency-shifted and sampled such that the passband extends from $f_s/8$ to $3f_s/8$, as shown in Figure 61.4(a). Before additional processing can be performed, the negative-frequency passband must be eliminated to produce an analytic signal having the positive-only spectrum shown in Figure 61.4(b). The coefficients for a 31-tap FIR Hilbert transformer can be obtained using the following segment of MATLAB code:

```
b=firpm(30,[0.25 0.75],[1 1],'h');
b=0.5*j*b;
b(16)=b(16)+0.5;
```

The response of the Hilbert transformer is shown in Figure 61.5, and the response of the corresponding ASG filter is shown in Figure 61.6.

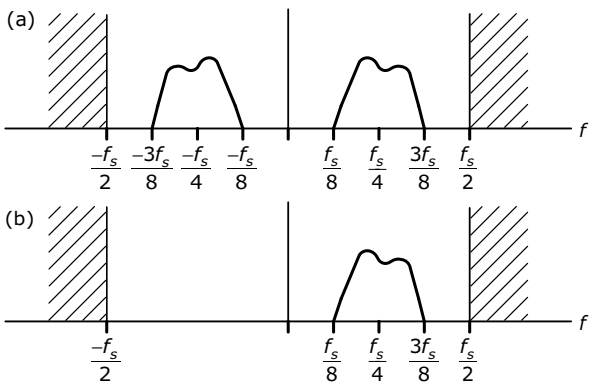


Figure 61.4 Spectra for Example 61.3: (a) spectrum for typical bandpass signal, (b) spectrum for corresponding analytic signal

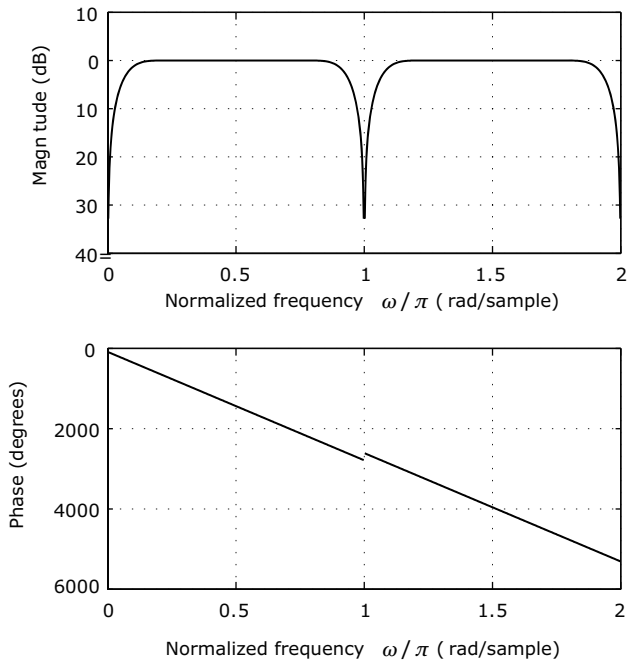


Figure 61.5 Frequency response for Hilbert transformer from Example 61.2

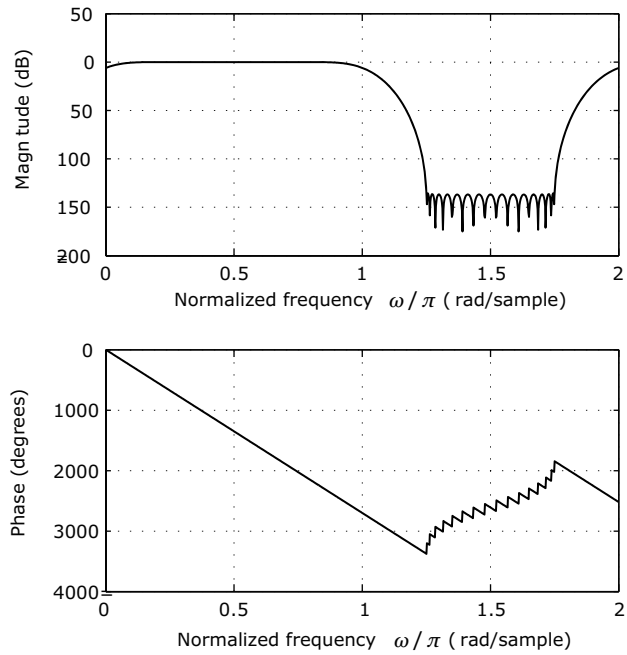


Figure 61.6 Frequency response for analytic signal generator from Example 61.2

Generating Analytic Signals with Frequency-Shifted FIR Lowpass Filters

A useful technique for designing FIR analytic signal generation (ASG) filters is presented by Reilly et al. in [1]. The basic idea is to design a "standard" lowpass filter having real-valued coefficients, and then phase-shift the impulse response coefficients such that the filter's passband is shifted to cover only positive frequencies and the stopband is shifted to cover only negative frequencies. The phase-shifted coefficients are complex-valued. The details are provided in Design Procedure 62.1.

Example 62.1 attempts to implement a broadband, or "almost-all-positive pass" ASG filter using the approach of Design Procedure 62.1. Example 62.2 implements a more pronouncedly bandpass design that is only concerned with cancelling negative frequencies in the band from $-3f_s/8$ to $-f_s/8$. The more generous transition bands in this second case result in an ASG filter with much higher stopband attenuation.

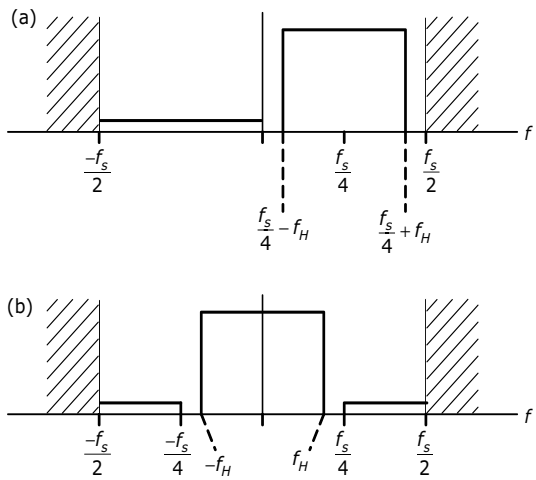


Figure 62.1 Desired frequency responses for Design Procedure 62.1: (a) analytic signal generating filter response, (b) corresponding prototype lowpass response

Design Procedure 62.1

Complex FIR ASG Filter Design

1. Map out a desired ASG filter response similar to the one shown in Figure 62.1(a). The stopband must cover the entire negative-frequency Nyquist zone, and the passband must be centered on $f_s/4$, as shown. The passband extends from f_1 to f_2 where

$$f_1 = \frac{f_s}{4} - f_H$$
$$f_2 = \frac{f_s}{4} + f_H \qquad \text{(DP 62.1)}$$

2. Using the value of f_H that yields the desired passband as given by Eq. (DP62.1), construct a response mask for a "standard" lowpass filter like the one shown in Figure 62.1(b).

3. Using a standard FIR design technique like least-squares or Parks-McClellan, obtain the coefficients, $h[n]$, for a filter that approximates the response in Figure 62.1(b).

4. Compute coefficients, $h[n]$, for the frequency-shifted filter that approximates the response in Figure 62.1(a):

$$h'[n] = h[n]\exp\left(j\pi n\frac{f_s}{4}\right)$$

Example 62.1

"Almost-All-Positive-Pass" ASG Filter

The desired response has a positive-frequency passband from $0.025f_s$ to $0.475f_s$, thus, $f_H = 0.225f_s$. The coefficients for a 61-tap lowpass filter can be obtained using the following segment of MATLAB code:

```
b=firpm(60,[0 0.45 0.5 1],[1 1 0 0]);
```

The response of this filter is shown in Figure 62.2. The vector of phase-shifting factors can be generated as

```
bm=0:60;
bm=exp(j*pi*bm/2);
```

Finally, the shifted filter's coefficients are obtained as

```
bc=b.*bm;
```

The response of the ASG filter is shown in Figure 62.3.

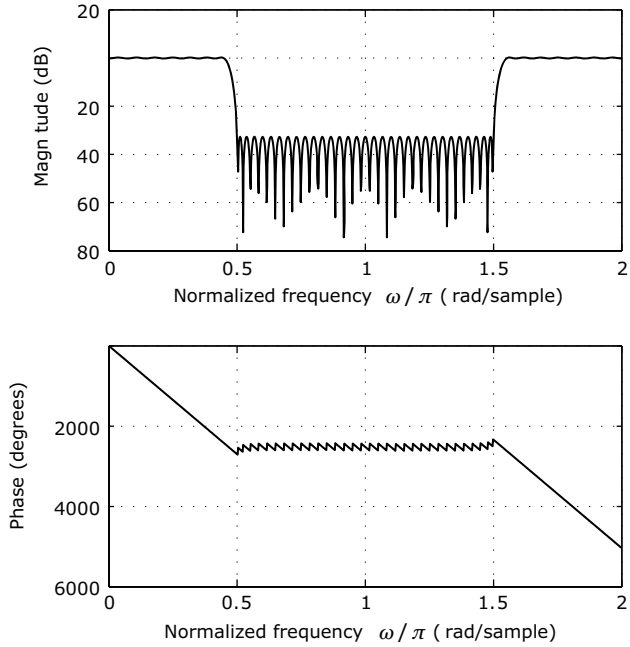


Figure 62.2 Frequency response for prototype lowpass filter from Example 62.1

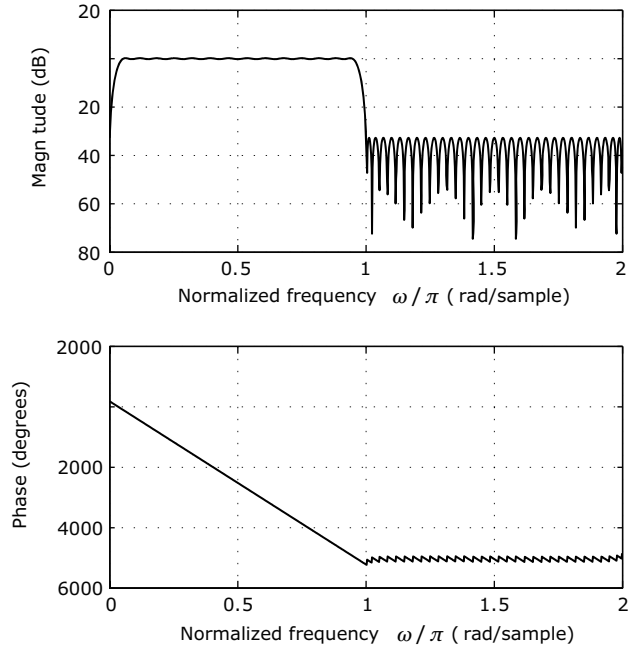


Figure 62.3 Frequency response for analytic signal filter from Example 62.1

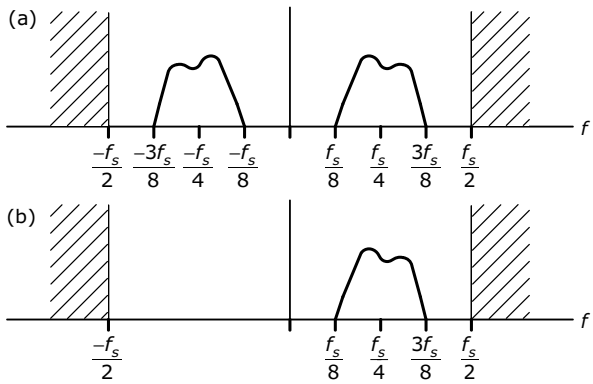


Figure 62.4 Spectra for Example 62.2: (a) spectrum for typical bandpass signal, (b) spectrum for corresponding analytic signal

Example 62.2

Bandpass ASG Filter

A digital I/Q scheme described in Note 65 involves a real-valued bandpass signal that has been frequency-shifted and sampled such that the passband extends from $f_s/8$ to $3f_s/8$, as shown in Figure 62.4(a).

Before additional processing can be performed, the negative-frequency passband must be eliminated to produce an analytic signal having the positive-only spectrum shown in Figure 62.4(b). The coefficients for a 61-tap prototype lowpass filter can be obtained using the following segment of MATLAB code:

```
b=firpm(60,[0 0.25 0.5 1],[1 1 0 0]);
```

The response of this filter is shown in Figure 62.5. The vector of phase-shifting factors can be generated as

```
bm=0:60;
bm=exp(j*pi*bm/2);
```

Finally, the shifted filter's coefficients are obtained as

```
bc=b.*bm;
```

The response of the ASG filter is shown in Figure 62.6.

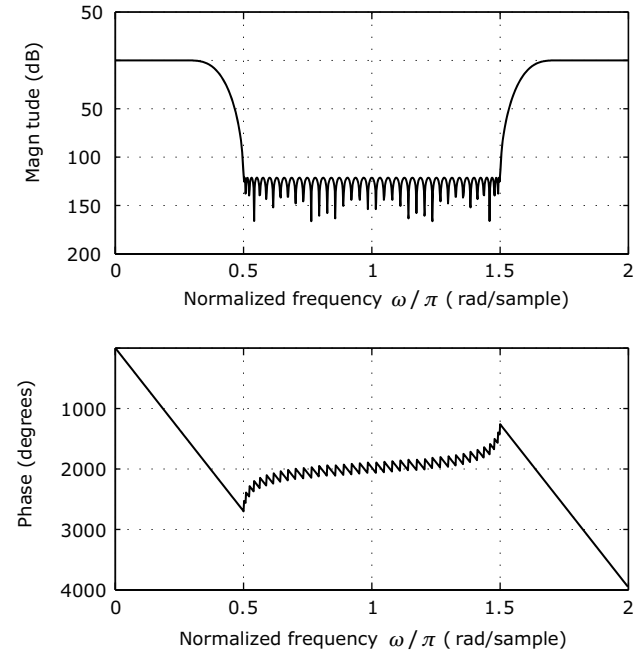


Figure 62.5 Frequency response for prototype lowpass filter from Example 62.2

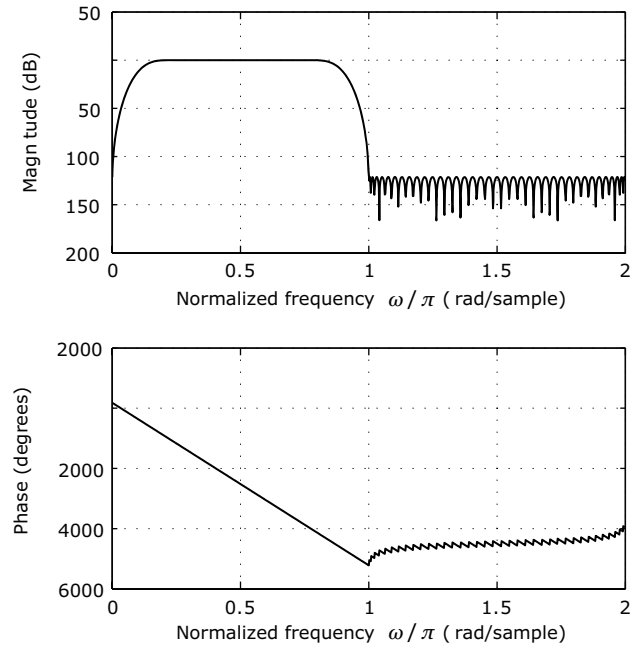


Figure 62.6 Frequency response for analytic signal filter from Example 62.2

Reference

1. A. Reilly, G. Frazer, and B. Boashash, "Analytic Signal Generation—Tips and Traps," *IEEE Trans. Signal Proc.*, vol. 42, n. 11, Nov. 1994, pp. 3241–3245.

IIR Phase-Splitting Networks for Generating Analytic Signals

This note describes the design of IIR phase-splitting networks that can be used to generate complex-valued analytic output signals from real-valued bandpass input signals. Refer to Note 60 for a discussion of analytic signals and their properties.

The analytic associate, $x_a[n]$, of a real bandpass signal, $x[n]$, can be formed as

$$x_a[n] = \tfrac{1}{2}\Big[\mathcal{P}\{x[n]\} + j\mathcal{Q}\{x[n]\}$$

where $\mathcal{P}$ and $\mathcal{Q}$ are all-pass filters that together comprise a phase-splitting network such that the phase of $\mathcal{Q}$ lags the phase of $\mathcal{P}$ by 90 degrees for all positive frequencies, and the phase of $\mathcal{Q}$ leads the phase of $\mathcal{P}$ by 90 degrees for all negative frequencies. Then when multiplication by j advances the phase of $\mathcal{Q}\{x[n]\}$ by 90 degrees, the phases of $\mathcal{P}\{x[n]\}$ and $j\,\mathcal{Q}\{x[n]\}$ are inphase for positive frequencies and 180 degrees out of phase for negative frequencies. The difficulty in this approach lies in the design of suitable phase-shifting filters for $\mathcal{P}$ and $\mathcal{Q}$.

A technique, based on Jacobian elliptic functions and the bilinear transform, that can be used to design suitable filter pairs for $\mathcal{P}$ and $\mathcal{Q}$ was first described by Storer [1], and later extended by Gold and Rader [2]. Rader [3] subsequently showed that a fifth-order phase splitter can be efficiently implemented as a pair of efficient IIR filters, provided that the desired response is of the form shown in Figure 63.1, where the passband is symmetric about $\pi/2$ and the stopband is at the corresponding negative frequencies and symmetric about $-\pi/2$. This symmetry causes poles of each branch to occur in pairs with poles in each pair having equal magnitudes and opposite signs, thus allowing a single multiplier to implement a pair of poles, as shown in Figure 63.2. The multiplication by a^2 implements

Essential Facts

- A phase-splitting network consists of a pair of IIR filters designed to generate a complex-valued analytic output signal from a real-valued bandpass input signal.
- Wide transition bands of low-order IIR filters make this approach ill suited for use on lowpass input signals.
- The IIR structures used in this approach present lower computational burdens than do FIR approaches for generating analytic signals.
- The architecture of the implementation provides an opportunity to perform downsampling as a natural part of the filter's operation.
- The phase response is only approximately linear across the passband and has jump discontinuities of π radians at nulls in the stopband.
- The most efficient realizations depend upon the specified passband being symmetric about a normalized radian frequency of $\pi/2$.
- Design procedures require evaluation of *complete elliptic integrals* and *Jacobian elliptic functions*. (MATLAB can handle both of these evaluations.)

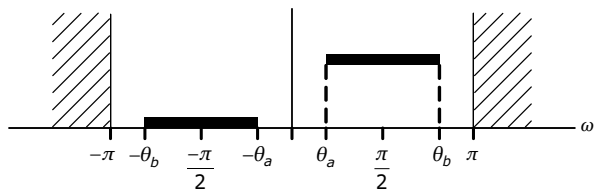


Figure 63.1 Spectrum symmetries required for Rader's efficient IIR implementation. The passband and stopband must be symmetric about $\pi/2$ and $-\pi/2$ respectively.

one pole pair, and the multiplication by b^2 implements a second pole pair. The fifth pole occurs at $z=0$, and it is implemented by the extra single delay element in the upper branch. The upper branch, which generates the real part of $\hat{x}[n]$, implements the transfer function

$$H_P(z) = z^{-1}\frac{z^{-2}-a^2}{1-a^2z^{-2}} \tag{63.1}$$

The lower branch, which generates the imaginary part of $\hat{x}[n]$, implements the transfer function

$$H_Q(z) = \frac{b^2-z^{-2}}{1-b^2z^{-2}} \tag{63.2}$$

Combining Eqs. (63.1) and (63.2) yields the overall complex response $H(z)$ as

$$\begin{aligned} H(z) &= H_P(z) + jH_L(z) \\ &= \frac{z^{-3}-a^2z^{-1}}{1-a^2z^{-2}} - j\frac{z^{-2}-b^2}{1-b^2z^{-2}} \\ &= \frac{jb^2 - a^2z^{-1} - j(1+a^2b^2)z^{-2} + (1+a^2b^2)z^{-3} + ja^2z^{-4} - b^2z^{-5}}{1-(a^2+b^2)z^{-2}+a^2b^2z^{-4}} \end{aligned} \tag{63.3}$$

In [4], Rader repeats the fifth-order example from [3] and also provides an efficient IIR implementation for a ninth-order phase splitter, which is shown in Figure 63.3.

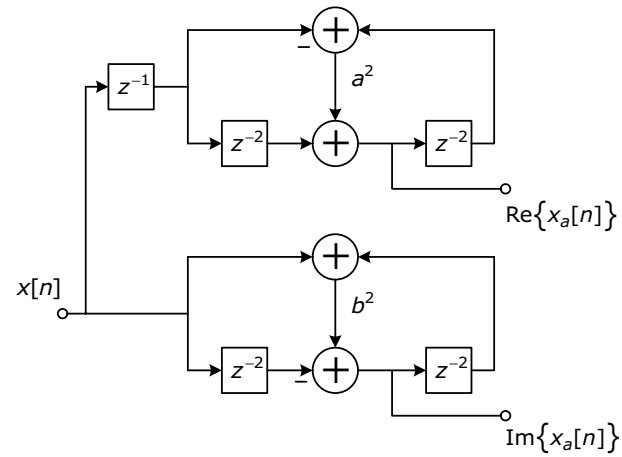


Figure 63.2 Efficient IIR realization for fifth-order phase-splitting network

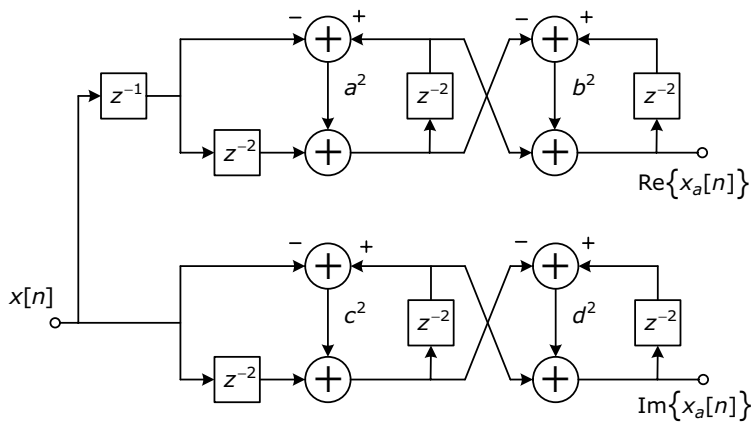


Figure 63.3 Efficient IIR realization for ninth-order phase-splitting network

63.1 The Approach

Before the coefficients for the phase splitter can be computed, Design Procedure 63.1 must be used to determine the minimum network order that will provide a desired level of stopband attenuation for a set of critical frequencies, θ_a and θ_b, specified as shown in Figure 63.1. Once the minimum order is determined, this value should be rounded up to either 5 or 9 before Design Procedure 63.2 is used to compute values for the filter coefficients.

The filter in Example 63.1 has a symmetric passband, and can therefore be implemented using the efficient IIR structure shown in Figure 63.2. Example 63.2 features a case in which the passband is not symmetric.

Design Procedure 63.1

Elliptic Phase-splitting Network: Computing the Minimum Order

1. Select values for the critical normalized frequencies, θ_a and θ_b, which are depicted in Figure 63.1. Selecting values such that $\theta_a = \pi - \theta_b$ forces the filter poles to occur in conjugate pairs.
2. Select the error factor ε. The passband gain is very close to 2, while the peak stopband gain is sin ε.
3. Compute
$$m = \left(\frac{\tan(\theta_a/2)}{\tan(\theta_b/2)}\right)^2 \qquad m_1 = \left(\frac{1-\tan(\varepsilon/2)}{1+\tan(\varepsilon/2)}\right)^4$$
4. Compute the *complete elliptic integrals*, $K(m)$ and $K(m_1)$. The MATLAB call
```
[K,E]=ellipke(m);
```
returns $K(m)$ in the variable K.
5. Compute the complete elliptic integrals, $K'(m) = K(1-m)$ and $K'(m_1) = K(1-m_1)$.
6. Compute the required number of poles and zeros as
$$N = \left\lceil \frac{K'(m)K(m_1)}{K'(m_1)K(m)} \right\rceil$$

Design Procedure 63.2

Elliptic Phase-splitting Network: Computing the Filter Coefficients

1. For $k = 0, 1, \ldots, N-1$, compute
$$p_k = -\tan\left(\frac{\theta_a}{2}\right)\frac{\mathrm{sn}(x, 1-m)}{\mathrm{cn}(x, 1-m)}$$
where
$$x = \frac{(4k+1)K'(m)}{2N}$$
and sn and cn are *Jacobian elliptic functions* that can be computed using the MATLAB call
```
[sn,cn,dn]=ellipj(x,1-m);
```
2. For negative values of the p_k obtained in step 1, compute the network coefficients z_k as
$$z_k = \frac{1+p_k}{1-p_k}$$
Use these coefficients to form $H_p(z)$ as
$$H_p(z) = \prod_{\substack{\text{all } k \text{ for} \\ \text{which} \\ p_k<0}} \left(\frac{z^{-1}-z_k}{1-z_k z^{-1}}\right) \qquad \text{(DP63.1)}$$
When the critical frequencies θ_a and θ_b are symmetric with respect to $\pi/2$, the coefficients occur in pairs with both coefficients in the pair having the same magnitude, but with the opposite sign. When two values of z_k, say, a and $-a$ comprise such a pair, the corresponding factors from Eq. (DP63.1) can be combined into a biquadratic factor:
$$\left(\frac{z^{-1}-a}{1-az^{-1}}\right)\left(\frac{z^{-1}-(-a)}{1-(-a)z^{-1}}\right) = \frac{z^{-2}-a^2}{1-a^2z^{-2}}$$
3. For positive values of the p_k obtained in step 1, compute z_k as
$$z_k = \frac{1-p_k}{1+p_k}$$
Use these coefficients to form $H_Q(z)$ as
$$H_Q(z) = \prod_{\substack{\text{all } k \text{ for} \\ \text{which} \\ p_k>0}} \left(\frac{z^{-1}-z_k}{1-z_k z^{-1}}\right)$$

Example 63.1

Phase-splitter with Passband Symmetric about π/4

Consider a signal having the bandpass spectrum shown in Figure 63.4. As part of a demodulation process, we want to eliminate the entire negative-frequency portion of the spectrum before shifting the positive-frequency band to be centered around zero. This application is the same one covered by Rader in [3]. To eliminate the negative-frequency portion of the spectrum, we need to design a filter having a passband and a stopband configured as shown in Figure 63.4, with $\theta_a = \pi/4$ and $\theta_b = 3\pi/4$.

First, determine the minimum order using the following steps, which are numbered to correspond with the steps listed in Design Procedure 63.1.

2. The critical frequencies are $\theta_a = \pi/4$ and $\theta_b = 3\pi/4$.

3. Suppose we need the ratio of passband gain to stopband gain to be greater than 60 dB. Thus,

$$\frac{2}{\sin\epsilon} > 60 \text{ dB}$$
$$\frac{2}{\sin\epsilon} > 10^{60/20}$$
$$\epsilon < 0.002$$

4. The parameters m and m_1 are obtained as

$$m = \left(\frac{\tan(\pi/8)}{\tan(3\pi/8)} \right)^2 = 0.02943725$$
$$m_1 = \left(\frac{1-\tan(0.002)}{1+\tan(0.002)} \right)^4 = 0.99203825$$

5. The complete elliptic integrals $K(m)$ and $K(m_1)$ are obtained from the MATLAB `ellipke` function as

$$K(m) = 1.58255172$$
$$K(m_1) = 3.80845011$$

6. The complete elliptic integrals $K'(m)$ and $K'(m_1)$ are obtained from `ellipke` as

$$K'(m) = 3.16510348$$
$$K'(m_1) = 1.56766507$$

7. The required number of poles and zeros is obtained as

$$N = \lceil 4.86 \rceil = 5$$

Next, obtain the coefficients for a fifth-order phase-splitting network via the following steps, which are numbered to correspond with the steps listed in Design Procedure 63.2.

1. The values for $\text{sn}(x, 1-m)$ and $\text{cn}(x, 1-m)$ obtained from the MATLAB `ellipj` function, along with the resulting values for p_k, are listed in Table 63.1.

2. The negative values of p_k result in the network coefficients

$$z_0 = 0.76464583$$
$$z_1 = 0.0$$
$$z_2 = -0.76464583$$

The coefficients, z_0 and z_2, can be combined into the single biquadratic factor

$$a^2 = 0.5846832$$

3. The positive values of p_k result in the network coefficients

$$z_3 = -0.3715172$$
$$z_4 = 0.3715172$$

which can be combined into the single biquadratic factor

$$b^2 = 0.1380250$$

Using MATLAB variables `a2` and `b2` to represent a^2 and b^2 obtained in steps 2 and 3 above, the vectors of numerator and denominator coefficients for the combined response can be written as:

```
a2=0.5846832;
b2=0.138025;
B=[j*b2,-a2,-j*(1+a2+b2),1+a2*b2,j*a2,-b2];
A=[1,0,-(a2+b2),0,a2*b2];
```

These definitions for `A` and `B` can be used in a call to `freqz` to calculate the magnitude and phase responses shown in Figure 63.5:

```
freqz(B,A,1024,'whole');
```

The stopband attenuation is about 62.2 dB, which is slightly better than the design goal of 60 dB.

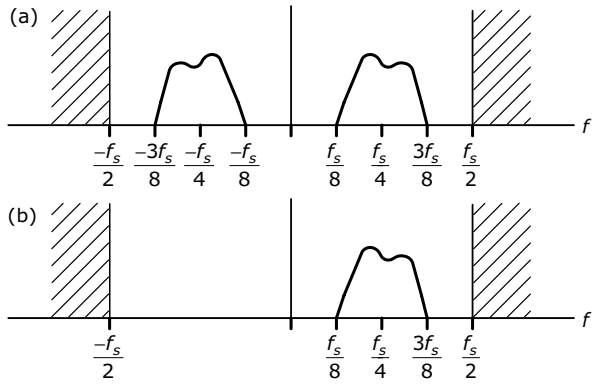


Figure 63.4 Spectra for Example 63.1: (a) spectrum for typical bandpass signal, (b) spectrum for corresponding analytic signal

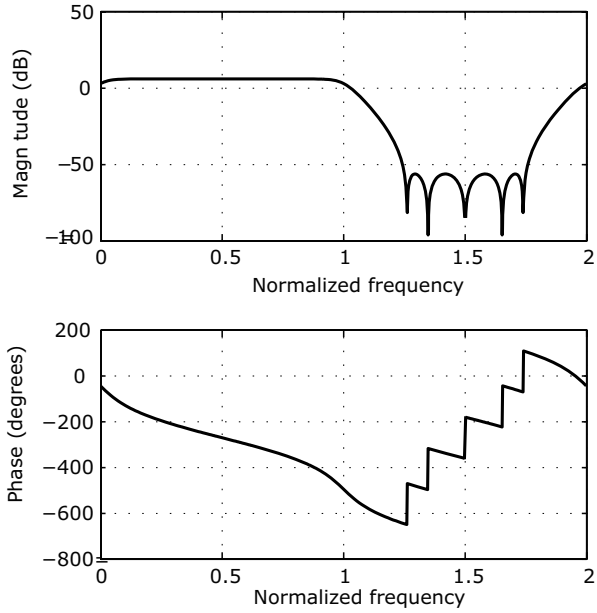


Figure 63.5 Magnitude and phase response for Example 63.1

Table 63.1 Values of Jacobian Elliptic Functions Used to Calculate the Filter Coefficients for Example 63.1

k	$\mathbf{sn}(x, 1-m)$	$\mathbf{cn}(x, 1-m)$	p_k
0	0.306491992	0.95187323	– 0.1333719
1	0.9238795	0.382683428	– 0.99999998
2	0.998477509	0.055160338	– 7.497831
3	0.9824588703	– 0.186479404	2.18226667
4	0.7418447897	– 0.6705716277	0.45823915

Example 63.2

Phase-splitter with Passband Not Symmetric about π/4

In Example 63.1, the critical frequencies are $\theta_a = \pi/4$ and $\theta_b = 3\pi/4$. If these frequencies are shifted to $\theta_a = \pi/5$ and $\theta_b = 7\pi/10$, the filter coefficients will not occur in conjugate pairs, as demonstrated by the current example.

First, determine the minimum order using the following steps, which are numbered to correspond with the steps listed in Design Procedure 63.1.

1. The critical frequencies are $\theta_a = \pi/5$ and $\theta_b = 7\pi/10$.
2. Again, assume that we need the ratio of passband gain to stopband gain to be greater than 60 dB. Thus, we use a value of $\epsilon = 0.002$, as in Example 63.1.
3. The parameters m and m_1 are obtained as

$$m = \left(\frac{\tan(\pi/10)}{\tan(7\pi/10)} \right)^2 = 0.02740841$$

$$m_1 = \left(\frac{1-\tan(0.002)}{1+\tan(0.002)} \right)^4 = 0.9920319$$

4. The complete elliptic integrals, $K(m)$ and $K(m_1)$, are obtained from the MATLAB `ellipke` function as

$$K(m) = 1.58172875$$
$$K(m_1) = 3.80805517$$

5. The complete elliptic integrals, $K'(m)$ and $K'(m_1)$, are obtained from `ellipke` as

$$K'(m) = 3.19993436$$
$$K'(m_1) = 1.5739395$$

6. The required number of poles and zeros is obtained as

$$N = \left\lceil \frac{(3.19993436)(3.80805517)}{(1.5739395)(1.58172875)} \right\rceil = 5$$

Next, obtain the coefficients for a fifth-order phase-splitting network via the following steps which are numbered to correspond with the steps listed in Design Procedure 63.2.

1. The values for $\mathrm{sn}(x, 1-m)$ and $\mathrm{cn}(x, 1-m)$ obtained from the MATLAB `ellipj` function, along with the resulting values for p_k, are listed in Table 63.2.
2. The negative values of p_k result in the network coefficients

$$z_0 = 0.80863244$$
$$z_1 = 0.11200351$$
$$z_2 = -0.71537849$$

3. The positve values of p_k result in the network coefficients

$$z_3 = -0.27265279$$
$$z_4 = 0.46578851$$

To use a standard response-plotting utility such as the MATLAB `freqz` function, the responses for $H_P(z)$ and $H_Q(z)$ must be combined into a single ratio of polynomials in z^{-1}. The algebraic manipulations needed to perform the combination manually can be very tedious when, as in this example, the coefficients do not occur in pairs. In such cases, the MATLAB-based aproach shown in Computer Listing 63.1 can be used to compute the polynomial coefficients from the factored-form coefficients, z_k. The resulting response plots for the current example are shown in Figure 63.6. The quality of this response is comparable to that for Example 63.1.

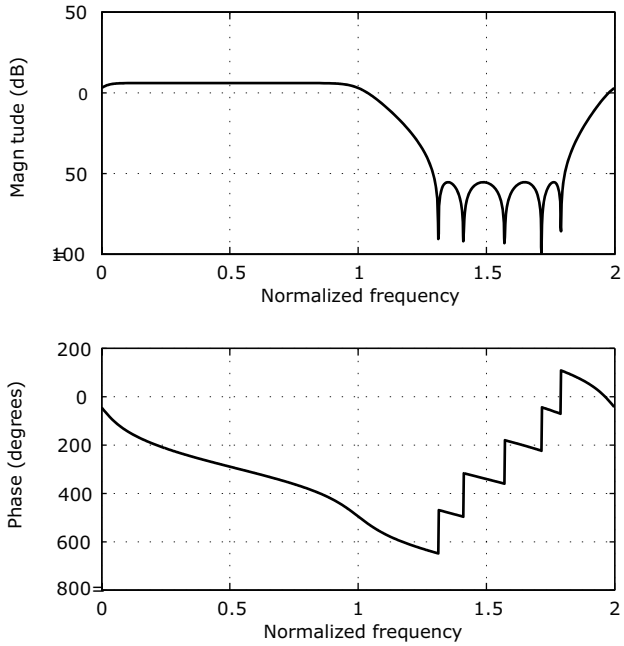


Figure 63.6 Magnitude and phase responses for Example 63.2

Table 63.2 Values of Jacobian Elliptic Functions Used in Calculating the Filter Coefficients for Example 63.2

k	$\mathrm{sn}(x, 1-m)$	$\mathrm{cn}(x, 1-m)$	p_k
0	0.309639142	0.95085414	−0.10580788
1	0.926261563	0.3768813	−0.79855547
2	0.998549919	0.05383362	−6.0268756
3	0.98319156	−0.182576978	1.7497184
4	0.746431003	−0.665462814	0.36445332

Computer Listing 63.1

Forming Combined Network Function for Plotting Phase-Splitter Response

```
%
% z_a and z_b contain coefficients that
%     were computed in Design Example #2
%
% Upper Branch
%
z_a = [0.80863244 0.11200351 -0.71537849];

HP_denom = poly(z_a);
HP_numer = fliplr(HP_denom);

%
% Lower Branch
%
z_b = [-0.27265279 0.46578851];

HQ_denom = poly(z_b);
HQ_numer = -fliplr(HQ_denom);

%
% Combined transfer function
%
H_numer_real = conv(HP_numer, HQ_denom);
H_numer_imag = conv(HP_denom, HQ_numer);
H_numer = H_numer_real + j*H_numer_imag;
H_denom = conv(HP_denom, HQ_denom);

%
% plot magnitude and phase response
%
figure (1);
freqz(H_numer,H_denom,1024,'whole');
%
% plot pole and zero locations
%
figure (2);
zplane(H_numer,H_denom);
```

References

1. J. E. Storer, *Passive Network Synthesis,* McGraw-Hill, 1957.
2. B. Gold and C. M. Rader, *Digital Processing of Signals,* McGraw-Hill, 1969. (Note: There are several errors in the section on phase-splitter design in this reference. The corrections for these errors are given by Rader in [3].)
3. C. M. Rader, "A Simple Method for Sampling In-Phase and Quadrature Components," *IEEE Trans. Aerospace and Electronic Syst.,* vol. 20, no. 6, November 1984, pp. 821–824.
4. C. M. Rader, "Method and Apparatus for Sampling In-Phase and Quadrature Components," U. S. Patent 4,794,556, dated Dec. 27, 1988.

Generating Analytic Signals with Complex Equiripple FIR Filters

This note describes the design and use of *complex equiripple FIR filters* to generate discrete-time analytic signals of the sort discussed in Note 60.

Such filters are designed using a complex-valued extension of the Parks-McClellan algorithm and offer more flexibility than other analytic signal generation (ASG) techniques when it comes to specifying the passband, stopband, and transition bands of the filter. However, this flexibility has a price—extra constraints placed on the filter's configuration usually demand an increased number of taps to meet the specified performance.

The complex filter coefficients for an FIR analytic signal generator can be determined using the MATLAB `cfirpm` (formerly `cremez`) function. This function requires that the user provide a piecewise specification of the desired response. For generating analytic signals, the ideal desired response has the characteristics listed in List 64.1.

Example 64.1 deals with the bandpass ASG case that is featured in the digital I/Q scheme discussed in Note 65.

List 64.1

Ideal Response Characteristics for ASG Filters

- A passband with a flat response across the bandwidth of the signal's sideband that is to be retained.
- A stopband with a zero response across the bandwidth of the signal's sideband that is to be removed.
- A stopband with zero response across a small range of frequencies around zero to remove any direct current (dc) component that may have been introduced by analog processing prior to sampling.
- Unconstrained transition bands between any two bands having different specified desired responses. These bands must be left unconstrained to give the design program the "wiggle room" it needs to achieve the best possible passband and stopband performance.
- Any frequency band not otherwise covered by characteristics listed above should have a desired response of zero to reject stray noise and interference.

Example 64.3

Equiripple Bandpass ASG Filter

A digital I/Q scheme, described in Note 60, involves a real-valued bandpass signal that has been frequency-shifted and sampled such that the passband extends from $f_s/8$ to $3f_s/8$, as shown in Figure 64.1(a). Before additional processing can be performed, the negative-frequency passband must be eliminated to produce an analytic signal having the positive-only spectrum shown in Figure 64.1(b).

- A desired response that meets the criteria in List 64.1 is shown in Figure 64.2(a).
- The `cfirpm` function requires the band-edge frequencies to be specified in terms of a normalized radian frequency, ω/π, so all specified values will range from −1.0 to +1.0. Figure 64.2(b) shows the desired response recast in frequency units acceptable to `cfirpm`. A value of 0.05 has been assumed for f_L.
- The following segment of MATLAB code designs the filter:

```
F=[-1,0.05,0.25,0.75,0.95,1];
A=[0,0,1,1,0,0];
C=cfirpm(90,F,A);
```

- This specification yields a 91-tap filter with the response shown in Figure 64.3.

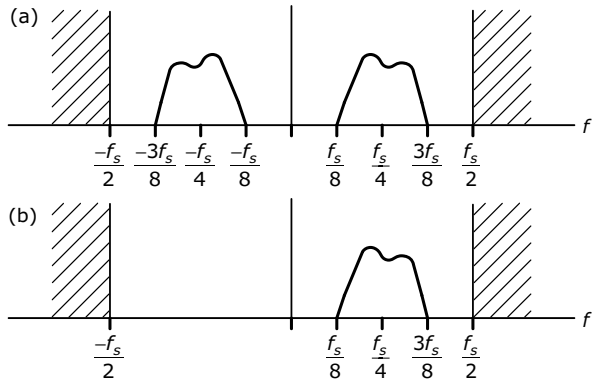


Figure 64.1 Spectra for Example 64.1: (a) spectrum for typical bandpass signal, (b) spectrum for corresponding analytic signal

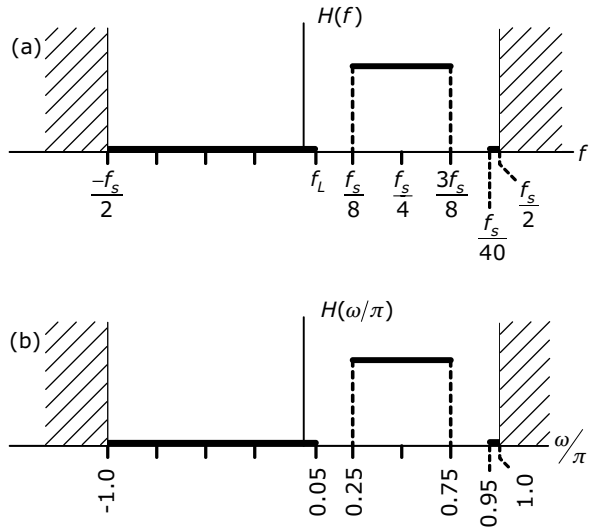


Figure 64.2 Desired response for the complex filter in Example 64.1: (a) band edges specified in terms of the sampling rate, (b) band edges specified in units acceptable to MATLAB `cfirpm` function

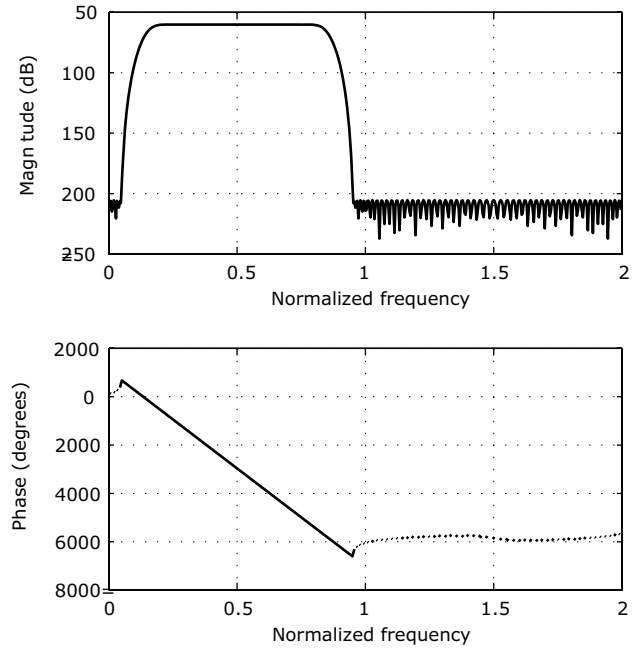


Figure 64.3 Frequency response for 91-tap filter designed in Example 64.1

Reference

1. M. Z. Komodromos, S. F. Russell, and P. T. P. Tang, "Design of FIR Filters with Complex Desired Frequency Response Using a Generalized Remez Algorithm," *IEEE Trans. Circuits and Systems*, vol. 42, no. 4, April 1995.

Generating I and Q Channels Digitally: Rader's Approach

This note presents an approach for generating inphase and and quadrature channels from a digitized, real-valued bandpass signal.

Radar and communications systems often involve quadrature receivers, which typically exhibit mismatch or imbalances between the amplitude and phase responses of their inphase and quadrature output channels. These imbalances introduce distortion into the signals of interest that are derived from the I and Q outputs of the receiver. Rader's approach is one of several digital I/Q techniques that have been devised to form the I and Q signals directly from a single real bandpass signal and thereby avoid the I/Q matching limitations of analog quadrature receivers.

Essential Facts

- Analog quadrature receivers exhibit imbalance between their *inphase* (I) and *quadrature* (Q) components.
- It is difficult and expensive to control the imbalances well enough to avoid severe distortion in the signals of interest (SOI).
- Rader's approach
  - avoids I/Q imbalance by digitizing a real-valued bandpass signal and creating the I and Q channels digitally
  - requires an analog mixing operation that moves the I/F down to a center frequency that is equal to the SOI's bandwidth of B Hz
  - requires A/D conversion at a rate of $4B$ samples per second
  - provides a convenient opportunity to remove any dc bias that may be introduced by the analog mixing operation

65.1 The Approach

Recipe 65.1 implements a digital I/Q approach depicted in Figure 65.1 and originally described by Rader in [1, 2]. The analytic signal generation (ASG) filter shown in the diagram can be implemented using any of the methods described in Notes 60 through 64. However, in [1], Rader initially implements this filter as a phase-splitting network consisting of a pair of low-order IIR filters, as shown in Figure 65.3. The top branch, which generates the real part of $\hat{x}_s[n]$, implements the transfer function

$$H_1(z) = z^{-1}\frac{z^{-2}-a^2}{1-a^2z^{-2}} \qquad (65.1)$$
$$a^2 = 0.5846832$$

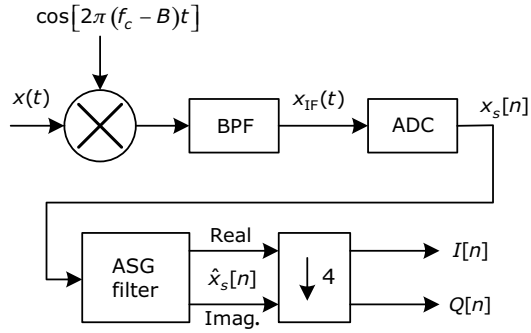


Figure 65.1 Block diagram of processing steps in Rader's digital I/Q approach

Recipe 65.1

Generating Digital I and Q

A block diagram of this processing sequence is shown in Figure 65.1. The RF signal, $x(t)$, is assumed to have a bandwidth of B Hz and a carrier frequency of $f_c \gg B$, as depicted in Figure 65.2.

1. The multiplier and bandpass filter form a mixer that translates the signal down to an intermediate frequency of B Hz to yield the signal spectrum shown in Figure 65.2(b).
2. The ADC samples the signal at a rate of $4B$ samples per second to yield a discrete-time signal with the spectrum shown in Figure 65.2(c).
3. The ASG filter forms the analytic associate of $x_s[n]$ by eliminating the negative frequency sideband to yield the result shown in Figure 65.2(d).
4. Downsampling by a factor of 4 causes the positive-frequency sideband to be shifted to a center frequency of zero, as shown in Figure 65.2(e). The real part of the corresponding complex signal is the inphase output, and the imaginary part is the quadrature output.

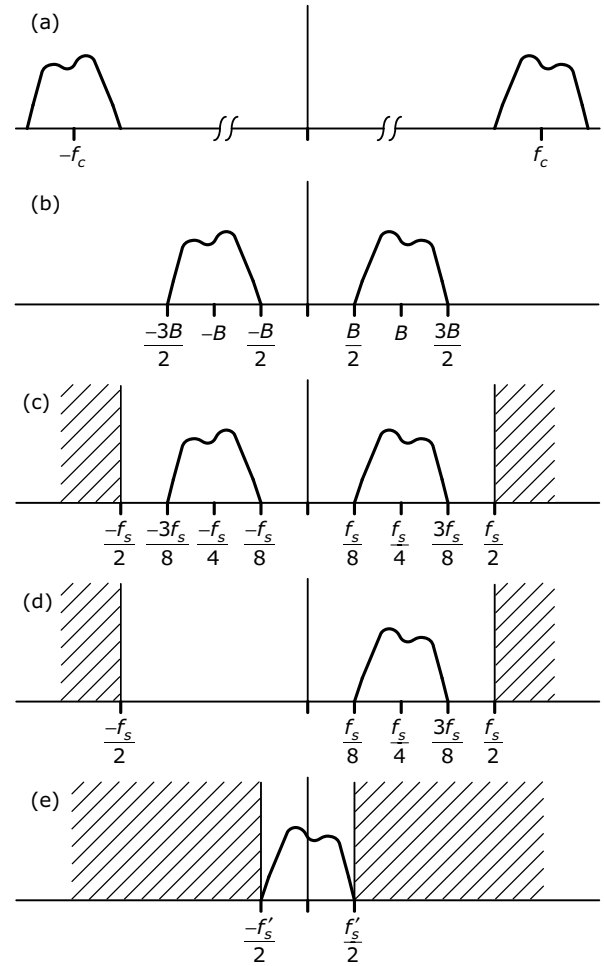


Figure 65.2 Spectra of signals at various points in Rader's digital I/Q approach: (a) spectrum of bandpass RF input signal, $x(t)$, with a bandwidth of B and a carrier frequency of f_c, (b) spectrum of IF signal, $x_{IF}(t)$, produced by the mixer, (c) baseband image of spectrum for $x_s[n]$ created from $x_{IF}(t)$ by sampling at a rate of f_s samples per second, (d) spectrum of the analytic signal, $\hat{x}_s[n]$, formed by complex filtering of $x_s[n]$, (e) spectrum of final quadrature baseband signal after down sampling by a factor of 4. The hashed areas indicate frequencies outside the bandwidth supported by the sampling rate.

The bottom branch, which generates the imaginary part of $\hat{x}_s[n]$, implements the transfer function

$$H_2(z) = \frac{z^{-2} - b^2}{1 - b^2 z^{-2}} \qquad (65.2)$$
$$b^2 = 0.1380250$$

The coefficients a^2 and b^2 can be obtained using the design procedure presented in Note 63. Rader points out that because the filter output, $\hat{x}_s[n]$, is destined to be downsampled by a factor of 4, it is possible to build a more efficient filter that absorbs the downsampling operation and generates only every fourth output. This improved realization is shown in Figure 65.4. The downsampling is accomplished by the switches on the output lines.

Careful examination of Figure 65.3 reveals that even-indexed samples of $\mathrm{Re}\{\hat{x}_s[n]\}$ depend only upon odd-indexed samples of $\hat{x}_s[n]$, and even-indexed samples of $\mathrm{Im}\{\hat{x}_s[n]\}$ depend only upon even-indexed samples of $\hat{x}_s[n]$. The input switch shown in Figure 65.4 routes odd-indexed samples to the top branch and even-indexed samples to the bottom branch, thereby enabling each branch to run at half the original sample rate. Because of the reduced rate, the z^{-2} delays become z^{-1} delays. The z^{-1} delay in the top branch of Figure 65.3 is accomplished by the relative phasing of the output switch closures in Figure 65.4. To establish the proper phasing, the sample of $Q[n']$ that is generated when $n = 0$ is discarded so that $I[0]$ is generated when $n = 1$, and $Q[0]$ is generated when $n = 2$.

Normally, shifting a bandpass spectrum such as Figure 65.2(d) to create a baseband spectrum such as Figure 65.2(e) would require some sort of mixing operation.

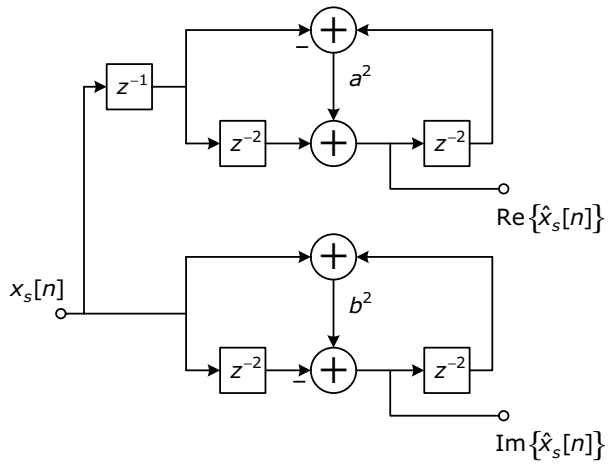


Figure 65.3 IIR realization for the complex filter

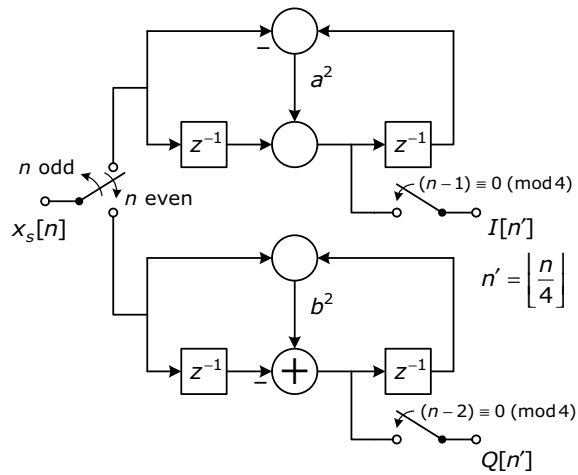


Figure 65.4 Improved IIR realization for the complex filter

However, the particular combination of center frequency and sampling rate used in Rader's approach completely eliminates the need for a final mixing step. If the signal corresponding to the spectrum in Figure 65.5(a) is downsampled by a factor of 4, two things happen:

1. Images of the spectrum are created at intervals of the new sampling rate, $f'_S = f_S/4$, as shown in Figure 65.5(b).
2. The supported passband of the system changes from $\pm f_S/2$ to $\pm f'_S/2 = \pm f_S/8$, with exactly one image of the signal's spectrum falling in this new passband, as shown in Figure 65.5(c).

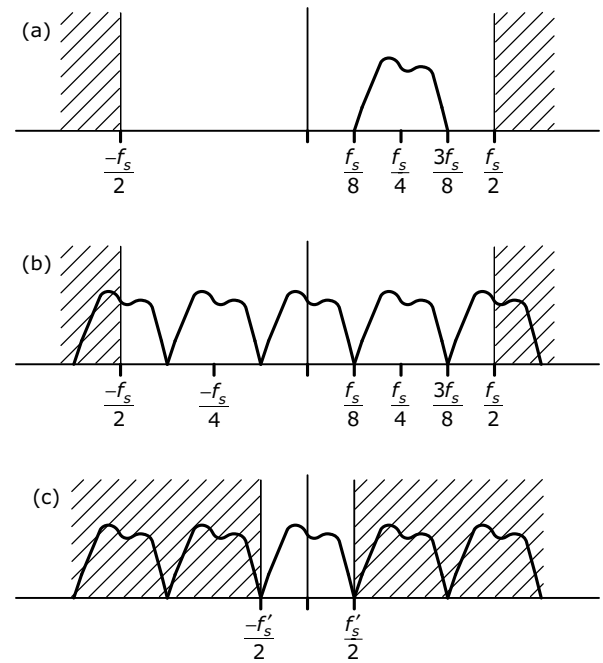


Figure 65.5 When the signal having the spectrum in (a) is downsampled by a factor of 4, images are created at intervals of $f_s/4$, as in (b), and the supported passband shrinks, as depicted in (c), from $\pm f_s/2$ to $\pm f'_s/2$ where $f'_s = f_s/4$.

References

1. C. M. Rader, "A Simple Method for Sampling In-Phase and Quadrature Components," *IEEE Trans. Aerospace and Electronic Syst.*, vol. 20, no. 6, Nov. 1984, pp. 821–824.
2. C. M. Rader, "Method and Apparatus for Sampling In-Phase and Quadrature Components," U. S. Patent 4,794,556, dated Dec. 27, 1988.
3. B. Gold and C. M. Rader, *Digital Processing of Signals*, McGraw-Hill, 1969.

Generating I and Q Channels Digitally: Generalization of Rader's Approach

Rader's digital I and Q approach presented in Note 65 requires that the real bandpass IF signal be sampled at a rate equal to four times the signal's bandwidth. This note shows how Rader's scheme can be generalized to support sampling rates that are not four times the signal bandwidth.

Rader's efficient IIR design for the complex filter depends upon downsampling by a factor of four, which in turn drives the requirement that the sampled I/F bandpass signal be centered at a frequency equal to one-fourth the sampling rate. This constraint is required so that downsampling by a factor of four shifts the surviving sideband to a center frequency of zero. If an FIR filter, or even a "standard" IIR filter, is used in place of Rader's IIR phase-splitting design, this constraint on the I/F can be relaxed.

If Rader's efficient IIR structure is not used, the important constraint is that the sample rate, f_s, be an integer multiple of the intermediate frequency, and that after filtering, the signal is downsampled by the same integer factor so that the new sample rate, f_s', equals the intermediate frequency:

$$f_s = Mf_{\text{IF}}$$
$$f_s' = \frac{f_s}{M} = f_{\text{IF}} \qquad (66.1)$$

The relationship between f_s, f_{IF}, and M is constrained this way to ensure that the downsampling operation always creates a spectral image at zero frequency. However, Eq. (66.1) says nothing about the relationship between f_{IF} and the bandwidth B. To avoid aliasing, the final sample rate, f_s', must exceed the bandwidth:

$$f_s' > B$$
$$f_s/D > B$$
$$f_s > BD$$

For convenience, let's say that $f_s = QB$, where $Q \geq D$. For Rader's original frequency plan, $Q = D = 4$, but there are many other possible combinations if downsampling by four is not needed to support the special IIR filter structure. The design case presented in the next section considers one of these alternatives.

66.1 Design Case

Figure 66.1 shows a sequence of models extracted from a simulation of pulse Doppler processing for a radar receiver. This simulation was constructed using some of the `PracSim` models discussed in [1]. Assume a

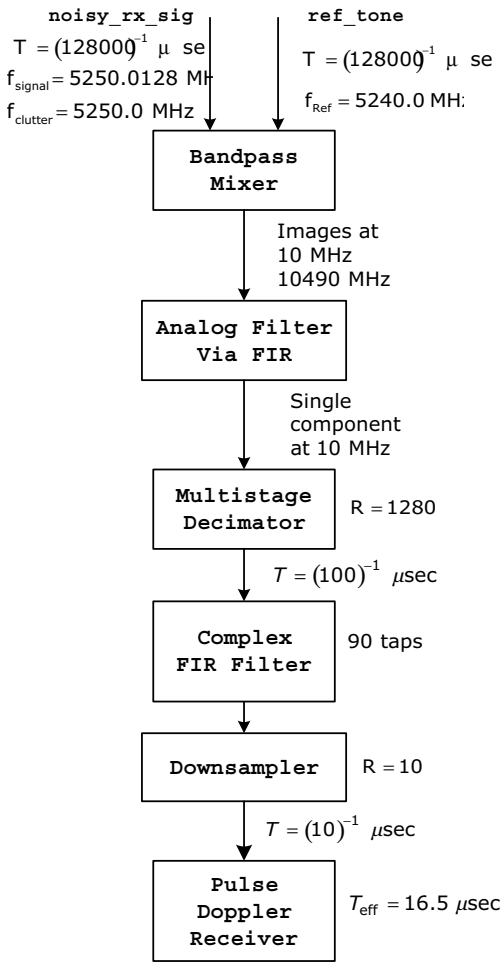


Figure 66.1 Block diagram for simulation of digital I/Q generation

transmit frequency of 5250.0 MHz. One of the inputs to the mixer is a received signal consisting of a tone at 5250.0 MHz, representing a return from stationary clutter, plus a tone at 5250.0128 MHz, representing a moving-target return that exhibits an inbound Doppler shift of 12.8 kHz. The reference tone is assumed to come from a frequency and timing source that is common to both the transmitter and receiver portions of the actual radar.

The bandwidth of the analog received signal varies depending upon the amount of Doppler shift created by moving targets in the radar's field of view. Let's assume that the bandwidth of the signal is guaranteed to be less than 2 MHz. Following Rader's original frequency plan, we would specify a sampling rate of 8 MHz. However, let's assume that this particular radar has several different operating modes, and one of these modes needs a sampling rate higher than 91 MHz. To avoid changing the ADC rate for different operating modes, we can design the pulse Doppler operation using an ADC rate of f_s = 100 MHz and a final sample rate of f_s' =10 MHz:

$$f_{IF} = f_s' = 10^7$$

$$D = \frac{f_s}{f_s'} = \frac{10^8}{10^7} = 10$$

$$Q = \frac{f_s}{B} = \frac{10^8}{2\times 10^6} = 50$$

For a received center frequency of 5250 MHz and a reference tone at 5240 MHz, the bandpass mixer produces signals centered at 10 MHz and at 10,490 MHz. The bandpass filter removes the high-frequency signal, leaving only the signal at 10 MHz. The mixer inputs, mixer output, and bandpass filter output are all analog signals that are simulated in this reference design using a sampling rate that corresponds to 1.28×10^{11} samples per second.

The multistage decimator performs filtering and downsampling to reduce the sample rate to 10^8 samples per second. The decimator output in the simulation corresponds to the ADC output in Figure 65.1 in Note 65. (Quantization performed by the ADC is not included in this simulation.)

The power spectral densities of the mixer output and decimator output are shown in Figures 66.2 and 66.3, respectively. The complex FIR filter model implements the 90-tap design having the response shown in Figures 66.4 and 66.5 that was developed in Example 63.1 of Note 63. The PSD for the complex filter's output is shown in Figure 66.6. The attenuated negative-frequency sideband is clearly visible. A zoomed view of the PSD in the vicinity of 10 MHz is shown in Figure 66.7. All of the fine structure shown in this view is due to the pulse gating of the radar signal. Figure 66.8 shows the corresponding view of the PSD when the gating is disabled

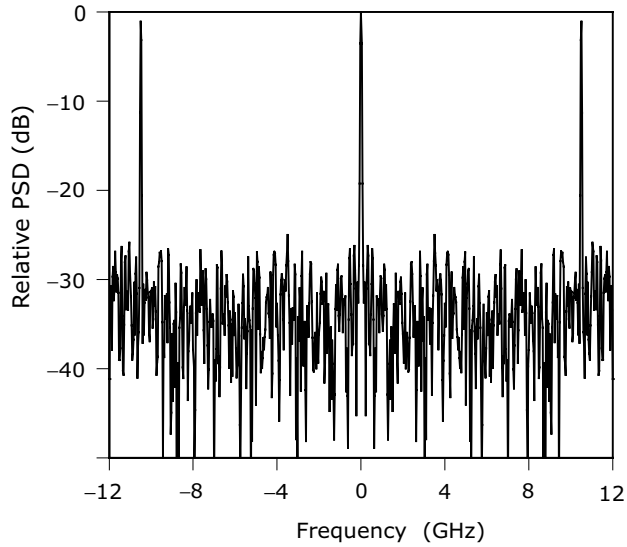


Figure 66.2 Power spectral density for output of mixer model

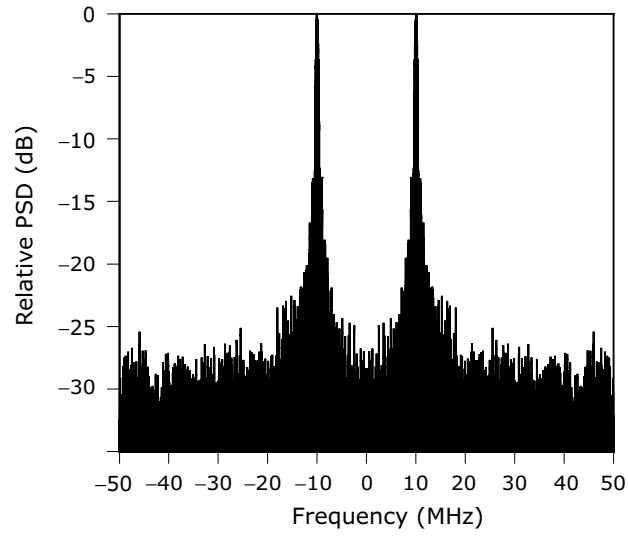


Figure 66.3 Power spectral density at decimator output

and a continuous tone is transmitted. The center-to-center spacing of the fingers in Figure 66.7 is 60.6 kHz, which corresponds to the *pulse repetition frequency* (PRF) of the gating pulses. The amplitudes of these fingers follow an envelope that corresponds to the spectrum of a single rectangular gating pulse.

The downsampler implements our choice of $D = 10$ to yield a final sample rate of 10^7. Figure 66.9 shows the low-frequency details for the PSD of the downsampler output. The components at zero and 12.8 kHz are the ones we're expecting to find based on the makeup of the original received signal. The other components are images due to the pulse gating.

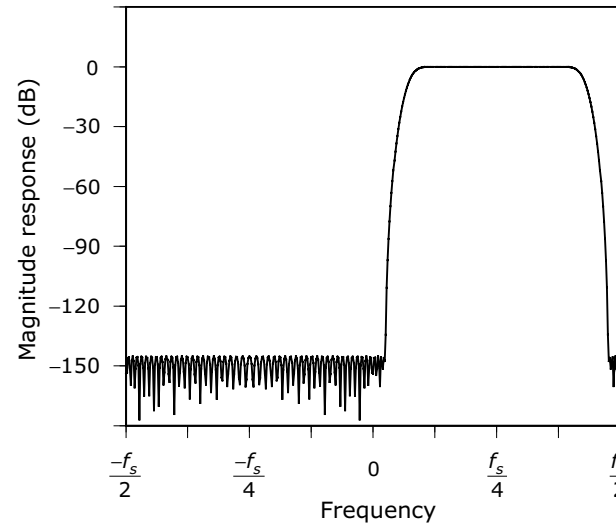


Figure 66.4 Magnitude response of 90-tap complex filter designed in Example 63.1 of Note 63

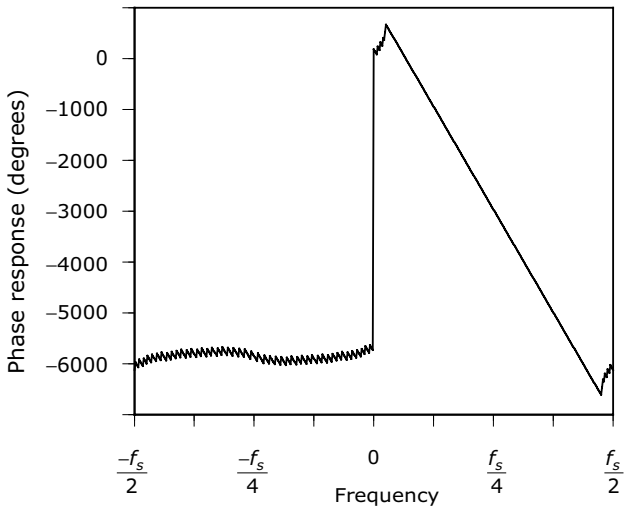


Figure 66.5 Phase response of 90-tap complex filter designed in Example 63.1 of Note 63

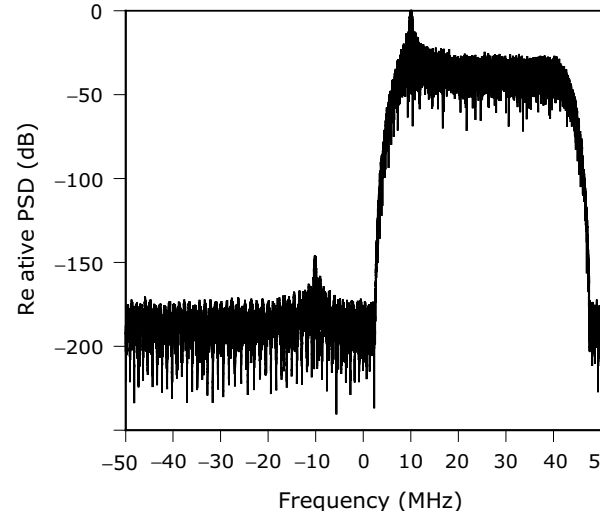


Figure 66.6 Power spectral density at output of the complex filter

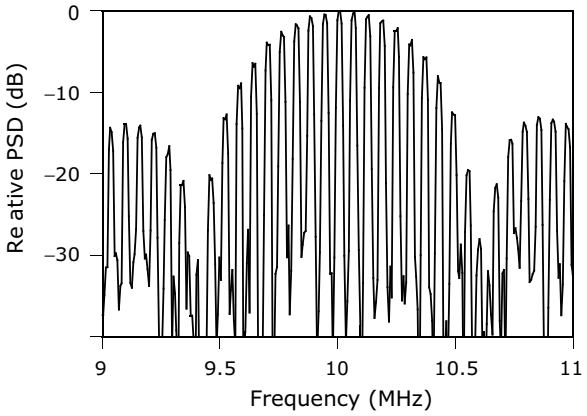


Figure 66.7 Enlarged view of the detail near 10 MHz for the PSD shown in Figure 66.6

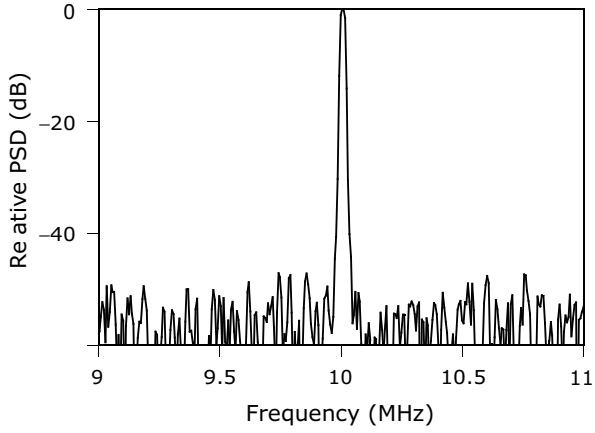


Figure 66.8 Enlarged view of the detail near 10 MHz for the PSD of the complex filter output when pulse gating of the transmitter signal is disabled

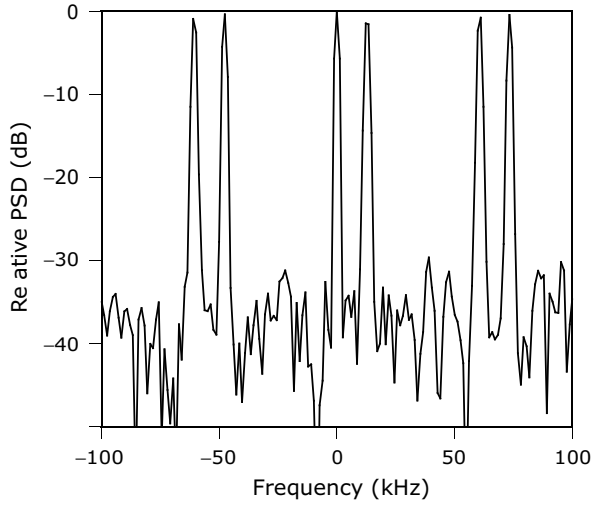


Figure 66.9 Power spectral density at output of the final downsampler

Reference

1. C. B. Rorabaugh, *Simulating Wireless Communication Systems*, Prentice Hall, 2004.

Parametric Modeling of Discrete-Time Signals

Many sampled real-world signals can be modeled as the output of a discrete-time filter that is driven by a simple input sequence. Many techniques for the analysis and characterization of signals are based upon such signal models. This note serves as an introduction to the basic model classes that are explored in subsequent notes.

Parametric modeling is the name given to the concept of modeling a discrete-time signal as the output of a discrete-time filter that is driven by a specified input sequence—usually either white noise or the unit-sample function. Many techniques used in the areas of statistical signal processing, spectral estimation, and adaptive filtering are built upon the theoretical foundations provided by parametric modeling. The discussions in this note focus on the various signal models that can be built around a filter that exhbits the following properties.

- The filter is linear.
- The filter is causal.
- The filter is time-invariant.
- Implementation of the filter requires a finite number of memory elements.
- Computation of each of the filter's output samples can be accomplished using a finite number of arithmetic operations.

Any filter that exhbits all of these properties can be represented by a constant-coefficient, linear difference equation of the form

$$y[n] = -\sum_{k=1}^{p} a_p[k]\,y[n-k] + \sum_{k=0}^{q} b_q[k]\,x[n-k] \quad \textbf{(67.1)}$$

List 67.1

Types of Parametric Models

- When $q = 0$ with $p > 0$, the resulting *all-pole* (AP) recursive filter can be driven by a unit sample function to construct an all-pole model of a deterministic signal.
- When $q = 0$ with $p > 0$, the resulting all-pole recursive filter can be driven by a white noise source to construct an *autoregressive* (AR) model of a random process.
- When $p = 0$ with $q > 0$, the resulting *all-zero* (AZ) transversal filter can be driven by a unit sample function to construct an all-zero model of a deterministic signal.
- When $p = 0$ with $q > 0$, the resulting all-zero transversal filter can be driven by a white noise source to construct a *moving-average* (MA) model of a random process.
- When both $p > 0$ and $q > 0$, the resulting *pole-zero* (PZ) filter can be driven by a unit sample function to construct a pole-zero model of a deterministic signal.
- When both $p > 0$ and $q > 0$, the resulting *pole-zero* filter can be driven by a white noise source to construct an *autoregressive-moving-average* (ARMA) model of a random process.

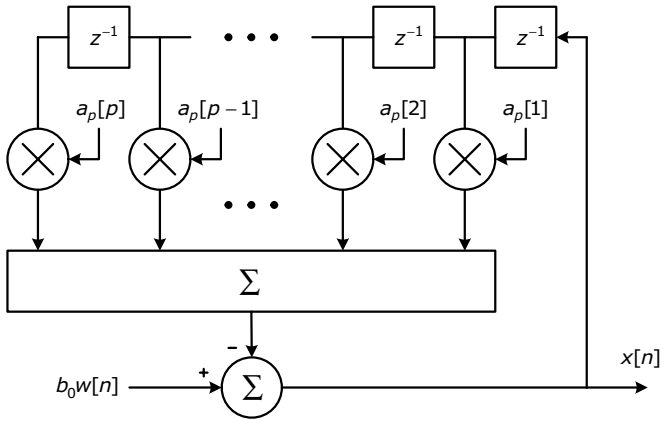


Figure 67.1 All-pole filter configured for generating an AR(p) process

where $x[n]$ is the input to the filter, and $y[n]$ is the output. The corresponding system function is a ratio of polynomials in z^{-1}:

$$H(z)=\frac{Y(z)}{X(z)} =\frac{\sum_{k=0}^{q} b_q[k]z^{-k}}{1+\sum_{k=1}^{p} a_p[k]z^{-k}}=\frac{B(z)}{A(z)} \tag{67.2}$$

The system function can be expressed in terms of its poles, p_k, and its zeros, z_k, by factoring the numerator and denominator polynomials to obtain

$$H(z)=\frac{B(z)}{A(z)}=G\frac{\prod_{k=1}^{q}\left(1-z_k z^{-1}\right)}{\prod_{k=1}^{p}\left(1-p_k z^{-1}\right)} \tag{67.3}$$

There are several items in Eq. (67.3) that require special mention to avoid possible confusion.

- The index for the numerator product in Eq. (67.3) begins at $k=1$, even though the corresponding summation index in Eq. (67.2) begins at $k=0$. If $b_q[0]\neq 0$, it can be factored out so that the numerator in Eq. (67.2) becomes

$$B(z)=b_q[0]\left(1+\sum_{k=1}^{q}\frac{b_q[k]}{b_q[0]}z^{-k}\right) \tag{67.4}$$

The factor, $b_q[0]$, is absorbed into G, and the zeros of the system function are the q roots of the polynomial:

$$1+\sum_{k=1}^{q}\frac{b_q[k]}{b_q[0]}z^{-k} \tag{67.5}$$

- The subscripted variable, z_k, represents the kth zero of the system function, while the unsubscripted variable z represents the discrete-frequency variable used in the z-transform. This dual use of z may be confusing, but it is fairly standard throughout the DSP literature.
- The subscripted variable, p_k, represents the kth pole of the system function, while the unsubsripted variable p represents the order of the denominator polynomial in both Eqs. (67.2) and (67.3). This second usage follows the convention (that is almost universal in the statistical literature) of using p to represent the autoregressive order of autoregressive and autoregressive-moving-average random processes (both of which will be discussed later in this note).

Typically three different specializations of Eq. (67.1) are considered, along with two different types of input signals, resulting in the six different types of parametric models given in List 67.1. Some texts, such as [1] and [2], carefully distinguish

between models for determinstic signals (all-pole, all-zero, and pole-zero) and models for stochastic signals (autoregressive, moving-average, and autoregressive-moving-average), as we have done in List 67.1. Other texts, especially some of the older works like [3] or [4], categorize models as AR, MA, or ARMA regardless of the signals that are meant to be modeled or the nature of the input signals used to drive the filters.

Although there is considerable overlap, different mathematical tools and analysis techniques are associated with each of these six models. Parametric modeling is best known for its role in modeling stochastic signals, where it allows complicated signals to be characterized by the statistics of the input noise and the coefficients of the filter.

In some cases, the parametric model's ouput signal is the true goal of the effort, and the coefficients are merely a means to an end. Such cases can be said to comprise the *synthesis role* of parametric modeling, because the primary goal is to *synthesize* the output signal. An example of the synthesis role would be a signal generator that produces a signal with particular characteristics that might be needed for testing some new DSP algorithm that is separate and distinct from the parametric model itself. In the synthesis role, the filter coefficients often are obtained from a mathematical specification of either the desired output signal or the output signal's autocorrelation sequence.

However, in many cases, the model's filter coefficients are themselves the true goal of the effort, and the filter itself is never actually implemented. Such cases can be said to comprise the *analysis role* of parametric modeling. Many modern spectrum estimation algorithms are based on taking the z-transform of the coefficients from a parametric model that has been fitted to a sequence of captured signal samples.

In parametric modeling, the filter coefficients are obtained using specialized methods that are different from the usual FIR and IIR design techniques. The methods discussed in this note are appropriate for fitting models to signals for which either the signal of interest or its autocorrelation sequence can be specified mathematically over all time. This condition typically exists for applications that use parametric modeling in its synthesis role. In cases where knowledge about the signal is limited to a finite sequence of measured values, as it usually is in the the analysis role, other techniques, such as those presented in Notes 69 through 71, must be used.

67.1 Pole-Zero and Autoregressive-Moving-Average Models

The pole-zero filter is the most general of three filters with rational system functions that are commonly used for signal modeling. As shown in Figure 67.2, a pole-zero filter has a portion that is structured like a conventional FIR filter and a portion that is structured like a conventional IIR filter.

The autoregressive-moving-average (ARMA) model results from using a white noise source to drive a pole-zero filter that has been designed to produce an output signal having particular autocorrelation properties. The ARMA model for the time series $x[n]$ is given by

$$x[n] = -\sum_{k=1}^{p} a_p[k]x[n-k] + \sum_{k=0}^{q} b_q[k]w[n-k] \qquad (67.6)$$

where $w[n]$ is the input driving sequence. The first summation in Eq. (67.6) is in the form of an IIR filter and constitutes the autoregressive portion of the model. The second summation is in the form of an FIR filter and constitutes the moving-average portion of the model. When the input, $w[n]$, is the unit sample, the corresponding system function has the form

$$H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{k=0}^{q} b_q[k]z^{-k}}{1 + \sum_{k=1}^{p} a_p[k]z^{-k}} \qquad (67\ 7)$$

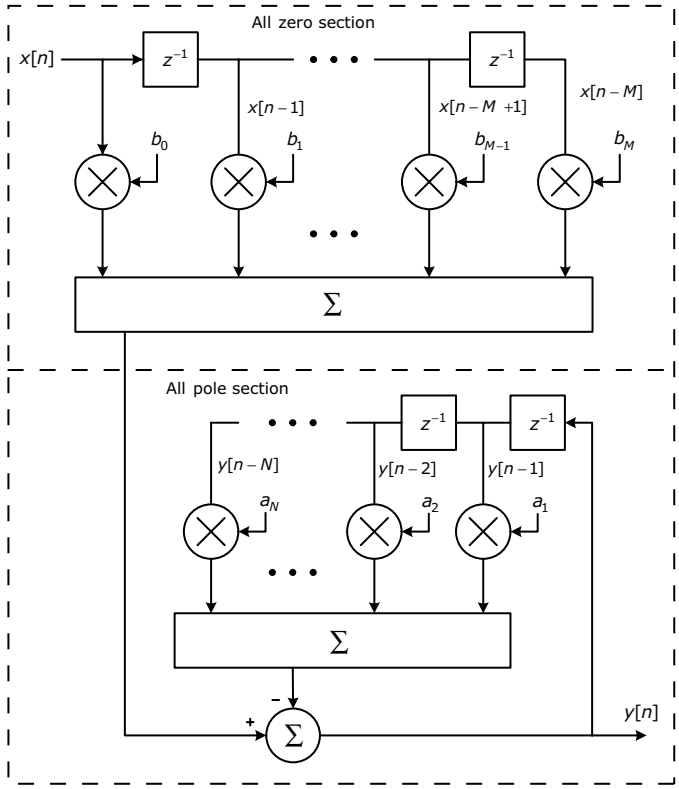


Figure 67.2 Block diagram for a pole-zero filter

The a_p in Eqs. (67.6) and (67.7) are the autoregressive parameters, and the b_q are the moving-average parameters. The subscripts on a and b may seem unnecessary in this context, but they are used here for consistency with the notation used in the generalized *Levinson recursion* in which each iteration of the recursion increments the order of the filter being estimated. The notation $a_p[n]$ represents the nth AR coefficient for a filter having an autoregressive order of p. Similarly, $b_q[m]$ represents the mth MA coefficient for a filter having a moving-average order of q.

An ARMA model having p autoregressive parameters and $q+1$ moving-average parameters is denoted as ARMA(p,q). The moving-average order of an ARMA model equals the highest power of z^{-1} appearing in the denominator of the system function. Likewise, the autoregressive order of an ARMA model equals the highest power of z^{-1} appearing in the numerator of the system function. Thus an ARMA(p,q) model has $q+1$ moving-average coefficients, but only p autoregressive coefficients because there is no coefficient $a_p[0]$.

If the input is a white noise process, the output is a wide-sense stationary random process known as an *autoregressive-moving-average process* of order (p,q). The power spectral density of the ARMA process is given by

$$P_{\text{ARMA}}(f) = T\sigma_w^2 \left|\frac{B(f)}{A(f)}\right|^2 = T\sigma_w^2 \frac{\sum_{k=0}^{q} b_q[k]\exp(-j2\pi fk)}{\sum_{k=1}^{p} a_p[k]\exp(-\boldsymbol{j}2\pi fk)} \tag{67.8}$$

where σ_w^2 is the variance of $w[n]$ and T is the sampling interval. If a particular signal can be

adequately represented by an ARMA model that is fitted to the signal, then the coefficients of the model can be used in Eq. (67.8) to estimate the PSD of the signal.

67.2 All-Pole and Autoregressive Models

In its most common form, which is depicted in Figure 67.1, *autoregressive* (AR) *modeling* involves using an all-pole IIR filter driven by a white noise source to produce an output signal having specified autocorrelation properties. The AR model for the time series, $x[n]$, is given by

$$x[n] = b_0 w[n] - \sum_{k=1}^{p} a_p[k]x[n-k] \tag{67.9}$$

When the input, $w[n]$, is the unit sample, the corresponding system function has the form

$$H(z) = \frac{b_0}{1 + \sum_{k=1}^{p} a_p[k]z^{-k}} \tag{67.10}$$

If the input driving sequence, $w[n]$, is a white noise process, the output is a wide-sense stationary random process known as an autoregressive process of order p. The power spectral density of the AR(p) process, $x[n]$, is given by

$$P_{\mathrm{AR}}(f) = \frac{T|b_0|^2 \sigma_w^2}{|A(f)|^2} = \frac{T|b_0|^2 \sigma_w^2}{\left|1 + \sum_{k=1}^{p} a_p[k]\exp(-j2\pi fk)\right|^2} \tag{67.11}$$

where σ_w^2 is the variance of $w[n]$ and T is the sampling interval.

67.3 All-Zero and Moving-Average Models

The *moving-average* (MA) model for the time series, $x[n]$, is given by

$$x[n] = \sum_{k=0}^{q} b_q[k]w[n-k] \tag{67.12}$$

When the input, $w[n]$, is the unit sample, the corresponding system function has the form

$$H(z) = \sum_{k=0}^{q} b_q[k]z^{-k} \tag{67.13}$$

If the input driving sequence, $w[n]$, is a white noise process, the output is a wide-sense stationary random process known as a *moving-average process* of order q. The power spectral density of the MA(q) process, $x[n]$, is given by

$$P_{\mathrm{MA}}(f) = T\sigma_w^2 |B(f)|^2 = T\sigma_w^2 \left| \sum_{k=0}^{q} b_q[k]\exp(-j2\pi fk) \right|^2 \tag{67.14}$$

References

1. M. H. Hayes, *Statistical Digital Signal Processing and Modeling*, Wiley, 1996.
2. D. G. Manolakis, V. K. Ingle, S. M. Kogon, *Statistical and Adaptive Signal Processing*, Artech House, 2005.
3. S. M. Kay, *Modern Spectral Estimation: Theory and Applications*, Prentice Hall, 1988.
4. S. L. Marple, *Digital Spectral Analysis with Applications*, Prentice Hall, 1987.
5. N. Levinson, "The Wiener RMS (Root Mean Square) Error Criterion in Filter Design and Prediction," *J. Math Phys.*, vol. 25, 1947, pp. 261–278.
6. J. Durbin, "The Fitting of Time Series Models," *Rev. Inst. Int. Statist.*, vol. 28, 1960, pp. 233–243.

Autoregressive Signal Models

Many real-world signals can be modeled as autoregressive (AR) processes. The properties of AR processes have led to the development of numerous techniques for analyzing and characterizing of such signals. This note serves as an introduction to these techniques, which are explored in subsequent notes.

An *autoregressive* process of order p (often denoted as an "AR(p) process") can be generated using a p-stage all-pole filter driven by a white noise source, as shown in Figure 68.1. The configuration shown in the figure implements the difference equation (68.1) given in Math Box 68.1, and is sometimes referred to as an *autoregressive signal model*.

The AR model can be used in either a *synthesis* role or an *analysis* role. In the synthesis role, the model order p, the coefficients, b_0 and $a_p[k]$, and the white noise variance, σ_w^2, are all assumed to be known, and the filter is used to generate the signal sequence, $x[n]$, recursively. An example of the synthesis role is the implementation of a signal generator that produces a signal with particular statistical

Essential Facts

- The autoregressive signal model consists of an all-pole filter driven by a white noise source.
- The *Yule-Walker* normal equations provide a critical link between the AR model parameters and the *autocorrelation sequence* (ACS) that is used to characterize the desired output of the model.
- The matrix of autocorrelation values that appears in the Yule-Walker equation is *Toeplitz*, and therefore a computationally efficient technique called the *Levinson-Durbin* recursion can be used to solve for the coefficients $a_p[k]$.
- When the AR signal model is used in a *synthesis* role, the ACS values needed for the Yule-Walker equations usually are specified directly by the designer based on the theoretical ACS of the desired model output.
- When the AR signal model is used in an *analysis* role, the ACS values needed for the Yule-Walker equations usually are estimated from a sampled segment of the signal being analyzed. Use of the AR model in an analysis role is discussed further in Notes 69 through 73.

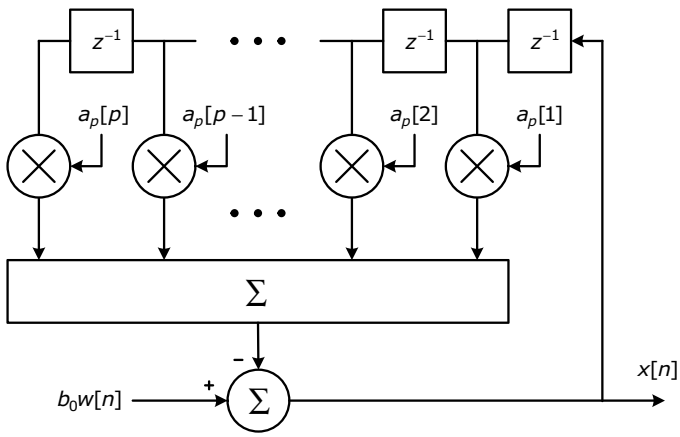


Figure 68.1 All-pole filter configured for generating an AR(p) process

characteristics that might be needed for testing or simulation purposes.

In the analysis role, a signal, $x[n]$, is known for some values of n, and is assumed to be an autoregressive signal. The problem is to find values for $a_p[k]$ and σ_w^2 that yield the estimated signal

$$\check{x}[n] = b_0 w[n] - \sum_{k=1}^{p} a_p[k] x[n-k]$$

that is in some sense the "best" estimate for $x[n]$. In most cases where an AR signal model is used in an analysis role, the model's filter coefficients are themselves the true goal of the effort, and the filter itself is never actually implemented. For example, Eq. (MB 68.3), given in Math Box 68.1, uses the filter coefficients to estimate the power spectral density of an autoregressive signal.

Math Box 68.1

Equations for Characterizing Autoregressive Processes

Difference Equation

$$x[n] = b_0 w[n] - \sum_{k=1}^{p} a_p[k] x[n-k] \qquad \text{(MB 68.1)}$$

System Function

$$H(z) = \frac{b_0}{1 + \sum_{k=1}^{p} a_p[k] z^{-k}} \qquad \text{(MB 68 2)}$$

Power Spectral Density

$$P_{\mathrm{AR}}(f) = \frac{T|b_0|^2 \sigma_w^2}{|A(f)|^2} = \frac{T|b_0|^2 \sigma_w^2}{\left|1 + \sum_{k=1}^{p} a_p[k] \exp(-j2\pi fkT)\right|^2} \qquad \text{(MB 68 3)}$$

References

1. M. H. Hayes, *Statistical Digital Signal Processing and Modeling*, Wiley, 1996.
2. N. Levinson, "The Wiener RMS (Root Mean Square) Error Criterion in Filter Design and Prediction," *J. Math Phys.*, vol. 25, 1947, pp. 261–278.
3. J. Durbin, "The Fitting of Time Series Models," *Rev. Inst. Int. Statist.*, vol. 28, 1960, pp. 233–243.

Fitting AR Models to Stochastic Signals: The Yule-Walker Method

This note describes the Yule-Walker method for finding the parameters needed to fit an autoregressive model to a finite sequence of samples obtained from a stochastic signal. This method is conceptually straightforward; it is based on the simple idea of substituting an estimated autocorrelation sequence (ACS) for the true ACS in the Yule-Walker equations that are presented in Note 68. Other methods for fitting an AR model to a finite sequence of signal samples are presented in Notes 72 and 73.

The Yule-Walker method is a technique for fitting an autoregressive model to a stochastic signal that is assumed to be autoregressive, but where knowledge about the signal is limited to a sequence of N samples, $x[0]$ through $x[N-1]$. The Yule-Walker equations for an AR process are given as

$$\mathbf{Ra} = -\mathbf{r} \tag{69.1}$$

where

$$\mathbf{R} = \begin{bmatrix} r_x[0] & r_x[-1] & \cdots & r_x[-p+1] \\ r_x[0] & r_x[0] & & r_x[-p+2] \\ \vdots & \vdots & \ddots & \vdots \\ r_x[p-1] & r_x[p-2] & & r_x[0] \end{bmatrix}$$

$$\mathbf{a} = \begin{bmatrix} a_p[1] \\ a_p[2] \\ \vdots \\ a_p[p] \end{bmatrix} \qquad \mathbf{r} = \begin{bmatrix} r_x[1] \\ r_x[2] \\ \vdots \\ r_x[p] \end{bmatrix}$$

The Yule-Walker method is based on using *estimated* values for the autocorrelation sequence (ACS) to populate the **R** matrix in Eq. (69.1). Assuming that $x[n]$ is known only for N values $x[0]$ through $x[N-1]$, the estimated ACS values, $r_x[-p+1]$ through $r_x[p]$, can be obtained as

$$\check{r}_x[k] = \begin{cases} \frac{1}{N} \sum_{n=0}^{N-1-k} x^*[n]x[n+k] & k \geq 0 \\ \check{r}_x^*[-k] & k < 0 \end{cases} \tag{69.2}$$

Because the matrix **R** is Toeplitz, the Levinson recursion can be used, as shown in Recipe 69.1, to solve for the coefficients $a_p[1]$ through $a_p[p]$.

Essential Facts

- The Yule-Walker method is based on simply using estimated values of the ACS to form the **R** matrix that appears in the Yule-Walker equations.
- The **R** matrix that appears in the Yule-Walker equation is Toeplitz, and therefore a computationally efficient technique called the Levinson recursion can be used to solve for the coefficients a_p.
- The Yule-Walker method does not perform as well as the Burg method or other lattice-based methods in situations where N, the number of available samples, is small.
- The Yule-Walker method for autoregressive modeling of random processes as described in the current topic is computationally identical to the autocorrelation method for all-pole modeling of deterministic signals described in Note 70. Because of this equivalence, some authors use the names *autocorrelation* and *Yule-Walker* interchangeably, even though the genesis of each approach is different.

69.1 About Recipe 69.1

Equation (69.3) is the *biased* estimate of the ACS for finite N. The *unbiased* estimate is the same, except for a normalizing factor of $1/(N-k)$ that replaces the factor of $1/N$ in Eq. (69.3). Normally, it is preferable to use unbiased rather than biased estimators, but for values of k that approach the value of N, the unbiased ACS estimator can produce results where the autocorelation estimate at lag zero is smaller than the estimate at one or more non-zero lags. This result can sometimes lead to matrix equations that cannot be solved [4]. Therefore, the biased ACS estimate is almost always the one used in the Yule-Walker method.

Along the way to producing the coefficients, $a_p[i]$, for an AR(p) model, the Levinson recursion generates coefficients for all of the lesser-order models, AR(1) through AR($p-1$).

Recipe 69.1

Yule-Walker Method Using the Levinson Recursion

1. Use signal samples, $x[0]$ through $x[N-1]$, to compute the sequence of autocorrelation values, $r_x[0]$ through $r_x[p]$, as

$$r_x[k] = \frac{1}{N}\sum_{n=0}^{N-1-k} x^*[n]x[n+k] \qquad (69.3)$$

2. Initialize:

$$a_1[1] = \frac{r_x[1]}{r_x[0]}$$

$$\rho_1 = \left(1-\left|a_1[1]\right|^2\right) r_x[0]$$

3. For $k = 2, 3, \ldots, p$ in succession, compute

$$a_k[k] = -\frac{r_x[k] + \sum_{m=1}^{k-1} a_{k-1}[m] r_x[k-m]}{\rho_k - 1}$$

$$a_k[i] = a_{k-1}[i] + a_k[k]\, a^*_{k-1}[k-i]$$

for $i = 1, 2, \ldots, k-1$

$$\rho_k = \left(1-\left|a\,[k]\right|^2\right) \rho_{k-1}$$

4. The coefficients $a_p[i]$ are the desired result.

References

1. M. H. Hayes, *Statistical Digital Signal Processing and Modeling*, Wiley, 1966.
2. N. Levinson, "The Wiener RMS (Root Mean Square) Error Criterion in Filter Design and Prediction," *J. Math Phys.*, vol. 25, 1947, pp. 261–278.
3. J. Durbin, "The Fitting of Time Series Models," *Rev. Inst. Int. Statist.*, vol. 28, 1960, pp. 233–243.
4. S. L. Marple, *Digital Spectral Analysis with Applications*, Prentice Hall, 1987.

Fitting All-Pole Models to Deterministic Signals: Autocorrelation Method

This note describes the autocorrelation method for finding the parameters needed to fit an all-pole model to a finite sequence of samples obtained from a deterministic signal. Although the genesis of each method is different, the autocorrelation method is computationally identical to the Yule-Walker method described in Note 69.

The autocorrelation method is a technique for fitting an all-pole model to a deterministic signal that is assumed to be autoregressive, but where knowledge about the signal is limited to a sequence of N samples, $x[0]$ through $x[N-1]$. This technique is based on normal equations that result from perfroming an error minimization over the semi-infinite interval, $0 \le n$, while assuming that the data sequence equals zero for $n < 0$ and for $n \ge N$. Recipe 70.1 implements the autocorrelation method by using the Levinson-Durbin recursion to solve the normal equations that result from this particular error-minimization strategy.

70.1 Background

The all-pole model has the form

$$\check{x}[n] = -\sum_{k=1}^{p} a_p[k]x[n-k] \qquad (70.1)$$

where the coefficients, $a_p[k]$, are selected to minimize the error

$$E_p = -\sum_{n=0}^{\infty} \left| e[n] \right|^2 \qquad (70.2)$$

with

$$\begin{aligned} e[n] &= x[n] - \check{x}[n] \\ &= x[n] + \sum_{k=1}^{p} a_p[k]x[n-\mathrm{k}] \end{aligned} \qquad (70.3)$$

Evaluation of Eq. (70.2) calls for some values of $x[n]$ that fall outside of the known sequence, $x[0]$ through $x[N-1]$. In the autocorrelation method, this difficulty is addressed by assuming that $x[n]$ equals zero for $n < 0$ and for $n \ge N$. Under this assumption, the normal equations become

$$\sum_{k=1}^{p} a_p[k]x[n-\mathrm{k}] = -r_x[n] \quad \text{for } n = 0,1,\ldots,p \qquad (70.4)$$

Essential Facts

- The autocorrelation method is derived by performing an error minimization over the semi-infinite interval, $0 \le n$, but the data sequence, $x[n]$, is assumed to equal zero for $n < 0$ and for $n \ge N$.
- Most authors claim that the autocorrelation method is less accurate and exhibits poorer resolution than the covariance method desribed in Note 69. However, such claims are based on observed performance and do not have strong theoretical support.
- The **R** matrix that appears in the Yule-Walker equation is Toeplitz, and therefore a computationally efficient technique called the Levinson recursion can be used to solve for the coefficients, $a_p[k]$.
- Using the unbiased ACS estimate to form the **R** matrix can result in matrix equations that cannot be solved. Therefore, the biased ACS estimate [Eq. (70.7)] is usually used in the autocorrelation method.
- The autocorrelation method for all-pole modeling of deterministic signals as described in this note is computationally identical to the Yule-Walker method for autoregressive modeling of random signals that is described in Note 69. Because of this equivalence, some authors use the names *autocorrelation* and *Yule-Walker* interchangeably, even though the genesis of each approach is different.

where

$$r_x[k] = \begin{cases} \frac{1}{N}\sum_{n=0}^{N-1-k} x^*[n]x[n+k] & k \geq 0 \\ r_x^*[-k] & k < 0 \end{cases} \quad (70.5)$$

In matrix form, the normal equations are

$$\mathbf{Ra} = -\mathbf{r} \quad (70.6)$$

where

$$\mathbf{R} = \begin{bmatrix} r_x[0] & r_x[-1] & \cdots & r_x[-p+1] \\ r_x[0] & r_x[0] & & r_x[-p+2] \\ \vdots & \vdots & \ddots & \vdots \\ r_x[p-1] & r_x[p-2] & & r_x[0] \end{bmatrix}$$

$$\mathbf{a} = \begin{bmatrix} a_p[1] \\ a_p[2] \\ \vdots \\ a_p[p] \end{bmatrix} \qquad \mathbf{r} = \begin{bmatrix} r_x[1] \\ r_x[2] \\ \vdots \\ r_x[p] \end{bmatrix}$$

70.2 About Recipe 70.1

Equation (70.7) is the *biased* estimate of the ACS for finite N. The *unbiased* estimate is the same, except for a normalizing factor of $1/(N - k)$ that replaces the factor of $1/N$. Normally, it is preferable to use unbiased rather than biased estimators, but for values of k that approach the value of N, the unbiased ACS estimator can produce results where the autocorelation estimate at lag zero is smaller than the estimate at one or more non-zero lags. This result can sometimes lead to matrix equations that cannot be solved [4]. Therefore, the biased ACS estimate is almost always the one used in the autocorrelation method.

Along the way to producing the coefficients, $a_p[i]$, for an AR(p) model, the Levinson recursion generates coefficients for all of the lesser-order models, AR(1) through AR($p-1$).

Recipe 70.1

Autocorrelation Method Using the Levinson Recursion

1. Use signal samples, $x[0]$ through $x[N-1]$, to compute the sequence of autocorrelation values, $r_x[0]$ through $r_x[p]$, as

$$r_x[k] = \frac{1}{N}\sum_{n=0}^{N-1-k} x^*[n]x[n+k] \quad (70.7)$$

2. Initialize:

$$a_1[1] = \frac{r_x[1]}{r_x[0]}$$

$$\rho_1 = \left(1 - |a_1[1]|^2\right) r_x[0]$$

3. For $k = 2, 3, \ldots, p$ in succession, compute

$$a_k[k] = -\frac{r_x[k] + \sum_{m=1}^{k-1} a_{k-1}[m] r_x[k-m]}{\rho_{k-1}}$$

$$a_k[i] = a_{k-1}[i] + a_k[k] a_{k-1}^*[k-i] \quad \text{for } i = 1, 2, \ldots, k-1$$

$$\rho_k = \left(1 - |a_k[k]|^2\right) \rho_{k-1}$$

4. The coefficients $a_p[i]$ are the desired result.

References

1. M. H. Hayes, *Statistical Digital Signal Processing and Modeling*, Wiley, 1966.
2. N. Levinson, "The Wiener RMS (Root Mean Square) Error Criterion in Filter Design and Prediction," *J. Math Phys.*, vol. 25, 1947, pp. 261–278.
3. J. Durbin, "The Fitting of Time Series Models," *Rev. Inst. Int. Statist.*, vol. 28, 1960, pp. 233–243.
4. S. L. Marple, *Digital Spectral Analysis with Applications*, Prentice Hall, 1987.

Fitting All-Pole Models to Deterministic Signals: Covariance Method

This note describes the covariance method for finding the parameters needed to fit an all-pole model to a finite sequence of samples obtained from a deterministic signal.

The covariance method is a technique for fitting an all-pole model to a deterministic signal that is assumed to be autoregressive, but where knowledge about the signal is limited to a sequence of N samples, $x[0]$ through $x[N-1]$. This method is an alternative to the autocorrelation method described in Note 70. Rather than optimizing the total error over all non-negative n and assuming that $x[n] = 0$ for n outside the interval $[0, n-1]$, as the autocorrelation method does, the covariance method is based on minimizing the error over only those values of n for which the error can be evaluated from the known data sequence $x[0]$, $x[1]$, . . . , $x[N-1]$. Under these constraints, the matrix appearing in the normal equations is not Toeplitz, and consequently the Levinson-Durbin recursion cannot be used to solve for the coefficients, $a_p[k]$.

Essential Facts

- The covariance method is derived by performing an error minimization over the finite interval, $[0, N-1]$, for which the data sequence is known.
- Most authors claim that the covariance method is more accurate and offers better resolution than the autocorrelation method described in Note 70. However, such claims are based on observed performance and do not have strong theoretical support.
- The inferior performance of the autocorrelation method is usually explained as being due to "end effects" resulting from setting all unknown values of $x[n]$ equal to zero.
- Because the **R** matrix is not Toeplitz, solving for the coefficients, $a_p[k]$, using the covariance method is more computationally expensive than for the autocorrelation method.
- In theory, the filter defined by the coefficients, a_p, obtained from the covariance method may be unstable, but this is rarely encountered in practice.
- The covariance method is very similar to the (modern) Prony method.

71.1 Background

In the autocorrelation method discussed in Note 70, the error to be minimized is

$$E_p = \sum_{n=0}^{\infty} \left| e[n] \right|^2 \tag{71.1}$$

where

$$e[n] = x[n] + \sum_{k=1}^{p} a_p[k] x[n-k] \tag{71.2}$$

Ouitside of the range, $0 \le n \le N-1$, the value for $x[n]$ is assumed to be zero. The covariance method minimizes the error only over the range of indices for which the necessary values of $x[n]$ are available. Computing $e[n]$ for values of n less than p requires values of $x[n]$ for n less than zero; therefore the lower limit on the summation in Eq. (71.1) must be changed to p. Similarly,

computing $e[n]$ for $n > N-1$ requires values of $x[n]$ for $n > N-1$, so the upper limit on the summation must be changed to N −1. Under these constraints, the error to be minimized becomes

$$E_p = \sum_{n=p}^{N-1} \left| e[n] \right|^2 \tag{71.3}$$

Using this criterion results in the normal equations

$$\mathbf{Ra} = -\mathbf{r} \tag{71.4}$$

where

$$\mathbf{R} = \begin{bmatrix} r_x[1,1] & r_x[1,2] & r_x[1,3] & r_x[1,p] \\ r_x[2,1] & r_x[2,2] & r_x[2,3] & r_x[2,p] \\ r_x[3,1] & r_x[3,2] & r_x[3,3] & r_x[3,p] \\ \vdots & \vdots & & \vdots \\ r_x[p,1] & r_x[p,2] & r_x[p,3] & r_x[p,p] \end{bmatrix}$$

$$\mathbf{a} = \begin{bmatrix} a_p[1] \\ a_p[2] \\ a_p[3] \\ \\ a_p[p] \end{bmatrix} \qquad \mathbf{r} = \begin{bmatrix} r_x[1,0] \\ r_x[2,0] \\ r_x[3,0] \\ \\ r_x[p,0] \end{bmatrix}$$

and

$$r_x[k,m] = \frac{1}{N-p} \sum_{n=p}^{N-1} x[n-m]\, x^*[n-k] \qquad \text{for } k = 1, 2, \ldots, p \quad m = 0, 1, \ldots, p \tag{71.5}$$

The matrix in Eq. (71.4) has the properties of a covariance matrix, but it is not really a covariance matrix according to the usual definition. Nevertheless, in most of the literature concerning linear prediction and speech processing, this matrix is referred to as a covariance matrix, and hence the method using this matrix is called the covariance method. The matrix is not Toeplitz, and therefore the Levinson-Durbin recursion cannot be used to solve for the coefficients

Recipe 71.1

Solving the Covariance Normal Equations Using the Cholesky Decomposition

All values of $r_x[i, j]$ needed for this recipe are computed using

$$r_x[k,m] = \frac{1}{N-p} \sum_{n=p}^{N-1} x[n-m]\, x^*[n-k] \qquad \text{for } k = 1, 2, \ldots, p \quad m = 0, 1, \ldots, p$$

1. Decompose the **R** matrix as $\mathbf{R} = \mathbf{LDL}^H$:
   (a) Initialize $d_1 = r_x[1, 1]$.
   (b) For $i = 2, 3, \ldots, p$ in succession, compute

$$l_{ij} = \frac{r_x[i,1]}{d_1} \qquad \text{for } j = 1$$

$$l_{ij} = \frac{r_x[i,j]}{d_j} \qquad \text{for } j = 2, 3, \ldots, i-$$

$$d_i = r_x[i,i] - \sum_{k=1}^{i-1} d_k \left| l_{ik} \right|^2$$

2. Compute y as follows.
   (a) Initialize $y_1 = r_x[1, 0]$.
   (b) For $k = 2, 3, \ldots, p$ in succession, compute

$$y_k = r_x[k,0] - \sum_{j=1}^{k-1} l_{kj} y_j$$

3. Solve for a as follows.
   (a) Initialize

$$a_p[p] = \frac{y_p}{d_p}$$

   (b) For $k = p-1, p-1, \ldots, 1$ in succession, compute

$$a_p[p] = \frac{y_k}{d_k} - \sum_{j=k+1}^{p} l_{jk}^* a_p[j]$$

$a_p[k]$. However, the matrix is Hermitian (conjugate symmetric), so Eq. (71.4) can be solved using Cholesky decomposition, which imposes a computational burden that is about half that of Gaussian elimination.

71.2 Notes Regarding Recipe 71.1

Morf, et al. [2] have shown that the form of the **R** matrix in Eq. (71.4) is the product of two non-square Toeplitz matrices, and that this fact can be exploited to devise an inversion algorithm that is not as efficient as the Levinson-Durbin recursion, but is more efficient than Cholesky decomposition.

Alternative derivations of similar algorithms are provided by McClellan in [3] and by Marple in [4] and [5]. The phrase "FIR System Identification" might make the title of Marple's paper [4] seem unrelated to the problem of estimating coefficients for an IIR filter in an AR model. However, as discussed in Note 70, there is an equivalence between an FIR filter used for linear prediction and an IIR filter used for autoregressive modeling.

References

1. M. H. Hayes, *Statistical Digital Signal Processing and Modeling*, Wiley, 1996.
2. M. Morf, B. Dickinson, T. Kailath, and A. Viera, "Efficient Solution of Covariance Equations for Linear Prediction," *IEEE Trans. ASSP*, vol. 25, no. 5, Oct. 1977, pp. 429–433.
3. J. H. McClellan, "Parametric Signal Modeling," Chapter 1 in *Advanced Topics in Signal Processing*, J. S. Lim and A. V. Oppenheim, eds., Prentice Hall, 1988.
4. S. L. Marple, "Efficient Least Squares FIR System Identification," *IEEE Trans. ASSP*, vol. 29, Feb. 1981, pp. 62–73.
5. S. L. Marple, *Digital Spectral Analysis with Applications*, Prentice Hall, 1987.

Autoregressive Processes and Linear Prediction Analysis

Linear prediction is a topic that is closely related to the generation of an autoregressive (AR) process. Techniques developed for linear prediction yield results that can be used in autoregressive modeling. *Forward* linear prediction estimates the value of the sample, $x[n]$, as a weighted combination of the m previously observed samples, $x[n-1]$ through $x[n-m]$

$$\hat{x}_m^f[n] = -\sum_{k=1}^{m} a_m^f[k]\,x[n-k] \qquad (72.1)$$

where the "hat" notation is used to distiguish estimated values, and the superscript f is used to indicate forward prediction. An FIR filter corresponding to Eq. (72.1) is shown in Figure 72.1. Of course, when the estimated sequence, $\hat{x}_m^f[n]$, is compared to the observed sequence, $x[n]$, there are, in general, some nonzero errors, $e_m^f[n]$, given by

$$e_m^f[n] = x[n] - \hat{x}_m^f[n] \qquad (72.2)$$

Combining Eqs. (72.1) and (72.2) yields

$$e_m^f[n] = x[n] + \sum_{k=1}^{m} a_m^f[k]\,x[n-k] \qquad (72.3)$$

which corresponds to the FIR *prediction error filter* shown in Figure 72.2. Rearranging terms in (72.3) yields

$$x[n] = -\left[\sum_{k=1}^{m} a_m^f[k]\,x[n-k]\right] + e_m^f[n] \qquad (72.4)$$

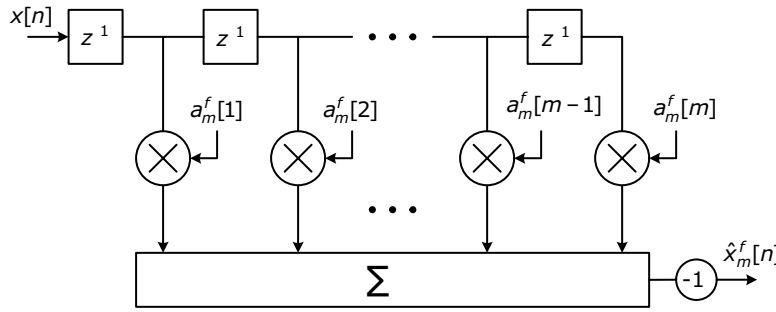


Figure 72.1 Linear prediction filter

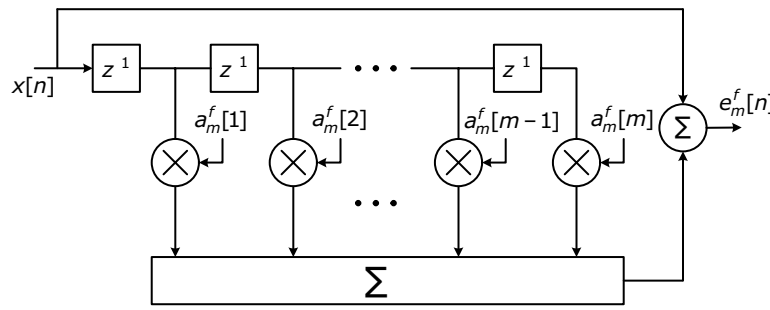


Figure 72.2 Prediction error filter

The algebraic structure of Eq. (72.4) is identical to the algebraic structure of the difference equation

$$x[n] = -\left[\sum_{k=1}^{p} a_p[k]x[n-k]\right] + b_0 w[n] \quad (72.5)$$

which is used in Note 68 to define an AR process. Equation (72.5) corresponds to the all-pole IIR filter shown in Figure 72.3. It might seem odd that of two equations having identical algebraic structures, one represents an FIR filter and one represents an IIR filter. The critical difference is the reversed role of inputs and outputs between the two filters. In Eq. (72.3), $x[n]$ represents the input, and the noise-like process, $e_m^f[n]$, represents the output. In Eq. (72.5), $x[n]$ represents the output, and the noise process, $b_0\, w[n]$, represents the input. In a sense, the two filters can be viewed as inverses of each other. If the filter defined by Eq. (72.5) is used to generate an AR(p) process, and this process is used as input to Eq. (72.3), with m set equal to p, and $a_m^f[k]$ set equal to $a_p[k]$ for each value of k from 1 through p, the resulting error sequence $e_m^f[n]$ exactly equals the driving sequence, $b_0\, w[n]$. The prediction filter "undoes" the work done by the AR process generator, thereby obtaining the original driving sequence as the end result. However, if the input to Eq. (72.3) is not an AR(p) process, then the error, $e_m^f[n]$, is not, in general, a white process.

Backward linear prediction estimates the value of the sample, $x[n]$, as a weighted combination of the m

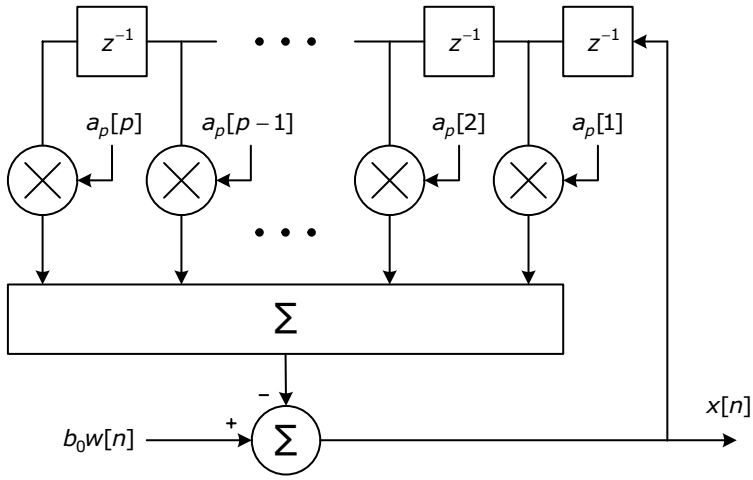


Figure 72.3 All-pole filter for generating an AR(p) process

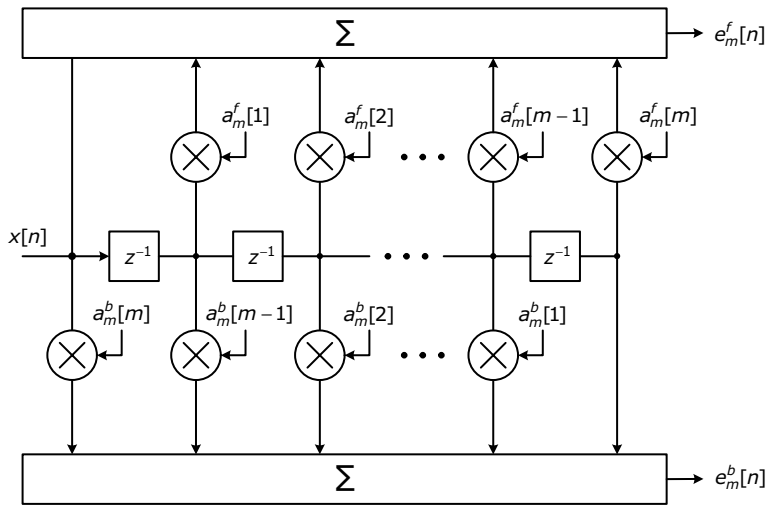


Figure 72.4 Linear prediction error filter

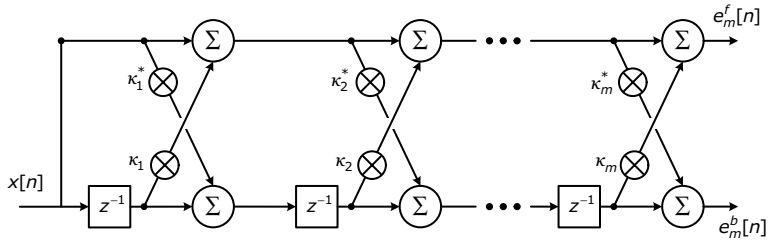


Figure 72.5 Lattice implementation for linear prediction error filter

Recipe 72.1

Generating Prediction-error-filter Tap Weights and Reflection Coefficients for the Corresponding Lattice

1. Initialize:

$$\kappa_1 = a_1[1] = \frac{r_{xx}[1]}{r_{xx}[0]} \tag{72.8}$$

$$\rho_1 = \left(1 - |\kappa_1|^2\right) r_{xx}[0] \tag{72.9}$$

2. For $m = 2, 3, \ldots M$ in succession compute

$$k_m = \frac{r_{xx}[m] + \sum_{k=1}^{m-1} a_{m-1}[k] r_{xx}[m-k]}{\rho_{m-1}} \tag{72.10}$$

$$a_m[i] = a_{m-1}[i] + \kappa_m a_{m-1}^*[m-k] \qquad k = 1, 2, \ldots, m-1 \tag{72.11}$$

$$a_m[m] = \kappa_m \tag{72.12}$$

$$\rho_m = \left(1 - |k_m|^2\right) \rho_{m-1} \tag{72.13}$$

Recipe 72.2

Computing Prediction-error-filter Tap Weights from Reflection Coefficients for the Corresponding Lattice

The reflection coefficients, $\kappa_1, \kappa_2, \ldots, \kappa_M$, and the ACS value at lag zero, $r_{xx}[0]$, are known.

1. Initialize:

$$a_1[1] = \kappa_1 \tag{72.14}$$

2. For $m = 2, 3, \ldots, M$ in succession, compute

$$a_m[k] = a_{m-1}[k] + \kappa_m a_{m-1}^*[m-k] \qquad k = 1, 2, \ldots, m-1 \tag{72.15}$$

$$a_m[m] = \kappa_m \tag{72.16}$$

subsequently observed samples, $x[n+1]$ through $x[n+m]$:

$$\hat{x}_m^b[n] = -\sum_{k=1}^{m} a_m^b[k] x[n-k] \tag{72.6}$$

For any given sequence, $x[n]$, and estimation order, m, the set of forward coefficients, $a_m^f[k]$, that minimizes the variance of $e_m^f[n]$, and the set of backward coefficients, $a_m^b[k]$, that minimizes the variance of $e_m^b[n]$ are related via complex conjugation:

$$a_m^b[k] = \left(a_m^f[k]\right)^* \tag{72 7}$$

Furthermore, the minimized variance of $e_m^f[n]$ equals the minimized variance of $e_m^b[n]$. Figure 72.4 shows a dual-output FIR filter that generates both the forward and backward prediction errors. It can be shown [5] that such a prediction error filter also can be

Recipe 72.3

Inverse Levinson Recursion for Computing Reflection Coefficients from the Prediction-error-filter Tap Weights

Tap weights, $a_M[1]$, $a_M[2], \ldots,$ $a_M[M]$, are known.

1. Initialize:

$$\kappa_m = a_m[m] \tag{72.17}$$

2. For $m = M, M-1, \ldots, 2$ in succession compute

$$a_{m-1}[k] = \frac{a_m[k] - \kappa_m a_m^*[m-k]}{1 - |\kappa_m|^2} \qquad k = 1, 2, \ldots, m-1 \tag{72.18}$$

$$\kappa_{m-1} = a_{m-1}[m-1] \tag{72.19}$$

implemented as a lattice of the form shown in Figure 72.5. The lattice implementation tends to be more tolerant of quantization and roundoff.

A prediction error filter of order M is completely specified by either the set of tap weights, $a_M[k]$ for $k = 1, 2, \ldots, M$, or by the set of reflection coefficients, κ_k, for $k = 1, 2, \ldots, M$, plus input variance, $r_{xx}[0]$. Based on the relationships between these alternative specifications, there are four possible analysis scenarios.

1. The ACS of the input process is either known or able to be estimated for lags 0 through M, and the requirement is to compute the set of tap weights, $a_M[1]$, $a_M[2], \ldots, a_M[M]$, for a prediction filter of order M.
2. The ACS of the input process is either known or able to be estimated for lags 0 through M, and the requirement is to compute the set of reflection coefficients, $\kappa_1, \kappa_2, \ldots, \kappa_M$.
3. The reflection coefficients $\kappa_1, \kappa_2, \ldots, \kappa_M$, are known, and the ACS of the input process is known for lag zero. The requirement is to compute the set of tap weights, $a_M[1]$, $a_M[2], \ldots, a_M[M]$ for a prediction filter of order M.
4. The set of tap weights, $a_M[1]$, $a_M[2], \ldots, a_M[M]$, for a prediction filter of order M is known, and the requirement is to compute the corresponding set of reflection coefficients, $\kappa_1, \kappa_2, \ldots, \kappa_M$.

The first two scenarios are handled by Recipe 72.1. For the third scenario, where the coefficients are known, Recipe 72.1 can be simplified to obtain Recipe 72.2.

For the fourth scenario, in which the tap weights of the Mth order prediction error filter are known, it is necessary to use the inverse form of the Levinson recursion to compute the tap weights for the prediction error filters of orders $M-1, M-2, \ldots 1$. The reflection coefficients are then obtained as

$$\kappa_m = a_m[m] \quad m = 1, 2, \ldots, M$$

This approach for handling the fourth scenario is implemented in Recipe 72.3.

References

1. M. H. Hayes, *Statistical Digital Signal Processing and Modeling*, Wiley, 1996.
2. M. Morf, B. Dickinson, T. Kailath, and A. Viera, "Efficient Solution of Covariance Equations for Linear Prediction," *IEEE Trans. ASSP*, vol. 25, no. 5, Oct. 1977, pp. 429–433.
3. J. H. McClellan, "Parametric Signal Modeling," Chapter 1 in *Advanced Topics in Signal Processing*, J. S. Lim and A. V. Oppenheim, eds., Prentice Hall, 1988.
4. S. L. Marple, "Efficient Least Squares FIR System Identification," *IEEE Trans. ASSP*, vol. 29, Feb. 1981, pp. 62–73.
5. S. L. Marple, *Digital Spectral Analysis with Applications*, Prentice Hall, 1987.

Estimating Coefficients for Autoregressive Models: Burg Algorithm

Essential Facts

- Like other lattice-based algorithms for fitting all-pole models, the Burg algorithm works directly with the signal's sample values. For short signal segments, algorithms using this direct approach tend to perform better than non-lattice algorithms that must explicitly estimate values in the signal's autocorrelation sequence.
- The Burg algorithm can be implemented efficiently using a modified form of the Levinson recursion.
- AR spectral estimates produced using coefficients obtained via the Burg algorithm have been known to exhibit some anomalous behaviors such as spurious spectral lines, line-splitting, and phase-dependent frequency shifting of some spectral lines.
- There are several techniques for mitigating the anomalous behaviors, but these techniques can significantly reduce the implementation efficiency.

The Burg algorithm [1, 2] is a technique for fitting an AR model to a signal that is represented by a sequence of N measured samples, $x[0]$ through $x[N-1]$. What sets the Burg method apart from other techniques for estimating AR model parameters are the assumptions made about signal values $x[n]$ for $n > 0$ and for $n \geq N$. The autocorrelation method described in Note 70 assumes that unknown values of $x[n]$ are zero. The covariance method described in Note 71 makes no assumptions about unknown values for $x[n]$, but uses an optimization strategy that is structured to use only the N measured values. FFT-based methods assume the same periodic extension of values that is implicit in the DFT . The Burg method does not make any upfront assumptions about the unknown values for $x[n]$, but instead adopts values that do not add any information or entropy to the signal. For this reason, Burg's method is sometimes called the *maximum entropy method* (MEM), and spectrum analysis using Burg's method is sometimes called *maximum entropy spectrum analysis* (MESA).

The strategy of the algorithm is to characterize the AR model in terms of reflection coefficients for the equivalent lattice filter implementation. The Burg method is distinguished from other methods using a similar strategy by the optimization criteria used in estimating the coefficients. In Burg's method, the reflection coefficients are computed sequentially using a variant of the Levinson recursion to minimize, at each stage, the sum of the forward and backward prediction error powers.

Specifically, the Burg algorithm computes the reflection coefficient, κ_k, as

$$\kappa_k = \frac{-2\sum_{n=k}^{N-1} e_{k-1}^{f}[n]\, e_{k-1}^{b^*}[n-1]}{\sum_{n=k}^{N-1}\left(\left|e_{k-1}^{f}[n]\right|^2 + \left|e_{k-1}^{b}[n-1]\right|^2\right)} \tag{73.1}$$

Recipe 73.1

Burg Algorithm for AR Coefficient Estimation

1. Initialize:

$$\rho_0 = \frac{1}{N}\sum_{n=0}^{N-1}\left|x[n]\right|^2$$

$$e_k^f[n] = e_{k-1}^f[n] + \kappa_k e_{k-1}^b[n-1] \quad n = k+1, k+2, \ldots, N-1$$

$$e_k^b[n] = e_{k-1}^b[n-1] + \kappa_k^* e_{k-1}^f[n] \quad n = k, k+1, \ldots, N-2$$

2. For $k = 1, 2, \ldots, p$ compute

$$\kappa_k = \frac{-2\sum_{n=k}^{N-1} e_{k-1}^f[n] e_{k-1}^{b*}[n-1]}{\sum_{n=k}^{N-1}\left(\left|e_{k-1}^f[n]\right|^2 + \left|e_{k-1}^b[n-1]\right|^2\right)}$$

$$\rho_k = \left(1 - \left|\kappa_k\right|^2\right)\rho_{k-1}$$

$$a_k[i] = \begin{cases} a_{k-1}[i] + \kappa_k a_{k-1}^*[k-i] & i = 1, 2, \ldots, k-1 \\ \kappa_k & i = k \end{cases}$$

$$e_k^f[n] = e_{k-1}^f[n] + \kappa_k e_{k-1}^b[n-1] \quad n = k+1, k+2, \ldots, N-1$$

$$e_k^b[n] = e_{k-1}^b[n-1] + \kappa_k^* e_{k-1}^f[n] \quad n = k, k+1, \ldots, N-2$$

3. The coefficients κ_k for $k = 1, 2, \ldots, p$ are the desired result for a lattice filter implementation. The coefficients $a_p[i]$ for $i = 1, 2, \ldots, p$ are the desired result for an IIR implementaion.

where the forward estimation error and backward estimation error at each stage are obtained from results of the prior stage as

$$e_k^f[n] = e_{k\ 1}^f[n] + \kappa_k e_{k\ 1}^b[n-1]$$

$$e_k^b[n] = e_{k-1}^b[n-1] + \kappa_k^* e_{k-1}^f[n]$$

Assuming that values for $\kappa_1, \kappa_2, \ldots, \kappa_{k-1}$ have been selected and remain fixed, the value for κ_k computed by Eq. (73.1) minimizes the stage k error, which is given by

$$E_k = \sum_{n=k}^{N-1}\left|e_{k-1}^f[n]\right|^2 + \sum_{n=k}^{N-1}\left|e_{k-1}^b[n]\right|^2$$

Derivations showing that coefficients computed by Eq. (73.1) do in fact minimize E_k can be found in [3] and [4].

Using the reflection coefficients obtained from Eq. (73.1), the coefficients for an AR filter implementation can be obtained recursively (in model order) as

$$a_k[i] = \begin{cases} a_{k-1}[i] + \kappa_k a_{k-1}^*[k-i] & i < k \\ \kappa_k & i = k \end{cases} \quad (73.2)$$

where k denotes the model order, and i is the delay index of the particular coefficient. A filter constructed using these coeffcients will be both stable and minimum-phase.

The complete Burg algorithm is provided in Recipe 73.1.

73.1 Line Splitting

It has been observed [5] that AR spectra produced using the Burg algorithm sometimes exhibit *line splitting*, a phenomenon in which a signal component at a single frequency gives rise to spectral lines at several closely-spaced but distinct frequencies. Line splitting is most likely to occur under the following conditions:

1. The signal-to-noise ratio (SNR) is high.
2. The initial phase of sinusoidal signal component(s) is an odd multiple of $\pi/4$.
3. The signal segment used for estimating the AR coefficients contains an odd number of quarter cycles for sinusoidal signal components.
4. The number of AR coefficients being estimated is a large fraction of the number of samples in the signal segment being used for estimating the AR coefficients.

References

1. J. P. Burg, "Maximum entropy spectral analysis," *Proc. 37th Meeting, Society of Exploration Geophysics*, 1967. Reprinted in *Modern Spectrum Analysis*, IEEE Press, 1978.
2. J. P. Burg, "Maximum entropy spectral analysis," Ph.D. dissertation, Dept. Geophysics, Stanford University, Stanford, Calif., May 1975.
3. M. H. Hayes, *Statistical Digital Signal Processing and Modeling*, Wiley, 1996.
4. S. L. Marple, *Digital Spectral Analysis with Applications*, Prentice Hall, 1987.
5. P. F. Fougere, E. J. Zawalick, H. R. Radoski, "Spontaneous line splitting in maximum entropy power spectrum analysis," *Phys. Earth and Planetary Interiors*, vol. 12, August 1976, pp. 201–207.
6. S. L. Marple, "A new autoregressive spectrum analysis algorithm," *IEEE Trans. ASSP*, vol. 28, August 1980, pp. 441–454.

This page intentionally left blank

Index

C

D

E

F

G

H

I

J–K

L

M

N

R

S

T

U

V

W

Y

Z

This page intentionally left blank