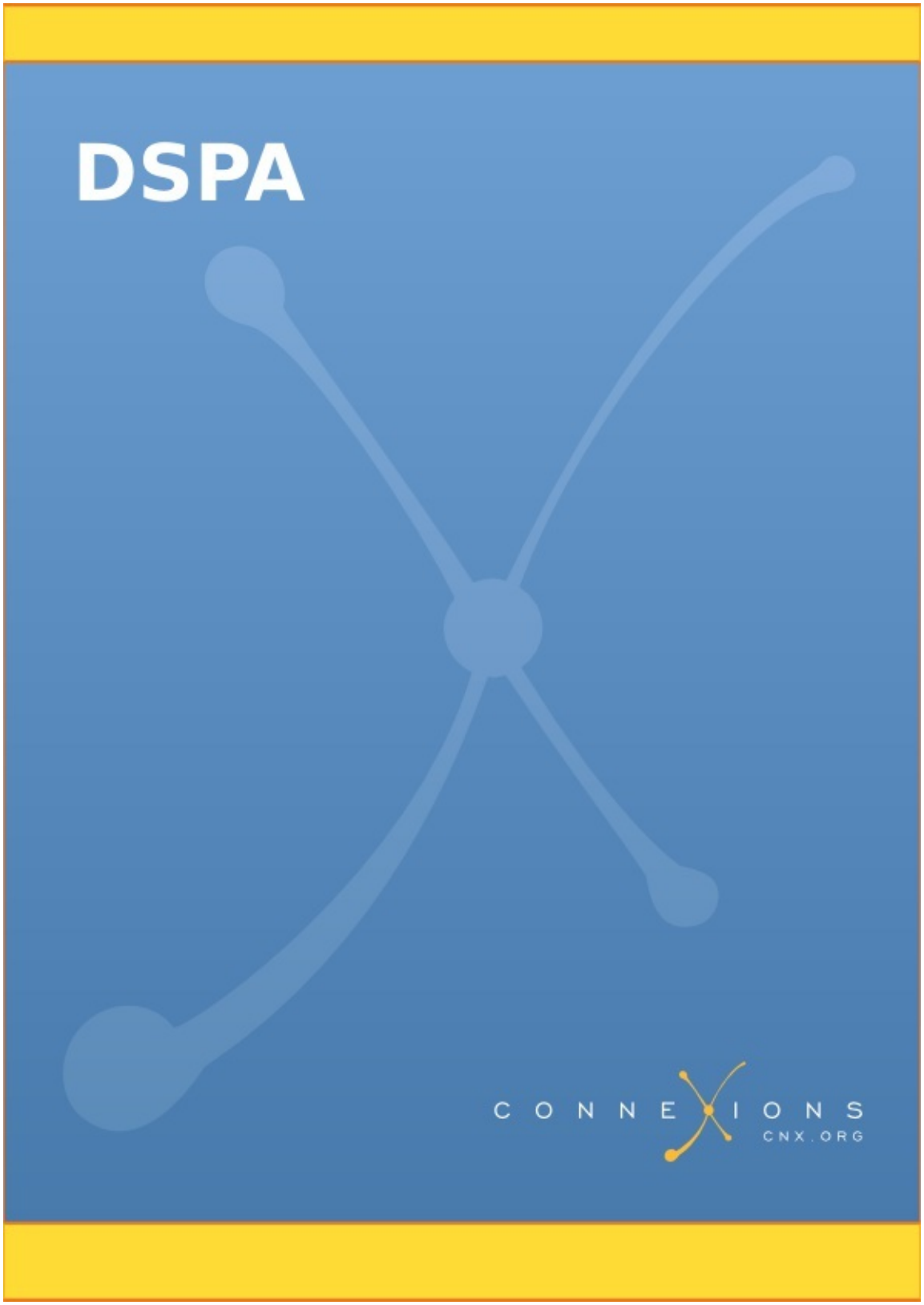
DSPA

DOUGLAS JONES, DON JOHNSON, ET AL



DSPA

Collection edited by: Janko Calic

Content authors: Douglas Jones, Don Johnson, Ricardo Radaelli-Sanchez, Richard Baraniuk, Stephen Kruzick, Catherine Elder, Melissa Selik, Robert Nowak, Anders Gjendemsjø, Michael Haag, Benjamin Fite, Ivan Selesnick, and Phil Schniter

Online: < http://cnx.org/content/col10599/1.5>

Collection structure revised: 2010/05/18

DSPA

Table of Contents

Preface for Digital Signal Processing: A User's Guide

Digital signal processing (DSP) has matured in the past few decades from an obscure research discipline to a large body of practical methods with very broad application. Both practicing

engineers and students specializing in signal processing need a clear exposition of the ideas and methods comprising the core signal processing "toolkit" so widely used today.

This text reflects my belief that the skilled practitioner must understand the key ideas underlying the algorithms to select, apply, debug, extend, and innovate most effectively; only with real insight can the engineer make novel use of these methods in the seemingly infinite range of new problems and applications. It also reflects my belief that the needs of the typical student and the practicing engineer have converged in recent years; as the discipline of signal processing has matured, these core topics have become less a subject of active research and more a set of tools applied in the course of other research. The modern student thus has less need for exhaustive coverage of the research literature and detailed derivations and proofs as preparation for their own research on these topics, but greater need for intuition and practical guidance in their most effective use. The majority of students eventually become practicing engineers themselves and benefit from the best preparation for their future careers.

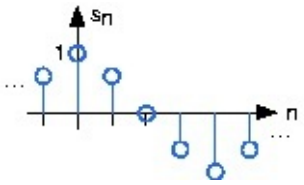
This text both explains the principles of classical signal processing methods and describes how they are used in engineering practice. It is thus much more than a recipe book; it describes the ideas behind the algorithms, gives analyses when they enhance that understanding, and includes derivations that the practitioner may need to extend when applying these methods to new situations. Analyses or derivations that are only of research interest or that do not increase intuitive understanding are left to the references. It is also much more than a theory book; it contains more description of common applications, discussion of actual implementation issues, comments on what really works in the real world, and practical "know-how" than found in the typical academic textbook. The choice of material emphasizes those methods that have found widespread practical use; techniques that have been the subject of intense research but which are rarely used in practice (for example, RLS adaptive filter algorithms) often receive only limited coverage.

The text assumes a familiarity with basic signal processing concepts such as ideal sampling theory, continuous and discrete Fourier transforms, convolution and filtering. It evolved from a set of notes for a second signal processing course, ECE 451: Digital Signal Processing II, in Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, aimed at second-semester seniors or first-semester graduate students in signal processing. Over the years, it has been enhanced substantially to include descriptions of common applications, sometimes hard-won knowledge about what actually works and what doesn't, useful tricks, important extensions known to experienced engineers but rarely discussed in academic texts, and other relevant "know-how" to aid the real-world user. This is necessarily an ongoing process, and I continue to expand and refine this component as my own practical knowledge and experience grows. The topics are the core signal processing methods that are used in the majority of signal processing applications; discrete Fourier analysis and FFTs, digital filter design, adaptive filtering, multirate signal processing, and efficient algorithm implementation and finite-precision issues. While many of these topics are covered at an introductory level in a first course, this text aspires to cover all of the methods, both basic and advanced, in these areas which see widespread use in practice. I have also attempted to make the individual modules and sections somewhat self-sufficient, so that those who seek specific information on a single topic can quickly find what they need. Hopefully these aspirations will eventually be achieved; in the meantime, I welcome your comments, corrections, and feedback so that I can continue to improve this text.

As of August 2006, the majority of modules are unedited transcriptions of handwritten notes and may contain typographical errors and insufficient descriptive text for documents unaccompanied by an oral lecture; I hope to have all of the modules in at least presentable shape by the end of the year.

Publication of this text in Connexions would have been impossible without the help of many people. A huge thanks to the various permanent and temporary staff at Connexions is due, in

particular to those who converted the text and equations from my original handwritten notes into CNXML and MathML. My former and current faculty colleagues at the University of Illinois who have taught the second DSP course over the years have had a substantial influence on the evolution of the content, as have the students who have inspired this work and given me feedback. I am very grateful to my teachers, mentors, colleagues, collaborators, and fellow engineers who have taught me the art and practice of signal processing; this work is dedicated to you.



Chapter 1. Background, Review, and Reference

1.1. Discrete-Time Signals and Systems*

Mathematically, analog signals are functions having as their independent variables continuous quantities, such as space and time. Discrete-time signals are functions defined on the integers; they are sequences. As with analog signals, we seek ways of decomposing discrete-time signals into simpler components. Because this approach leading to a better understanding of signal structure, we can exploit that structure to represent information (create ways of representing information with signals) and to extract information (retrieve the information thus represented). For symbolic-valued signals, the approach is different: We develop a common representation of all symbolic-valued signals so that we can embody the information they contain in a unified way. From an information representation perspective, the most important issue becomes, for both real-valued and symbolic-valued signals, efficiency: what is the most parsimonious and compact way to represent information so that it can be extracted later.

Real- and Complex-valued Signals

A discrete-time signal is represented symbolically as $s(n)$, where $n=\{ \ldots, -1, 0, 1, \ldots\}$.

Figure 1.1. Cosine

The discrete-time cosine signal is plotted as a stem plot. Can you find the formula for this signal?

We usually draw discrete-time signals as stem plots to emphasize the fact they are functions defined only on the integers. We can delay a discrete-time signal by an integer just as with analog ones. A signal delayed by *m* samples has the expression $s(n-m)$.

Complex Exponentials

The most important signal is, of course, the **complex exponential sequence**.

()

$s(n)=e^{i2\pi fn}$

[redacted]

[redacted]

[redacted]

[redacted]

Note that the frequency variable *f* is dimensionless and that adding an integer to the frequency of the discrete-time complex exponential has no effect on the signal's value.

()

This derivation follows because the complex exponential evaluated at an integer multiple of 2π equals one. Thus, we need only consider frequency to have a value in some unit-length interval.

Sinusoids

Discrete-time sinusoids have the obvious form $s(n)=A\cos(2\pi fn+\varphi)$. As opposed to analog complex exponentials and sinusoids that can have their frequencies be any real value, frequencies of their discrete-time counterparts yield unique waveforms **only** when *f* lies in the interval

.

This choice of frequency interval is arbitrary; we can also choose the frequency to lie in the

interval [0, 1) . How to choose a unit-length interval for a sinusoid's frequency will become evident later.

Unit Sample

The second-most important discrete-time signal is the **unit sample**, which is defined to be

()

Figure 1.2. Unit sample

The unit sample.

Examination of a discrete-time signal's plot, like that of the cosine signal shown in Figure 1.1, reveals that all signals consist of a sequence of delayed and scaled unit samples. Because the value of a sequence at each integer m is denoted by $s(m)$ and the unit sample delayed to occur at m is written $\delta(n- m)$, we can decompose **any** signal as a sum of unit samples delayed to the appropriate location and scaled by the signal value.

()

This kind of decomposition is unique to discrete-time signals, and will prove useful subsequently.

Unit Step

The **unit sample** in discrete-time is well-defined at the origin, as opposed to the situation with analog signals.

()

Symbolic Signals

An interesting aspect of discrete-time signals is that their values do not need to be real numbers. We do have real-valued discrete-time signals like the sinusoid, but we also have signals that denote the sequence of characters typed on the keyboard. Such characters certainly aren't real numbers, and as a collection of possible signal values, they have little mathematical structure other than that they are members of a set. More formally, each element of the **symbolic-valued**

signal $s(n)$ takes on one of the values $\{a_1, \ldots, a_K\}$ which comprise the **alphabet** A. This technical terminology does not mean we restrict symbols to being members of the English or Greek alphabet. They could represent keyboard characters, bytes (8-bit quantities), integers that convey daily temperature. Whether controlled by software or not, discrete-time systems are ultimately constructed from digital circuits, which consist **entirely** of analog circuit elements. Furthermore, the transmission and reception of discrete-time signals, like e-mail, is accomplished with analog signals and systems. Understanding how discrete-time and analog signals and systems intertwine is perhaps the main goal of this course.

Discrete-Time Systems

Discrete-time systems can act on discrete-time signals in ways similar to those found in analog signals and systems. Because of the role of software in discrete-time systems, many more different systems can be envisioned and "constructed" with programs than can be with analog signals. In fact, a special class of analog signals can be converted into discrete-time signals, processed with software, and converted back into an analog signal, all without the incursion of error. For such signals, systems can be easily produced in software, with equivalent analog realizations difficult, if not impossible, to design.

1.2. Systems in the Time-Domain*

A discrete-time signal $s(n)$ is **delayed** by n_0 samples when we write $s(n-n_0)$, with $n_0>0$. Choosing n_0 to be negative advances the signal along the integers. As opposed to **analog delays**, discrete-time delays can **only** be integer valued. In the frequency domain, delaying a signal corresponds to a linear phase shift of the signal's discrete-time Fourier transform:

$(s(n-n_0) \leftrightarrow e^{-(i 2\pi f n_0)} S(e^{i 2\pi f}))$.

Linear discrete-time systems have the superposition property.

(1.1)

Superposition

$S(a_1 x_1(n)+a_2 x_2(n))=a_1 S(x_1(n))+a_2 S(x_2(n))$ A discrete-time system is called **shift-**

invariant (analogous to **time-invariant analog systems)** if delaying the input delays the corresponding output.

(1.2)

Shift-Invariant

If $S(x(n))=y(n)$, Then $S(x(n-n_0))=y(n-n_0)$ We use the term shift-invariant to emphasize that delays can only have integer values in discrete-time, while in analog signals, delays can be arbitrarily valued.

We want to concentrate on systems that are both linear and shift-invariant. It will be these that allow us the full power of frequency-domain analysis and implementations. Because we have no physical constraints in "constructing" such systems, we need only a mathematical specification. In analog systems, the differential equation specifies the input-output relationship in the time-domain. The corresponding discrete-time specification is the **difference equation**.

(1.3)

The Difference Equation

$y(n)=a_1 y(n-1)+\ldots+a_p y(n-p)+b_0 x(n)+b_1 x(n-1)+\ldots+b_q x(n-q)$ Here, the output signal $y(n)$ is related to its **past** values $y(n-l)$, $l=\{1,\ldots,p\}$, and to the current and past values of the input signal $x(n)$. The system's characteristics are determined by the choices for the number of coefficients p and q and the coefficients' values $\{a_1,\ldots,a_p\}$ and $\{b_0,b_1,\ldots,b_q\}$.

There is an asymmetry in the coefficients: where is a_0 ? This coefficient would multiply the $y(n)$ term in **the difference equation**. We have essentially divided the equation by it, which does not change the input-output relationship. We have thus created the convention that a_0 is always one.

As opposed to differential equations, which only provide an **implicit** description of a system (we must somehow solve the differential equation), difference equations provide an **explicit** way of computing the output for any input. We simply express the difference equation by a program that calculates each output from the previous output values, and the current and previous inputs.

1.3. Discrete Time Convolution*

Introduction

Convolution, one of the most important concepts in electrical engineering, can be used to determine the output a system produces for a given input signal. It can be shown that a linear time invariant system is completely characterized by its impulse response. The sifting property of the discrete time impulse function tells us that the input signal to a system can be represented as a sum of scaled and shifted unit impulses. Thus, by linearity, it would seem reasonable to compute of the output signal as the sum of scaled and shifted unit impulse responses. That is exactly what the operation of convolution accomplishes. Hence, convolution can be used to determine a linear time invariant system's output from knowledge of the input and the impulse response.

Convolution and Circular Convolution

Convolution

Operation Definition

Discrete time convolution is an operation on two discrete time signals defined by the integral

(1.4)

for all signals f, g defined on Z. It is important to note that the operation of convolution is commutative, meaning that

(1.5)

$$f * g = g * f$$

for all signals f, g defined on Z. Thus, the convolution operation could have been just as easily stated using the equivalent definition

(1.6)

for all signals f, g defined on Z. Convolution has several other important properties not listed here but explained and derived in a later module.

Definition Motivation

The above operation definition has been chosen to be particularly useful in the study of linear time invariant systems. In order to see this, consider a linear time invariant system H with unit impulse response h. Given a system input signal x we would like to compute the system output signal $H(x)$. First, we note that the input can be expressed as the convolution

(1.7)

by the sifting property of the unit impulse function. By linearity

(1.8)

Since $H\delta(n-k)$ is the shifted unit impulse response $h(n-k)$, this gives the result

(1.9)

Hence, convolution has been defined such that the output of a linear time invariant system is given by the convolution of the system input with the system unit impulse response.

Graphical Intuition

It is often helpful to be able to visualize the computation of a convolution in terms of graphical processes. Consider the convolution of two functions f, g given by

(1.10)

The first step in graphically understanding the operation of convolution is to plot each of the functions. Next, one of the functions must be selected, and its plot reflected across the $k=0$ axis. For each real t, that same function must be shifted left by t. The product of the two resulting plots is then constructed. Finally, the area under the resulting curve is computed.

Example 1.1.

Recall that the impulse response for a discrete time echoing feedback system with gain a is

(1.11)

$$h(n) = a^n u(n),$$

and consider the response to an input signal that is another exponential

(1.12)

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

We know that the output for this input is given by the convolution of the impulse response with the input signal

(1.13)

$$y(n) = x(n) * h(n).$$

We would like to compute this operation by beginning in a way that minimizes the algebraic complexity of the expression. However, in this case, each possible coice is equally simple. Thus, we would like to compute

(1.14)

The step functions can be used to further simplify this sum. Therefore,

(1.15)

$y(n) = 0$

for $n<0$ and

(1.16)

for $n \geq 0$. Hence, provided $ab \neq 1$, we have that

(1.17)

Circular Convolution

Discrete time circular convolution is an operation on two finite length or periodic discrete time signals defined by the integral

(1.18)

for all signals f, g defined on $Z[0, N-1]$ where

are periodic extensions of f and g. It is important

to note that the operation of circular convolution is commutative, meaning that

(1.19)

$f * g = g * f$

for all signals f, g defined on $Z[0, N-1]$. Thus, the circular convolution operation could have been just as easily stated using the equivalent definition

(1.20)

for all signals f, g defined on $Z[0, N-1]$ where

are periodic extensions of f and g. Circular

convolution has several other important properties not listed here but explained and derived in a later module.

[redacted]

[redacted]

[redacted]

Alternatively, discrete time circular convolution can be expressed as the sum of two summations given by

(1.21)

for all signals f, g defined on $Z[0, N–1]$.

Meaningful examples of computing discrete time circular convolutions in the time domain would involve complicated algebraic manipulations dealing with the wrap around behavior, which would ultimately be more confusing than helpful. Thus, none will be provided in this section. Of course, example computations in the time domain are easy to program and demonstrate. However, disrete time circular convolutions are more easily computed using frequency domain tools as will be shown in the discrete time Fourier series section.

Definition Motivation

The above operation definition has been chosen to be particularly useful in the study of linear time invariant systems. In order to see this, consider a linear time invariant system H with unit impulse response h. Given a finite or periodic system input signal x we would like to compute the system output signal $H(x)$. First, we note that the input can be expressed as the circular convolution

(1.22)

by the sifting property of the unit impulse function. By linearity,

(1.23)

Since $H\delta(n– k)$ is the shifted unit impulse response $h(n– k)$, this gives the result

(1.24)

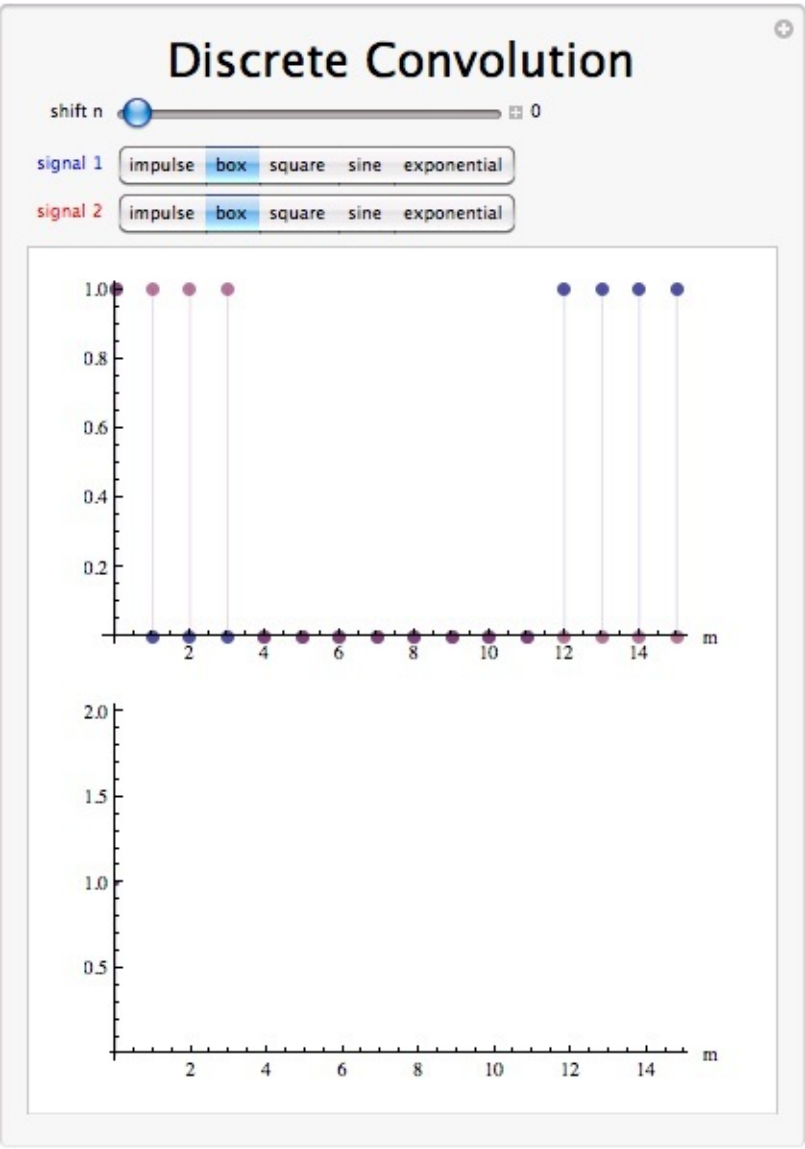
Hence, circular convolution has been defined such that the output of a linear time invariant system

is given by the convolution of the system input with the system unit impulse response.

Graphical Intuition

It is often helpful to be able to visualize the computation of a circular convolution in terms of graphical processes. Consider the circular convolution of two finite length functions f, g given by

(1.25)



The first step in graphically understanding the operation of convolution is to plot each of the periodic extensions of the functions. Next, one of the functions must be selected, and its plot reflected across the k=0 axis. For each $k \in \mathbb{Z}[0, N-1]$, that same function must be shifted left by k. The product of the two resulting plots is then constructed. Finally, the area under the resulting curve on $\mathbb{Z}[0, N-1]$ is computed.

Interactive Element

Figure 1.3.

Interact (when online) with the Mathematica CDF demonstrating Discrete Linear Convolution. To download, right click and save file as .cdf

Convolution Summary

Convolution, one of the most important concepts in electrical engineering, can be used to determine the output signal of a linear time invariant system for a given input signal with knowledge of the system's unit impulse response. The operation of discrete time convolution is defined such that it performs this function for infinite length discrete time signals and systems.

[redacted]

[redacted]

The operation of discrete time circular convolution is defined such that it performs this function for finite length and periodic discrete time signals. In each case, the output of the system is the convolution or circular convolution of the input signal with the unit impulse response.

1.4. Introduction to Fourier Analysis*

Fourier's Daring Leap

Fourier postulated around 1807 that any periodic signal (equivalently finite length signal) can be built up as an infinite linear combination of harmonic sinusoidal waves.

i.e. Given the collection

(1.26)

any

(1.27)

$f(t) \in L2[0, T)$

can be approximated arbitrarily closely by

(1.28)

Now, The issue of exact convergence did bring **Fourier** much criticism from the French Academy of

Science (Laplace, Lagrange, Monge and LaCroix comprised the review committee) for several years after its presentation on 1807. It was not resolved for also a century, and its resolution is interesting and important to understand from a practical viewpoint. See more in the section on **Gibbs Phenomena**.

Fourier analysis is fundamental to understanding the behavior of signals and systems. This is a result of the fact that sinusoids are **Eigenfunctions** of **linear, time-invariant (LTI)** systems. This is to say that if we pass any particular sinusoid through a LTI system, we get a scaled version of that same sinusoid on the output. Then, since Fourier analysis allows us to redefine the signals in terms of sinusoids, all we need to do is determine how any given system effects all possible sinusoids (its **transfer function**) and we have a complete understanding of the system. Furthermore, since we are able to define the passage of sinusoids through a system as multiplication of that sinusoid by the transfer function at the same frequency, we can convert the passage of any signal through a system from **convolution** (in time) to multiplication (in frequency). These ideas are what give Fourier analysis its power.

Now, after hopefully having sold you on the value of this method of analysis, we must examine exactly what we mean by Fourier analysis. The four Fourier transforms that comprise this analysis are the **Fourier Series**, **Continuous-Time Fourier Transform**, **Discrete-Time Fourier Transform** and **Discrete Fourier Transform**. For this document, we will view the **Laplace Transform** and **Z-Transform** as simply extensions of the CTFT and DTFT respectively. All of these transforms act essentially the same way, by converting a signal in time to an equivalent signal in frequency (sinusoids). However, depending on the nature of a specific signal *i.e.* whether it is finite- or infinite-length and whether it is discrete- or continuous-time) there is an appropriate transform to convert the signal into the frequency domain. Below is a table of the four Fourier transforms and when each is appropriate. It also includes the relevant convolution for the specified space.

Table 1.1. Table of Fourier Representations

Time

Frequency

Transform

Convolution

Domain

Domain

Continuous-Time

Continuous-Time Fourier Series

L 2([0, T))

l 2(ℤ)

Circular

Continuous-Time Fourier

L 2(ℝ)

L 2(ℝ)

Continuous-Time Linear

Transform

Discrete-Time Fourier Transform

l 2(ℤ)

L 2([0, 2 π))

Discrete-Time Linear

Discrete Fourier Transform

l 2([0, N−1])

l 2([0, N−1])

Discrete-Time Circular

1.5. Continuous Time Fourier Transform (CTFT)*

Introduction

In this module, we will derive an expansion for any arbitrary continuous-time function, and in doing so, derive the **Continuous Time Fourier Transform** (CTFT).

Since **complex exponentials** are **eigenfunctions of linear time-invariant (LTI) systems**, calculating the output of an LTI system $\mathcal{H}$ given e^{st} as an input amounts to simple multiplication, where $H(s) \in \mathbb{C}$ is the eigenvalue corresponding to s. As shown in the figure, a simple exponential input would yield the output

()

$y(t) = H(s)\, e^{st}$

Using this and the fact that $\mathcal{H}$ is linear, calculating $y(t)$ for combinations of complex exponentials is also straightforward.

$c_1 e^{s_1 t} + c_2 e^{s_2 t} \rightarrow c_1 H(s_1) e^{s_1 t} + c_2 H(s_2) e^{s_2 t}$ The action of H on an input such as those in the two equations above is easy to explain. $\mathcal{H}$ **independently scales** each exponential component e^{snt} by a different complex number $H(sn) \in \mathbb{C}$.

As such, if we can write a function $f(t)$ as a combination of complex exponentials it allows us to easily calculate the output of a system.

Now, we will look to use the power of complex exponentials to see how we may represent arbitrary signals in terms of a set of simpler functions by superposition of a number of complex exponentials. Below we will present the **Continuous-Time Fourier Transform** (CTFT), commonly referred to as just the Fourier Transform (FT). Because the CTFT deals with nonperiodic signals, we must find a way to include all real frequencies in the general equations. For the CTFT we simply utilize integration over real numbers rather than summation over integers

in order to express the aperiodic signals.

Fourier Transform Synthesis

Joseph Fourier demonstrated that an arbitrary *s*(*t*) can be written as a linear combination of harmonic complex sinusoids

()

where

is the fundamental frequency. For almost all *s*(*t*) of practical interest, there exists

cn to make Equation true. If *s*(*t*) is finite energy (*s*(*t*)∈ *L* 2[0, *T*]), then the equality in Equation

holds in the sense of energy convergence; if *s*(*t*) is continuous, then Equation holds pointwise.

Also, if *s*(*t*) meets some mild conditions (the Dirichlet conditions), then Equation holds pointwise everywhere except at points of discontinuity.

The *cn* - called the Fourier coefficients - tell us "how much" of the sinusoid ℮*jω* 0 *nt* is in *s*(*t*) . The

[redacted]

formula shows *s*(*t*) as a sum of complex exponentials, each of which is easily processed by an LTI

system (since it is an eigenfunction of **every** LTI system). Mathematically, it tells us that the set

of complex exponentials { ℮*jω* 0 *nt* , *n*∈ℤ } form a basis for the space of T-periodic continuous

time functions.

Equations

Now, in order to take this useful tool and apply it to arbitrary non-periodic signals, we will have to delve deeper into the use of the superposition principle. Let $sT(t)$ be a periodic signal having period T. We want to consider what happens to this signal's spectrum as the period goes to infinity. We denote the spectrum for any assumed value of the period by $cn(T)$. We calculate the spectrum according to the Fourier formula for a periodic signal, known as the Fourier Series (for more on this derivation, see the section on **Fourier Series**.)

()

where

and where we have used a symmetric placement of the integration interval about the origin for subsequent derivational convenience. We vary the frequency index n proportionally as we increase the period. Define

()

making the corresponding Fourier Series

()

As the period increases, the spectral lines become closer together, becoming a continuum. Therefore,

()

with

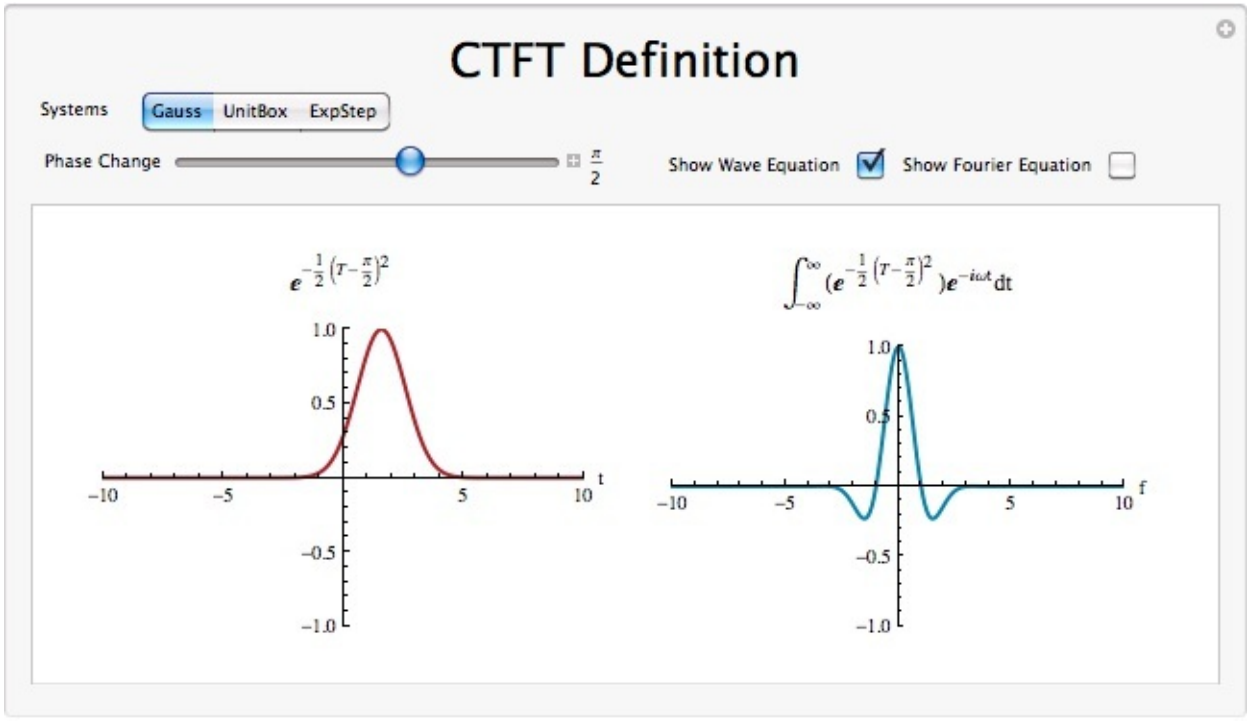
()

(1.29)

Continuous-Time Fourier Transform

(1.30)

Inverse CTFT



Warning

It is not uncommon to see the above formula written slightly different. One of the most

common differences is the way that the exponential is written. The above equations use the

radial frequency variable Ω in the exponential, where $\Omega=2\pi f$, but it is also common to include

the more explicit expression, $-j2\pi ft$, in the exponential. **Click here** for an overview of the notation used in Connexion's DSP modules.

Example 1.2.

We know from Euler's formula that

CTFT Definition Demonstration

Figure 1.4.

Interact (when online) with a Mathematica CDF demonstrating Continuous Time Fourier Transform. To Download, right-click and

save as .cdf.

Example Problems

Exercise 1.

Find the Fourier Transform (CTFT) of the function

()

In order to calculate the Fourier transform, all we need to use is Equation 1.29, **complex exponentials**, and basic calculus.

()

()

Exercise 2.

Find the inverse Fourier transform of the ideal lowpass filter defined by

()

Here we will use Equation 1.30 to find the inverse FT given that $t \neq 0$.

()

()

Fourier Transform Summary

Because complex exponentials are eigenfunctions of LTI systems, it is often useful to represent signals using a set of complex exponentials as a basis. The continuous time Fourier series synthesis formula expresses a continuous time, periodic function as the sum of continuous time, discrete frequency complex exponentials.

()

The continuous time Fourier series analysis formula gives the coefficients of the Fourier series expansion.

()

In both of these equations

is the fundamental frequency.

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

1.6. Discrete-Time Fourier Transform (DTFT)*

The Fourier transform of the discrete-time signal *s*(*n*) is defined to be

()

Frequency here has no units. As should be expected, this definition is linear, with the transform of a sum of signals equaling the sum of their transforms. Real-valued signals have conjugate-symmetric spectra:

.

Exercise 3.

A special property of the discrete-time Fourier transform is that it is periodic with period one: $S(e^{i2\pi(f+1)}) = S(e^{i2\pi f})$. Derive this property from the definition of the DTFT.

(1.31)

Because of this periodicity, we need only plot the spectrum over one period to understand completely the spectrum's structure; typically, we plot the spectrum over the frequency range . When the signal is real-valued, we can further simplify our plotting chores by showing the spectrum only over ; the spectrum at negative frequencies can be derived from positive-frequency spectral values.

When we obtain the discrete-time signal via sampling an analog signal, the **Nyquist frequency** corresponds to the discrete-time frequency . To show this, note that a sinusoid having a frequency equal to the Nyquist frequency

has a sampled waveform that equals

The exponential in the DTFT at frequency equals

,

meaning that discrete-time frequency equals analog frequency multiplied by the sampling interval

()

$f_D = f_A T_s$

f_D and f_A represent discrete-time and analog frequency variables, respectively. The **aliasing figure** provides another way of deriving this result. As the duration of each pulse in the periodic

sampling signal $pT(t)$ narrows, the amplitudes of the signal's spectral repetitions, which are

s

governed by the **Fourier series coefficients** of $pT(t)$, become increasingly equal. Examination of s

the **periodic pulse signal** reveals that as Δ decreases, the value of c_0, the largest Fourier coefficient, decreases to zero:

. Thus, to maintain a mathematically viable Sampling

Theorem, the amplitude A must increase as , becoming infinitely large as the pulse duration

decreases. Practical systems use a small value of Δ, say $0.1 \cdot Ts$ and use amplifiers to rescale the

signal. Thus, the sampled signal's spectrum becomes periodic with period . Thus, the Nyquist

frequency

corresponds to the frequency .

Example 1.3.

Let's compute the discrete-time Fourier transform of the exponentially decaying sequence

$s(n) = a^n u(n)$, where $u(n)$ is the unit-step sequence. Simply plugging the signal's expression into the Fourier transform formula,

(1.32)

This sum is a special case of the **geometric series**.

()

Thus, as long as $|a|<1$, we have our Fourier transform.

(1.33)

Using Euler's relation, we can express the magnitude and phase of this spectrum.

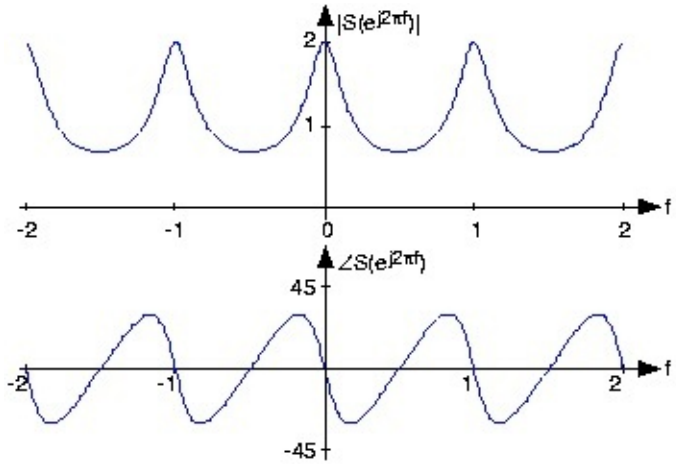
(1.34)

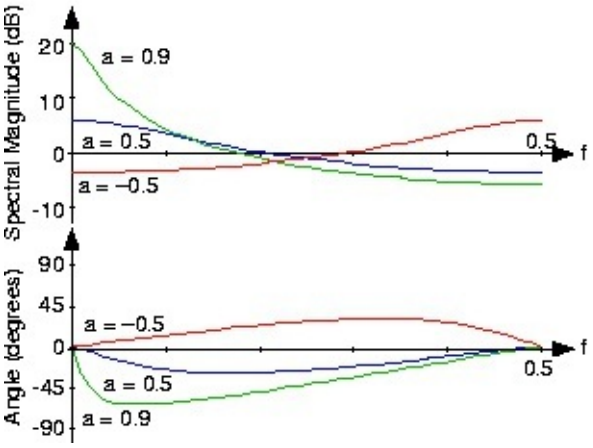
(1.35)

No matter what value of *a* we choose, the above formulae clearly demonstrate the periodic nature of the spectra of discrete-time signals. Figure 1.5 shows indeed that the spectrum is a periodic function. We need only consider the spectrum between

and to unambiguously

define it. When $a>0$, we have a lowpass spectrum—the spectrum diminishes as frequency increases from 0 to —with increasing *a* leading to a greater low frequency content; for $a<0$, we have a highpass spectrum (Figure 1.6).





[redacted]

[redacted]

Figure 1.5. Spectrum of exponential signal

The spectrum of the exponential signal (a=0.5) is shown over the frequency range [-2, 2], clearly demonstrating the periodicity of all discrete-time spectra. The angle has units of degrees.

Figure 1.6. Spectra of exponential signals

The spectra of several exponential signals are shown. What is the apparent relationship between the spectra for a=0.5 and a=–0.5?

Example 1.4.

Analogous to the analog pulse signal, let's find the spectrum of the length- N pulse sequence.

(1.36)

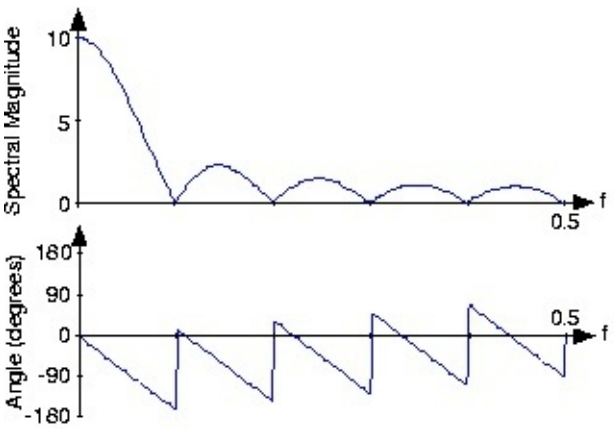
The Fourier transform of this sequence has the form of a truncated geometric series.

(1.37)

For the so-called finite geometric series, we know that

()

for **all** values of α.



[redacted]

Exercise 4.

Derive this formula for the finite geometric series sum. The "trick" is to consider the difference

between the series' sum and the sum of the series multiplied by α.

which, after manipulation, yields the geometric sum formula.

Applying this result yields (Figure 1.7.)

()

The ratio of sine functions has the generic form of

, which is known as the **discrete-time**

sinc function dsinc(x) . Thus, our transform can be concisely expressed as

$S(e^{i2\pi f}) = e^{-(i\pi f(N-1))}\text{dsinc}(\pi f)$. The discrete-time pulse's spectrum contains many ripples, the number of which increase with N, the pulse's duration.

Figure 1.7. Spectrum of length-ten pulse

The spectrum of a length-ten pulse is shown. Can you explain the rather complicated appearance of the phase?

The inverse discrete-time Fourier transform is easily derived from the following relationship:

()

Therefore, we find that

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

()

The Fourier transform pairs in discrete-time are

()

The properties of the discrete-time Fourier transform mirror those of the analog Fourier

transform. The **DTFT properties table** shows similarities and differences. One important common property is Parseval's Theorem.

()

To show this important property, we simply substitute the Fourier transform expression into the

frequency-domain expression for power.

()

Using the **orthogonality relation**, the integral equals $\delta(m- n)$, where $\delta(n)$ is the **unit sample**.

Thus, the double sum collapses into a single sum because nonzero values occur only when $n= m$,

giving Parseval's Theorem as a result. We term

the energy in the discrete-time signal

$s(n)$ in spite of the fact that discrete-time signals don't consume (or produce for that matter)

energy. This terminology is a carry-over from the analog world.

Exercise 5.

Suppose we obtained our discrete-time signal from values of the product $s(t) p Ts (t)$, where the duration of the component pulses in $p Ts (t)$ is Δ. How is the discrete-time signal energy related to the total energy contained in $s(t)$? Assume the signal is bandlimited and that the sampling rate

was chosen appropriate to the Sampling Theorem's conditions.

If the sampling frequency exceeds the Nyquist frequency, the spectrum of the samples equals the analog spectrum, but over the normalized analog frequency fT . Thus, the energy in the sampled signal equals the original signal's energy multiplied by T.

1.7. DFT as a Matrix Operation*

Matrix Review

Recall:

Vectors in $\mathbb{R}^N$:

Vectors in $\mathbb{C}^N$:

Transposition:

1. transpose:

2. conjugate:

Inner product:

1. real:

2. complex:

Matrix Multiplication:

Matrix Transposition:

Matrix transposition involved simply

swapping the rows with columns.

The above equation is Hermitian transpose.

[**A**T] $\boldsymbol{kn}$=**A** $\boldsymbol{nk}$

Representing DFT as Matrix Operation

Now let's represent the **DFT** in vector-matrix notation.

Here **x** is the

vector of time samples and **X** is the vector of DFT coefficients. How are **x** and **X** related:

where

so **X**= ***Wx*** where **X** is the DFT vector, *W* is the matrix

and **x** the time domain vector.

IDFT:

where

is the matrix Hermitian transpose. So,

where **x** is the time vector,

is the inverse DFT matrix, and **X** is the DFT vector.

1.8. Sampling theory

Introduction*

Contents of Sampling chapter

Introduction(Current module)

Proof

Illustrations

Matlab Example

Hold operation

System view

Aliasing applet

Exercises

Table of formulas

Why sample?

This section introduces sampling. Sampling is the necessary fundament for all digital signal processing and communication. Sampling can be defined as the process of measuring an analog signal at distinct points.

Digital representation of analog signals offers advantages in terms of

robustness towards noise, meaning we can send more bits/s

use of flexible processing equipment, in particular the computer

more reliable processing equipment

easier to adapt complex algorithms

Claude E. Shannon

Figure 1.8.

Claude Elwood Shannon (1916-2001)

Claude Shannon has been called the father of information theory, mainly due to his landmark papers on the **"Mathematical theory of communication"** . **Harry Nyquist** was the first to state the sampling theorem in 1928, but it was not proven until Shannon proved it 21 years later in the

paper **"Communications in the presence of noise" .**

Notation

In this chapter we will be using the following notation

Original analog signal $x(t)$

Sampling frequency Fs

Sampling interval Ts (Note that:

)

Sampled signal $xs(n)$. (Note that $xs(n)= x(nTs)$)

Real angular frequency Ω

Digital angular frequency ω. (Note that: $\omega= \Omega Ts$)

The Sampling Theorem

The Sampling theorem

When sampling an analog signal the sampling frequency must be greater than twice the highest frequency component of the analog signal to be able to reconstruct the original signal from the sampled version.

Finished? Have at look at: **Proof**; **Illustrations**; **Matlab Example**; **Aliasing applet**; **Hold operation**; **System view**; **Exercises**

Proof*

Sampling theorem

In order to recover the signal $x(t)$ from it's samples exactly, it is necessary to sample $x(t)$ at a rate greater than twice it's highest frequency component.

Introduction

As mentioned **earlier**, sampling is the necessary fundament when we want to apply digital signal processing on analog signals.

Here we present the proof of the sampling theorem. The proof is divided in two. First we find an expression for the spectrum of the signal resulting from sampling the original signal $x(t)$. Next we show that the signal $x(t)$ can be recovered from the samples. Often it is easier using the frequency

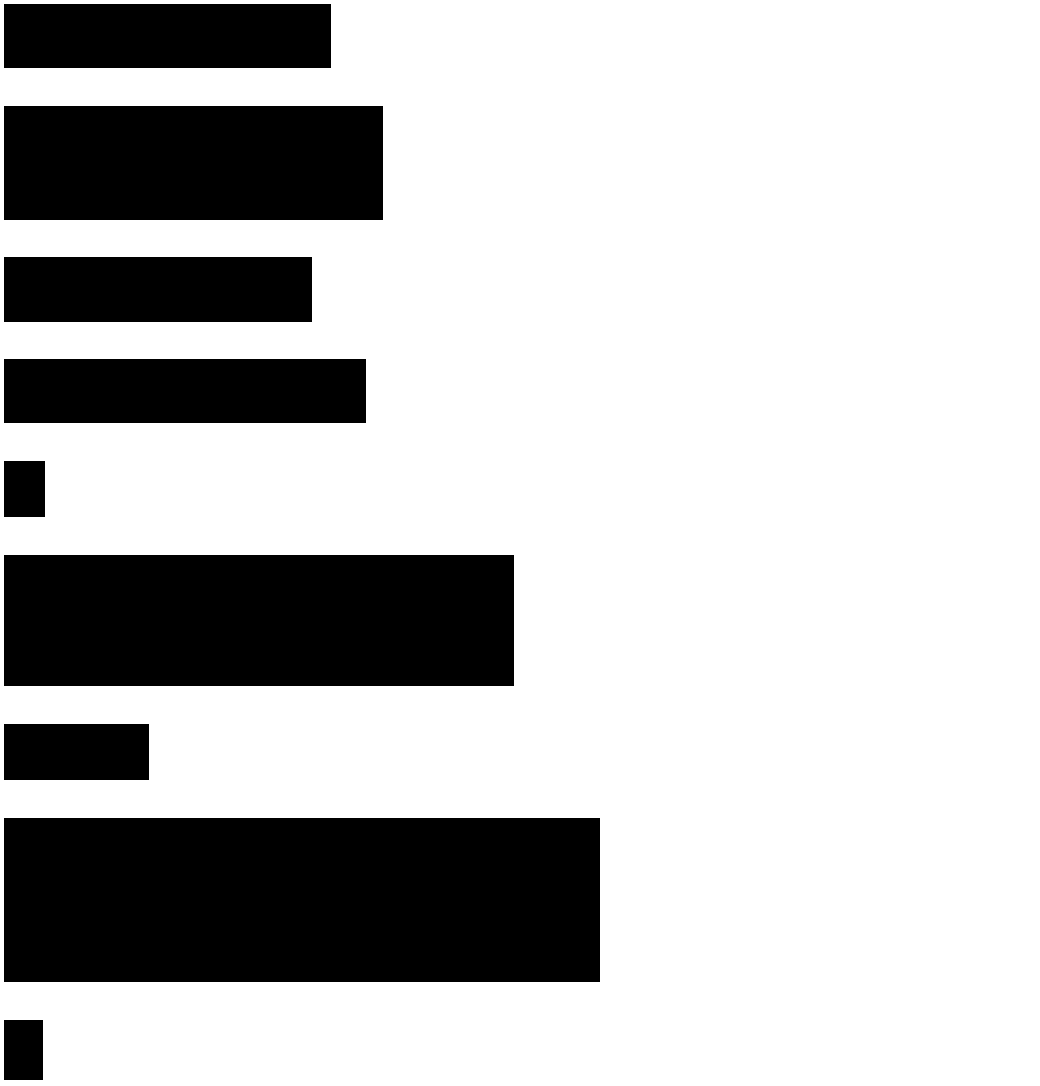
domain when carrying out a proof, and this is also the case here.

Key points in the proof

We find an **equation** for the spectrum of the sampled signal

We find a **simple method to reconstruct** the original signal

The sampled signal has a periodic spectrum...



The sampled signal has a periodic spectrum...

...and the period is $2\pi Fs$

Proof part 1 - Spectral considerations

By sampling $x(t)$ every Ts second we obtain $xs(n)$. The inverse fourier transform of this **time**

discrete signal is

()

For convenience we express the equation in terms of the real angular frequency Ω using $\omega = \Omega Ts$.

We then obtain

()

The inverse fourier transform of a continuous signal is

()

From this equation we find an expression for x (nTs)

()

To account for the difference in region of integration we split the integration in Equation into

subintervals of length

and then take the sum over the resulting integrals to obtain the complete

area.

()

Then we change the integration variable, setting

()

We obtain the final form by observing that $e^{i} 2\pi kn = 1$, reinserting $\eta = \Omega$ and multiplying by

[redacted]

()

To make $x_s(n) = x(nT_s)$ for all values of n, the integrands in Equation and Equation have to agreee, that is

()

This is a central result. We see that the digital spectrum consists of a sum of shifted versions of

the original, analog spectrum. Observe the periodicity!

We can also express this relation in terms of the digital angular frequency $\omega = \Omega T_s$

()

This concludes the first part of the proof. Now we want to find a reconstruction formula, so that

we can recover $x(t)$ from $x_s(n)$.

Proof part II - Signal reconstruction

For a **bandlimited** signal the inverse fourier transform is

()

In the interval we are integrating we have:

. Substituting this relation into Equation

we get

()

Using the **DTFT** relation for $X_s(e^{j\Omega T_s})$ we have

()

Interchanging integration and summation (under the assumption of convergence) leads to

()

Finally we perform the integration and arrive at the important reconstruction formula

()

(Thanks to R.Loos for pointing out an error in the proof.)

Summary

Spectrum sampled signal

Reconstruction formula

Go to **Introduction**; **Illustrations**; **Matlab Example**; **Hold operation**; **Aliasing applet**; **System view**; **Exercises** ?

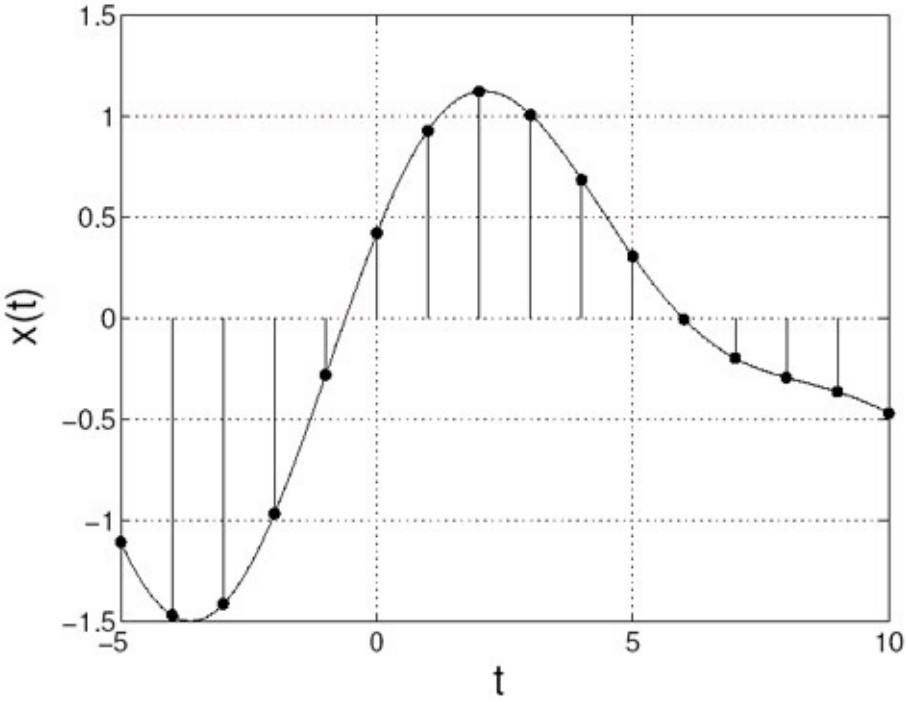
Illustrations*

In this module we illustrate the processes involved in sampling and reconstruction. To see how all these processes work together as a whole, take a look at the **system view**. In **Sampling and reconstruction with Matlab** we provide a Matlab script for download. The matlab script shows the process of sampling and reconstruction **live**.

Basic examples

Example 1.5.

To sample an analog signal with 3000 Hz as the highest frequency component requires sampling at 6000 Hz or above.



Example 1.6.

The sampling theorem can also be applied in two dimensions, i.e. for image analysis. A 2D sampling theorem has a simple physical interpretation in image analysis: Choose the sampling interval such that it is less than or equal to half of the smallest interesting detail in the image.

The process of sampling

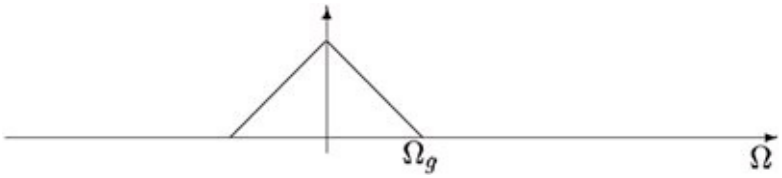
We start off with an analog signal. This can for example be the sound coming from your stereo at home or your friend talking.

The signal is then sampled uniformly. Uniform sampling implies that we sample every *Ts* seconds. In Figure 1.9 we see an analog signal. The analog signal has been sampled at times $t = nTs$.

Figure 1.9.

Analog signal, samples are marked with dots.

In signal processing it is often more convenient and easier to work in the frequency domain. So let's look at at the signal in frequency domain, Figure 1.10. For illustration purposes we take the frequency content of the signal as a triangle. (If you Fourier transform the signal in Figure 1.9 you will not get such a nice triangle.)



[redacted]

[redacted]

[redacted]

[redacted]

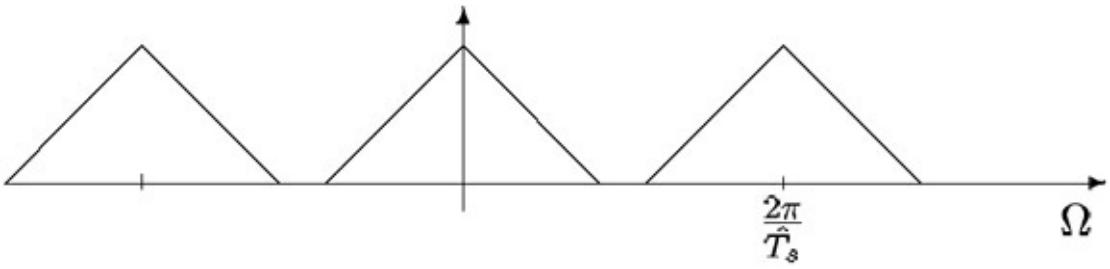


Figure 1.10.

The spectrum $X(i\Omega)$.

Notice that the signal in Figure 1.10 is bandlimited. We can see that the signal is bandlimited

because $X(i\Omega)$ is zero outside the interval $[-\Omega g, \Omega g]$. Equivalentely we can state that the signal has no angular frequencies above Ωg, corresponding to no frequencies above

.

Now let's take a look at the sampled signal in the frequency domain. While **proving** the sampling theorem we found the the spectrum of the sampled signal consists of a sum of shifted versions of

the analog spectrum. Mathematically this is described by the following equation:

()

Sampling fast enough

In Figure 1.11 we show the result of sampling $x(t)$ according to **the sampling theorem**. This means that when sampling the signal in Figure 1.9/Figure 1.10 we use $Fs \geq 2\,Fg$. Observe in

Figure 1.11 that we have the same spectrum as in Figure 1.10 for $\Omega \in [-\Omega g, \Omega g]$, except for the scaling factor . This is a consequence of the sampling frequency. As mentioned in the **proof** the spectrum of the sampled signal is periodic with period

.

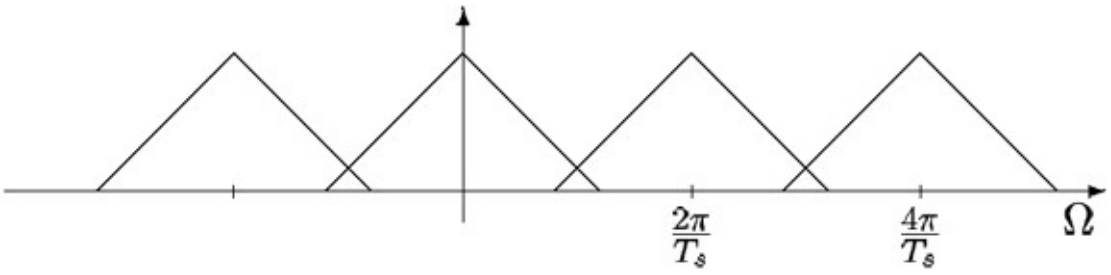
Figure 1.11.

The spectrum *Xs*. Sampling frequency is OK.

So now we are, according to **the sample theorem**, able to reconstruct the original signal **exactly**.

How we can do this will be explored further down under **reconstruction**. But first we will take a look at what happens when we sample too slowly.

Sampling too slowly



If we sample $x(t)$ too slowly, that is $Fs<2\,Fg$, we will get overlap between the repeated spectra, see

Figure 1.12. According to Equation the resulting spectra is the sum of these. This overlap gives rise to the concept of aliasing.

Aliasing

If the sampling frequency is less than twice the highest frequency component, then

frequencies in the original signal that are above half the sampling rate will be "aliased" and

will appear in the resulting signal as lower frequencies.

The consequence of aliasing is that we cannot recover the original signal, so aliasing has to be

avoided. Sampling too slowly will produce a sequence $xs(n)$ that could have orginated from a

number of signals. So there is **no** chance of recovering the original signal. To learn more about

aliasing, take a look at this **module**. (Includes an applet for demonstration!) Figure 1.12.

The spectrum *Xs*. Sampling frequency is too low.

To avoid aliasing we have to sample fast enough. But if we can't sample fast enough (possibly due

to costs) we can include an Anti-Aliasing filter. This will not able us to get an exact reconstruction

but can still be a good solution.

Anti-Aliasing filter

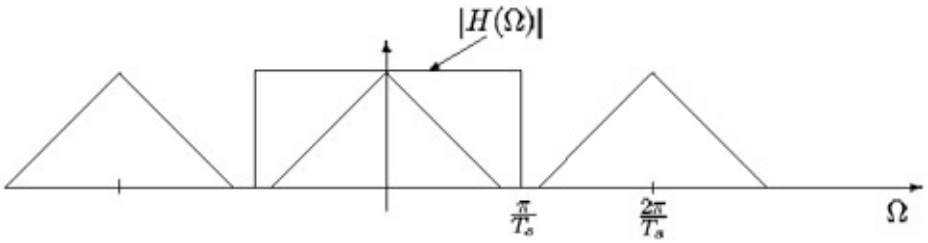
Typically a low-pass filter that is applied before sampling to ensure that no components with

frequencies greater than half the sample frequency remain.

Example 1.7.

The stagecoach effect

In older western movies you can observe aliasing on a stagecoach when it starts to roll. At first the spokes appear to turn forward, but as the stagecoach increase its speed the spokes



appear to turn backward. This comes from the fact that the sampling rate, here the number of frames per second, is too low. We can view each frame as a sample of an image that is changing continuously in time. (**Applet illustrating the stagecoach effect**) **Reconstruction**

Given the signal in Figure 1.11 we want to recover the original signal, but the question is how? When there is no overlapping in the spectrum, the spectral component given by k=0 (see Equation),is equal to the spectrum of the analog signal. This offers an oppurtunity to use a simple reconstruction process. Remember what you have learned about filtering. What we want is to change signal in Figure 1.11 into that of Figure 1.10. To achieve this we have to remove all the extra components generated in the sampling process. To remove the extra components we apply an ideal analog low-pass filter as shown in Figure 1.13 As we see the ideal filter is rectangular in the frequency domain. A rectangle in the frequency domain corresponds to a **sinc** function in time domain (and vice versa).

Figure 1.13.

$H(i\Omega)$ The ideal reconstruction filter.

Then we have reconstructed the original spectrum, and as we know **if two signals are identical in the frequency domain, they are also identical in the time domain**. End of reconstruction.

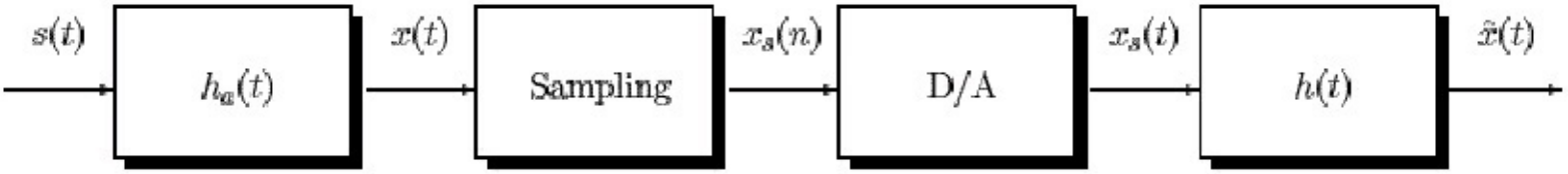
Conclusions

The Shannon sampling theorem requires that the input signal prior to sampling is band-limited to

at most half the sampling frequency. Under this condition the samples give an exact signal representation. It is truly remarkable that such a broad and useful class signals can be represented that easily!

We also looked into the problem of reconstructing the signals form its samples. Again the simplicity of the **principle** is striking: linear filtering by an ideal low-pass filter will do the job. However, the ideal filter is impossible to create, but that is another story...

Go to? **Introduction; Proof; Illustrations**; **Matlab Example; Aliasing applet**; **Hold operation;**



System view; **Exercises**

Systems view of sampling and reconstruction*

Ideal reconstruction system

Figure 1.14 shows the ideal reconstruction system based on the results of the Sampling theorem **proof**.

Figure 1.14 consists of a sampling device which produces a time-discrete sequence *xs*(*n*). The reconstruction filter, *h*(*t*), is an ideal analog **sinc** filter, with

. We can't apply the time-

discrete sequence *xs*(*n*) directly to the analog filter *h*(*t*). To solve this problem we turn the sequence into an analog signal using **delta functions**. Thus we write

.

Figure 1.14.

Ideal reconstruction system

But when will the system produce an output

? According to the **sampling theorem** we have

when the sampling frequency, *Fs*, is at least twice the highest frequency component of *x*(*t*).

Ideal system including anti-aliasing

To be sure that the reconstructed signal is free of aliasing it is customary to apply a lowpass filter, an **anti-aliasing filter**, before sampling as shown in Figure 1.15.

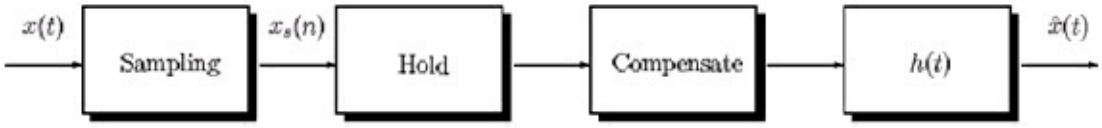
Figure 1.15.

Ideal reconstruction system with **anti-aliasing filter**

Again we ask the question of when the system will produce an output

? If the signal is

[redacted]



[redacted]

entirely confined within the passband of the lowpass filter we will get perfect reconstruction if *Fs* is high enough.

But if the anti-aliasing filter removes the "higher" frequencies, (which in fact is the job of the anti-aliasing filter), we will **never** be able to **exactly** reconstruct the original signal, *s*(*t*). If we sample

fast enough we can reconstruct $x(t)$, which in most cases is satisfying.

The reconstructed signal,

, will not have aliased frequencies. This is essential for further use

of the signal.

Reconstruction with hold operation

To make our reconstruction system realizable there are many things to look into. Among them are

the fact that any practical reconstruction system must input finite length pulses into the

reconstruction filter. This can be accomplished by the **hold operation**. To alleviate the distortion caused by the hold opeator we apply the output from the hold device to a compensator. The

compensation can be as accurate as we wish, this is cost and application consideration.

Figure 1.16.

More practical reconstruction system with a **hold component**

By the use of the hold component the reconstruction will not be exact, but as mentioned above we

can get as close as we want.

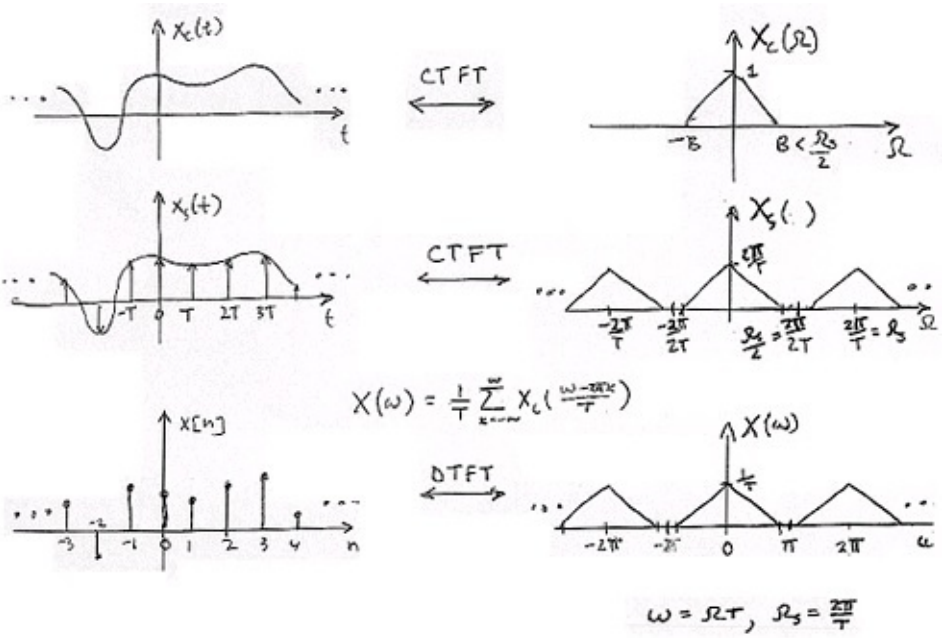
Introduction; **Proof**; **Illustrations**; **Matlab example**; **Hold operation**; **Aliasing applet**;

Exercises

Sampling CT Signals: A Frequency Domain Perspective*

Understanding Sampling in the Frequency Domain

We want to relate $xc(t)$ directly to $x[n]$. Compute the CTFT of



()

where $\omega \equiv \boldsymbol{\Omega T}$ and $X(\omega)$ is the DTFT of $x[n]$.

Recall

()

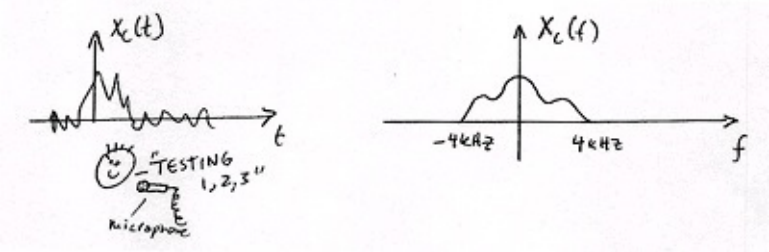
where this last part is 2π -periodic.

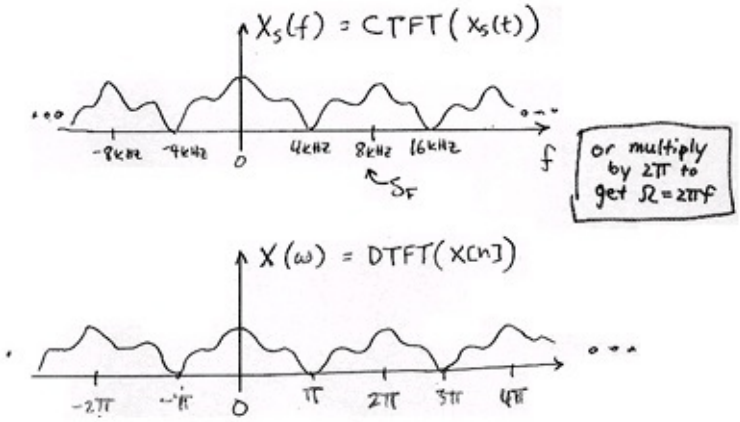
Sampling

Figure 1.17.

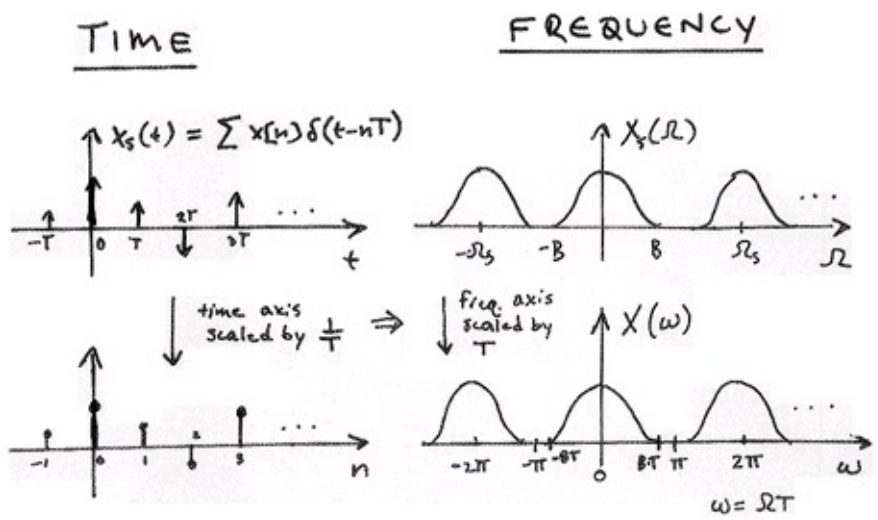
Example 1.8. Speech

Speech is intelligible if bandlimited by a CT lowpass filter to the band ±4 kHz. We can





[redacted]



sample speech as slowly as _____?

Figure 1.18.

Figure 1.19.

Note that there is no mention of *T* or *Ωs* !

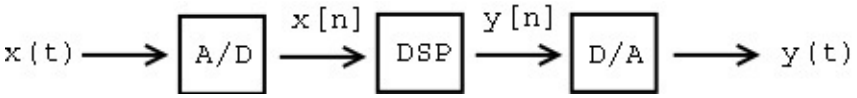
Relating x[n] to sampled x(t)

Recall the following equality:

Figure 1.20.

[redacted]

[redacted]



[redacted]

Recall the CTFT relation:

()

where α is a scaling of time and is a scaling in frequency.

()

$Xs(\ \Omega) \equiv X(\ \boldsymbol{\Omega T})$

The DFT: Frequency Domain with a Computer Analysis*

Introduction

We just covered ideal (and non-ideal) (time) **sampling of CT signals**. This enabled DT signal processing solutions for CT applications (Figure 1.21):

Figure 1.21.

Much of the theoretical analysis of such systems relied on frequency domain representations. How do we carry out these frequency domain analysis on the computer? Recall the following relationships:

where ω and Ω are continuous frequency variables.

Sampling DTFT

Consider the DTFT of a discrete-time (DT) signal $x[\ n]$. Assume $x[\ n]$ is of finite duration N (*i.e.* , an N-point signal).

()

where $X(\ \omega)$ is the continuous function that is indexed by the real-valued parameter $-\pi \leq \omega \leq \pi$. The other function, $x[\ n]$, is a discrete function that is indexed by integers.

We want to work with $X(\ \omega)$ on a computer. Why not just **sample** $X(\ \omega)$?

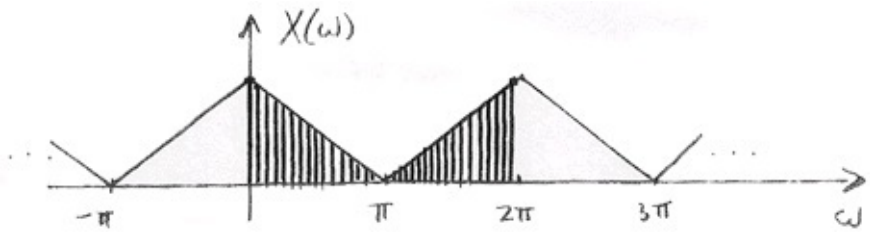
()

In Equation we sampled at

where $k=\{0, 1, \ldots, N-1\}$ and $X[k]$ for $k=\{0, \ldots, N-1\}$ is called

[redacted]

[redacted]



the **Discrete Fourier Transform (DFT)** of $x[n]$.

Example 1.9.

Figure 1.22. Finite Duration DT Signal

The DTFT of the image in Figure 1.22 is written as follows:

()

where ω is any 2π -interval, for example $-\pi \leq \omega \leq \pi$.

Figure 1.23. Sample X(ω)

where again we sampled at

where $k=\{0, 1, \ldots, M-1\}$. For example, we take $M=10$. In

the **following section** we will discuss in more detail how we should choose M, the number of samples in the 2π interval.

(This is precisely how we would plot $X(\omega)$ in Matlab.)

Choosing M

Case 1

Given N (length of $x[n]$), choose ($M \gg N$) to obtain a dense sampling of the DTFT (Figure 1.24):

Figure 1.24.

Case 2

Choose M as small as possible (to minimize the amount of computation).

In general, we require $M \geq N$ in order to represent all information in $x[n]$, $n=\{0, \ldots, N-1\}$ Let's concentrate on $M=N$:

for $n=\{0, \ldots, N-1\}$ and $k=\{0, \ldots, N-1\}$ numbers $\leftrightarrow$ N numbers

Discrete Fourier Transform (DFT)

Define

()

where N=length($x[n]$) and $k=\{0, \ldots, N-1\}$. In this case, $M=N$.

(1.38)

DFT

(1.39)

Inverse DFT (IDFT)

Interpretation

Represent $x[n]$ in terms of a sum of N **complex sinusoids** of amplitudes $X[k]$ and frequencies Think

Fourier Series with fundamental frequency

Remark 1

IDFT treats $x[\ n]$ as though it were N-periodic.

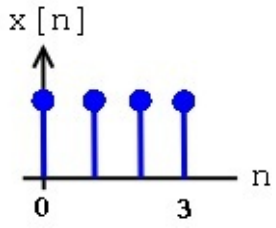
()

where $n \in \{0, \ldots, N-1\}$

Exercise 6.

What about other values of n?

[redacted]



[redacted]

[redacted]

[redacted]

$x[\ n+N]$=???

Remark 2

Proof that the IDFT inverts the DFT for $n \in \{0, \ldots, N-1\}$

()

Example 1.10. Computing DFT

Given the following discrete-time signal (Figure 1.25) with N=4 , we will compute the DFT

using two different methods (the DFT Formula and Sample DTFT):

Figure 1.25.

1. DFT Formula

()

Using the above equation, we can solve and get the following results:

$x[0]=4$ $x[1]=0$ $x[2]=0$ $x[3]=0$

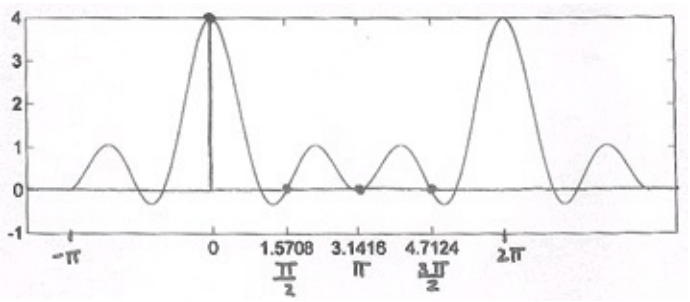
2. Sample DTFT. Using the same figure, Figure 1.25, we will take the DTFT of the signal and

get the following equations:

()

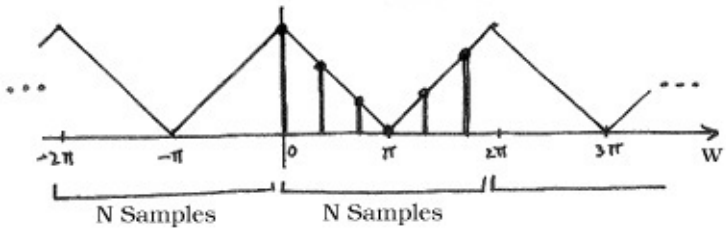
Our sample points will be:

where $k=\{0, 1, 2, 3\}$ (Figure 1.26).



[redacted]

[redacted]



[redacted]

Figure 1.26.

Periodicity of the DFT

DFT $X[k]$ consists of **samples** of DTFT, so $X(\omega)$, a 2π -periodic DTFT signal, can be converted to $X[k]$, an N-periodic DFT.

()

where

is an N-periodic basis function (See Figure 1.27).

Figure 1.27.

Also, recall,

()

Example 1.11. Illustration

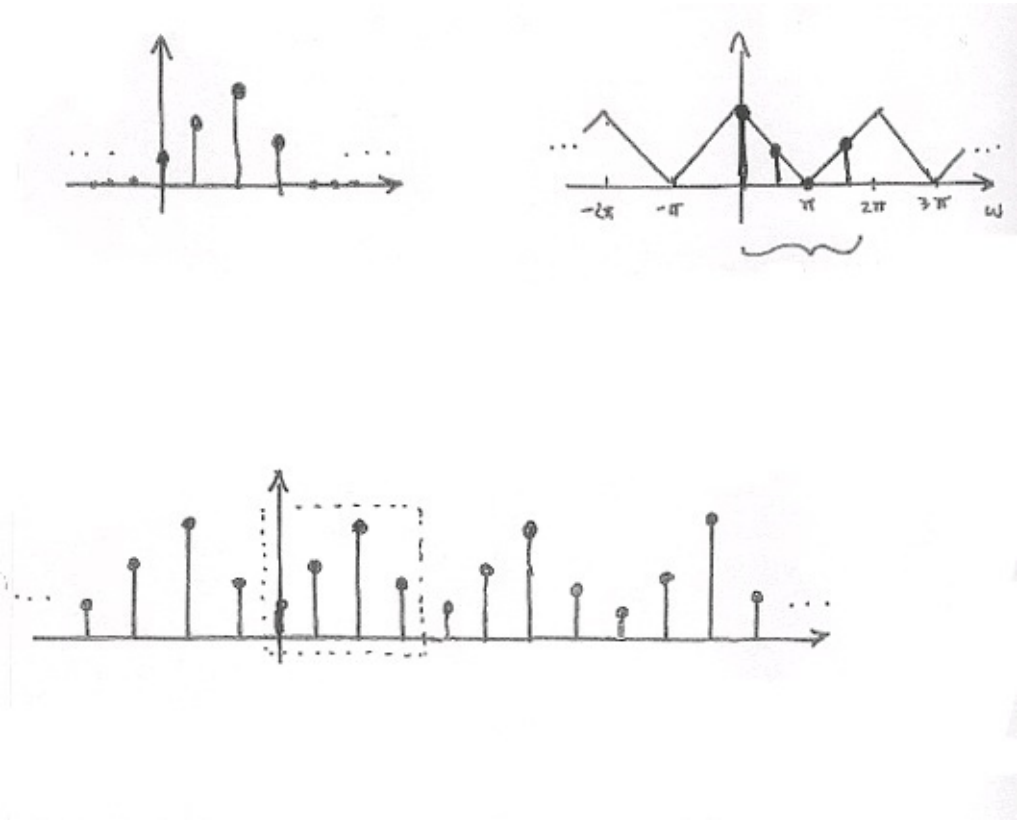


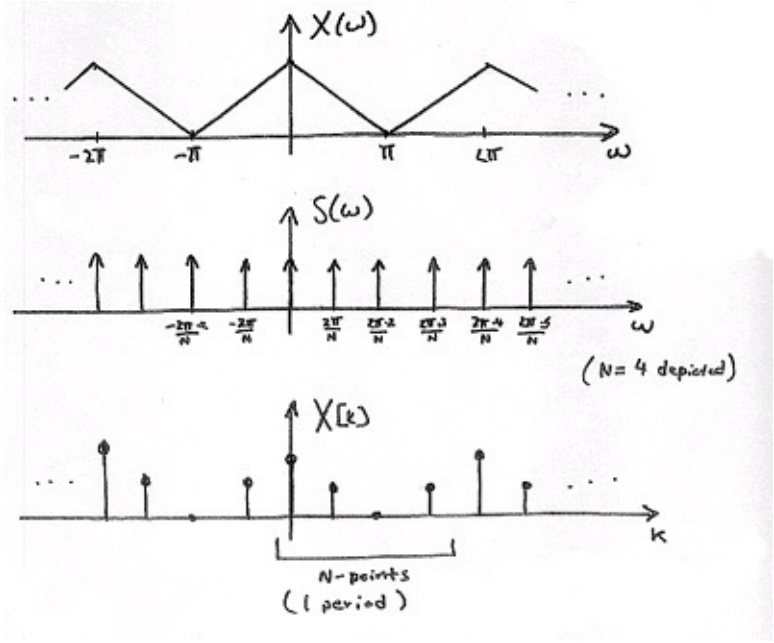
Figure 1.28.

When we deal with the DFT, we need to remember that, in effect, this treats the signal as

an N-periodic sequence.

A Sampling Perspective

Think of sampling the continuous function $X(\omega)$, as depicted in Figure 1.29. $S(\omega)$ will represent the sampling function applied to $X(\omega)$ and is illustrated in Figure 1.29 as well. This will result in our discrete-time sequence, $X[k]$.



[redacted]

[redacted]

[redacted]

Figure 1.29.

Recall

Remember the multiplication in the frequency domain is equal to convolution in the time domain!

Inverse DTFT of S(ω)

()

Given the above equation, we can take the DTFT and get the following equation:

()

Exercise 7.

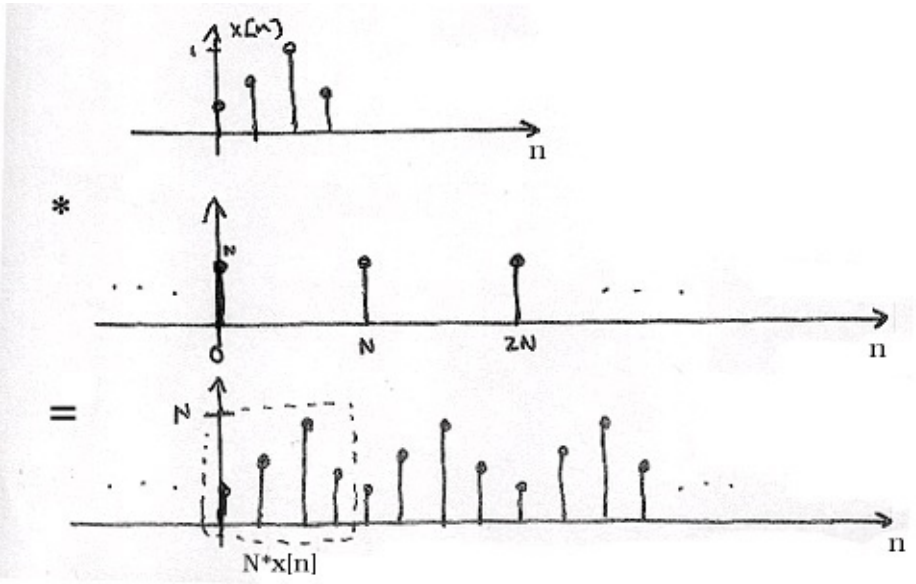
Why does Equation equal *S*[*n*] ?

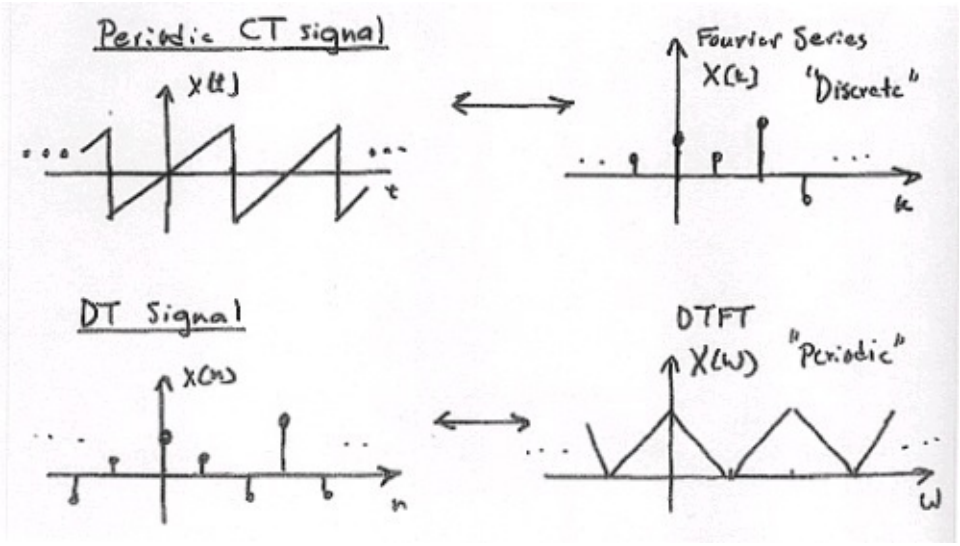
S[*n*] is *N*-periodic, so it has the following **Fourier Series**:

()

[redacted]

[redacted]





()

where the DTFT of the exponential in the above equation is equal to

.

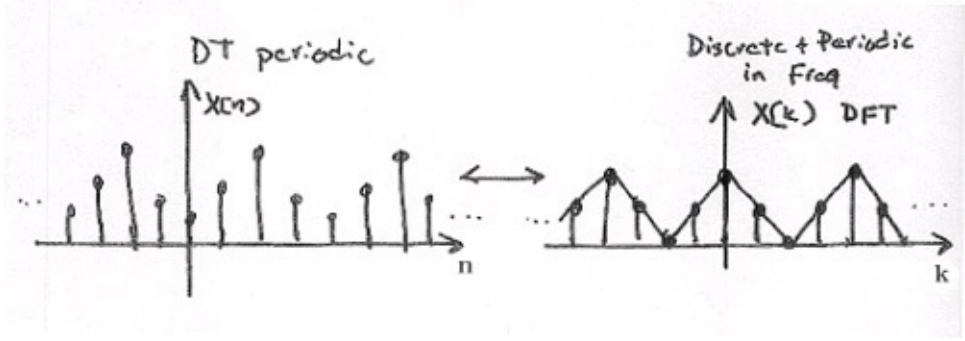
So, in the time-domain we have (Figure 1.30):

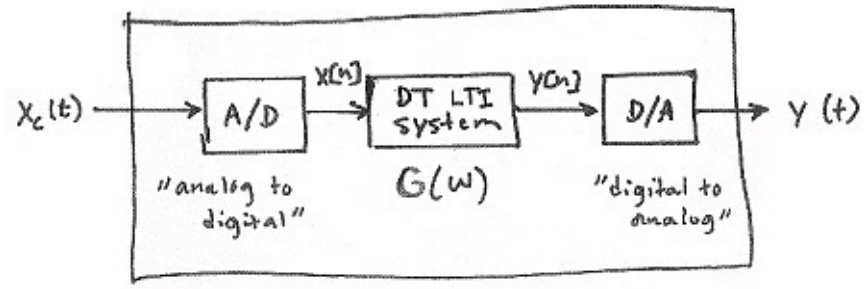
Figure 1.30.

Connections

Figure 1.31.

Combine signals in Figure 1.31 to get signals in Figure 1.32.





[redacted]

[redacted]

[redacted]

Figure 1.32.

Discrete-Time Processing of CT Signals*

DT Processing of CT Signals

Figure 1.33. DSP System

Analysis

()

$Yc(\Omega) = H\,\mathrm{LP}(\Omega)\,Y(\boldsymbol{\Omega T})$

where we know that $Y(\omega) = X(\omega)\,G(\omega)$ and $G(\omega)$ is the frequency response of the DT LTI system.

Also, remember that $\omega \equiv \boldsymbol{\Omega T}$ So,

()

$Yc(\Omega) = H\,\mathrm{LP}(\Omega)\,G(\boldsymbol{\Omega T})\,X(\boldsymbol{\Omega T})$

where $Yc(\Omega)$ and $H\,\mathrm{LP}(\Omega)$ are CTFTs and $G(\boldsymbol{\Omega T})$ and $X(\boldsymbol{\Omega T})$ are DTFTs.

Recall

Therefore our final output signal, $Yc(\Omega)$, will be:

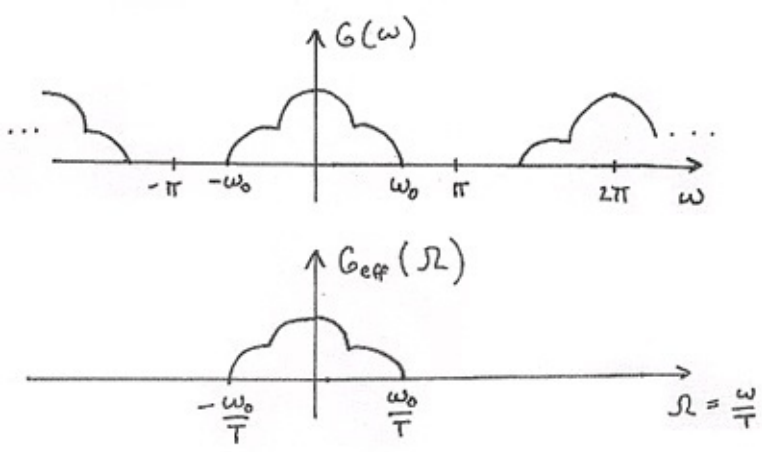
()

[redacted]

[redacted]

[redacted]

[redacted]



Now, if $Xc(\Omega)$ is bandlimited to

and we use the usual lowpass reconstruction filter in the

D/A, Figure 1.34:

Figure 1.34.

Then,

()

Summary

For bandlimited signals sampled at or above the Nyquist rate, we can relate the input and output of

the DSP system by:

()

$Yc(\Omega) = G\,\mathrm{eff}(\Omega)\, Xc(\Omega)$

where

Figure 1.35.

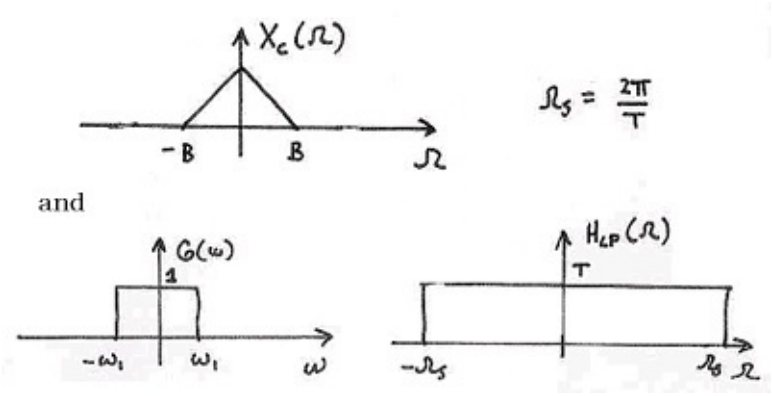
Note

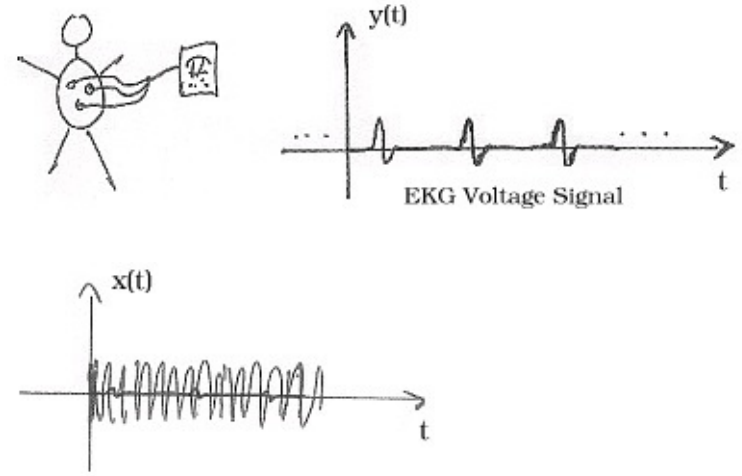
G eff(*Ω*) is LTI if and only if the following two conditions are satisfied:

1. *G*(*ω*) is LTI (in DT).

2. *Xc*(*T*) is bandlimited and sampling rate equal to or greater than Nyquist. For example, if we

had a simple pulse described by *Xc*(*t*)= *u*(*t*− *T* 0)− *u*(*t*− *T* 1) where *T* 1> *T* 0 . If the sampling period





T> *T* 1− *T* 0 , then some samples might "miss" the pulse while others might not be "missed." This is what we term **time-varying behavior**.

Example 1.12.

Figure 1.36.

If

and *ω* 1< ***BT*** , determine and sketch *Yc*(*Ω*) using Figure 1.36.

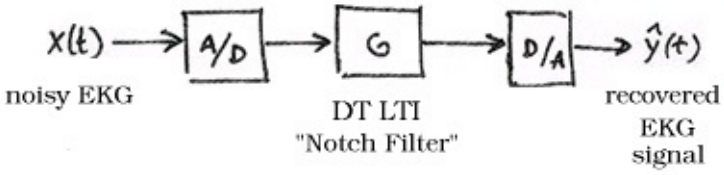
Application: 60Hz Noise Removal

Figure 1.37.

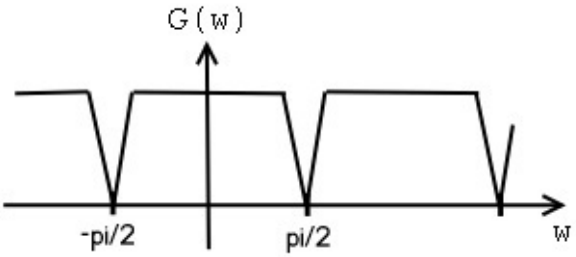
Unfortunately, in real-world situations electrodes also pick up ambient 60 Hz signals from lights, computers, *etc.* . In fact, usually this "60 Hz noise" is much greater in amplitude than the EKG signal shown in Figure 1.37. Figure 1.38 shows the EKG signal; it is barely noticeable as it has become overwhelmed by noise.

Figure 1.38.

Our EKG signal, $y(t)$, is overwhelmed by noise.



[redacted]



DSP Solution

Figure 1.39.

Figure 1.40.

Sampling Period/Rate

First we must note that $|Y(\Omega)|$ is **bandlimited** to ±60 Hz. Therefore, the minimum rate should be 120 Hz. In order to get the best results we should set fs= ***240Hz*** .

Figure 1.41.

Digital Filter

Therefore, we want to design a digital filter that will remove the 60Hz component and preserve

the rest.

Figure 1.42.

1.9. Z-Transform

[redacted]

[redacted]

Difference Equation*

Introduction

One of the most important concepts of DSP is to be able to properly represent the input/output relationship to a given LTI system. A linear constant-coefficient **difference equation** (LCCDE) serves as a way to express just this relationship in a discrete-time system. Writing the sequence of inputs and outputs, which represent the characteristics of the LTI system, as a difference equation help in understanding and manipulating a system.

Definition: difference equation

An equation that shows the relationship between consecutive values of a sequence and the differences among them. They are often rearranged as a recursive formula so that a systems output can be computed from the input signal and past outputs.

Example .

()

$y[n]+7\,y[n-1]+2\,y[n-2]=x[n]-4\,x[n-1]$

General Formulas for the Difference Equation

As stated briefly in the definition above, a difference equation is a very useful tool in describing and calculating the output of the system described by the formula for a given sample n. The key property of the difference equation is its ability to help easily find the transform, $H(z)$, of a system. In the following two subsections, we will look at the general form of the difference

equation and the general conversion to a z-transform directly from the difference equation.

Difference Equation

The general form of a linear, constant-coefficient difference equation (LCCDE), is shown below:

()

We can also write the general form to easily express a recursive output, which looks like this:

()

From this equation, note that $y[\ n-\ k]$ represents the outputs and $x[\ n-\ k]$ represents the inputs. The

value of N represents the **order** of the difference equation and corresponds to the memory of the system being represented. Because this equation relies on past values of the output, in order to compute a numerical solution, certain past outputs, referred to as the **initial conditions**, must be known.

Conversion to Z-Transform

Using the above formula, Equation, we can easily generalize the **transfer function**, $H(\ z)$, for any difference equation. Below are the steps taken to convert any difference equation into its transfer function, *i.e.* z-transform. The first step involves taking the **Fourier Transform** of all the terms in Equation. Then we use the linearity property to pull the transform inside the summation and the time-shifting property of the z-transform to change the time-shifting terms to exponentials. Once this is done, we arrive at the following equation: $a\ 0=1$.

()

()

Conversion to Frequency Response

Once the z-transform has been calculated from the difference equation, we can go one step further to define the frequency response of the system, or filter, that is being represented by the difference equation.

Remember that the reason we are dealing with these formulas is to be able to aid us in filter design. A LCCDE is one of the easiest ways to represent FIR filters. By being able to find the frequency response, we will be able to look at the basic properties of any filter represented by a simple LCCDE.

Below is the general formula for the frequency response of a z-transform. The conversion is simple a matter of taking the z-transform formula, $H(z)$, and replacing every instance of z with e^{jw} .

()

Once you understand the derivation of this formula, look at the module concerning **Filter Design from the Z-Transform** for a look into how all of these ideas of the **Z-transform**, Difference Equation, and **Pole/Zero Plots** play a role in filter design.

Example

Example 1.13. Finding Difference Equation

Below is a basic example showing the opposite of the steps above: given a transfer function one can easily calculate the systems difference equation.

()

Given this transfer function of a time-domain filter, we want to find the difference equation.

To begin with, expand both polynomials and divide them by the highest order *z*.

()

From this transfer function, the coefficients of the two polynomials will be our *ak* and *bk* values found in the general difference equation formula, Equation. Using these coefficients and the above form of the transfer function, we can easily write the difference equation:

()

In our final step, we can rewrite the difference equation in its more common form showing the recursive nature of the system.

()

Solving a LCCDE

In order for a linear constant-coefficient difference equation to be useful in analyzing a LTI system, we must be able to find the systems output based upon a known input, $x(n)$, and a set of

initial conditions. Two common methods exist for solving a LCCDE: the **direct method** and the **indirect method**, the later being based on the z-transform. Below we will briefly discuss the formulas for solving a LCCDE using each of these methods.

Direct Method

The final solution to the output based on the direct method is the sum of two parts, expressed in the following equation:

()

$y(n) = yh(n) + yp(n)$

The first part, $yh(n)$, is referred to as the **homogeneous solution** and the second part, $yh(n)$, is referred to as **particular solution**. The following method is very similar to that used to solve many differential equations, so if you have taken a differential calculus course or used differential equations before then this should seem very familiar.

Homogeneous Solution

We begin by assuming that the input is zero, $x(n) = 0$. Now we simply need to solve the homogeneous difference equation:

()

In order to solve this, we will make the assumption that the solution is in the form of an exponential. We will use lambda, λ, to represent our exponential terms. We now have to solve the following equation:

()

We can expand this equation out and factor out all of the lambda terms. This will give us a large polynomial in parenthesis, which is referred to as the **characteristic polynomial**. The roots of this polynomial will be the key to solving the homogeneous equation. If there are all distinct roots, then the general solution to the equation will be as follows:

()

$$yh(n) = C1(\lambda 1)n + C2(\lambda 2)n + \ldots + CN(\lambda N)n$$

However, if the characteristic equation contains multiple roots then the above general solution will be slightly different. Below we have the modified version for an equation where $\lambda 1$ has K multiple roots:

()

$$yh(n) = C1(\lambda 1)n + C1\,n(\lambda 1)n + C1\,n\,2(\lambda 1)n + \ldots + C1\,nK-1(\lambda 1)n + C2(\lambda 2)n + \ldots + CN(\lambda N)n$$

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Particular Solution

The particular solution, $yp(n)$, will be any solution that will solve the general difference equation:

()

In order to solve, our guess for the solution to $yp(n)$ will take on the form of the input, $x(n)$. After guessing at a solution to the above equation involving the particular solution, one only needs to

plug the solution into the difference equation and solve it out.

Indirect Method

The indirect method utilizes the relationship between the difference equation and z-transform,

discussed **earlier,** to find a solution. The basic idea is to convert the difference equation into a z-transform, as described **above,** to get the resulting output, $Y(z)$. Then by inverse transforming this and using partial-fraction expansion, we can arrive at the solution.

(1.40)

$Z\{y(n+1)-y(n)\} = zY(z)-y(0)$ This can be interatively extended to an arbitrary order derivative as in Equation Equation 1.41.

(1.41)

Now, the Laplace transform of each side of the differential equation can be taken

(1.42)

which by linearity results in

(1.43)

and by differentiation properties in

(1.44)

Rearranging terms to isolate the Laplace transform of the output,

(1.45)

Thus, it is found that

(1.46)

In order to find the output, it only remains to find the Laplace transform $X(z)$ of the input, substitute the initial conditions, and compute the inverse Z-transform of the result. Partial fraction expansions are often required for this last step. This may sound daunting while looking at Equation 1.46, but it is often easy in practice, especially for low order difference equations. Equation 1.46 can also be used to determine the transfer function and frequency response.

As an example, consider the difference equation

(1.47)

$y[n-2] + 4y[n-1] + 3y[n] = \cos(n)$ with the initial conditions $y'(0)=1$ and $y(0)=0$ Using the method described above, the Z transform

of the solution $y[n]$ is given by

(1.48)

Performing a partial fraction decomposition, this also equals

(1.49)

Computing the inverse Laplace transform,

(1.50)

One can check that this satisfies that this satisfies both the differential equation and the initial conditions.

The Z Transform: Definition*

Basic Definition of the Z-Transform

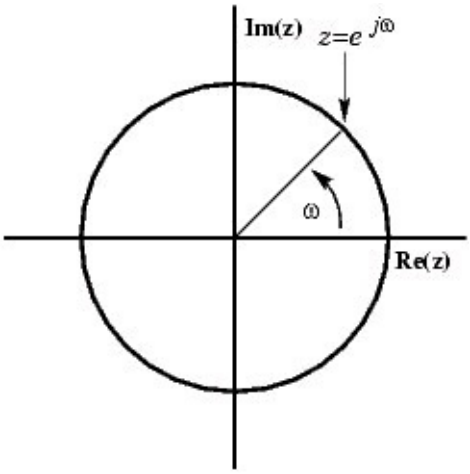
The **z-transform** of a sequence is defined as

()

Sometimes this equation is referred to as the **bilateral z-transform**. At times the z-transform is defined as

[redacted]

[redacted]



[redacted]

()

which is known as the **unilateral z-transform**.

There is a close relationship between the z-transform and the **Fourier transform** of a discrete time signal, which is defined as

()

Notice that that when the $z– n$ is replaced with $e–(iωn)$ the z-transform reduces to the Fourier

Transform. When the Fourier Transform exists, $z = e^{i\omega}$, which is to have the magnitude of z equal to unity.

The Complex Plane

In order to get further insight into the relationship between the Fourier Transform and the Z-Transform it is useful to look at the complex plane or **z-plane**. Take a look at the complex plane:

Figure 1.43. Z-Plane

The Z-plane is a complex plane with an imaginary and real axis referring to the complex-valued variable z. The position on the complex plane is given by $re^{i\omega}$, and the angle from the positive, real axis around the plane is denoted by ω. $X(z)$ is defined everywhere on this plane. $X(e^{i\omega})$ on the other hand is defined only where $|z|=1$, which is referred to as the unit circle. So for example,

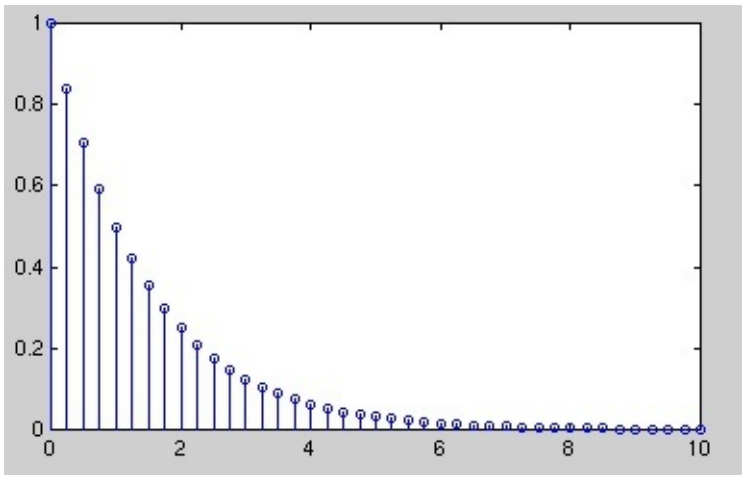
$\omega=1$ at $z=1$ and

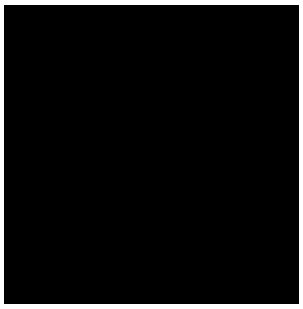
at $z=-1$. This is useful because, by representing the Fourier transform as the

z-transform on the unit circle, the periodicity of Fourier transform is easily seen.

Region of Convergence

The region of convergence, known as the **ROC**, is important to understand because it defines the





region where the z-transform exists. The ROC for a given $x[n]$, is defined as the range of z for which the z-transform converges. Since the z-transform is a **power series**, it converges when $x[n]z^{-n}$ is absolutely summable. Stated differently,

()

must be satisfied for convergence. This is best illustrated by looking at the different ROC's of the z-transforms of $\alpha^n u[n]$ and $\alpha^n u[n-1]$.

Example 1.14.

For

()

$x[n]=\alpha^n u[n]$

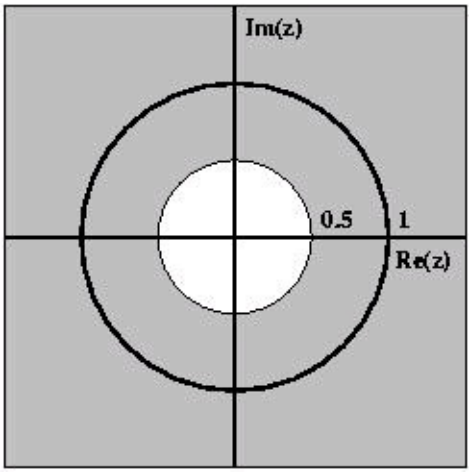
Figure 1.44.

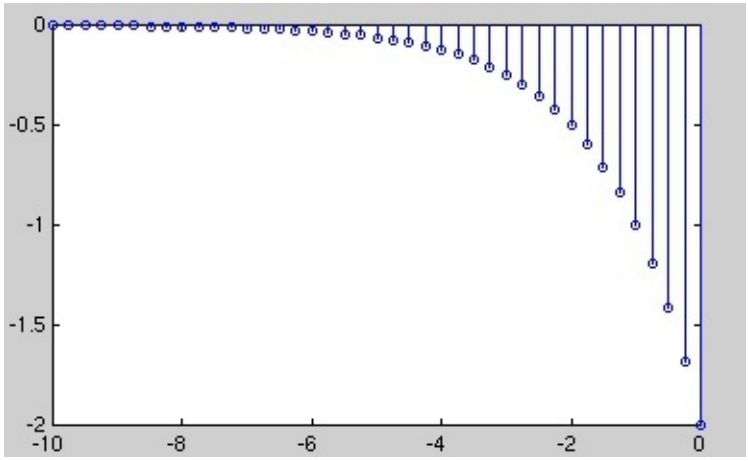
$x[n]=\alpha^n u[n]$ where $\alpha=0.5$.

()

This sequence is an example of a right-sided exponential sequence because it is nonzero for $n\geq 0$. It only converges when $|\alpha z^{-1}|<1$. When it converges,

()





If | αz-1|≥1 , then the series,

does not converge. Thus the ROC is the range of values

where

()

| αz-1|<1

or, equivalently,

()

| z|>| α|

Figure 1.45.

ROC for $x[\ n]= \alpha nu[\ n]$ where α=0.5

Example 1.15.

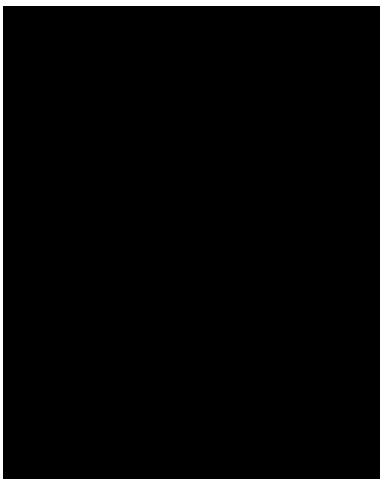
For

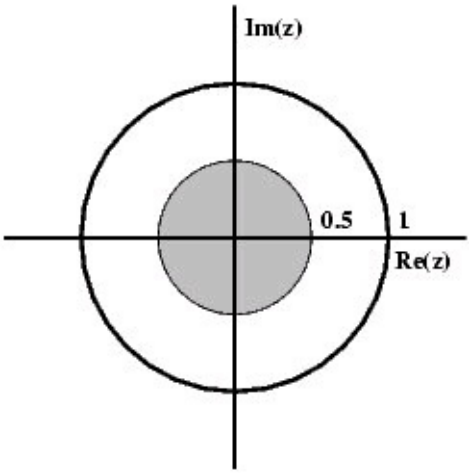
()

$x[n]=(-(\alpha n))\,u[-n-1]$

Figure 1.46.

$x[n]=(-(\alpha n))\,u[-n-1]$ where $\alpha=0.5$.





()

The ROC in this case is the range of values where

()

$|\alpha^{-1} z|<1$

or, equivalently,

()

$|z|<|\alpha|$

If the ROC is satisfied, then

()

Figure 1.47.

ROC for $x[n]=(-(\alpha n))\, u[-n-1]$

Table of Common z-Transforms*

The table below provides a number of unilateral and bilateral **z-transforms**. The table also specifies the **region of convergence**.

The notation for z found in the table below may differ from that found in other tables. For example, the basic z-transform of $u[n]$ can be written as either of the following two

expressions, which are equivalent:

()

Table 1.2.

Signal

Z-Transform ROC

$\delta[n-k]$

$z-k$

$u[n]$

$|z|>1$

$-(u[-n-1])$

$|z|<1$

$nu[n]$

$|z|>1$

$n\ 2\ u[n]$

$|z|>1$

$n\ 3\ u[n]$

$|z|>1$

$(-(\alpha n))\ u[-n-1]$

$|z|<|\alpha|$

$\alpha nu[n]$

$|z|>|\alpha|$

$n\alpha nu[n]$

$|z|>|\alpha|$

$n\ 2\ \alpha nu[n]$

$|z|>|\alpha|$

$\gamma n \cos(\boldsymbol{\alpha n}) u[n]$

| *z*|>| *γ*|

γn sin(***αn***) *u*[*n*]

| *z*|>| *γ*|

Understanding Pole/Zero Plots on the Z-Plane*

[redacted]

Introduction to Poles and Zeros of the Z-Transform

It is quite difficult to qualitatively analyze the **Laplace transform** and **Z-transform**, since mappings of their magnitude and phase or real part and imaginary part result in multiple

mappings of 2-dimensional surfaces in 3-dimensional space. For this reason, it is very common to

examine a plot of a **transfer function's** poles and zeros to try to gain a qualitative idea of what a system does.

Once the Z-transform of a system has been determined, one can use the information contained in

function's polynomials to graphically represent the function and easily observe many defining

characteristics. The Z-transform will have the below structure, based on **Rational Functions**: **()**

The two polynomials, *P*(*z*) and *Q*(*z*), allow us to find the **poles and zeros** of the Z-Transform.

Definition: zeros

1. The value(s) for *z* where

.

2. The complex frequencies that make the overall gain of the filter transfer function zero.

Definition: poles

1. The value(s) for *z* where

.

2. The complex frequencies that make the overall gain of the filter transfer function infinite.

Example 1.16.

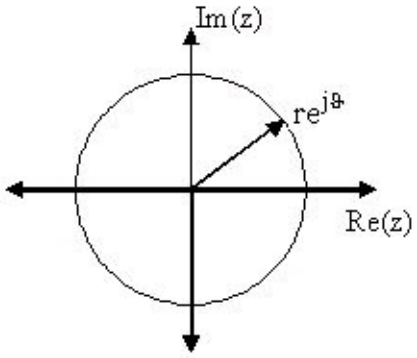
Below is a simple transfer function with the poles and zeros shown below it.

The zeros are: {–1}

The poles are:

The Z-Plane

Once the poles and zeros have been found for a given Z-Transform, they can be plotted onto the Z-Plane. The Z-plane is a complex plane with an imaginary and real axis referring to the complex-valued variable *z*. The position on the complex plane is given by $re^{j\theta}$ and the angle from the positive, real axis around the plane is denoted by θ. When mapping poles and zeros onto the plane, poles are denoted by an "x" and zeros by an "o". The below figure shows the Z-Plane, and examples of plotting zeros and poles onto the plane can be found in the following section.



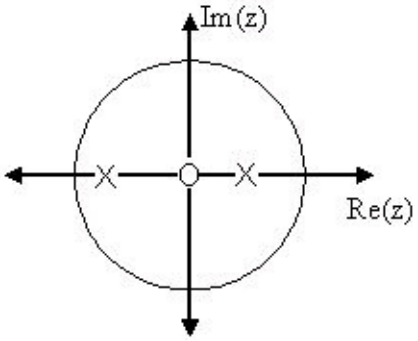


Figure 1.48. Z-Plane

Examples of Pole/Zero Plots

This section lists several examples of finding the poles and zeros of a transfer function and then plotting them onto the Z-Plane.

Example 1.17. Simple Pole/Zero Plot

The zeros are: {0}

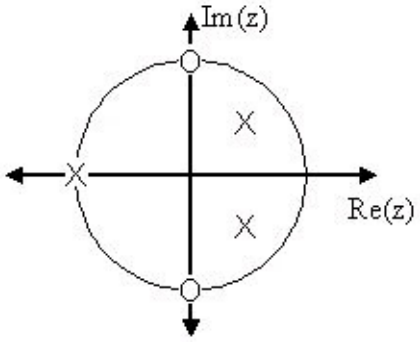
The poles are:

Figure 1.49. Pole/Zero Plot

Using the zeros and poles found from the transfer function, the one zero is mapped to zero and the two poles are placed at and

Example 1.18. Complex Pole/Zero Plot

The zeros are: { i, $-i$}



The poles are:

Figure 1.50. Pole/Zero Plot

Using the zeros and poles found from the transfer function, the zeros are mapped to $\pm i$, and the poles are placed at –1,

and

Example 1.19. Pole-Zero Cancellation

An easy mistake to make with regards to poles and zeros is to think that a function like

is the same as $s+3$. In theory they are equivalent, as the pole and zero at $s=1$ cancel

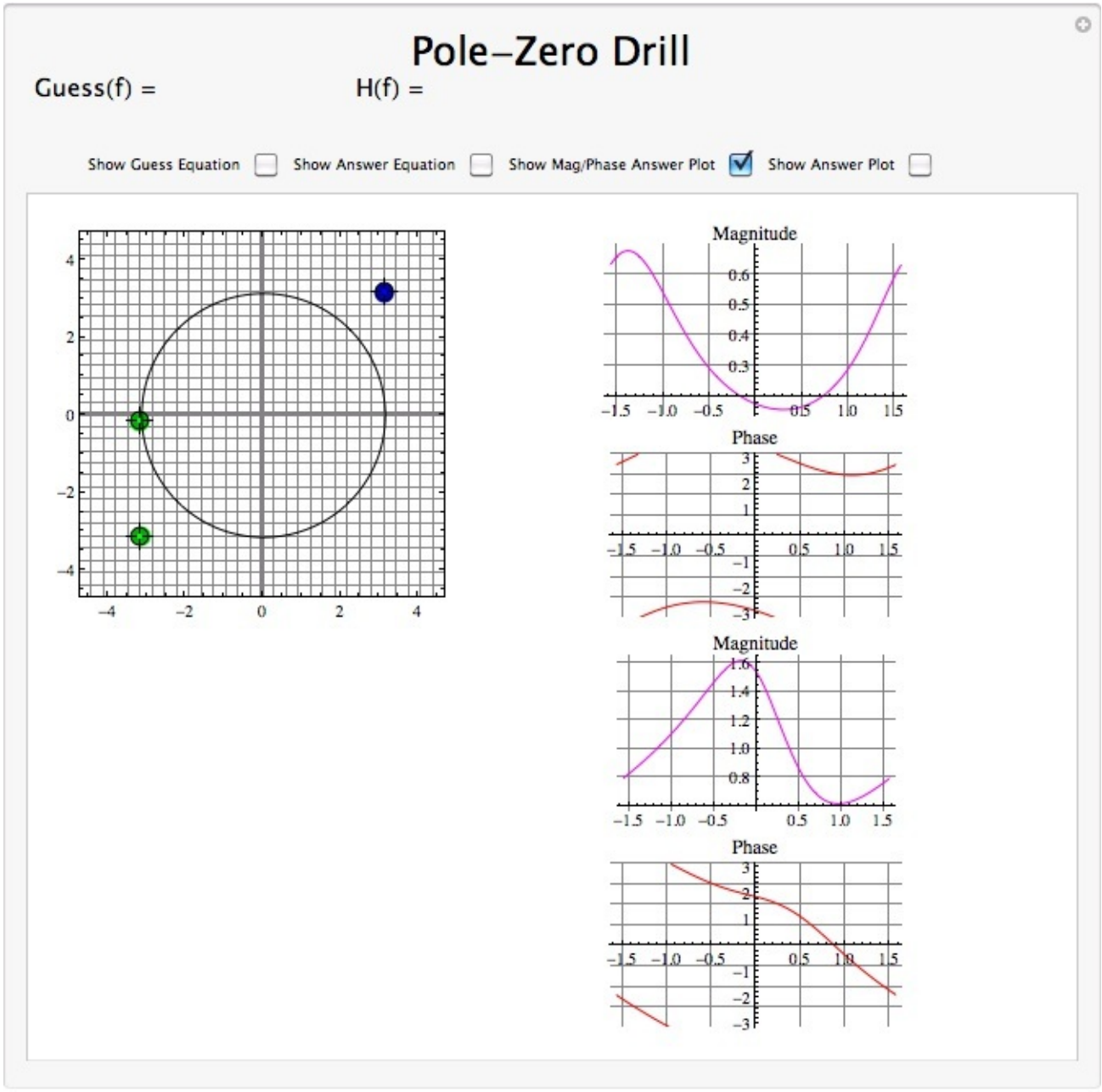
each other out in what is known as **pole-zero cancellation**. However, think about what may happen if this were a transfer function of a system that was created with physical circuits. In this case, it is very unlikely that the pole and zero would remain in exactly the same place. A minor temperature change, for instance, could cause one of them to move just slightly. If this were to occur a tremendous amount of volatility is created in that area, since there is a change from infinity at the pole to zero at the zero in a very small range of signals. This is generally a very bad way to try to eliminate a pole. A much better way is to use **control theory** to move the pole to a better place.

Repeated Poles and Zeros

It is possible to have more than one pole or zero at any given point. For instance, the discrete-time transfer function $H(z)= z^2$ will have two zeros at the origin and the continuous-time function

will have 25 poles at the origin.

MATLAB - If access to MATLAB is readily available, then you can use its functions to easily create pole/zero plots. Below is a short program that plots the poles and zeros from the above example onto the Z-Plane.




```
% Set up vector for zeros
z = [j ; -j];
% Set up vector for poles
p = [-1 ; .5+.5j ; .5-.5j];
figure(1);
zplane(z,p);
title('Pole/Zero Plot for Complex Pole/Zero Plot Example');
```

Interactive Demonstration of Poles and Zeros

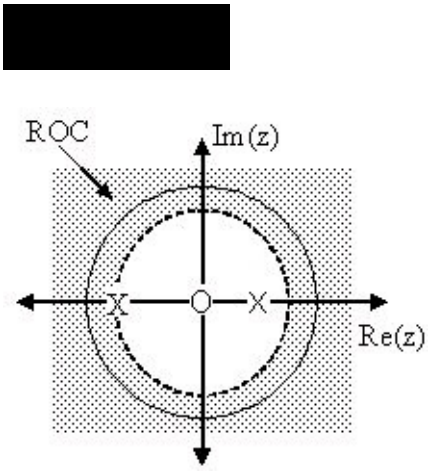
Figure 1.51.

Interact (when online) with a Mathematica CDF demonstrating Pole/Zero Plots. To Download, right-click and save target as .cdf.

Applications for pole-zero plots

Stability and Control theory

Now that we have found and plotted the poles and zeros, we must ask what it is that this plot gives us. Basically what we can gather from this is that the magnitude of the transfer function will be larger when it is closer to the poles and smaller when it is closer to the zeros. This provides us



with a qualitative understanding of what the system does at various frequencies and is crucial to the discussion of **stability.**

Pole/Zero Plots and the Region of Convergence

The region of convergence (ROC) for $X(z)$ in the complex Z-plane can be determined from the pole/zero plot. Although several regions of convergence may be possible, where each one corresponds to a different impulse response, there are some choices that are more practical. A ROC can be chosen to make the transfer function causal and/or stable depending on the pole/zero plot.

Filter Properties from ROC

If the ROC extends outward from the outermost pole, then the system is **causal**.

If the ROC includes the unit circle, then the system is **stable**.

Below is a pole/zero plot with a possible ROC of the Z-transform in the **Simple Pole/Zero Plot** discussed earlier. The shaded region indicates the ROC chosen for the filter. From this figure, we

can see that the filter will be both causal and stable since the above listed conditions are both met.

Example 1.20.

Figure 1.52. Region of Convergence for the Pole/Zero Plot

The shaded area represents the chosen ROC for the transfer function.

Frequency Response and Pole/Zero Plots

The reason it is helpful to understand and create these pole/zero plots is due to their ability to help us easily design a filter. Based on the location of the poles and zeros, the magnitude response of the filter can be quickly understood. Also, by starting with the pole/zero plot, one can design a filter and obtain its transfer function very easily.

Glossary

Definition: difference equation

An equation that shows the relationship between consecutive values of a sequence and the differences among them. They are often rearranged as a recursive formula so that a systems output can be computed from the input signal and past outputs. **Example .**

simplemath *mathml-miitalicsy*[*mathml-miitalicsn*]+7 *mathml-miitalicsy*[*mathml-miitalicsn*−1]+2

Definition: poles

1. The value(s) for simplemath *mathml-miitalicsz* z where Qz= .

2. The complex frequencies that make the overall gain of the filter transfer function infinite.

Definition: zeros

1. The value(s) for simplemath *mathml-miitalicsz* z where Pz= .

2. The complex frequencies that make the overall gain of the filter transfer function zero.

Solutions

Chapter 2. Digital Filter Design

2.1. Overview of Digital Filter Desig[n*]

Advantages of FIR filters

1. Straight forward conceptually and simple to implement

2. Can be implemented with fast convolution

3. Always stable

4. Relatively insensitive to quantization

5. Can have linear phase (same time delay of all frequencies)

Advantages of IIR filters

1. Better for approximating analog systems

2. For a given magnitude response specification, IIR filters often require much less computation

than an equivalent FIR, particularly for narrow transition bands

Both FIR and IIR filters are very important in applications.

Generic Filter Design Procedure

1. Choose a desired response, based on application requirements

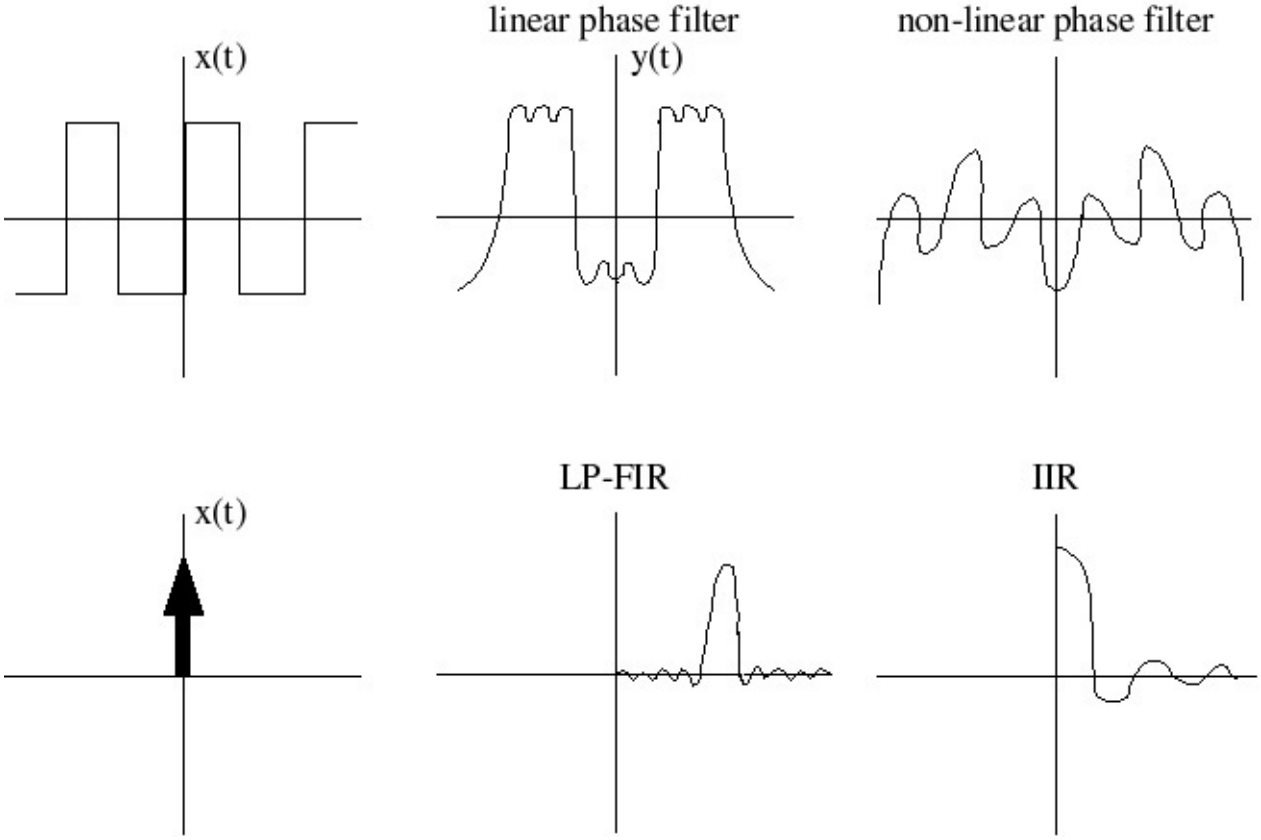
2. Choose a filter class

3. Choose a quality measure

4. Solve for the filter in class 2 optimizing criterion in 3

Perspective on FIR filtering

Most of the time, people do $L\infty$ optimal design, using the **Parks-McClellan algorithm**. This is probably the second most important technique in "classical" signal processing (after the Cooley-



Tukey (**radix-2**) FFT).

Most of the time, FIR filters are designed to have linear phase. The most important advantage of FIR filters over IIR filters is that they can have exactly linear phase. There are advanced design techniques for minimum-phase filters, constrained *L* 2 optimal designs, *etc.* (see chapter 8 of text). However, if only the **magnitude** of the response is important, IIR filers usually require much fewer operations and are typically used, so the bulk of FIR filter design work has concentrated on linear phase designs.

2.2. FIR Filter Design

Linear Phase Filters*

In general, for $-\pi \le \omega \le \pi$ $H(\omega)=|H(\omega)|e^{-j\theta(\omega)}$ Strictly speaking, we say $H(\omega)$ is linear phase if $H(\omega)=|H(\omega)|e^{-j\omega K}e^{-j\theta 0}$ Why is this important? A linear phase response gives the **same time delay for ALL frequencies**! (Remember the shift theorem.) This is very desirable in many applications, particularly when the appearance of the time-domain waveform is of interest, such as in an oscilloscope. (see Figure 2.1)

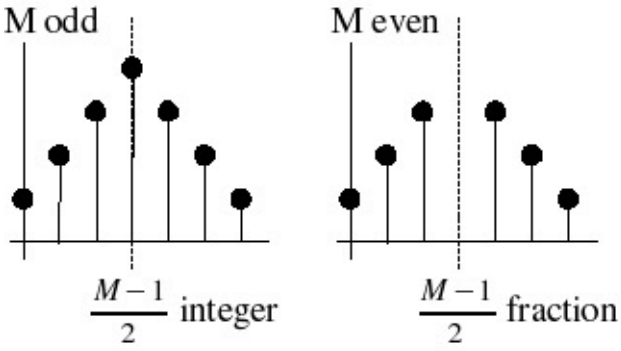


Figure 2.1.

Restrictions on h(n) to get linear phase

()

For linear phase, we require the right side of Equation to be ℮–(ⅈθ 0)(real,positive function of ω) .

For θ 0=0 , we thus require

$h(0)+h(M-1)$=real number $h(0)-h(M-1)$=pure imaginary number $h(1)+h(M-2)$=pure real number $h(1)-h(M-2)$=pure imaginary number ⋮ Thus $h(k)=h^*(M-1-k)$ is a **necessary** condition for the right side of Equation to be real valued, for θ 0=0 .

For

, or ℮–(ⅈθ 0)=– ⅈ , we require

$h(0)+h(M-1)$=pure imaginary $h(0)-h(M-1)$=pure real number ⋮ ⇒ $h(k)=-(h^*(M-1-k))$ Usually, one is interested in filters with **real-valued** coefficients, or see Figure 2.2 and Figure 2.3.

Figure 2.2.

θ 0=0 (Symmetric Filters). $h(k)=h(M-1-k)$.

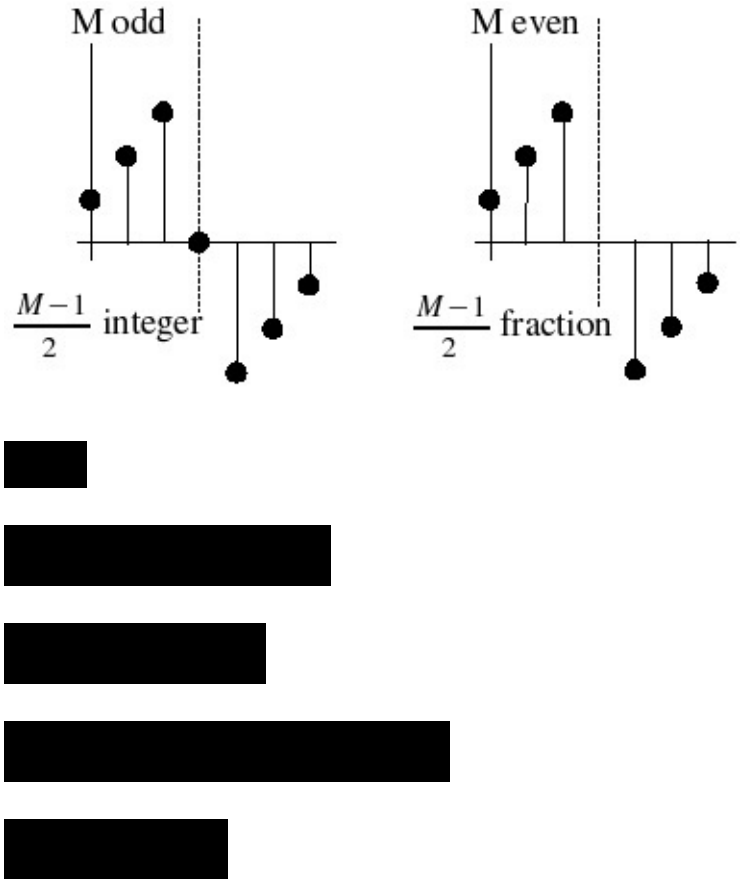


Figure 2.3.

(Anti-Symmetric Filters). $h(k) = -(h(M-1-k))$.

Filter design techniques are usually slightly different for each of these four different filter types. We will study the most common case, symmetric-odd length, in detail, and often leave the others for homework or tests or for when one encounters them in practice. Even-symmetric filters are often used; the anti-symmetric filters are rarely used in practice, except for special classes of filters, like differentiators or Hilbert transformers, in which the desired response is anti-symmetric.

So far, we have satisfied the condition that

where $A(\omega)$ is **real-valued**.

However, we have **not** assured that $A(\omega)$ is **non-negative**. In general, this makes the design techniques much more difficult, so most FIR filter design methods actually design filters with **Generalized Linear Phase**:

, where $A(\omega)$ is **real-valued**, but possible negative.

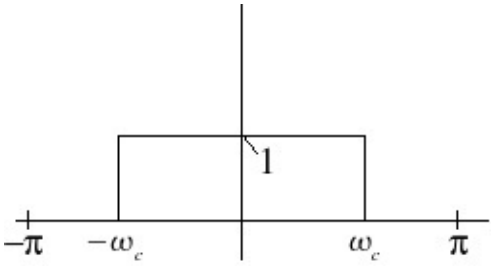
$A(\omega)$ is called the **amplitude of the frequency response**.

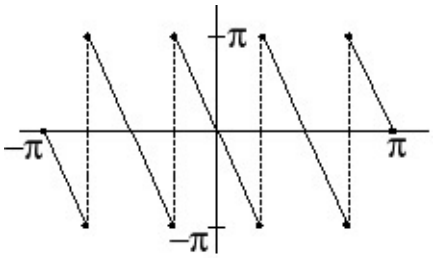
Excuse

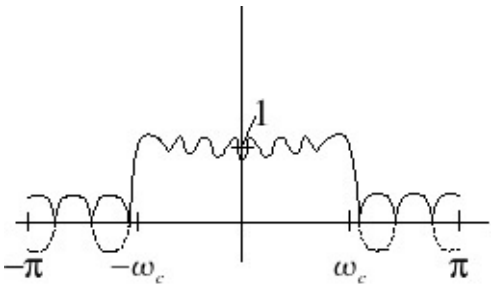
$A(\omega)$

Example 2.1.

Lowpass Filter







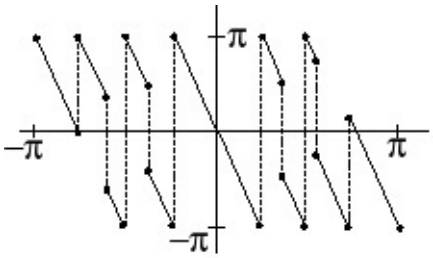


Figure 2.4. Desired |H(ω)|

Figure 2.5. Desired ∠H(ω)

The slope of each line is

.

Figure 2.6. Actual |H(ω)|

$A(\omega)$ goes negative.

Figure 2.7. Actual ∠H(ω)

2 π phase jumps due to periodicity of phase. π phase jumps due to sign change in $A(\omega)$.

Time-delay introduces generalized linear phase.

For odd-length FIR filters, a linear-phase design procedure is equivalent to a zero-phase design procedure followed by an

-sample delay of the **impulse response**. For even-length filters, the delay is non-integer, and the linear phase must be incorporated directly in the desired response!

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Window Design Method*

The truncate-and-delay design procedure is the simplest and most obvious FIR design procedure.

Exercise 1.

Is it any Good?

Yes; in fact it's optimal! (in a certain sense)

L2 optimization criterion

find $h[\ n]$, $0 \leq n \leq M-1$, maximizing the energy difference between the desired response and the actual response: i.e., find

by **Parseval's relationship**

()

Since

this becomes

$h[\ n]$

The best we can do is let

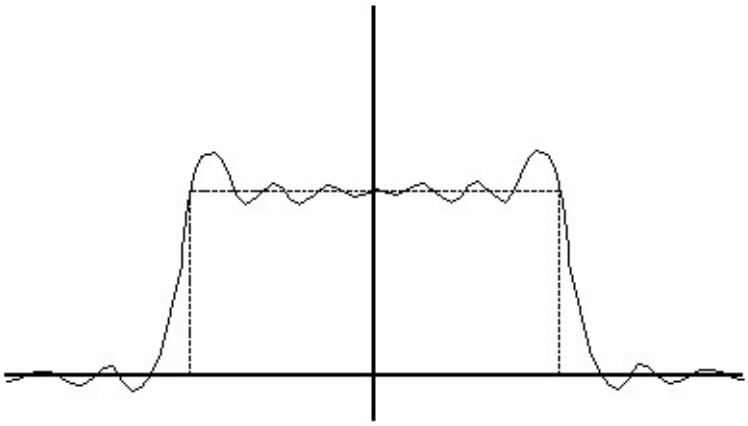
Thus $h[\ n] = hd[\ n]\ w[\ n]$,

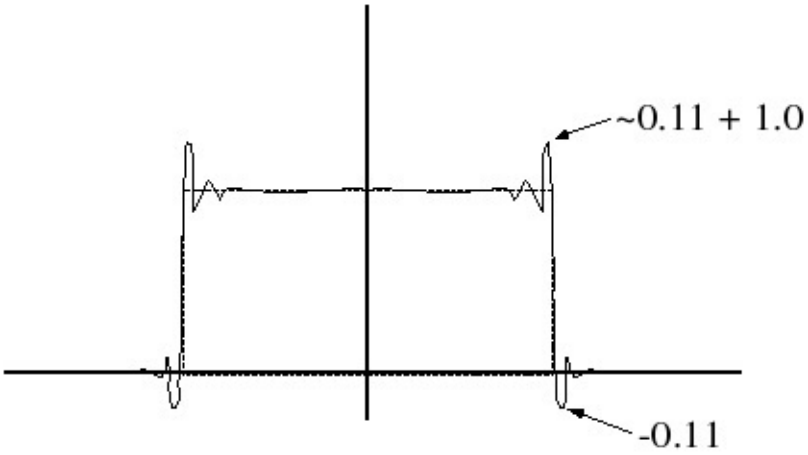
is

optimal in a least-total-sqaured-error ($L\ 2$, or energy) sense!

Exercise 2.

Why, then, is this design often considered undersirable?





(b) $A(\omega)$, large M

(a) $A(\omega)$, small M

Figure 2.7.

For desired spectra with discontinuities, the least-square designs are poor in a minimax (worst-case, or $L\infty$) error sense.

Window Design Method

Apply a more gradual truncation to reduce "ringing" (**Gibb's Phenomenon)**

$H(\omega) = Hd(\omega) * W(\omega)$

The window design procedure (except for the boxcar window) is ad-hoc and not optimal in any usual sense. However, it is very simple, so it is sometimes used for "quick-and-dirty" designs of if the error criterion is itself heurisitic.

Frequency Sampling Design Method for FIR filters*

Given a desired frequency response, the frequency sampling design method designs a filter with a frequency response **exactly** equal to the desired response at a particular set of frequencies ωk .

(2.1)

Procedure

Desired Response must incluce linear phase shift (if linear phase is desired)

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Exercise 3.

What is $Hd(\omega)$ for an ideal lowpass filter, cotoff at ωc ?

This set of linear equations can be written in matrix form

(2.2)

(2.3)

or

So

(2.4)

$W\ N = M\ \omega i \neq \omega j + \mathbf{2\pi l}\ i \neq j$

Important Special Case

What if the frequencies are equally spaced between 0 and 2 π , i.e.

Then

so

or

Important Special Case #2

$h[\ n]$ symmetric, linear phase, and has real coefficients. Since $h[\ n]= h[\ M- n]$, there are only degrees of freedom, and only linear equations are required.

(2.5)

Removing linear phase from both sides yields

Due to

symmetry of response for real coefficients, only ωk on $\omega \in [0, \pi)$ need be specified, with the frequencies $-\omega k$ thereby being implicitly defined also. Thus we have **real-valued** simultaneous linear equations to solve for $h[\, n]$.

Special Case 2a

$h[\, n]$ symmetric, odd length, linear phase, real coefficients, and ωk equally spaced:

(2.6)

To yield real coefficients, $A(\, \omega)$ mus be symmetric $A(\, \omega) = A(-\omega) \Rightarrow A[\, k] = A[\, M - k]$

(2.7)

Simlar equations exist for even lengths, anti-symmetric, and

filter forms.

Comments on frequency-sampled design

This method is simple conceptually and very efficient for equally spaced samples, since $h[\,n]$ can be computed using the IDFT.

$H(\,\omega)$ for a frequency sampled design goes **exactly** through the sample points, but it may be very far off from the desired response for $\omega \neq \omega k$. This is the main problem with frequency sampled design.

Possible solution to this problem: specify more frequency samples than degrees of freedom, and minimize the total error in the frequency response at all of these samples.

Extended frequency sample design

For the samples $H(\,\omega k)$ where $0 \leq k \leq M-1$ and $N > M$, find $h[\,n]$, where $0 \leq n \leq M-1$ minimizing

$\| Hd(\,\omega k) - H(\,\omega k) \|$

For $\| l \|_\infty$ norm, this becomes a linear programming problem (standard packages availble!)

Here we will consider the $\| l \|_2$ norm.

To minimize the $\| l \|_2$ norm; that is,

, we have an overdetermined set of linear

equations:

or

The minimum error norm solution is well known to be

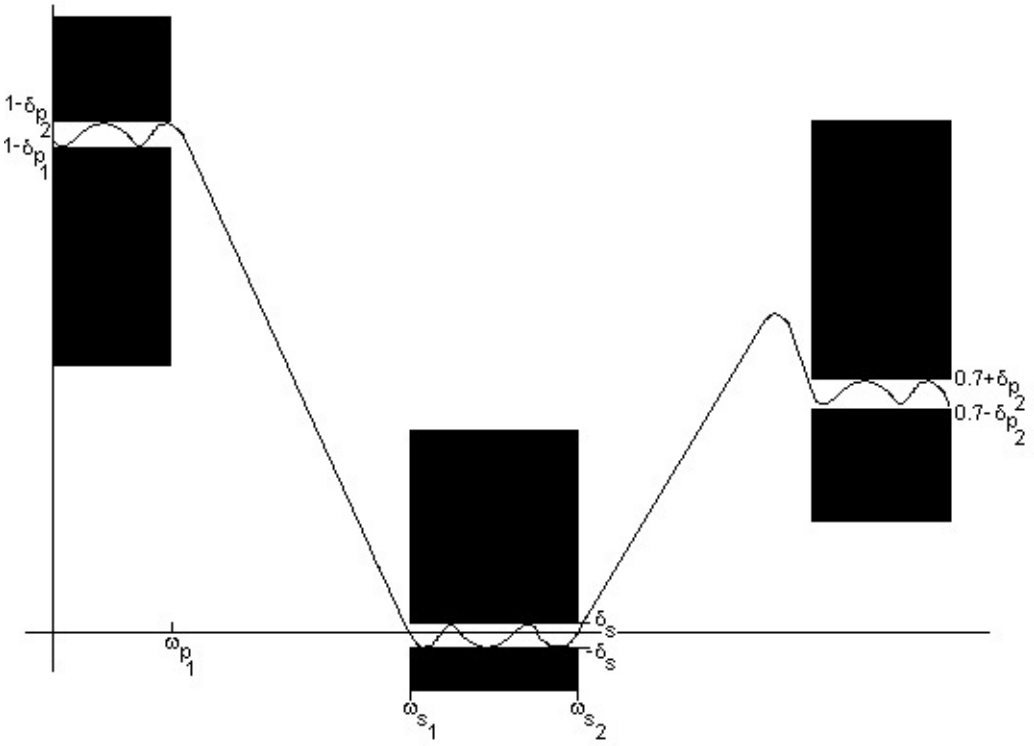
;

is well known as the

pseudo-inverse matrix.

Extended frequency sampled design discourages radical behavior of the frequency response between samples for sufficiently closely spaced samples. However, the actual frequency response may no longer pass exactly through **any** of the $Hd(\,\omega k)$.

Parks-McClellan FIR Filter Design*



The approximation tolerances for a filter are very often given in terms of the maximum, or worst-case, deviation within frequency bands. For example, we might wish a lowpass filter in a (16-bit) CD player to have no more than -bit deviation in the pass and stop bands.

The Parks-McClellan filter design method efficiently designs linear-phase FIR filters that are optimal in terms of worst-case (minimax) error. Typically, we would like to have the shortest-length filter achieving these specifications. Figure Figure 2.8 illustrates the amplitude frequency response of such a filter.

Figure 2.8.

The black boxes on the left and right are the passbands, the black boxes in the middle represent the stop band, and the space between the boxes are the transition bands. Note that overshoots may be allowed in the transition bands.

Exercise 4.

Must there be a transition band?

Yes, when the desired response is discontinuous. Since the frequency response of a finite-length filter must be continuous, without a transition band the worst-case error could be no less than half the discontinuity.

Formal Statement of the L-∞ (Minimax) Design Problem

[redacted]

[redacted]

[redacted]

For a given filter length (M) and type (odd length, symmetric, linear phase, for example), and a relative error weighting function $W(\omega)$, find the filter coefficients minimizing the maximum error

where $E(\omega)= W(\omega)(Hd(\omega)- H(\omega))$ and F is a compact subset of

$\omega\in[0, \pi]$ (*i.e.* , all ω in the passbands and stop bands).

Typically, we would often rather specify $\| E(\omega)\|_\infty \leq \delta$ and minimize over M and **h**; however, the design techniques minimize δ for a given M. One then repeats the design procedure for different M until the minimum M satisfying the requirements is found.

We will discuss in detail the design only of odd-length symmetric linear-phase FIR filters. Even-length and anti-symmetric linear phase FIR filters are essentially the same except for a slightly different implicit weighting function. For arbitrary phase, exactly optimal design procedures have only recently been developed (1990).

Outline of L-∞ Filter Design

The Parks-McClellan method adopts an indirect method for finding the minimax-optimal filter coefficients.

1. Using results from Approximation Theory, simple conditions for determining whether a given filter is $L\infty$ (minimax) optimal are found.

2. An iterative method for finding a filter which satisfies these conditions (and which is thus

optimal) is developed.

That is, the $L\infty$ filter design problem is actually solved **indirectly**.

Conditions for L-∞ Optimality of a Linear-phase FIR Filter

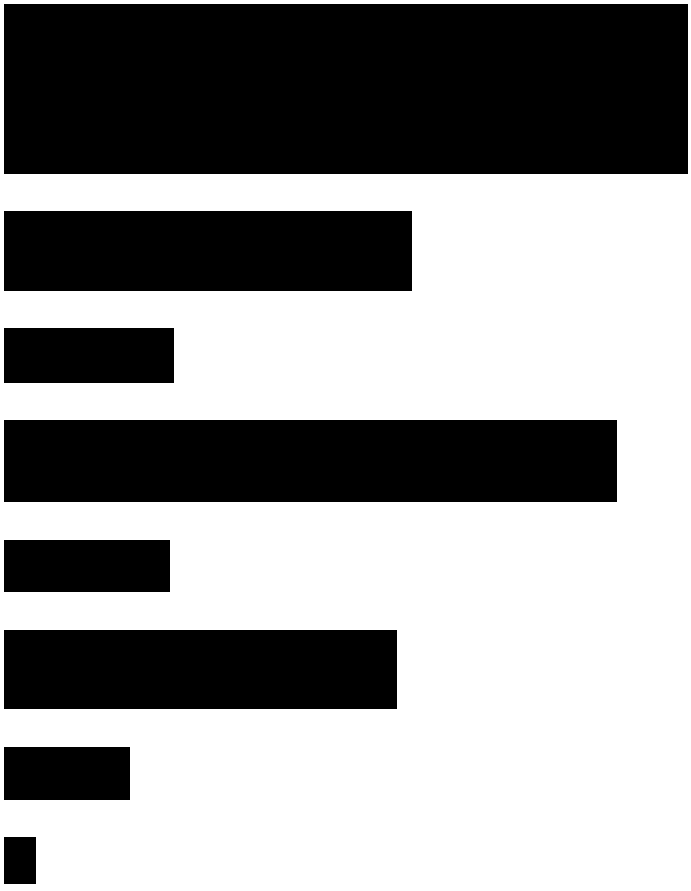
All conditions are based on Chebyshev's "Alternation Theorem," a mathematical fact from

polynomial approximation theory.

Alternation Theorem

Let F be a compact subset on the real axis x, and let $P(x)$ be and L th-order polynomial

Also, let $D(x)$ be a desired function of x that is continuous on F, and $W(x)$ a positive, continuous weighting function on F. Define the error $E(x)$ on F as $E(x)= W(x)(D(x)- P(x))$ and A necessary and sufficient condition that $P(x)$ is the unique L th-order



polynomial minimizing $\| E(x)\|\infty$ is that $E(x)$ exhibits **at least** $L+2$ "alternations;" that is, there

must exist at least $L+2$ values of x, $xk \in F$, $k=[0, 1, \ldots, L+1]$, such that $x\ 0< x\ 1< \ldots < x\ L + 2$ and such

that $E(xk)=-(E(x\ k + 1))=\pm(\| E\|\infty)$

Exercise 5.

What does this have to do with linear-phase filter design?

It's the same problem! To show that, consider an odd-length, symmetric linear phase filter.

(2.8)

(2.9)

Where

.

Using trigonometric identities (such as $\cos(\boldsymbol{n\alpha})=2\cos((n-1)\alpha)\cos(\alpha)-\cos((n-2)\alpha)$), we can rewrite $A(\omega)$ as

where the αk are related to the $h(n)$ by a

linear transformation. Now, let $x=\cos(\omega)$. This is a one-to-one mapping from $x\in[-1, 1]$ onto $\omega\in[0, \pi]$. Thus $A(\omega)$ is an L th-order polynomial in $x=\cos(\omega)$!

Implication

The alternation theorem holds for the $L\infty$ filter design problem, too!

Therefore, to determine whether or not a length- M, odd-length, symmetric linear-phase filter is optimal in an $L\infty$ sense, simply count the alternations in $E(\omega)= W(\omega)(Ad(\omega)- A(\omega))$ in the pass and stop bands. If there are

or more alternations, $h(n)$, $0\leq n\leq M-1$ is the optimal filter!

Optimality Conditions for Even-length Symmetric Linear-phase Filters

For M even,

where

Using the trigonometric identity

$\cos(\alpha+\beta)=\cos(\alpha-\beta)+2\cos(\alpha)\cos(\beta)$ to pull out the term and then using the **other trig**

[redacted]

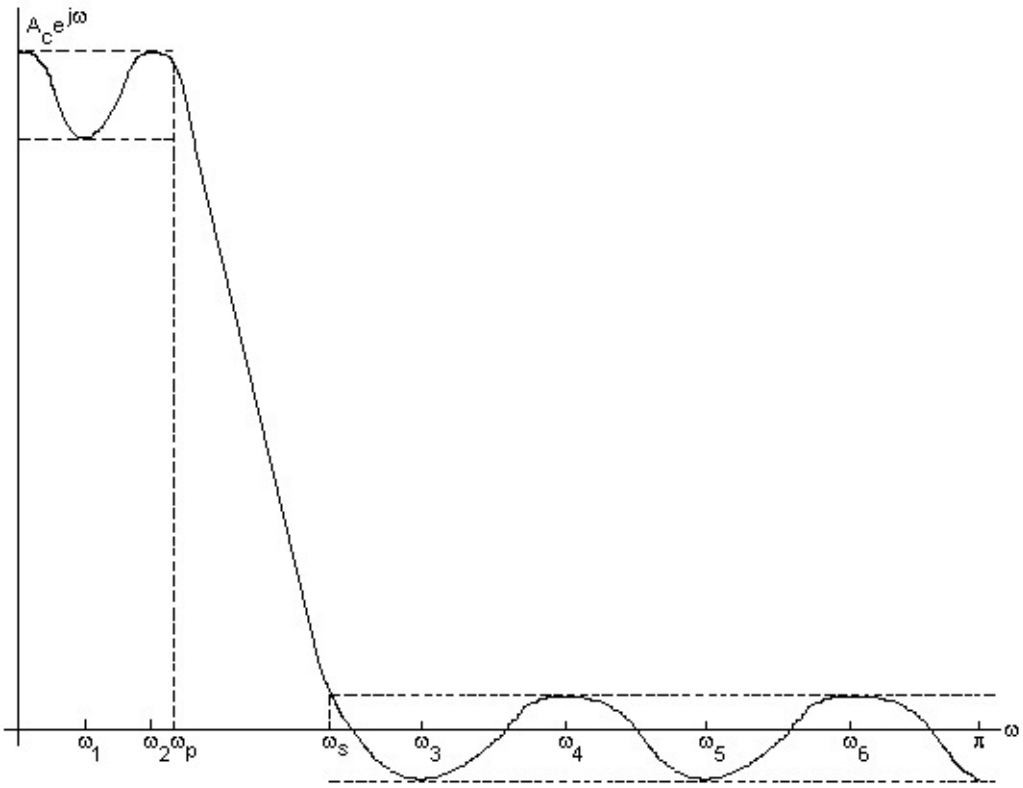
[redacted]

[redacted]

[redacted]

[redacted]

[redacted]



identities, it can be shown that $A(\omega)$ can be written as

Again, this is a

polynomial in $x=\cos(\omega)$, except for a weighting function out in front.

()

which implies

'

()

$E(x)= W'(x)(Ad(x)- P(x))$

where

and

Again, this is a polynomial approximation

problem, so the alternation theorem holds. If $E(\omega)$ has at least

alternations, the even-

length symmetric filter is optimal in an $L\infty$ sense.

The prototypical filter design problem:

See Figure 2.9.

Figure 2.9.

L-∞ Optimal Lowpass Filter Design Lemma

[redacted]

1. The maximum possible number of alternations for a lowpass filter is $L+3$: The proof is that

the extrema of a polynomial occur only where the derivative is zero:

. Since $P'(x)$ is an

($L-1$) th-order polynomial, it can have at **most** $L-1$ zeros. **However**, the mapping

$x=\cos(\omega)$ implies that

at $\omega=0$ and $\omega=\pi$, for two more possible alternation points.

Finally, the band edges can also be alternations, for a total of $L-1+2+2=L+3$ possible

alternations.

2. There must be an alternation at either $\omega=0$ or $\omega=\pi$.

3. Alternations must occur at ωp and ωs . See Figure 2.9.

4. The filter must be equiripple except at possibly ω=0 or ω= π . Again see Figure 2.9.

The alternation theorem doesn't directly suggest a method for computing the optimal filter. It simply tells us how to recognize that a filter **is** optimal, or **isn't** optimal. What we need is an intelligent way of guessing the optimal filter coefficients.

In matrix form, these L+2 simultaneous equations become

or

So, for the given set of

L+2 extremal frequencies, we can solve for **h** and δ via

. Using the FFT, we can

compute $A(\omega)$ of $h(n)$, on a dense set of frequencies. If the old ωk are, in fact the extremal locations of $A(\omega)$, then the alternation theorem is satisfied and $h(n)$ is **optimal**. If not, repeat the process with the new extremal locations.

Computational Cost

$O(L\ 3)$ for the matrix inverse and $N \log 2\ N$ for the FFT ($N \geq 32\ L$, typically), **per iteration**!

This method is expensive computationally due to the matrix inverse.

A more efficient variation of this method was developed by Parks and McClellan (1972), and is based on the Remez exchange algorithm. To understand the Remez exchange algorithm, we first need to understand **Lagrange Interpoloation**.

[redacted]

[redacted]

[redacted]

Now $A(\omega)$ is an L th-order polynomial in x=cos(ω) , so Lagrange interpolation can be used to **exactly** compute $A(\omega)$ from L+1 samples of $A(\omega k)$, k=[0, 1, 2, ..., L] .

Thus, given a set of extremal frequencies and knowing δ , samples of the amplitude response

$A(\omega)$ can be computed **directly** from the

()

without solving for the filter coefficients!

This leads to computational savings!

Note that Equation is a set of $L+2$ simultaneous equations, which can be solved for δ to obtain (Rabiner, 1975)

()

where

The result is the Parks-McClellan FIR filter design method, which is

simply an application of the Remez exchange algorithm to the filter design problem. See

Figure 2.10.

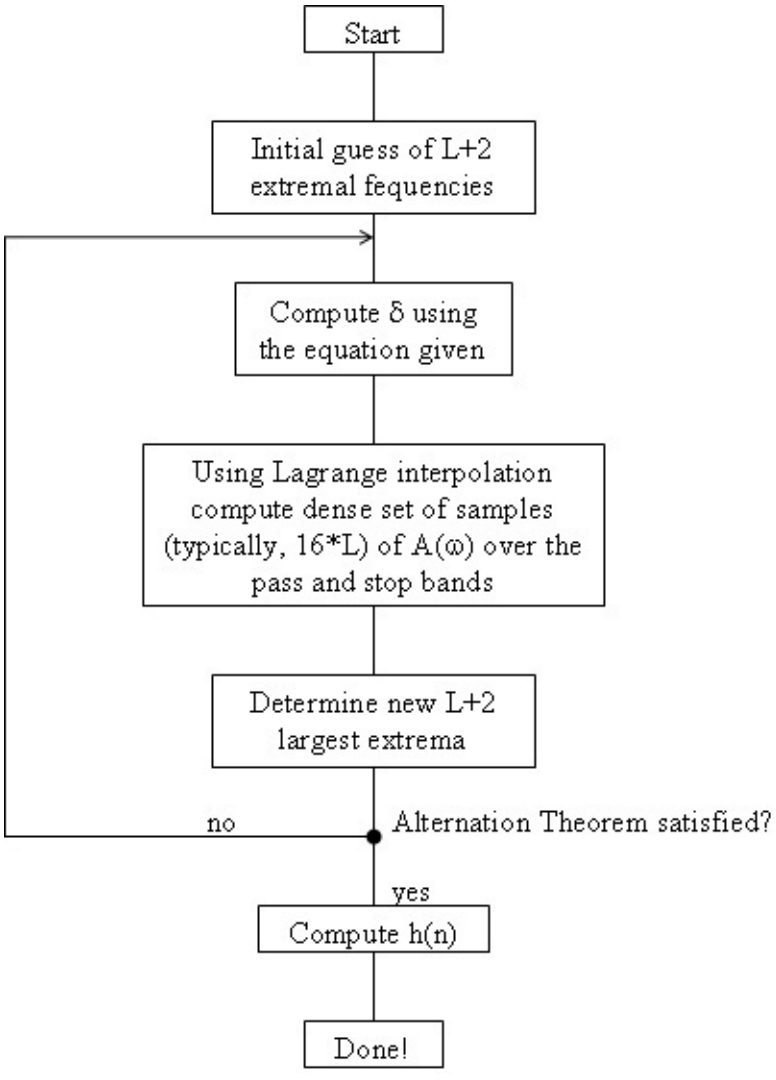


Figure 2.10.

The initial guess of extremal frequencies is usually equally spaced in the band. Computing δ costs $O(L^2)$. Using Lagrange interpolation costs $O(16L) \approx O(16L^2)$. Computing $h(n)$ costs $O(L^3)$, but it is only done once!

The cost per iteration is $O(16L^2)$, as opposed to $O(L^3)$; much more efficient for large L. Can also interpolate to DFT sample frequencies, take inverse FFT to get corresponding filter coefficients,

and zeropad and take longer FFT to efficiently interpolate.

2.3. IIR Filter Design

Overview of IIR Filter Design*

IIR Filter

[redacted]

IIR Filter Design Problem

Choose $\{a_i\}, \{b_i\}$ to **best** approximate some desired $|H_d(w)|$ or, (occasionally), $H_d(w)$.

As before, different design techniques will be developed for different approximation criteria.

Outline of IIR Filter Design Material

Bilinear Transform: Maps $\|L\|_\infty$ optimal (and other) **analog** filter designs to $\|L\|_\infty$ optimal

digital IIR filter designs.

Prony's Method: Quasi-$\|L\|_2$ optimal method for time-domain fitting of a desired impulse

response (*ad hoc*).

Lp Optimal Design: $\|L\|_p$ optimal filter design ($1 < p < \infty$) using non-linear optimization

techniques.

Comments on IIR Filter Design Methods

The bilinear transform method is used to design "typical" $\| L\|_\infty$ magnitude optimal filters. The $\| L\|_p$ optimization procedures are used to design filters for which classical analog prototype solutions don't exist. The program by Deczky (*DSP Programs Book*, IEEE Press) is widely used. Prony/Linear Prediction techniques are used often to obtain initial guesses, and are almost exclusively used in data modeling, system identification, and most applications involving the fitting of real data (for example, the impulse response of an unknown filter).

Prototype Analog Filter Design*

Analog Filter Design

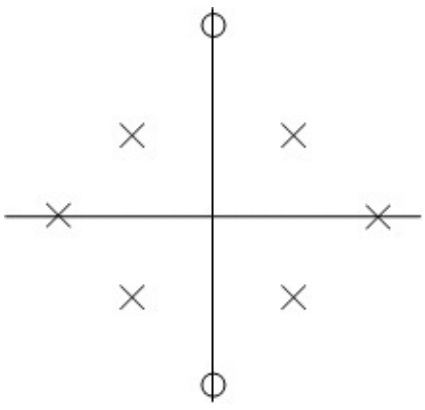
Laplace transform:

Note that the **continuous-time Fourier transform** is

$H(j\omega)$ (the Laplace transform evaluated on the imaginary axis).

Since the early 1900's, there has been a lot of research on designing analog filters of the form

A **causal** IIR filter **cannot** have linear phase (no possible symmetry



point), and design work for analog filters has concentrated on designing filters with equiriplle ($\| L\|_\infty$) **magnitude** responses. These design problems have been solved. We will not concern ourselves here with the design of the analog prototype filters, only with how these designs are mapped to discrete-time while preserving optimality.

An analog filter with **real** coefficients must have a magnitude response of the form

$(| H(\lambda)|)2= B(\lambda 2)$

()

Let $s= ⅈ\lambda$, note that the poles and zeros of $B(–(s 2))$ are symmetric around **both** the real and imaginary axes: that is, a pole at p 1 implies poles at p 1 , , $– p$ 1 , and

, as seen in Figure 2.11.

Figure 2.11. s-plane

Recall that an analog filter is stable and causal if all the poles are in the left half-plane, LHP, and is **minimum phase** if all zeros and poles are in the LHP.

$s= ⅈ\lambda$:

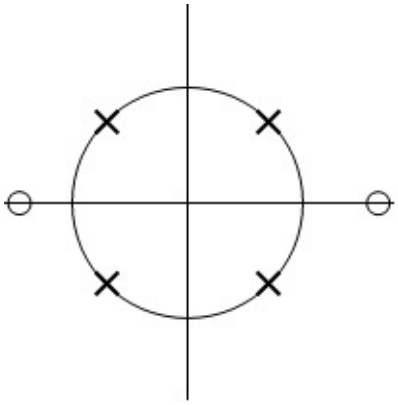
we can factor $B(–(s 2))$ into $H(s) H(– s)$, where

$H(s)$ has the left half plane poles and zeros, and $H(– s)$ has the RHP poles and zeros.

$(| H(s)|)2= H(s) H(– s)$ for $s= ⅈ\lambda$, so $H(s)$ has the magnitude response $B(\lambda 2)$. The trick to analog filter design is to design a good $B(\lambda 2)$, then factor this to obtain a filter with that **magnitude** response.

The traditional analog filter designs all take the form

, where F is a rational function in $\lambda 2$.



Example 2.2.

where

.

Roots of 1+ sN are N points equally spaced around the unit circle (Figure 2.12).

Figure 2.12.

Take $H(s)$=LHP factors:

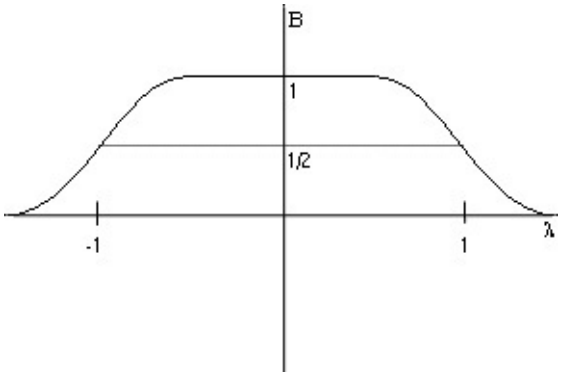
Traditional Filter Designs

Butterworth

Remember this for homework and rest problems!

"Maximally smooth" at $\lambda=0$ and $\lambda=\infty$ (maximum possible number of zero derivatives).

Figure 2.13. $B(\lambda 2)=(|H(\lambda)|)2$



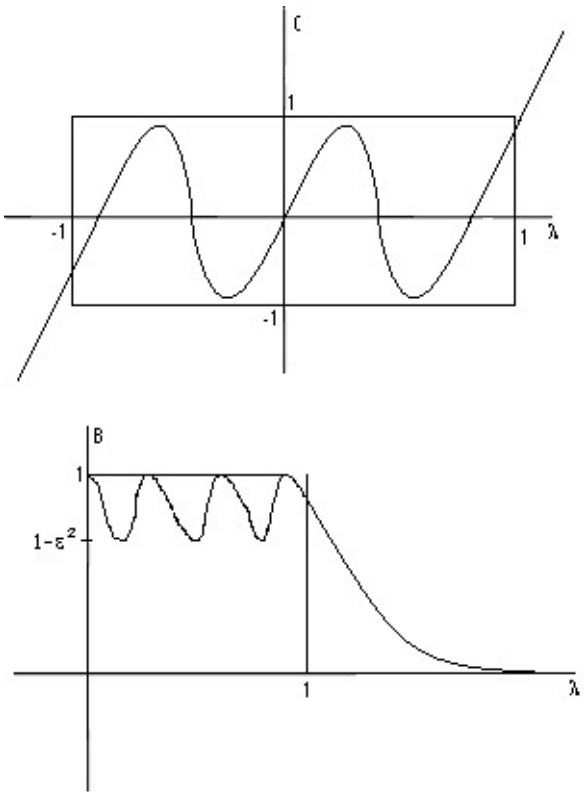


Figure 2.13.

Chebyshev

where *C* 2

M (*λ*) is an *M* th order Chebyshev polynomial. Figure 2.14.

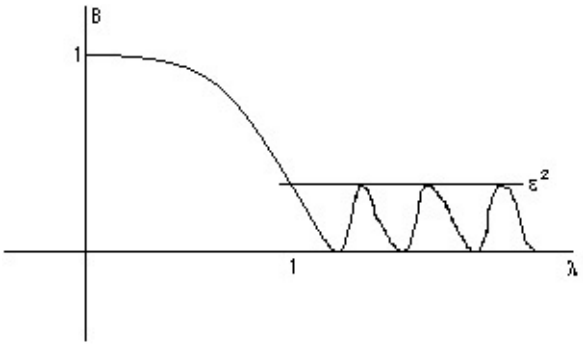
(a)

(b)

Figure 2.14.

Inverse Chebyshev

Figure 2.15.



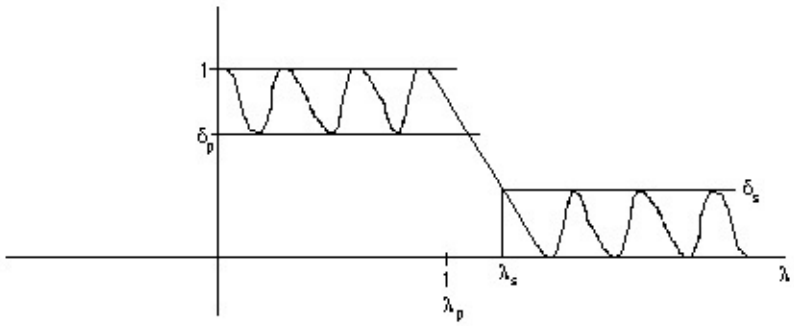


Figure 2.15.

Elliptic Function Filter (Cauer Filter)

where *JM* is the "Jacobi Elliptic Function." Figure 2.16.

Figure 2.16.

The Cauer filter is $\|\mathbf{L}\|\infty$ optimum in the sense that for a given *M*, *δp*, *δs*, and *λp*, the transition bandwidth is smallest.

That is, it is $\|\mathbf{L}\|\infty$ optimal.

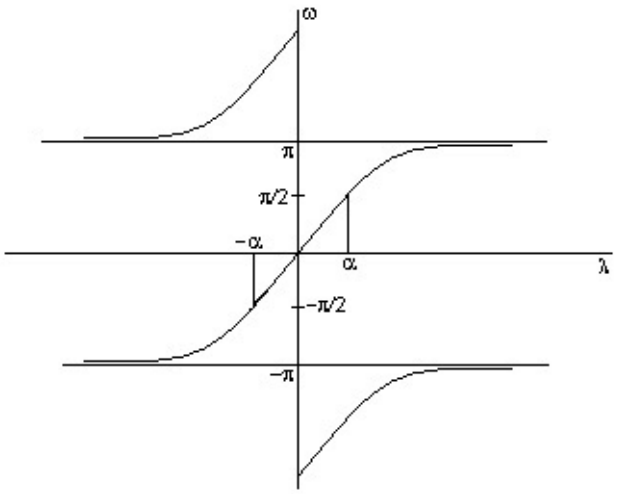
IIR Digital Filter Design via the Bilinear Transform*

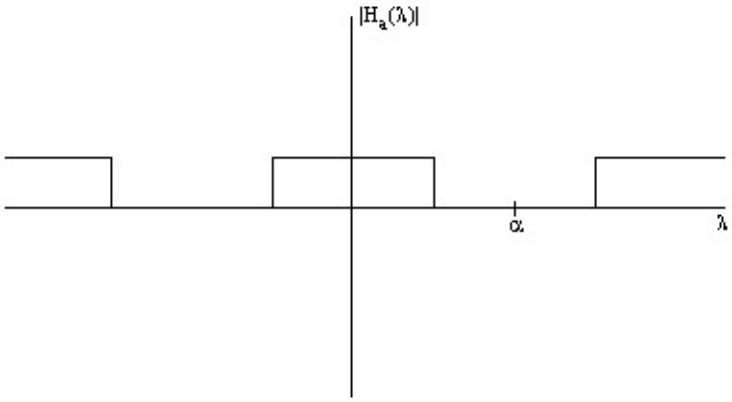
A **bilinear transform** maps an analog filter *Ha*(*s*) to a discrete-time filter *H*(*z*) of the same order.

If only we could somehow map these optimal analog filter designs to the digital world while preserving the magnitude response characteristics, we could make use of the already-existing body of knowledge concerning optimal analog filter design.

Bilinear Transformation

The Bilinear Transform is a nonlinear ($\mathbb{C} \rightarrow \mathbb{C}$) mapping that maps a function of the complex variable *s* to a function of a complex variable *z* . This map has the property that the LHP in *s* (





Re(*s*)<0) maps to the interior of the unit circle in *z* , and the $j\lambda = s$ axis maps to the unit circle $e^{j\omega}$ in *z* .

Bilinear transform:

Figure 2.17

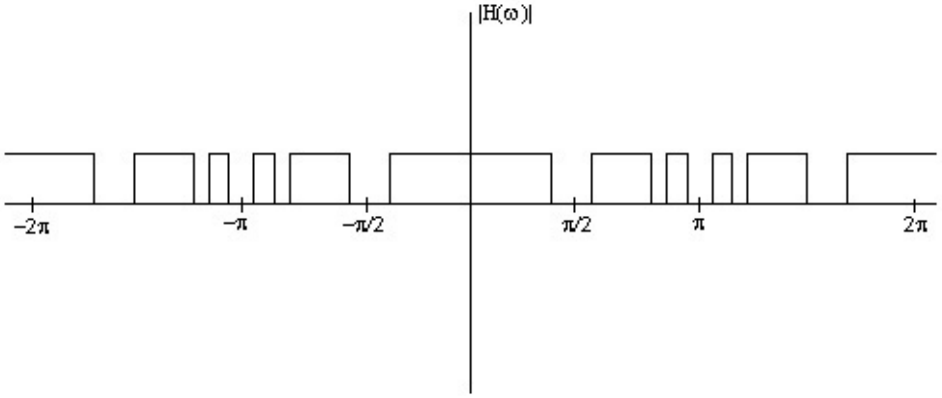
Figure 2.17.

The magnitude response doesn't change in the mapping from λ to ω , it is simply warped

nonlinearly according to

, Figure 2.18.

(a)



[redacted]

[redacted]

(b)

Figure 2.18.

The first image implies the second one.

This mapping preserves $\| L\|_\infty$ errors in (warped) frequency bands. Thus optimal Cauer ($\| L\|_\infty$) filters in the analog realm can be mapped to $\| L\|_\infty$ optimal discrete-time IIR filters using the bilinear transform! This is how IIR filters with $\| L\|_\infty$ optimal magnitude responses are designed.

The parameter α provides one degree of freedom which can be used to map a single λ 0 to any desired ω 0 :

or

This can be used, for example, to map the pass-band edge

of a lowpass analog prototype filter to any desired pass-band edge in ω . Often, analog prototype filters will be designed with λ=1 as a band edge, and α will be used to locate the band edge in ω . Thus an M th order optimal lowpass analog filter prototype can be used to design **any** M th order discrete-time lowpass IIR filter with the same ripple specifications.

Prewarping

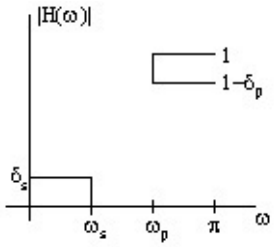
Given specifications on the frequency response of an IIR filter to be designed, map these to

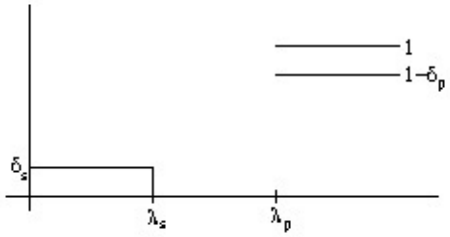
specifications in the analog frequency domain which are equivalent. Then a satisfactory analog prototype can be designed which, when transformed to discrete-time using the bilinear transformation, will meet the specifications.

Example 2.3.

The goal is to design a high-pass filter, $\omega s = \omega s$, $\omega p = \omega p$, $\delta s = \delta s$, $\delta p = \delta p$; pick up some $\alpha = \alpha\ 0$.

[redacted]





[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

The goal is to design a high-pass filter, $\omega s = \omega s$, $\omega p = \omega p$, $\delta s = \delta s$, $\delta p = \delta p$; pick up some $\alpha = \alpha\ 0$.

In Figure 2.19 the δi remain the same and the band edges are mapped by

.

(a)

(b)

Figure 2.19.

Where

and

.

Impulse-Invariant Design*

Pre-classical, adhoc-but-easy method of converting an analog prototype filter to a digital IIR

filter. Does not preserve any optimality.

Impulse invariance means that digital filter impulse response exactly equals samples of the analog

prototype impulse response: $h(n) = ha(\boldsymbol{nT})$ How is this done?

The impulse response of a causal, stable analog filter is simply a sum of decaying exponentials:

which implies

$ha(t) = (A\,1\,e^{s}\,1\,t + A\,2\,e^{s}\,2\,t + \ldots + Ape^{spt})\,u(t)$ For impulse invariance, we desire $h(n) = ha(\boldsymbol{nT}) = (A\,1\,e^{s}\,1\,nT + A\,2\,e^{s}\,2\,nT + \ldots + Ape^{spnT})\,u(n)$ Since where $|z| > |e^{skT}|$, and

where

.

This technique is used occasionally in digital simulations of analog filters.

Exercise 6.

What is the main problem/drawback with this design technique?

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

What is the main problem/drawback with this design technique?

Since it samples the non-bandlimited impulse response of the analog prototype filter, the frequency response **aliases**. This distorts the original analog frequency and destroys any optimal frequency properties in the resulting digital filter.

Digital-to-Digital Frequency Transformations*

Given a prototype **digital** filter design, transformations similar to the bilinear transform can also be developed.

Requirements on such a mapping $z\text{-}1 = g(z\text{-}1)$:

1. points inside the unit circle stay inside the unit circle (condition to preserve stability)

2. unit circle is mapped to itself (preserves frequency response)

This condition implies that $e^{-(i\omega 1)} = g(e^{-(i\boldsymbol{\omega})}) = |g(\omega)| e^{i\angle(g(\omega))}$ requires that $|g(e^{-(i\boldsymbol{\omega})})| = 1$ on the unit circle!

Thus we require an **all-pass** transformation:

where $|\alpha K| < 1$, which is required to

satisfy **this condition.**

Example 2.4. Lowpass-to-Lowpass

which maps original filter with a cutoff at ω

$'$

c to a new filter with cutoff ωc ,

Example 2.5. Lowpass-to-Highpass

which maps original filter with a cutoff at ωc to a frequency reversed filter with

cutoff ω'

c ,

(Interesting and occasionally useful!)

Prony's Method*

Prony's Method is a quasi-least-squares time-domain IIR filter design method.

First, assume $H(z)$ is an "all-pole" system:

()

and

where $h(n)=0$, $n<0$ for a causal system.

For $h=0$, $h(0)= b\ 0$.

Let's attempt to fit a desired impulse response (let it be **causal**, although one can extend this technique when it isn't) $hd(n)$.

A true least-squares solution would attempt to minimize

where $H(z)$ takes the

form in Equation. This is a difficult non-linear optimization problem which is known to be plagued by local minima in the error surface. So instead of solving this difficult non-linear problem, we solve the **deterministic linear prediction** problem, which is related to, **but not the same as**, the true least-squares optimization.

The deterministic linear prediction problem is a **linear** least-squares optimization, which is easy

to solve, but it minimizes the **prediction** error, not the (|desired−actual|)2 response error.

Notice that for $n>0$, with the all-pole filter

()

the right hand side of **this equation** is a **linear predictor** of $h(\ n)$ in terms of the M previous samples of $h(\ n)$.

For the desired reponse $hd(\ n)$, one can choose the recursive filter coefficients ak to minimize the

squared prediction error

where, in practice, the ∞ is replaced by an

N .

In matrix form, that's

or

The optimal solution is

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Now suppose $H(z)$ is an M th -order IIR (ARMA) system,

or

()

For $n > M$, this is just like the all-pole case, so we can solve for the best predictor coefficients as

before:

or

and

Having

determined the a 's, we can use them in Equation to obtain the bn 's:

where

$hd(n-k)=0$ for $n-k<0$.

For $N=2M$,

is square, and we can solve **exactly** for the ak 's with no error. The bk 's are also

chosen such that there is no error in the first $M+1$ samples of $h(n)$. Thus for $N=2M$, the first $2M+1$ points of $h(n)$ **exactly equal** $hd(n)$. This is called **Prony's Method**. Baron de Prony invented this in 1795.

For $N>2M$, $hd(n)=h(n)$ for $0\le n\le M$, the prediction error is minimized for $M+1<n\le N$, and whatever for $n\ge N+1$. This is called the **Extended Prony Method**.

One might prefer a method which tries to minimize an overall error with the numerator

coefficients, rather than just using them to exactly fit $hd(0)$ to $hd(M)$.

Shank's Method

1. Assume an all-pole model and fit $hd(n)$ by minimizing the prediction error $1\le n\le N$.

2. Compute $v(n)$, the impulse response of this all-pole filter.

3. Design an all-zero (MA, FIR) filter which fits $v(n)*hz(n)\approx hd(n)$ optimally in a least-squares sense (Figure 2.20).

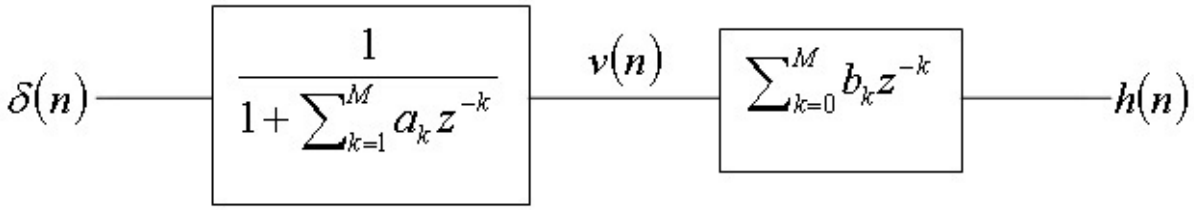


Figure 2.20.

Here, $h(n) \approx hd(n)$.

The final IIR filter is the cascade of the all-pole and all-zero filter.

This is is solved by

or in matrix form

Which has solution:

Notice that none of these methods solve the true least-squares problem:

which is a difficult non-linear optimization problem. The true least-squares problem can be

written as:

since the impulse response of an IIR filter is a sum of

exponentials, and non-linear optimization is then used to solve for the αi and βi .

Linear Prediction*

Recall that for the all-pole design problem, we had the overdetermined set of linear equations:

with solution

Let's look more closely at H H

$d\ Hd = R$. $r\ i\ j$ is related to the **correlation** of hd with itself:

Note also that:

where

so this takes the

form

, or $\boldsymbol{Ra} = -\mathbf{r}$, where R is $M \times M$, $\mathbf{a}$ is $M \times 1$, and $\mathbf{r}$ is also $M \times 1$.

Except for the changing endpoints of the sum, $r\ i\ j \approx r(\ i - j) = r(\ j - i)$. If we tweak the problem slightly to make $r\ i\ j = r(\ i - j)$, we get:

The matrix R is **Toeplitz**

(diagonal elements equal), and $\mathbf{a}$ can be solved for with $O(\ M\ 2)$ computations using Levinson's

[redacted]

[redacted]

[redacted]

[redacted]

recursion.

Statistical Linear Prediction

Used very often for forecasting (*e.g.* stock market).

Given a time-series $y(n)$, assumed to be produced by an auto-regressive (AR) (all-pole) system:

where $u(n)$ is a white Gaussian noise sequence which is stationary and has

zero mean.

To determine the model parameters $\{ ak\}$ minimizing the variance of the prediction error, we seek

()

The mean of $y(n)$ is zero.

()

()

Setting Equation equal to zero yields: $\boldsymbol{Ra}=-\mathbf{r}$ These are called the **Yule-Walker** equations. In practice, given samples of a sequence $y(n)$, we estimate $r(n)$ as

which is extremely similar to the deterministic least-squares technique.

Solutions

[redacted]

[redacted]

Chapter 3. The DFT, FFT, and Practical Spectral Analysis

3.1. The Discrete Fourier Transform

DFT Definition and Properties*

DFT

The **discrete Fourier transform (DFT)** is the primary transform used for numerical computation in digital signal processing. It is very widely used for **spectrum analysis**, **fast convolution**, and many other applications. The DFT transforms N discrete-time samples to the same number of discrete frequency samples, and is defined as

()

The DFT is widely used in part because it can be computed very efficiently using **fast Fourier transform (FFT)** algorithms.

IDFT

The inverse DFT (IDFT) transforms N discrete-frequency samples to the same number of discrete-time samples. The IDFT has a form very similar to the DFT,

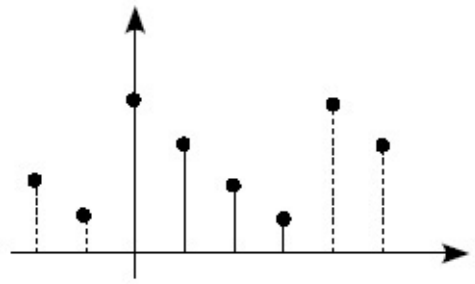
()

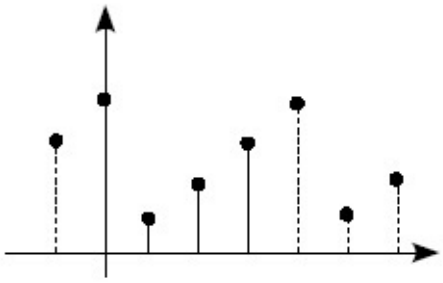
and can thus also be computed efficiently using **FFTs**.

DFT and IDFT properties

Periodicity

Due to the N-sample periodicity of the complex exponential basis functions in the DFT and IDFT, the resulting transforms are also periodic with N samples.





[redacted]

[redacted]

$X(k+N)=X(k)\ x(n)=x(n+N)$

Circular Shift

A shift in time corresponds to a phase shift that is linear in frequency. Because of the periodicity

induced by the DFT and IDFT, the shift is **circular**, or modulo N samples.

The modulus operator p mod N means the **remainder** of p when divided

by N. For example, 9mod5=4 and -1mod5=4

Time Reversal

$(x((-n)\text{mod}\ N)=x((N-n)\text{mod}\ N)\ X((N-k)\text{mod}\ N)=X((-k)\text{mod}\ N))$ Note: time-reversal maps (0 0) , (1 $N-1$) , (2 $N-2$) , etc. as illustrated in the figure below.

(a) Original signal

(b) Time-reversed

Figure 3.1.

Illustration of circular time-reversal

Complex Conjugate

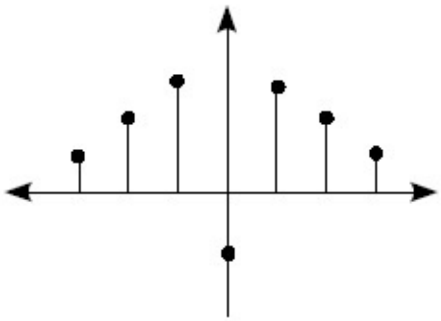
Circular Convolution Property

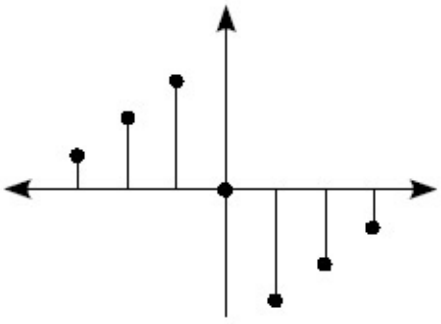
Circular convolution is defined as

Circular convolution of two discrete-time signals corresponds to multiplication of their DFTs:

($x(n)$* $h(n)$ $X(k)$ $H(k)$)

[redacted]





Multiplication Property

A similar property relates multiplication in time to circular convolution in frequency.

Parseval's Theorem

Parseval's theorem relates the energy of a length- N discrete-time signal (or one period) to the energy of its DFT.

Symmetry

The **continuous-time Fourier transform,** the **DTFT,** and **DFT** are all defined as transforms of complex-valued data to complex-valued spectra. However, in practice signals are often real-valued. The DFT of a real-valued discrete-time signal has a special symmetry, in which the real

part of the transform values are **DFT even symmetric** and the imaginary part is **DFT odd**

symmetric, as illustrated in the equation and figure below.

$x(n)$ real

(This implies $X(0)$,

are real-valued.)

<db:title>Real part of X(k) is even</db:title>

(a) Even-symmetry in DFT sense

<db:title>Imaginary part of X(k) is odd</db:title>

(b) Odd-symmetry in DFT sense

Figure 3.2.

DFT symmetry of real-valued signal

3.2. Spectrum Analysis

Spectrum Analysis Using the Discrete Fourier Transform*

Discrete-Time Fourier Transform

The **Discrete-Time Fourier Transform (DTFT)** is the primary theoretical tool for understanding

the frequency content of a discrete-time (sampled) signal. The **DTFT** is defined as

()

The inverse DTFT (IDTFT) is defined by an integral formula, because it operates on a continuous-

frequency DTFT spectrum:

()

The DTFT is very useful for theory and analysis, but is not practical for numerically computing a spectrum digitally, because

1. infinite time samples means

infinite computation

infinite delay

2. The transform is continuous in the discrete-time frequency, ω

For practical computation of the frequency content of real-world signals, the Discrete Fourier Transform (DFT) is used.

Discrete Fourier Transform

The DFT transforms *N* samples of a discrete-time signal to the same number of discrete frequency samples, and is defined as

()

The DFT is invertible by the inverse discrete Fourier transform (IDFT):

()

The **DFT** and **IDFT** are a self-contained, one-to-one transform pair for a length- N discrete-time signal. (That is, the **DFT** is not **merely** an approximation to the **DTFT** as discussed next.) However, the **DFT** is very often used as a practical approximation to the **DTFT**.

Relationships Between DFT and DTFT

DFT and Discrete Fourier Series

The **DFT** gives the discrete-time Fourier series coefficients of a periodic sequence (

$x(n) = x(n+N)$) of period N samples, or

()

as can easily be confirmed by computing the inverse DTFT of the corresponding line spectrum:

(3.1)

The DFT can thus be used to **exactly** compute the relative values of the N line spectral

components of the DTFT of any periodic discrete-time sequence with an integer-length period.

DFT and DTFT of finite-length data

When a discrete-time sequence happens to equal zero for all samples except for those between 0

and $N-1$, the infinite sum in the **DTFT** equation becomes the same as the finite sum from 0 to $N-1$ in the **DFT** equation. By matching the arguments in the exponential terms, we observe that the DFT values **exactly** equal the DTFT for specific DTFT frequencies

. That is, the DFT

computes exact samples of the DTFT at N equally spaced frequencies

, or

[redacted]

DFT as a DTFT approximation

In most cases, the signal is neither exactly periodic nor truly of finite length; in such cases, the DFT of a finite block of N consecutive discrete-time samples does **not** exactly equal samples of the DTFT at specific frequencies. Instead, the **DFT** gives frequency samples of a windowed (truncated) **DTFT**

where

Once

again, $X(k)$ **exactly** equals $X(\omega k)$ a DTFT frequency sample only when $x(n)=0$, $n \notin [0, N-1]$

Relationship between continuous-time FT and DFT

The goal of spectrum analysis is often to determine the frequency content of an analog (continuous-time) signal; very often, as in most modern spectrum analyzers, this is actually accomplished by sampling the analog signal, windowing (truncating) the data, and computing and plotting the magnitude of its DFT. It is thus essential to relate the DFT frequency samples back to the original analog frequency. Assuming that the analog signal is bandlimited and the sampling frequency exceeds twice that limit so that no frequency aliasing occurs, the relationship between the continuous-time Fourier frequency Ω (in radians) and the DTFT frequency ω imposed by sampling is $\omega = \boldsymbol{\Omega T}$ where T is the sampling period. Through the relationship

between the

DTFT frequency ω and the DFT frequency index k, the correspondence between the DFT frequency index and the original analog frequency can be found:

or in terms of analog

frequency f in Hertz (cycles per second rather than radians)

for k in the range k between 0

and . It is important to note that correspond to **negative** frequencies due to the periodicity of

the DTFT and the DFT.

Exercise 1.

In general, will DFT frequency values $X(k)$ **exactly** equal samples of the analog Fourier transform Xa at the corresponding frequencies? That is, will

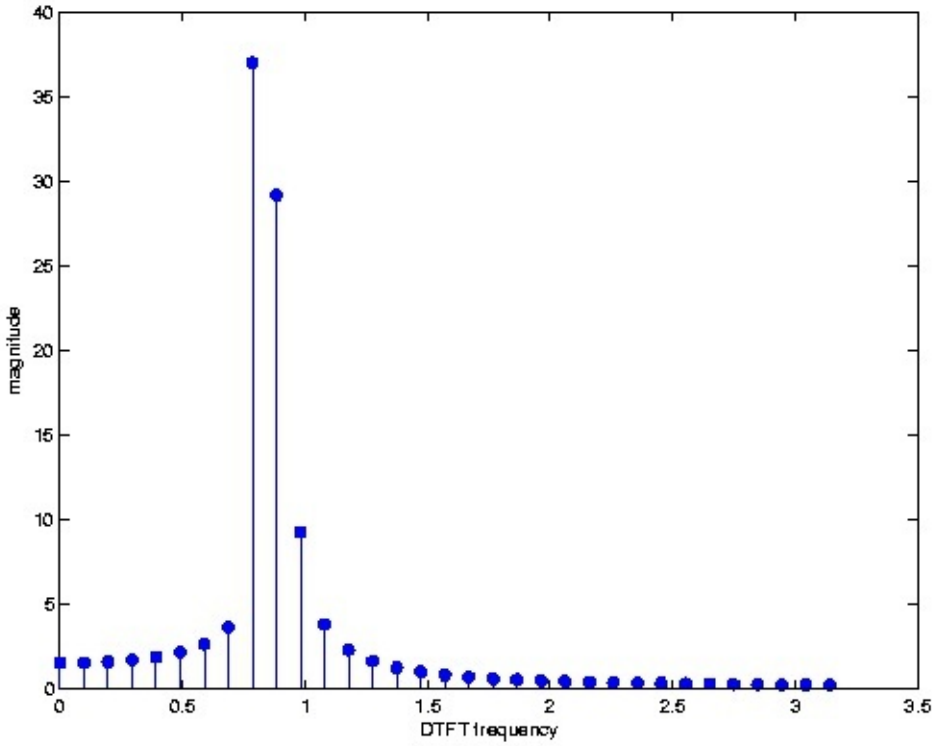
?

In general, **NO**. The DTFT exactly corresponds to the continuous-time Fourier transform only when the signal is bandlimited and sampled at more than twice its highest frequency. The DFT frequency values exactly correspond to frequency samples of the DTFT only when the discrete-time signal is time-limited. However, a bandlimited continuous-time signal cannot be time-limited, so in general these conditions cannot both be satisfied.

It can, however, be true for a small class of analog signals which are not time-limited but happen to exactly equal zero at all **sample times** outside of the interval . The sinc function with a bandwidth equal to the Nyquist frequency and centered at $t= 0$ is an example.

[redacted]

[redacted]



Zero-Padding

If more than *N* equally spaced frequency samples of a length- *N* signal are desired, they can easily be obtained by **zero-padding** the discrete-time signal and computing a DFT of the longer length.

In particular, if ***LN*** **DTFT** samples are desired of a length- *N* sequence, one can compute the length- ***LN*** **DFT** of a length- ***LN*** zero-padded sequence

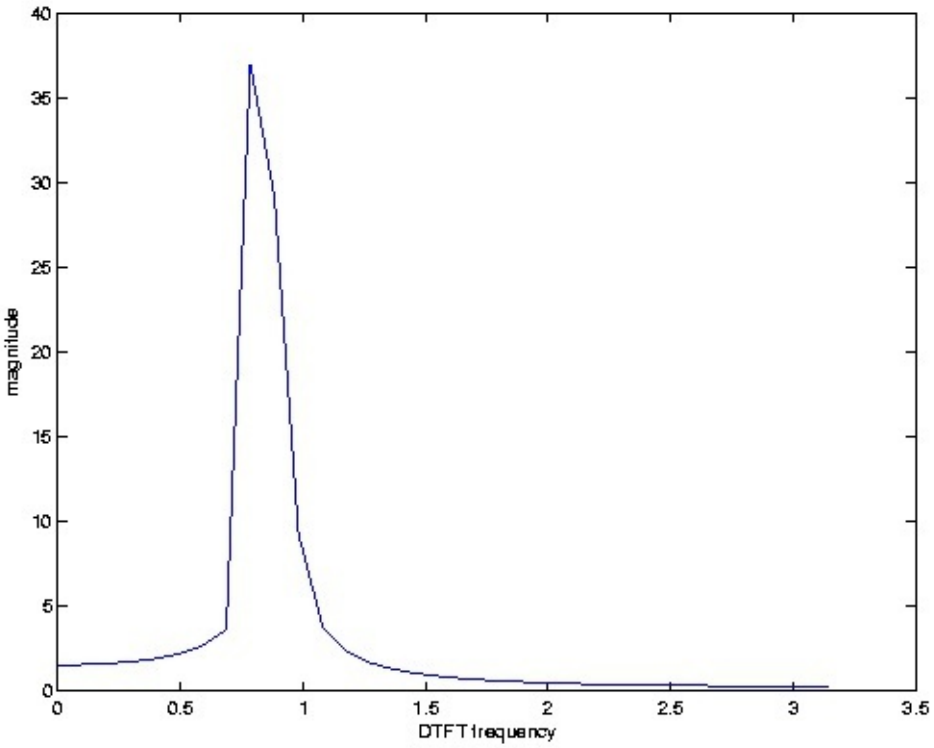
Note that zero-padding **interpolates** the spectrum. One should always zero-pad (by about at least a factor of 4) when using the **DFT** to approximate the **DTFT** to get a clear picture of the **DTFT**. While performing computations on zeros may at first seem inefficient, using **FFT** algorithms, which generally expect the same number of input and output samples, actually makes this approach very efficient.

Figure 3.3 shows the magnitude of the DFT values corresponding to the non-negative frequencies of a real-valued length-64 DFT of a length-64 signal, both in a "stem" format to emphasize the discrete nature of the DFT frequency samples, and as a line plot to emphasize its use as an approximation to the continuous-in-frequency DTFT. From this figure, it appears that the signal has a single dominant frequency component.

<db:title>Stem plot</db:title>

(a)

<db:title>Line Plot</db:title>



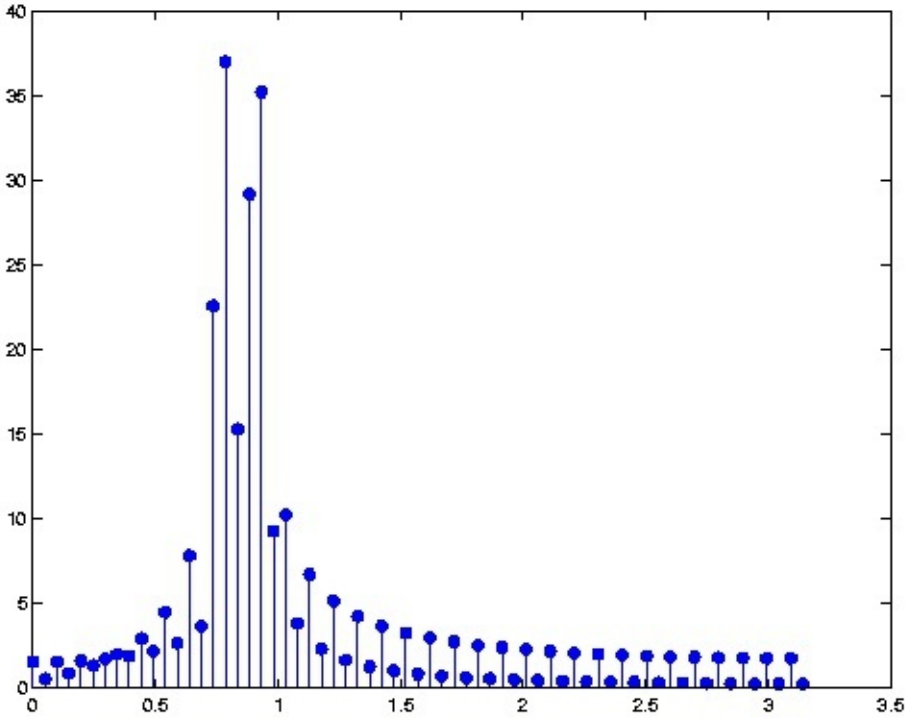
(b)

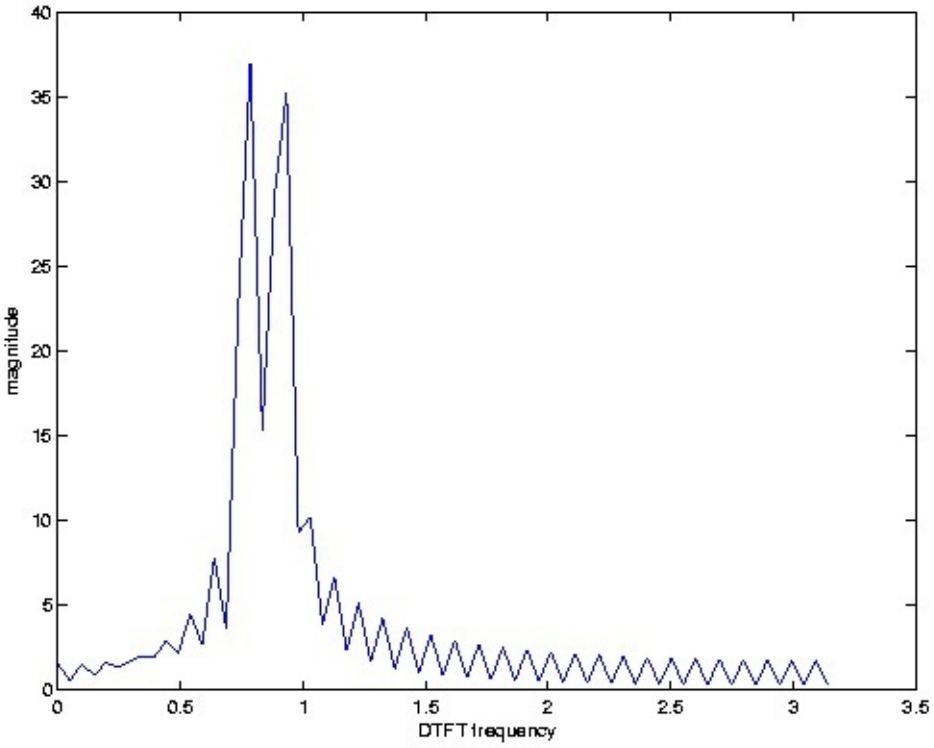
Figure 3.3. Spectrum without zero-padding

Magnitude DFT spectrum of 64 samples of a signal with a length-64 DFT (no zero padding)

Zero-padding by a factor of two by appending 64 zero values to the signal and computing a length-128 DFT yields Figure 3.4. It can now be seen that the signal consists of at least two narrowband frequency components; the gap between them fell between DFT samples in Figure 3.3, resulting in a misleading picture of the signal's spectral content. This is sometimes called the **picket-fence effect**, and is a result of insufficient sampling in frequency. While zero-padding by a factor of two has revealed more structure, it is unclear whether the peak magnitudes are reliably rendered, and the jagged linear interpolation in the line graph does not yet reflect the smooth, continuously-differentiable spectrum of the DTFT of a finite-length truncated signal. Errors in the apparent peak magnitude due to insufficient frequency sampling is sometimes referred to as **scalloping loss**.

<db:title>Stem plot</db:title>





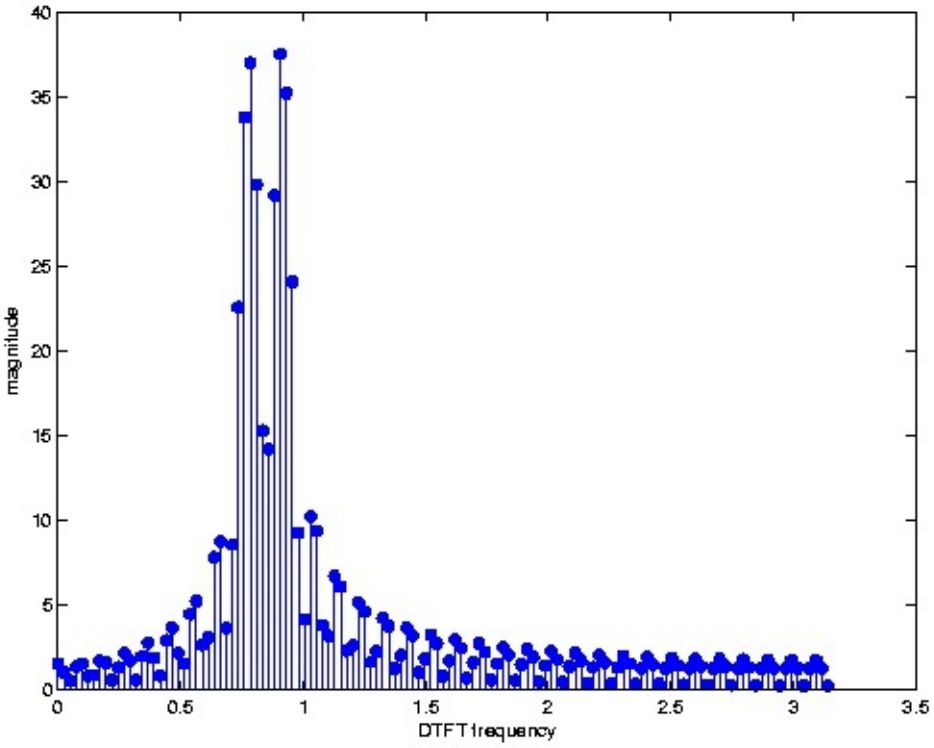
(a)

<db:title>Line Plot</db:title>

(b)

Figure 3.4. Spectrum with factor-of-two zero-padding

Magnitude DFT spectrum of 64 samples of a signal with a length-128 DFT (double-length zero-padding)

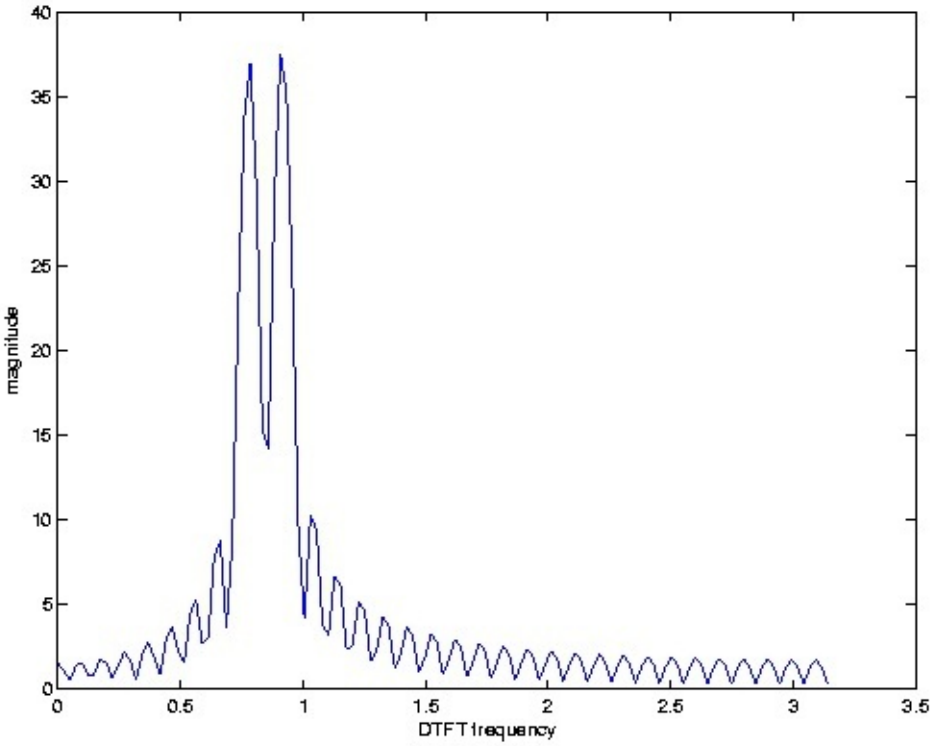


Zero-padding to four times the length of the signal, as shown in Figure 3.5, clearly shows the spectral structure and reveals that the magnitude of the two spectral lines are nearly identical. The line graph is still a bit rough and the peak magnitudes and frequencies may not be precisely captured, but the spectral characteristics of the truncated signal are now clear.

<db:title>Stem plot</db:title>

(a)

<db:title>Line Plot</db:title>



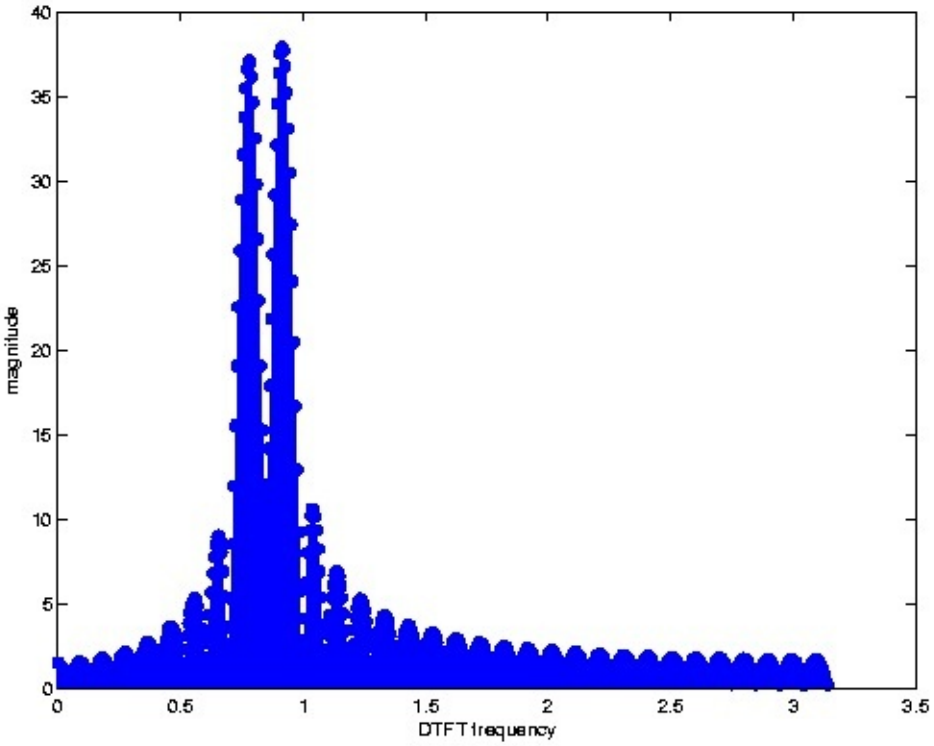
(b)

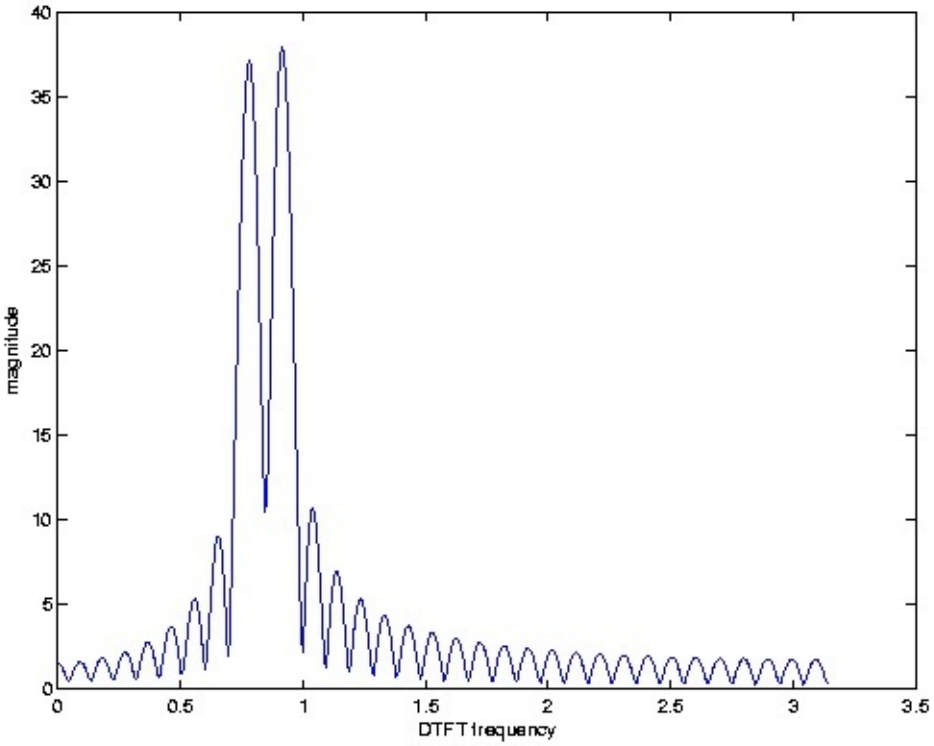
Figure 3.5. Spectrum with factor-of-four zero-padding

Magnitude DFT spectrum of 64 samples of a signal with a length-256 zero-padded DFT (four times zero-padding)

Zero-padding to a length of 1024, as shown in Figure 3.6 yields a spectrum that is smooth and continuous to the resolution of the computer screen, and produces a very accurate rendition of the DTFT of the **truncated** signal.

<db:title>Stem plot</db:title>





(a)

<db:title>Line Plot</db:title>

(b)

Figure 3.6. Spectrum with factor-of-sixteen zero-padding

Magnitude DFT spectrum of 64 samples of a signal with a length-1024 zero-padded DFT. The spectrum now looks smooth and

continuous and reveals all the structure of the DTFT of a truncated signal.

The signal used in this example actually consisted of two pure sinusoids of equal magnitude. The slight difference in magnitude of the two dominant peaks, the breadth of the peaks, and the sinc-like lesser **side lobe** peaks throughout frequency are artifacts of the truncation, or windowing, process used to practically approximate the DFT. These problems and partial solutions to them are discussed in the following section.

Effects of Windowing

Applying the DTFT multiplication property

we find that the

DFT of the windowed (truncated) signal produces samples not of the true (desired) DTFT spectrum $X(\omega)$, but of a **smoothed** verson $X(\omega)^* W(\omega)$. We want this to resemble $X(\omega)$ as closely as possible, so $W(\omega)$ should be as close to an impulse as possible. The **window** $w(n)$ need not be a simple **truncation** (or **rectangle**, or **boxcar**) window; other shapes can also be used as long as they limit the sequence to at most N consecutive nonzero samples. All good windows are impulse-like, and represent various tradeoffs between three criteria:

1. main lobe width: (limits resolution of closely-spaced peaks of equal height)

2. height of first sidelobe: (limits ability to see a small peak near a big peak)

3. slope of sidelobe drop-off: (limits ability to see small peaks further away from a big peak)

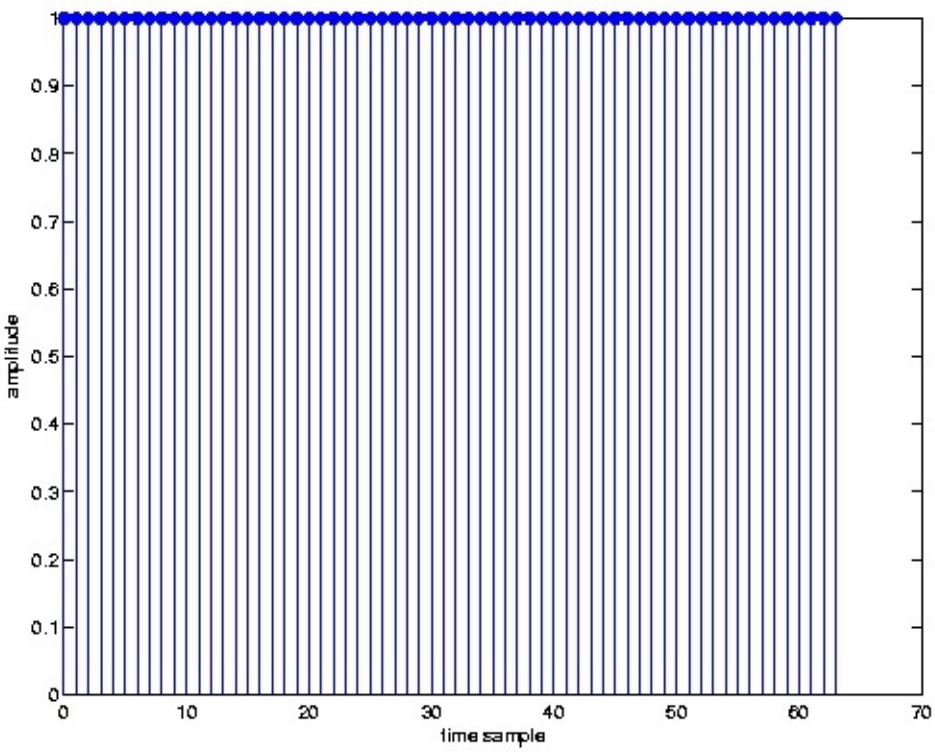
Many different **window functions** have been developed for truncating and shaping a length- N signal segment for spectral analysis. The simple **truncation** window has a periodic sinc DTFT, as shown in Figure 3.7. It has the narrowest main-lobe width,

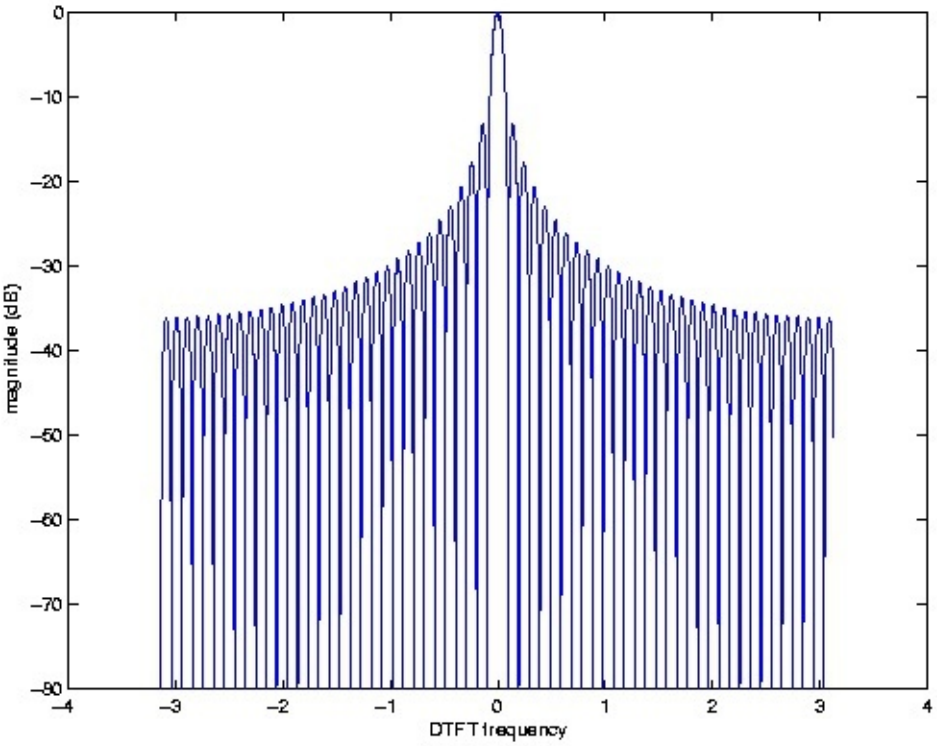
at the -3 dB level and

between

the two zeros surrounding the main lobe, of the common window functions, but also the largest side-lobe peak, at about -13 dB. The side-lobes also taper off relatively slowly.

<db:title>Rectangular window</db:title>





(a)

<db:title>Magnitude of boxcar window spectrum</db:title>

(b)

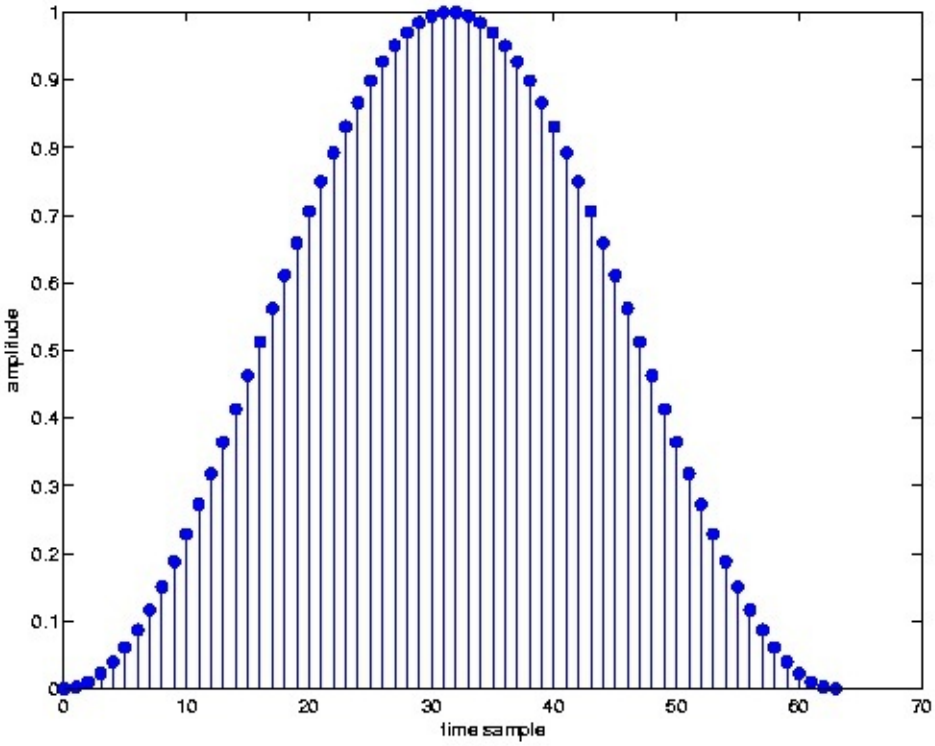
Figure 3.7.

Length-64 truncation (boxcar) window and its magnitude DFT spectrum

[redacted]

[redacted]

[redacted]



The **Hann window** (sometimes also called the **hanning** window), illustrated in Figure 3.8, takes the form

for n between 0 and $N-1$. It has a main-lobe width (about

at the -

3 dB level and

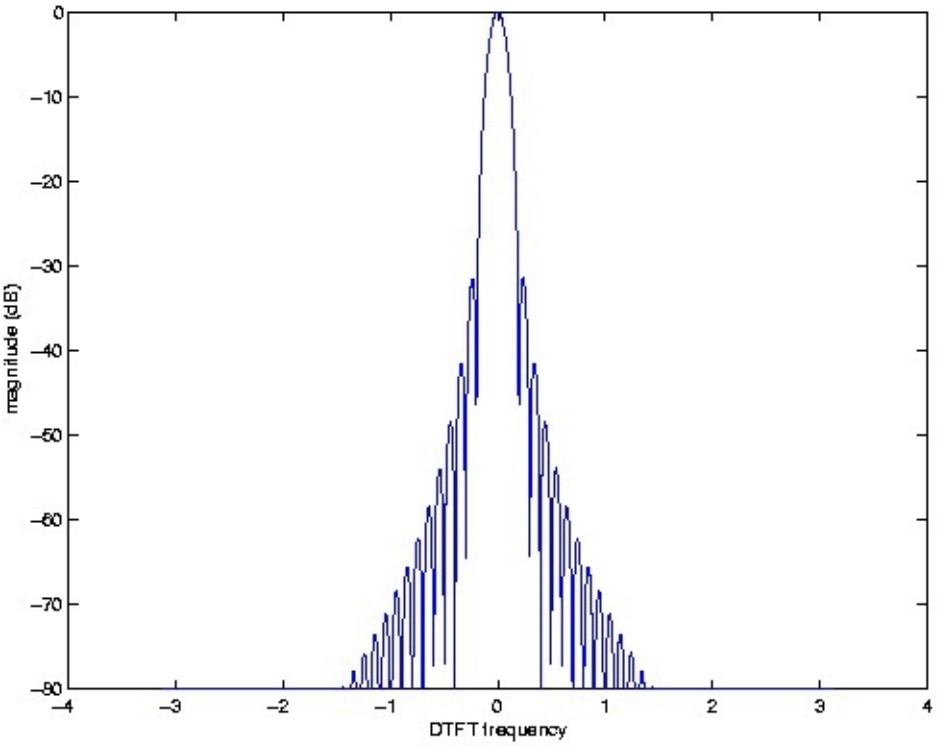
between the two zeros surrounding the main lobe) considerably larger than the

rectangular window, but the largest side-lobe peak is much lower, at about -31.5 dB. The side-lobes also taper off much faster. For a given length, this window is worse than the boxcar window at separating closely-spaced spectral components of similar magnitude, but better for identifying smaller-magnitude components at a greater distance from the larger components.

<db:title>Hann window</db:title>

(a)

<db:title>Magnitude of Hann window spectrum</db:title>



[redacted]

[redacted]

[redacted]

(b)

Figure 3.8.

Length-64 Hann window and its magnitude DFT spectrum

The **Hamming window**, illustrated in Figure 3.9, has a form similar to the Hann window but with slightly different constants:

for n between 0 and $N-1$. Since it is composed

of the same Fourier series harmonics as the Hann window, it has a similar main-lobe width (a bit

less than

at the -3 dB level and

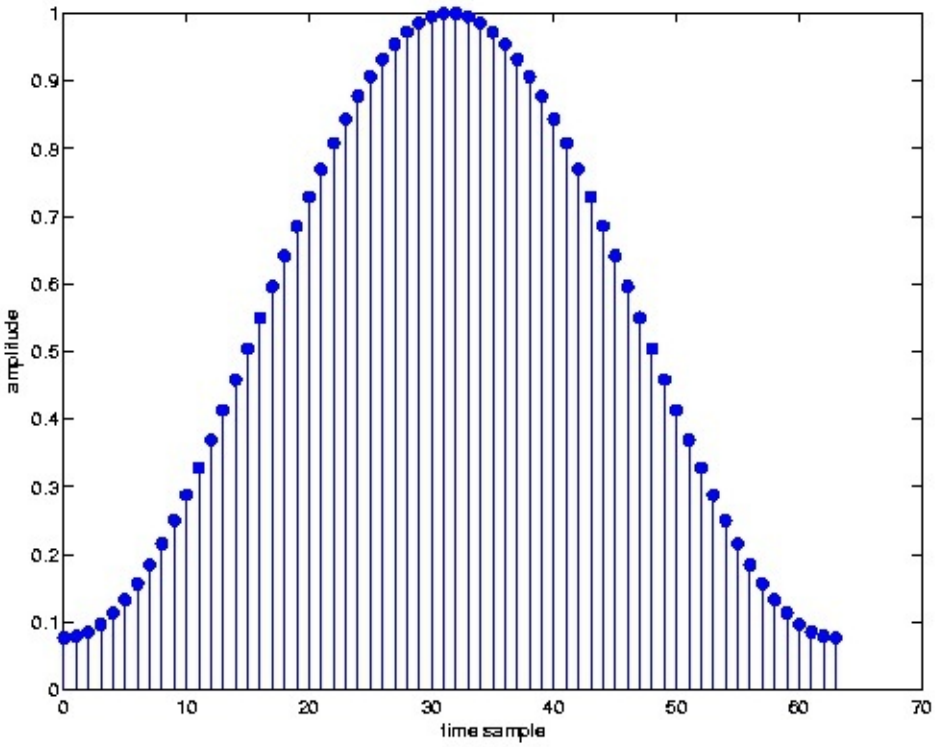
between the two zeros surrounding the main lobe), but the

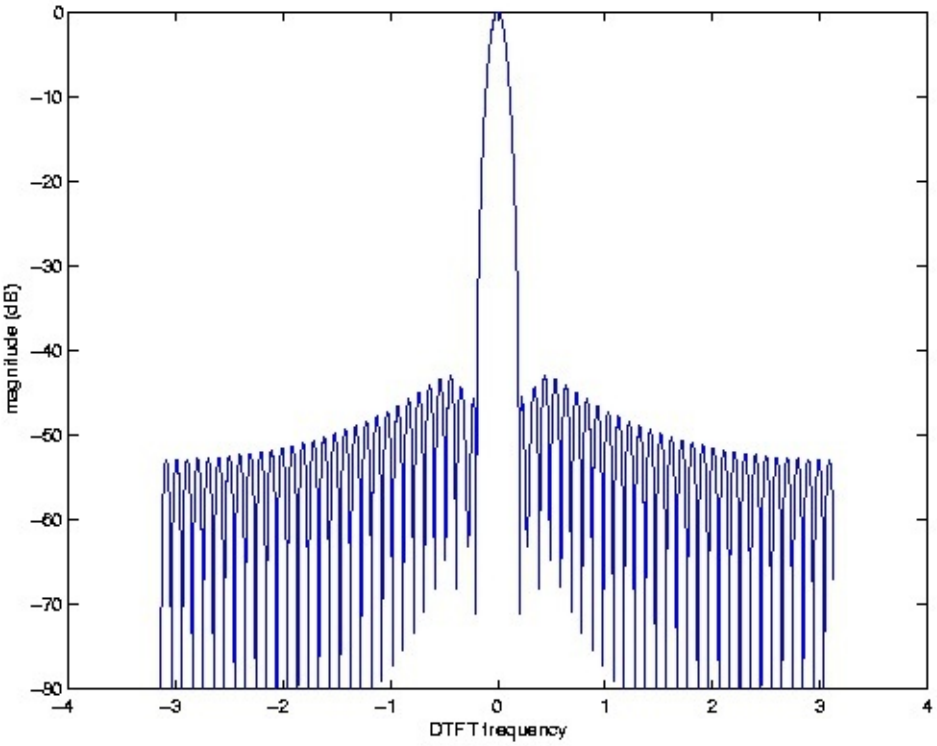
largest side-lobe peak is much lower, at about -42.5 dB. However, the side-lobes also taper off

much more slowly than with the Hann window. For a given length, the Hamming window is better

than the Hann (and of course the boxcar) windows at separating a small component relatively near to a large component, but worse than the Hann for identifying very small components at considerable frequency separation. Due to their shape and form, the Hann and Hamming windows are also known as **raised-cosine windows**.

<db:title>Hamming window</db:title>





(a)

<db:title>Magnitude of Hamming window spectrum</db:title>

(b)

Figure 3.9.

Length-64 Hamming window and its magnitude DFT spectrum

[redacted]

Standard even-length windows are symmetric around a point halfway between the window samples

and . For some applications such as **time-frequency analysis**, it may be

important to align the window perfectly to a sample. In such cases, a **DFT-symmetric** window that is symmetric around the -th sample can be used. For example, the DFT-symmetric

Hamming window is

. A DFT-symmetric window has a purely real-

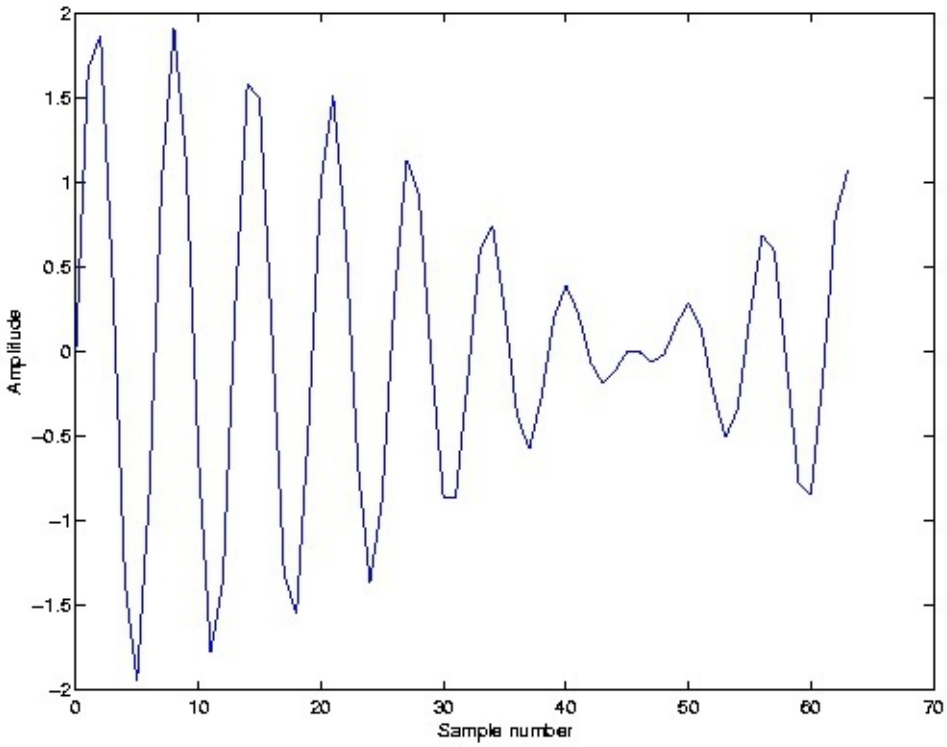
valued DFT and DTFT. DFT-symmetric versions of windows, such as the Hamming and Hann windows, composed of few discrete Fourier series terms of period N, have few non-zero DFT terms (only when **not** zero-padded) and can be used efficiently in **running FFTs**.

The main-lobe width of a window is an inverse function of the window-length N ; for any type of window, a longer window will always provide better resolution.

Many other windows exist that make various other tradeoffs between main-lobe width, height of largest side-lobe, and side-lobe rolloff rate. The **Kaiser window** family, based on a modified Bessel function, has an adjustable parameter that allows the user to tune the tradeoff over a continuous range. The Kaiser window has near-optimal time-frequency resolution and is widely used. A list of many different windows can be found **here**.

Example 3.1.

Figure 3.10 shows 64 samples of a real-valued signal composed of several sinusoids of various frequencies and amplitudes.



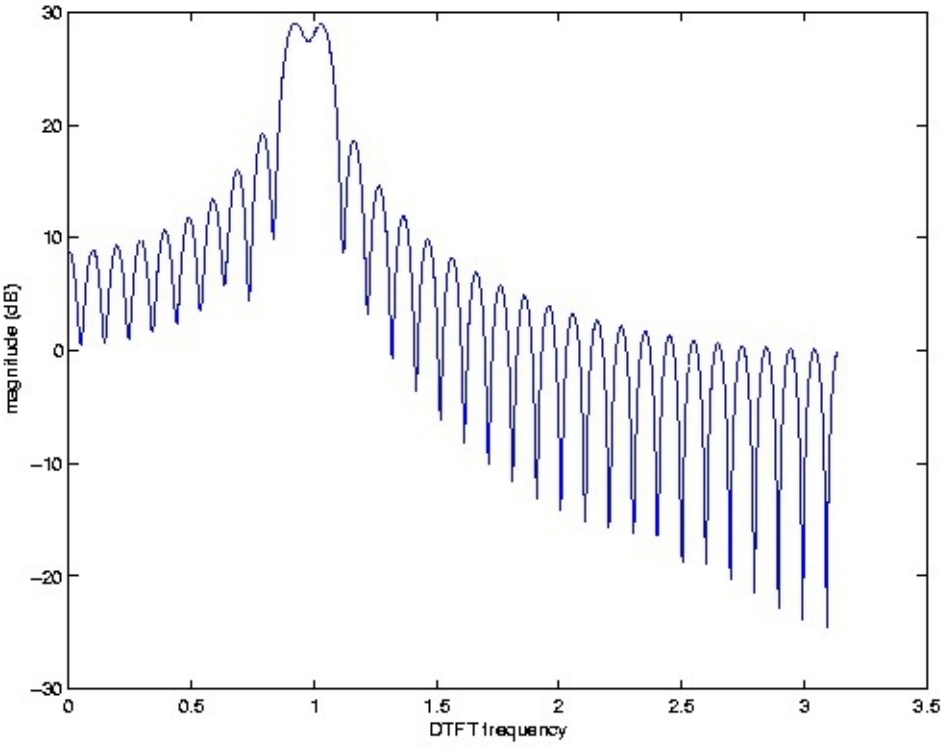


Figure 3.10.

64 samples of an unknown signal

Figure 3.11 shows the magnitude (in dB) of the positive frequencies of a length-1024 zero-padded DFT of this signal (that is, using a simple truncation, or rectangular, window).

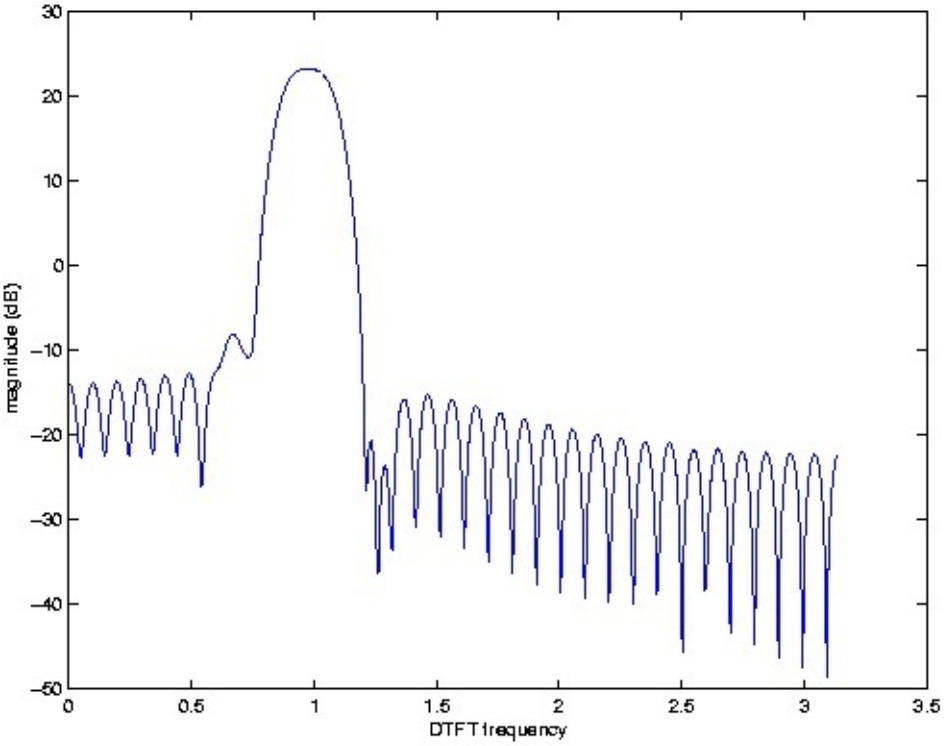


Figure 3.11.

Magnitude (in dB) of the zero-padded DFT spectrum of the signal in Figure 3.10 using a simple length-64 rectangular window From this spectrum, it is clear that the signal has two large, nearby frequency components

with frequencies near 1 radian of essentially the same magnitude.

Figure 3.12 shows the spectral estimate produced using a length-64 Hamming window applied

to the same signal shown in Figure 3.10.

Figure 3.12.

Magnitude (in dB) of the zero-padded DFT spectrum of the signal in Figure 3.10 using a length-64 Hamming window The two large spectral peaks can no longer be resolved; they blur into a single broad peak due

to the reduced spectral resolution of the broader main lobe of the Hamming window. However,

the lower side-lobes reveal a third component at a frequency of about 0.7 radians at about 35

dB lower magnitude than the larger components. This component was entirely buried under

the side-lobes when the rectangular window was used, but now stands out well above the much

lower nearby side-lobes of the Hamming window.

Figure 3.13 shows the spectral estimate produced using a length-64 Hann window applied to

the same signal shown in Figure 3.10.

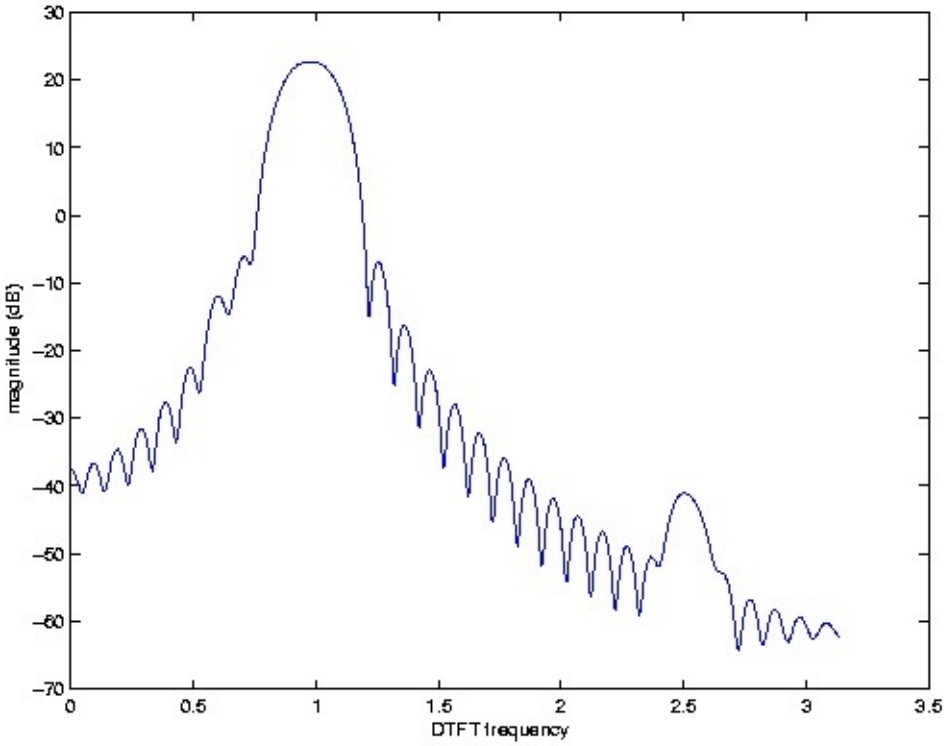


Figure 3.13.

Magnitude (in dB) of the zero-padded DFT spectrum of the signal in Figure 3.10 using a length-64 Hann window The two large components again merge into a single peak, and the smaller component observed with the Hamming window is largely lost under the higher nearby side-lobes of the Hann window. However, due to the much faster side-lobe rolloff of the Hann window's spectrum, a fourth component at a frequency of about 2.5 radians with a magnitude about 65 dB below that of the main peaks is now clearly visible.

This example illustrates that no single window is best for all spectrum analyses. The best window depends on the nature of the signal, and different windows may be better for different components of the same signal. A skilled spectrum analysist may apply several different windows to a signal to gain a fuller understanding of the data.

Classical Statistical Spectral Estimation*

Many signals are either partly or wholly stochastic, or random. Important examples include human speech, vibration in machines, and **CDMA** communication signals. Given the ever-present noise in electronic systems, it can be argued that almost **all** signals are at least partly stochastic.

Such signals may have a distinct **average** spectral structure that reveals important information

(such as for speech recognition or early detection of damage in machinery). Spectrum analysis of any single block of data using **window-based deterministic spectrum analysis**, however, produces a random spectrum that may be difficult to interpret. For such situations, the classical

[redacted]

[redacted]

statistical spectrum estimation methods described in this module can be used.

The goal in classical statistical spectrum analysis is to estimate $E[(|X(\omega)|)2]$, the **power spectral density (PSD)** across frequency of the stochastic signal. That is, the goal is to find the expected (mean, or average) energy density of the signal as a function of frequency. (For zero-mean signals, this equals the variance of each frequency sample.) Since the spectrum of each block of signal samples is itself random, we must average the squared spectral magnitudes over a number of blocks of data to find the mean. There are two main classical approaches, the **periodogram** and **auto-correlation** methods.

Periodogram method

The periodogram method divides the signal into a number of shorter (and often overlapped) blocks of data, computes the squared magnitude of the **windowed** (and usually **zero-padded**) **DFT**, $Xi(\omega k)$, of each block, and averages them to estimate the power spectral density. The squared magnitudes of the DFTs of L possibly overlapped length- N windowed blocks of signal (each probably with **zero-padding)** are averaged to estimate the power spectral density:

For a fixed **total** number of samples, this introduces a tradeoff: Larger individual data blocks provides better frequency resolution due to the use of a longer window, but it means there are less blocks to average, so the estimate has higher variance and appears more noisy. The best tradeoff depends on the application. Overlapping blocks by a factor of two to four increases the number of averages and reduces the variance, but since the same data is being reused, still more overlapping does not further reduce the variance. As with any **window-based**

spectrum estimation procedure, the window function introduces broadening and sidelobes into the power spectrum estimate. That is, the periodogram produces an estimate of the **windowed** spectrum

, not of $E[(|X(\omega)|)2]$.

Example 3.2.

Figure 3.14 shows the non-negative frequencies of the DFT (zero-padded to 1024 total samples) of 64 samples of a real-valued stochastic signal.

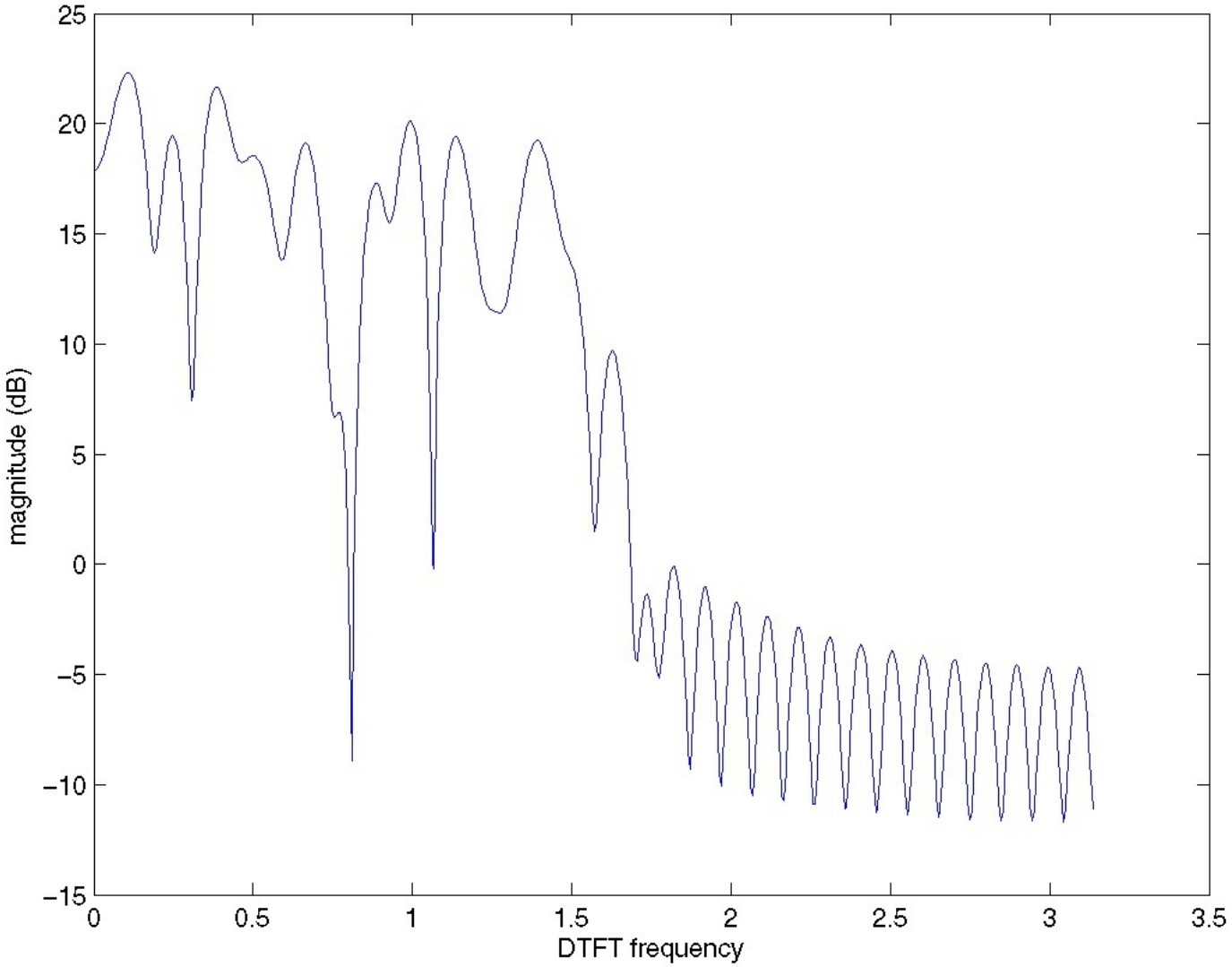


Figure 3.14.

DFT magnitude (in dB) of 64 samples of a stochastic signal

With no averaging, the power spectrum is very noisy and difficult to interpret other than

noting a significant reduction in spectral energy above about half the Nyquist frequency. Various peaks and valleys appear in the lower frequencies, but it is impossible to say from this figure whether they represent actual structure in the power spectral density (PSD) or simply random variation in this single realization. Figure 3.15 shows the same frequencies of a length-1024 DFT of a length-1024 signal. While the frequency resolution has improved, there is still no averaging, so it remains difficult to understand the power spectral density of this signal. Certain small peaks in frequency might represent narrowband components in the spectrum, or may just be random noise peaks.

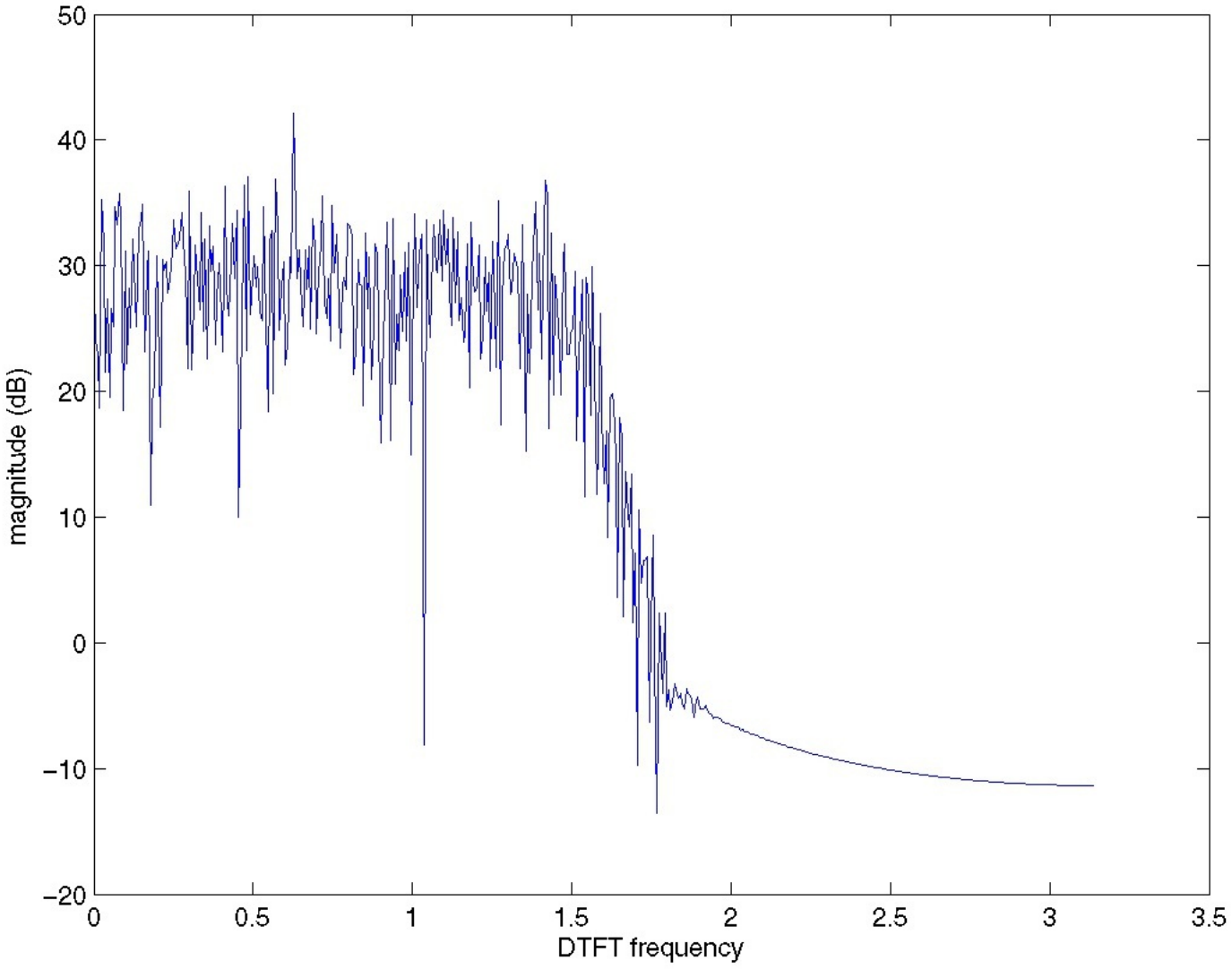


Figure 3.15.

DFT magnitude (in dB) of 1024 samples of a stochastic signal

In Figure 3.16, a power spectral density computed from averaging the squared magnitudes of length-1024 zero-padded DFTs of 508 length-64 blocks of data (overlapped by a factor of four, or a 16-sample step between blocks) are shown.

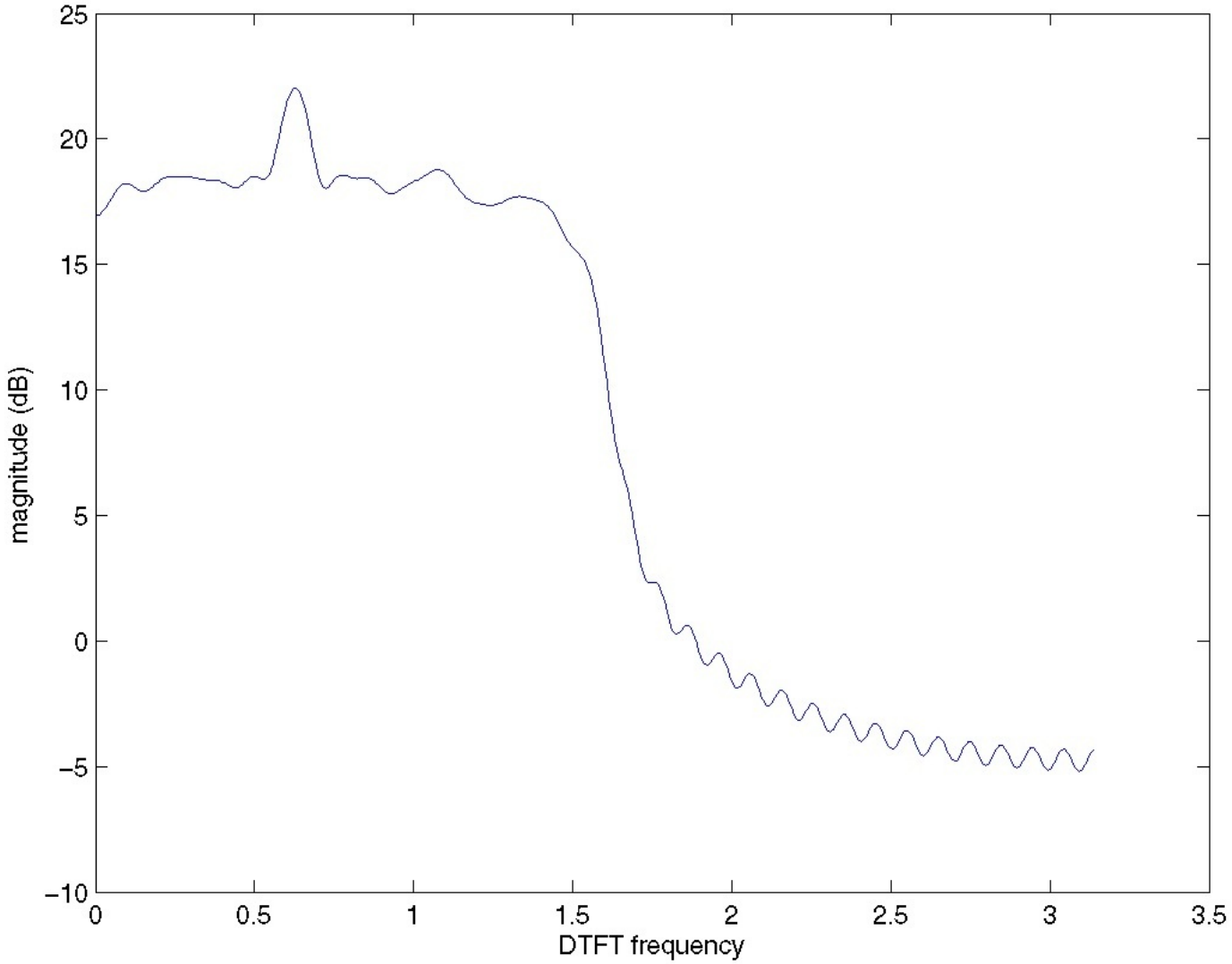


Figure 3.16.

Power spectrum density estimate (in dB) of 1024 samples of a stochastic signal

While the frequency resolution corresponds to that of a length-64 truncation window, the averaging greatly reduces the variance of the spectral estimate and allows the user to reliably conclude that the signal consists of lowpass broadband noise with a flat power spectrum up to half the Nyquist frequency, with a stronger narrowband frequency component at around 0.65 radians.

Auto-correlation-based approach

The averaging necessary to estimate a power spectral density can be performed in the discrete-time domain, rather than in frequency, using the auto-correlation method. The squared magnitude of the frequency response, from the DTFT multiplication and conjugation properties, corresponds in the discrete-time domain to the signal convolved with the time-reverse of itself, $((|X(\omega)|)2 = X(\omega)\, X^*(\omega) \leftrightarrow (x(n), x^*(-n)) = r(n))$ or its **auto-correlation** $r(n) = \sum(x(k)\, x^*(n+k))$ We can

thus compute the squared magnitude of the spectrum of a signal by computing the DFT of its auto-correlation. For stochastic signals, the power spectral density is an expectation, or average, and by linearity of expectation can be found by transforming the average of the auto-correlation. For a finite block of N signal samples, the average of the autocorrelation values, $r(n)$, is

Note that with increasing **lag**, n, fewer values are averaged, so they introduce more noise into the estimated power spectrum. By **windowing** the auto-correlation before transforming it to the frequency domain, a less noisy power spectrum is obtained, at the expense of less resolution. The multiplication property of the DTFT shows that the windowing smooths the resulting power spectrum via convolution with the DTFT of the window:

This yields another important interpretation of how the auto-correlation method works: it estimates the power spectral density by **averaging the power spectrum over nearby frequencies**, through convolution with the window function's transform, to reduce variance. Just as with the periodogram approach, there is always a variance vs. resolution tradeoff. The periodogram and the auto-correlation method give similar results for a similar amount of averaging; the user should simply note that in the periodogram case, the window introduces smoothing of the spectrum via frequency convolution **before** squaring the

magnitude, whereas the periodogram convolves the squared magnitude with $W(\omega)$.

Short Time Fourier Transform*

Short Time Fourier Transform

The Fourier transforms (FT, DTFT, DFT, *etc.*) do not clearly indicate how the frequency content of a signal changes over time.

That information is hidden in the phase - it is not revealed by the plot of the magnitude of the spectrum.

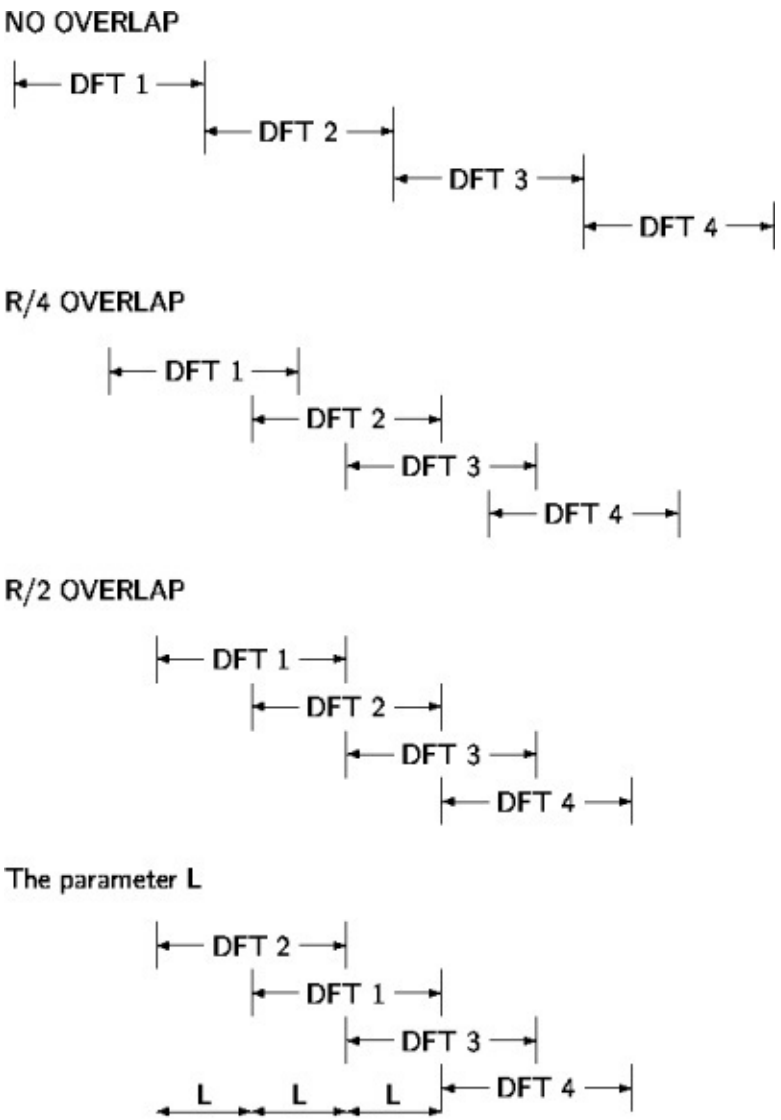
Note

To see how the frequency content of a signal changes over time, we can cut the signal into blocks and compute the spectrum of each block.

To improve the result,

1. blocks are overlapping

2. each block is multiplied by a window that is tapered at its endpoints.



Several parameters must be chosen:

Block length, *R*.

The type of window.

Amount of overlap between blocks. (Figure 3.17)

Amount of zero padding, if any.

Figure 3.17. STFT: Overlap Parameter

The short-time Fourier transform is defined as

()

[redacted]

[redacted]

where $w(n)$ is the window function of length R.

1. The STFT of a signal $x(n)$ is a function of two variables: time and frequency.

2. The block length is determined by the support of the window function $w(n)$.

3. A graphical display of the magnitude of the STFT, $|X(\omega, m)|$, is called the **spectrogram** of the signal. It is often used in speech processing.

4. The STFT of a signal is invertible.

5. One can choose the block length. A long block length will provide higher frequency resolution (because the main-lobe of the window function will be narrow). A short block length will provide higher time resolution because less averaging across samples is performed for each STFT value.

6. A **narrow-band spectrogram** is one computed using a relatively long block length R, (long window function).

7. A **wide-band spectrogram** is one computed using a relatively short block length R, (short window function).

Sampled STFT

To numerically evaluate the STFT, we sample the frequency axis ω in N equally spaced samples from $\omega=0$ to $\omega= \boldsymbol{2\pi}$.

()

We then have the discrete STFT,

()

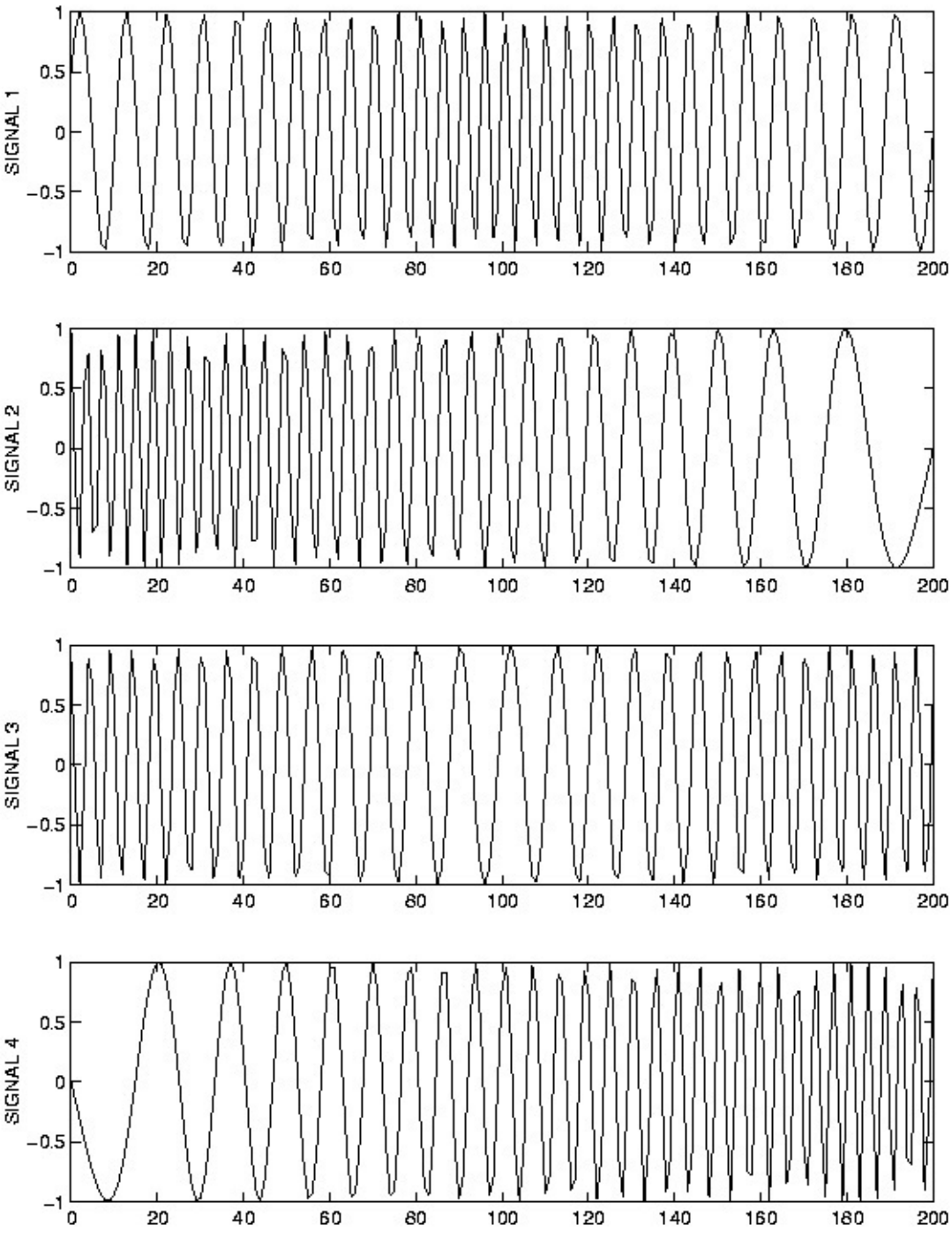
where $0,\ldots 0$ is $N- R$.

In this definition, the overlap between adjacent blocks is $R-1$. The signal is shifted along the

window one sample at a time. That generates more points than is usually needed, so we also

sample the STFT along the time direction. That means we usually evaluate $Xd(\ k, \boldsymbol{Lm})$ where L is the time-skip. The relation between the time-skip, the number of overlapping samples, and the

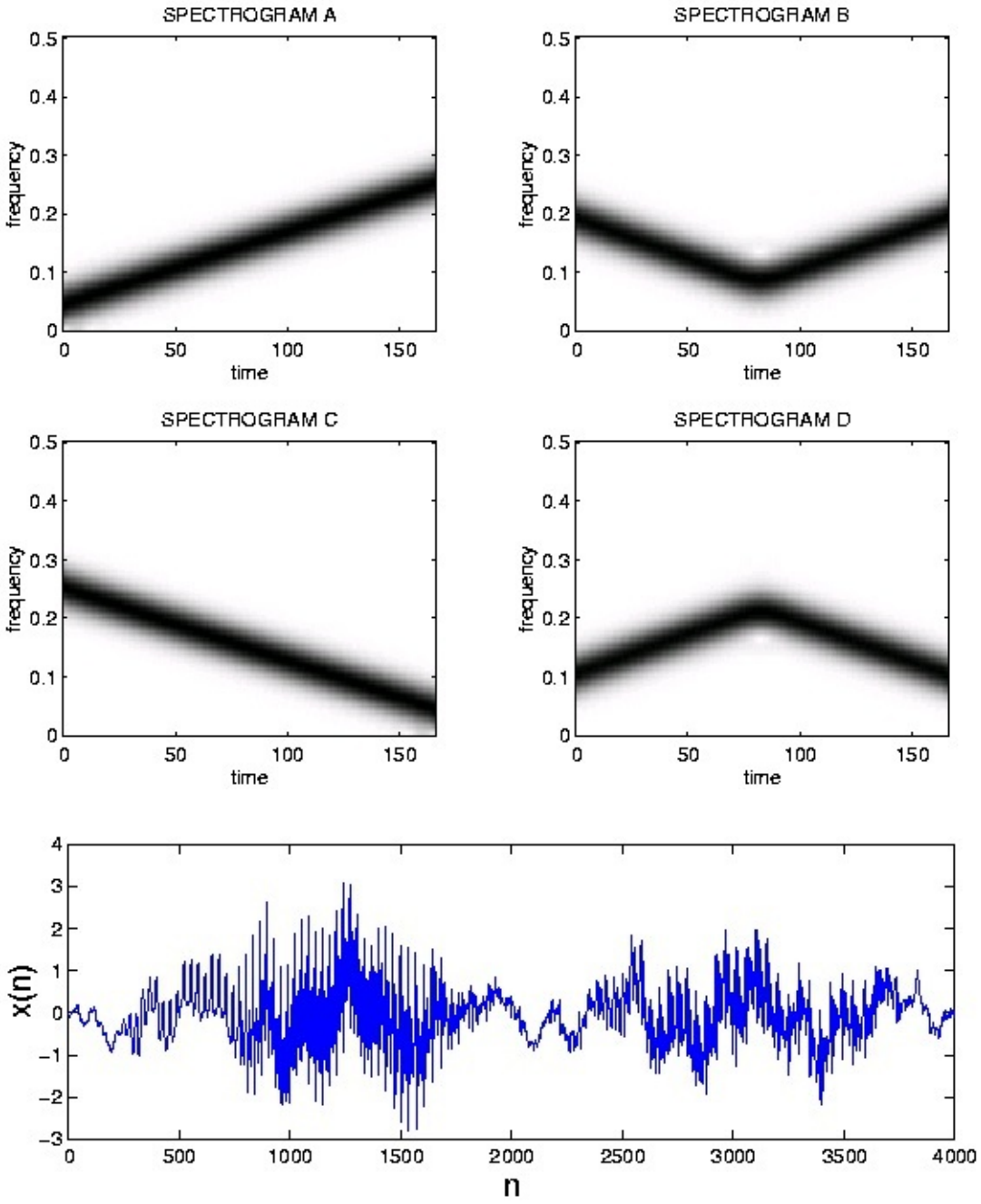
block length is Overlap= $R- L$



Exercise 2.

Match each signal to its spectrogram in Figure 3.17.

(a)



(b)

Figure 3.17.

Spectrogram Example

Figure 3.18.

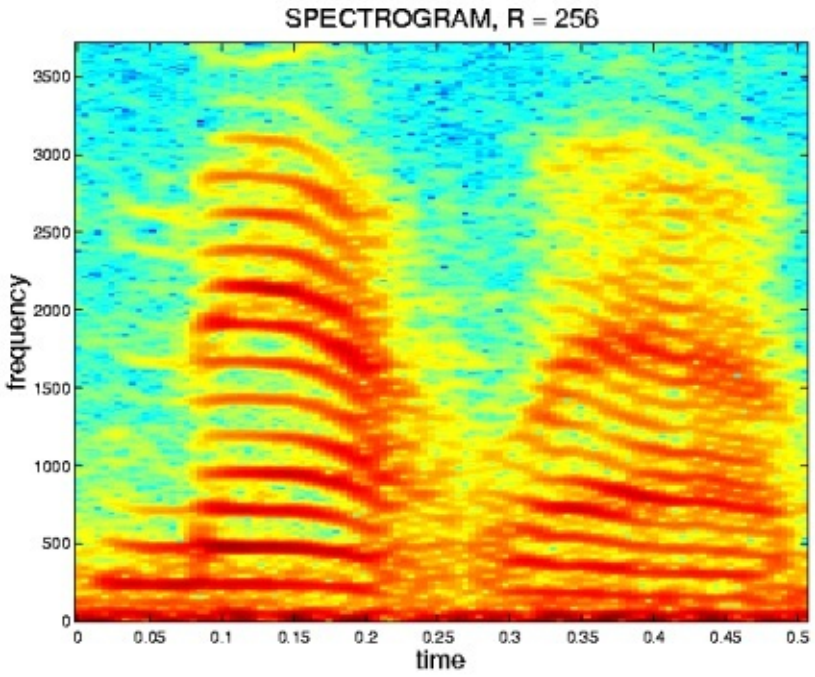
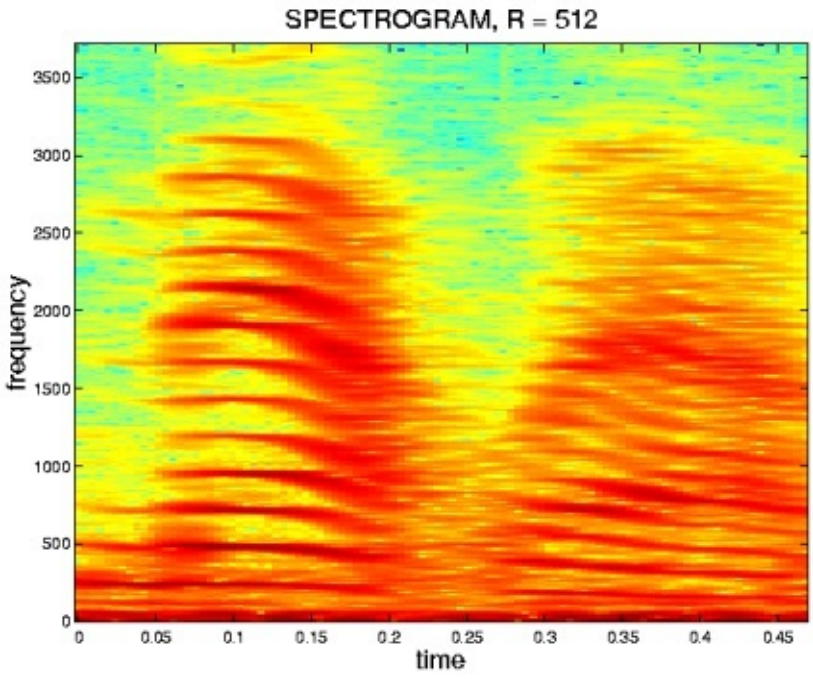


Figure 3.19.

The matlab program for producing the figures above (Figure 3.18 and Figure 3.19).

```
% LOAD DATA

load mtlb;

x = mtlb;

figure(1), clf

plot(0:4000,x)

xlabel('n')

ylabel('x(n)')

% SET PARAMETERS

R = 256; % R: block length

window = hamming(R); % window function of length R

N = 512; % N: frequency discretization

L = 35; % L: time lapse between blocks

fs = 7418; % fs: sampling frequency

overlap = R - L;

% COMPUTE SPECTROGRAM
```

```
[B,f,t] = specgram(x,N,fs,window,overlap);

% MAKE PLOT

figure(2), clf
```




```
imagesc(t,f,log10(abs(B)));

colormap('jet')

axis xy

xlabel('time')

ylabel('frequency')

title('SPECTROGRAM, R = 256')
```

Effect of window length R

Figure 3.20. Narrow-band spectrogram: better frequency resolution

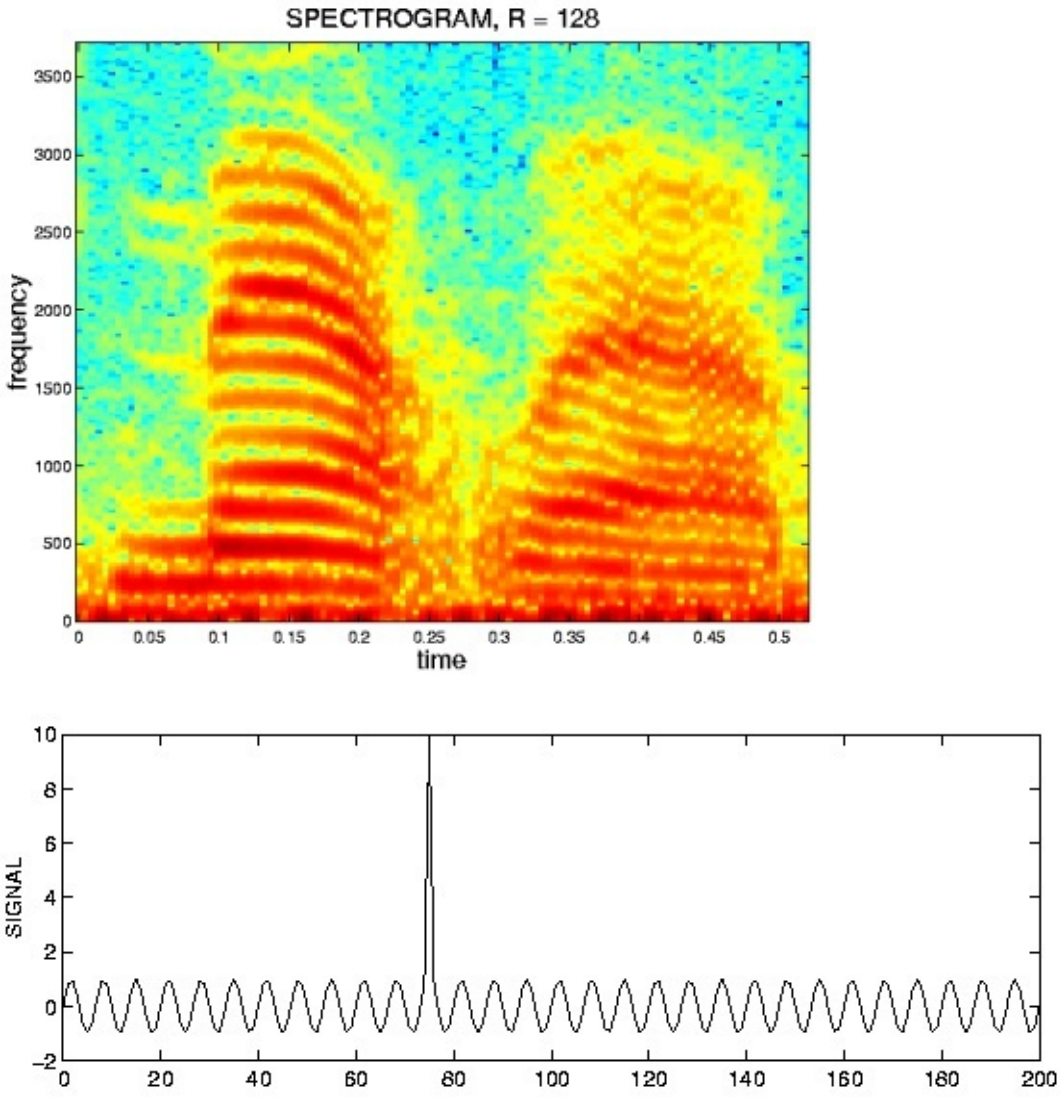
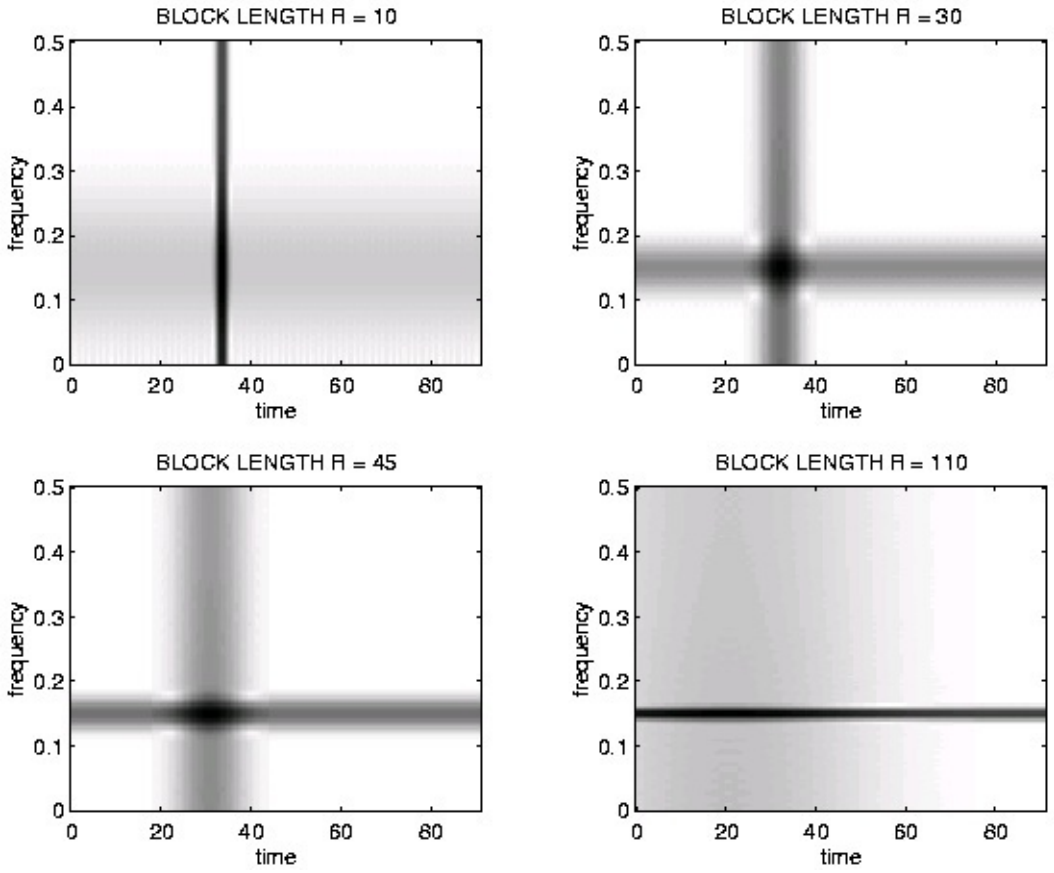


Figure 3.21. Wide-band spectrogram: better time resolution

Here is another example to illustrate the frequency/time resolution trade-off (See figures - Figure 3.20, Figure 3.21, and Figure 3.22).

(a)



(b)

Figure 3.22. Effect of Window Length R

Effect of L and N

A spectrogram is computed with different parameters: $L \in \{1, 10\}$ $N \in \{32, 256\}$

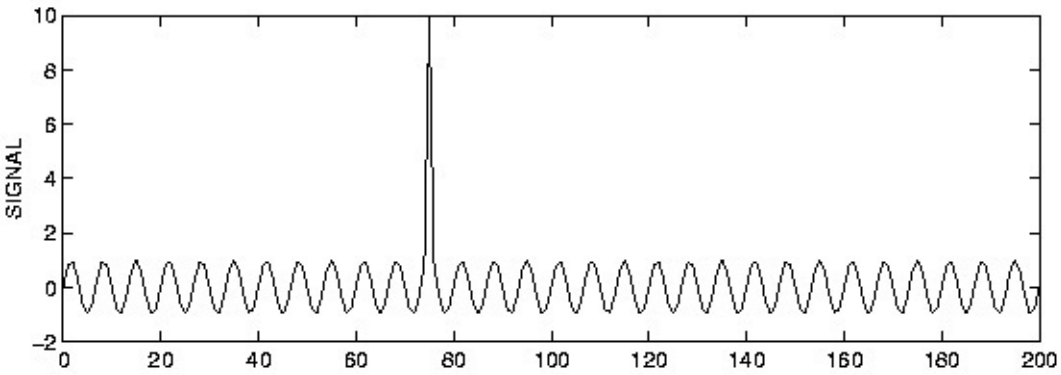
L = time lapse between blocks.

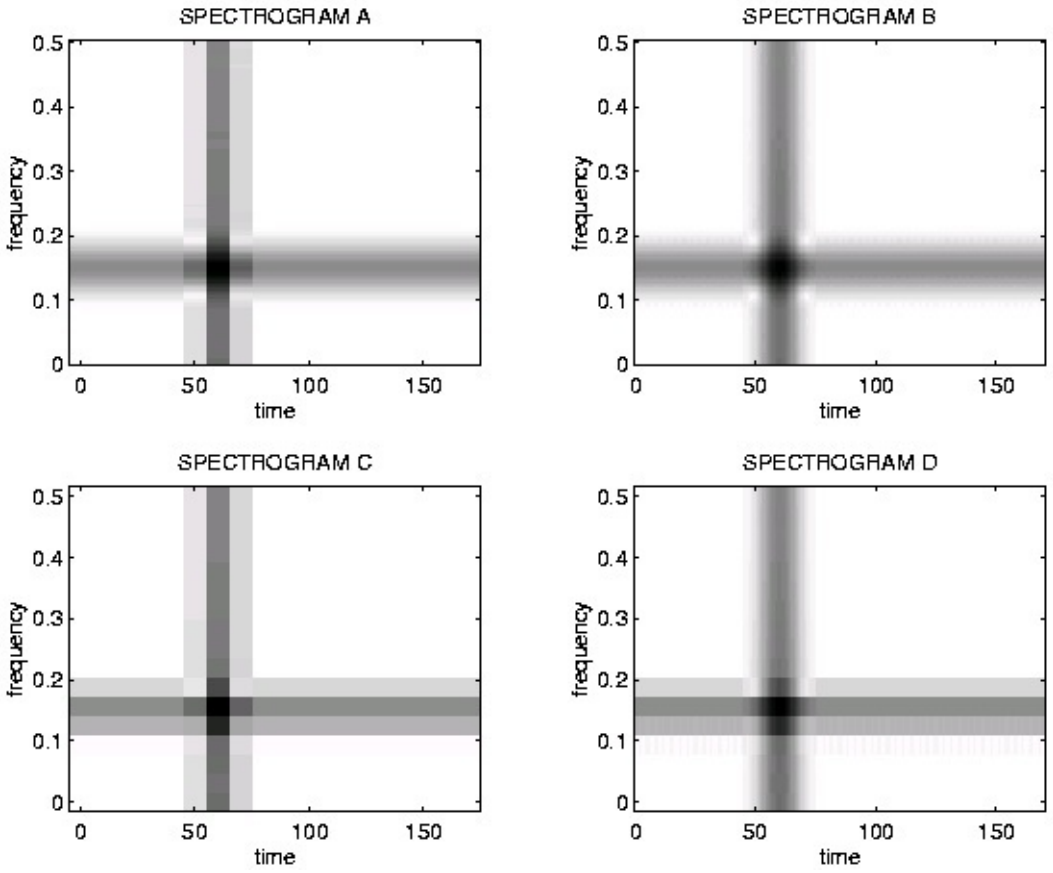
N = FFT length (Each block is zero-padded to length N.)

In each case, the block length is 30 samples.

Exercise 3.

For each of the four spectrograms in Figure 3.22 can you tell what L and N are?





(a)

(b)

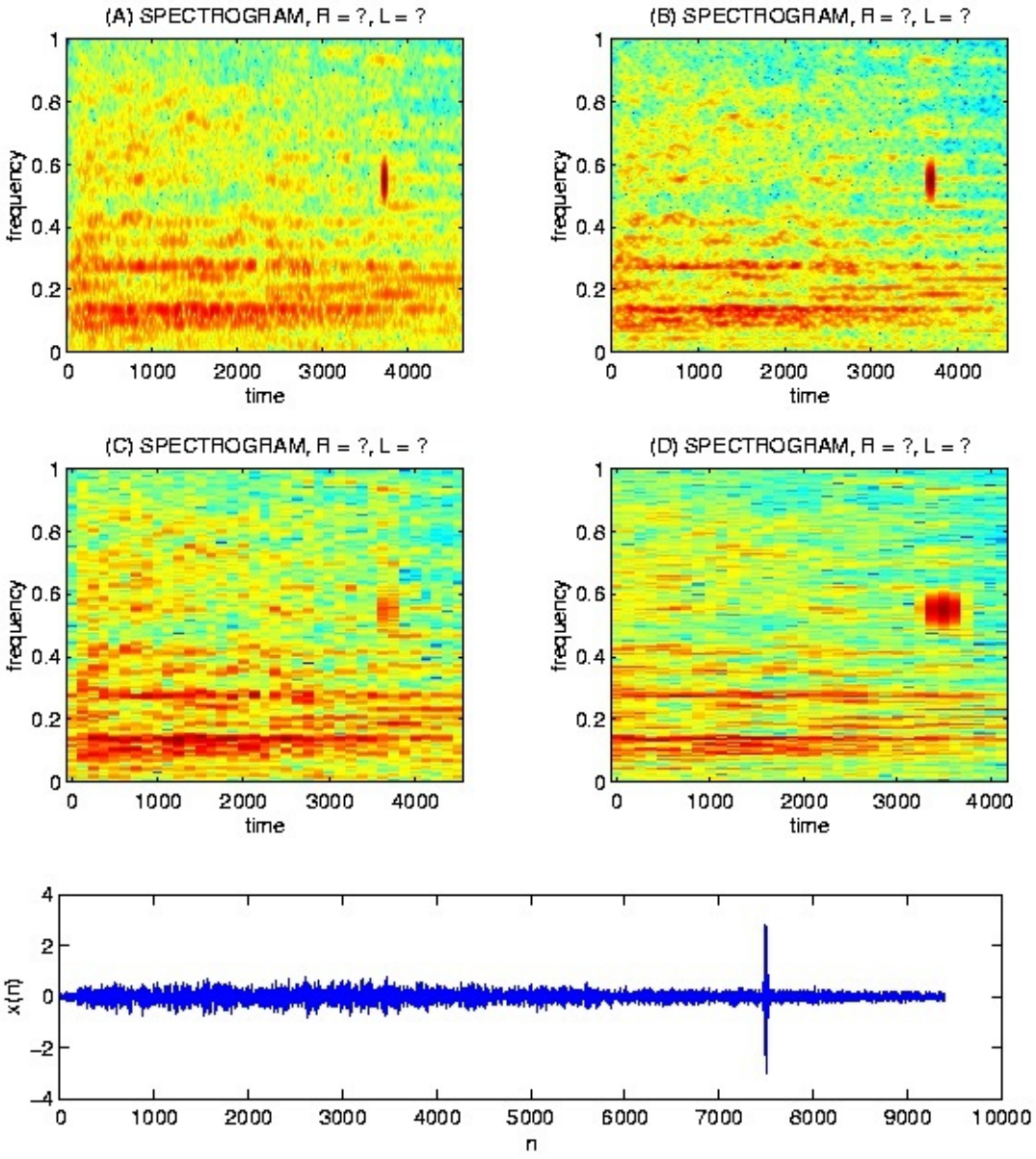
Figure 3.22.

L and *N* do not effect the time resolution or the frequency resolution. They only affect the 'pixelation'.

Effect of R and L

Shown below are four spectrograms of the same signal. Each spectrogram is computed using a different set of parameters. $R \in \{120, 256, 1024\}$ $L \in \{35, 250\}$ where

R = block length



L = time lapse between blocks.

Exercise 4.

For each of the four spectrograms in Figure 3.22, match the above values of *L* and *R*.

Figure 3.22.

If you like, you may listen to this signal with the soundsc command; the data is in the file:

stft_data.m. **Here** is a figure of the signal.

Figure 3.23.

3.3. Fast Fourier Transform Algorithms

[redacted]

Overview of Fast Fourier Transform (FFT) Algorithms*

A **fast Fourier transform**, or **FFT,** is not a new transform, but is a computationally efficient algorithm for the computing the **DFT.** The length- N DFT, defined as

()

where $X(k)$ and $x(n)$ are in general complex-valued and $0 \le k, n \le N-1$, requires N complex multiplies to compute each $X(k)$. Direct computation of all N frequency samples thus requires

N 2 complex multiplies and $N(N-1)$ complex additions. (This assumes precomputation of the DFT

coefficients

; otherwise, the cost is even higher.) For the large DFT lengths used in

many applications, N 2 operations may be prohibitive. (For example, digital terrestrial television

broadcast in Europe uses $N = 2048$ or 8192 OFDM channels, and the **SETI** project uses up to length-4194304 DFTs.) DFTs are thus almost always computed in practice by an **FFT algorithm.**

FFTs are very widely used in signal processing, for applications such as **spectrum analysis** and digital filtering via **fast convolution**.

History of the FFT

It is now known that **C.F. Gauss** invented an FFT in 1805 or so to assist the computation of planetary orbits via **discrete Fourier series.** Various FFT algorithms were independently invented over the next two centuries, but FFTs achieved widespread awareness and impact only with the

Cooley and Tukey algorithm published in 1965, which came at a time of increasing use of digital

computers and when the vast range of applications of numerical Fourier techniques was becoming

apparent. Cooley and Tukey's algorithm spawned a surge of research in FFTs and was also partly

responsible for the emergence of Digital Signal Processing (DSP) as a distinct, recognized

discipline. Since then, many different algorithms have been rediscovered or developed, and

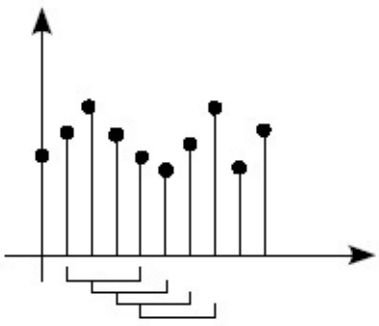
efficient FFTs now exist for all DFT lengths.

Summary of FFT algorithms

The main strategy behind most FFT algorithms is to factor a length- N DFT into a number of

shorter-length DFTs, the outputs of which are reused multiple times (usually in additional short-

length DFTs!) to compute the final results. The lengths of the short DFTs correspond to integer factors of the DFT length, *N*, leading to different algorithms for different lengths and factors. By far the most commonly used FFTs select $N=2^M$ to be a power of two, leading to the very efficient **power-of-two FFT algorithms**, including the **decimation-in-time radix-2 FFT** and the **decimation-in-frequency radix-2 FFT** algorithms, the **radix-4 FFT** ($N=4^M$), and the **split-radix FFT**. Power-of-two algorithms gain their high efficiency from extensive reuse of



intermediate results and from the low complexity of length-2 and length-4 DFTs, which require no multiplications. Algorithms for lengths with repeated **common factors** (such as 2 or 4 in the radix-2 and radix-4 algorithms, respectively) require extra **twiddle factor** multiplications between the short-length DFTs, which together lead to a computational complexity of $O(N\log N)$, a very considerable savings over direct computation of the DFT.

The other major class of algorithms is the **Prime-Factor Algorithms (PFA)**. In PFAs, the short-length DFTs must be of relatively prime lengths. These algorithms gain efficiency by reuse of intermediate computations and by eliminating twiddle-factor multiplies, but require more operations than the power-of-two algorithms to compute the short DFTs of various prime lengths. In the end, the computational costs of the prime-factor and the power-of-two algorithms are comparable for similar lengths, as illustrated in **Choosing the Best FFT Algorithm**. Prime-length DFTs cannot be factored into shorter DFTs, but in different ways both **Rader's conversion** and the **chirp z-transform** convert prime-length DFTs into convolutions of other lengths that can be computed efficiently using FFTs via **fast convolution**.

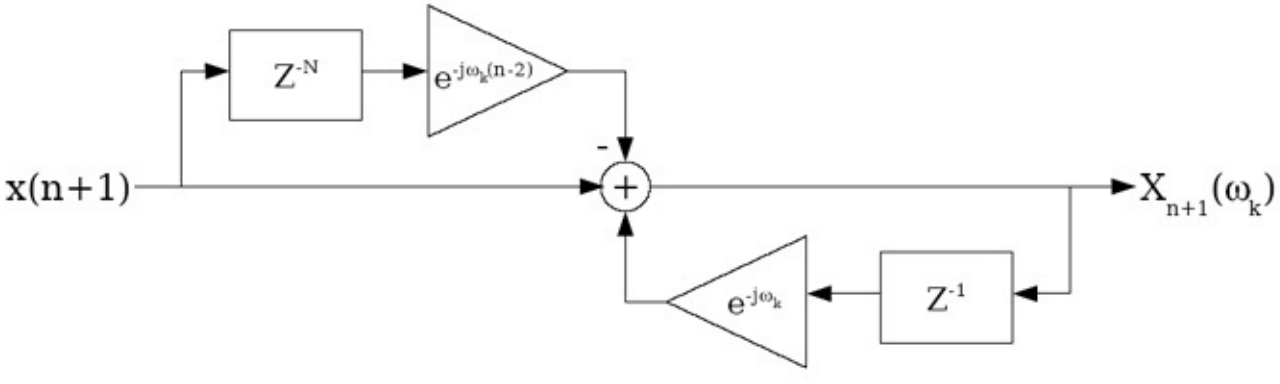
Some applications require only a few DFT frequency samples, in which case **Goertzel's**

algorithm halves the number of computations relative to the DFT sum. Other applications involve successive DFTs of overlapped blocks of samples, for which the **running FFT** can be more efficient than separate FFTs of each block.

Running FFT*

Some applications need **DFT** frequencies of the most recent *N* samples on an ongoing basis. One example is **DTMF**, or touch-tone telephone dialing, in which a detection circuit must constantly monitor the line for two simultaneous frequencies indicating that a telephone button is depressed. In such cases, most of the data in each successive block of samples is the same, and it is possible to efficiently update the DFT value from the previous sample to compute that of the current sample. Figure 3.24 illustrates successive length-4 blocks of data for which successive DFT values may be needed. The **running FFT** algorithm described here can be used to compute successive DFT values at a cost of only two complex multiplies and additions per DFT frequency.

Figure 3.24.



The running FFT efficiently computes DFT values for successive overlapped blocks of samples.

The running FFT algorithm is derived by expressing each DFT sample, $X\,n + 1\,(\,\omega k)$, for the next block at time n+1 in terms of the previous value, $Xn(\,\omega k)$, at time n.

Let $q = p - 1$:

Now let's add and subtract $e^{-(\,i\omega k(\,N-2))}\,x(\,n-\,N+1)$:

()

This running FFT algorithm requires only two complex multiplies and adds per update, rather than N if each DFT value were recomputed according to the DFT equation. Another advantage of this algorithm is that it works for **any** ωk , rather than just the standard DFT frequencies. This can make it advantageous for applications, such as DTMF detection, where only a few arbitrary frequencies are needed.

Successive computation of a specific DFT frequency for overlapped blocks can also be thought of as a length- N **FIR filter**. The running FFT is an efficient recursive implementation of this filter for this special case. Figure 3.25 shows a block diagram of the running FFT algorithm. The running FFT is one way to compute **DFT filterbanks**. If a window other than rectangular is desired, a running FFT requires either a fast recursive implementation of the corresponding windowed, modulated impulse response, or it must have few non-zero coefficients so that it can be applied after the running FFT update via frequency-domain convolution. **DFT-symmmetric raised-cosine windows** are an example.

Figure 3.25.

Block diagram of the running FFT computation, implemented as a recursive filter

[redacted]

[redacted]

Goertzel's Algorithm*

Some applications require only a few DFT frequencies. One example is **frequency-shift keying**

(FSK) demodulation, in which typically two frequencies are used to transmit binary data; another example is **DTMF**, or touch-tone telephone dialing, in which a detection circuit must constantly monitor the line for two simultaneous frequencies indicating that a telephone button is depressed.

Goertzel's algorithm [**link**] reduces the number of real-valued multiplications by almost a factor of two relative to direct computation via the **DFT equation**. Goertzel's algorithm is thus useful for computing a **few** frequency values; if many or most DFT values are needed, **FFT algorithms** that compute all DFT samples in $O(N\log N)$ operations are faster. Goertzel's algorithm can be derived

by converting the **DFT equation** into an equivalent form as a convolution, which can be

efficiently implemented as a digital filter. For increased clarity, in the equations below the

complex exponential is denoted as

. Note that because $W - N k$

N

always equals 1, the **DFT**

equation can be rewritten as a convolution, or filtering operation:

()

Note that this last expression can be written in terms of a recursive **difference equation**

$y(n)= W - k$

N

$y(n-1)+ x(n)$ where $y(-1)=0$. The DFT coefficient equals the output of the difference

equation at time $n= N$: $X(k)= y(N)$ Expressing the difference equation as a **z-transform** and multiplying both numerator and denominator by $1- W k$

$N\ z$–1 gives the transfer function

This system can be realized by the

structure in Figure 3.26

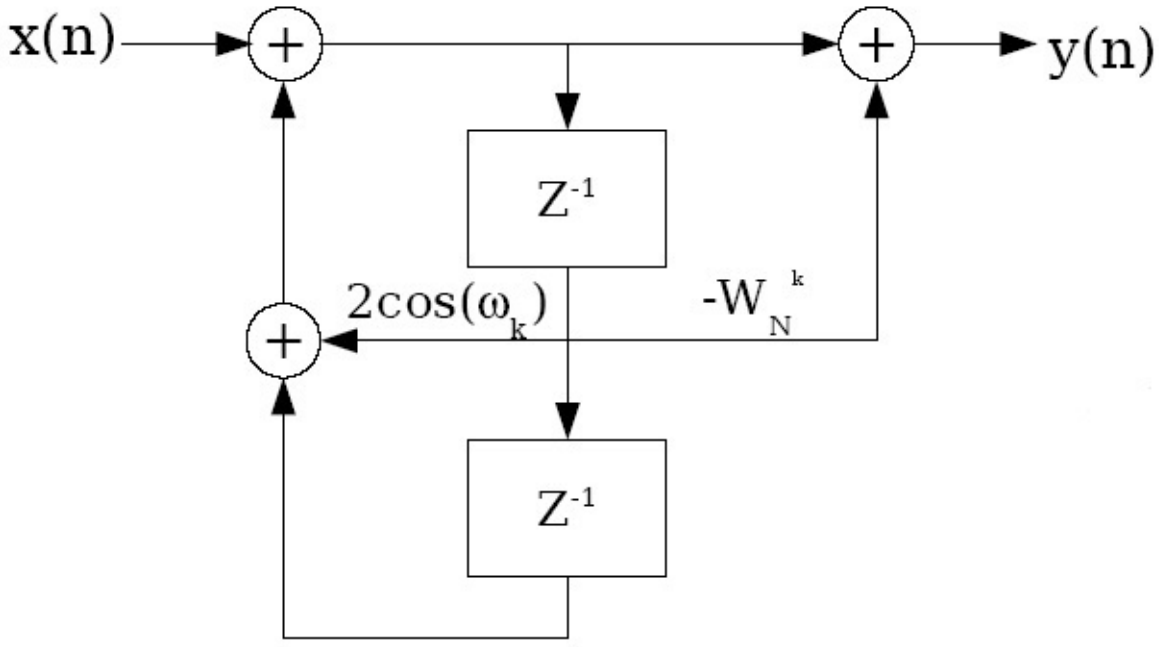


Figure 3.26.

We want $y(n)$ not for all n, but only for $n = N$. We can thus compute only the **recursive** part, or just the left side of the flow graph in Figure 3.26, for $n=[0, 1, \ldots, N]$, which involves only a **real/complex** product rather than a complex/complex product as in a direct **DFT,** plus one complex multiply to get $y(N) = X(k)$.

The input $x(N)$ at time $n = N$ must equal 0! A slightly more efficient **alternate implementation**

that computes the full recursion only through $n = N-1$ and combines the nonzero operations of

the final recursion with the final complex multiply can be found **here,** complete with pseudocode (for real-valued data).

If the data are real-valued, only real/real multiplications and real additions are needed until the

final multiply.

Cost

The computational cost of Goertzel's algorithm is thus $2N+2$ real multiplies and $4N-2$ real

adds, a reduction of almost a factor of two in the number of real multiplies relative to direct

computation via the DFT equation. If the data are real-valued, this cost is almost halved again.

For certain frequencies, additional simplifications requiring even fewer multiplications are

possible. (For example, for the DC (k=0) frequency, all the multipliers equal 1 and only

additions are needed.) A correspondence by *C.G. Boncelet, Jr.* [**link**] describes some of these additional simplifications. Once again, Goertzel's and Boncelet's algorithms are efficient for a few

DFT frequency samples; if more than ***logN*** frequencies are needed, *O*(*N**logN***) **FFT algorithms**

that compute all frequencies simultaneously will be more efficient.

References

1. G Goertzel. (1958). An Algorithm for the Evaluation of Finite Trigonomentric Series. *The*

American Mathematical Monthly.

2. C.G. Boncelet, Jr. (1986, Dec). A Rearranged DFT Algorithm Requiring N^2/6

Multiplications. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, *ASSP-34*(6), 1658-

1659.

Power-of-Two FFTs

Power-of-two FFTs*

FFTs of length N=2 M equal to a power of two are, by far, the most commonly used. These

algorithms are very efficient, relatively simple, and a single program can compute power-of-two

FFTs of different lengths. As with most FFT algorithms, they gain their efficiency by computing

all DFT points simultaneously through extensive reuse of intermediate computations; they are thus efficient when many DFT frequency samples are needed. The simplest power-of-two FFTs

are the **decimation-in-time radix-2 FFT** and the **decimation-in-frequency radix-2 FFT;** they reduce the length- N=2 M DFT to a series of length-2 DFT computations with **twiddle-factor**

complex multiplications between them. The **radix-4 FFT algorithm** similarly reduces a length-

N=4 M DFT to a series of length-4 DFT computations with twiddle-factor multiplies in between.

Radix-4 FFTs require only 75% as many complex multiplications as the radix-2 algorithms,

although the number of complex additions remains the same. Radix-8 and higher-radix FFT

algorithms can be derived using **multi-dimensional index maps** to reduce the computational

complexity a bit more. However, the **split-radix algorithm** and its recent extensions combine the best elements of the radix-2 and radix-4 algorithms to obtain lower complexity than either or than

any higher radix, requiring only two-thirds as many complex multiplies as the radix-2 algorithms.

All of these algorithms obtain huge savings over direct computation of the DFT, reducing the

complexity from $O(N^2)$ to $O(N \log N)$.

The efficiency of an FFT implementation depends on more than just the number of computations.

Efficient FFT programming tricks can make up to a several-fold difference in the run-time of

FFT programs. **Alternate FFT structures** can lead to a more convenient data flow for certain

hardware. As discussed in **choosing the best FFT algorithm**, certain hardware is designed for, and thus most efficient for, FFTs of specific lengths or radices.

Radix-2 Algorithms

Decimation-in-time (DIT) Radix-2 FFT*

The radix-2 decimation-in-time and **decimation-in-frequency** fast Fourier transforms (FFTs) are the simplest **FFT algorithms**. Like all FFTs, they gain their speed by reusing the results of

smaller, intermediate computations to compute multiple DFT frequency outputs.

Decimation in time

The radix-2 decimation-in-time algorithm rearranges the **discrete Fourier transform (DFT)**

equation into two parts: a sum over the even-numbered discrete-time indices $n=[0, 2, 4, \ldots, N-2]$

and a sum over the odd-numbered indices $n=[1, 3, 5, \ldots, N-1]$ as in Equation 3.2: **(3.2)**

The mathematical simplifications in Equation 3.2 reveal that all DFT frequency outputs $X(k)$ can be computed as the sum of the outputs of two length- DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where the odd-indexed short DFT is multiplied by a

so-called **twiddle factor** term

. This is called a **decimation in time** because the time

samples are rearranged in alternating groups, and a **radix-2** algorithm because there are two

groups. Figure 3.27 graphically illustrates this form of the DFT computation, where for

convenience the frequency outputs of the length- DFT of the even-indexed time samples are

denoted $G(k)$ and those of the odd-indexed samples as $H(k)$. Because of the periodicity with

frequency samples of these length- DFTs, $G(k)$ and $H(k)$ can be used to compute **two** of the

length- N DFT frequencies, namely $X(k)$ and

, but with a different twiddle factor. This reuse

of these short-length DFT outputs gives the FFT its computational savings.

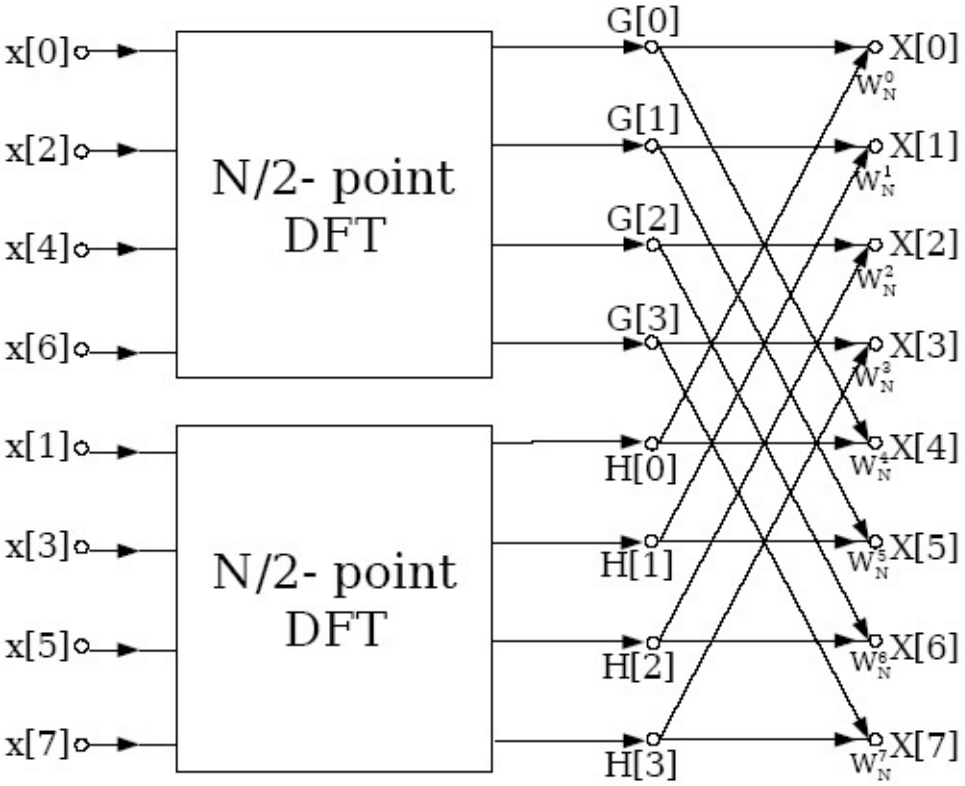


Figure 3.27.

Decimation in time of a length- N DFT into two length-

DFTs followed by a combining stage.

Whereas direct computation of all N DFT frequencies according to the **DFT equation** would require N 2 complex multiplies and N 2− N complex additions (for complex-valued data), by reusing

the results of the two short-length DFTs as illustrated in Figure 3.27, the computational cost is

now

New Operation Counts

complex multiplies

complex additions

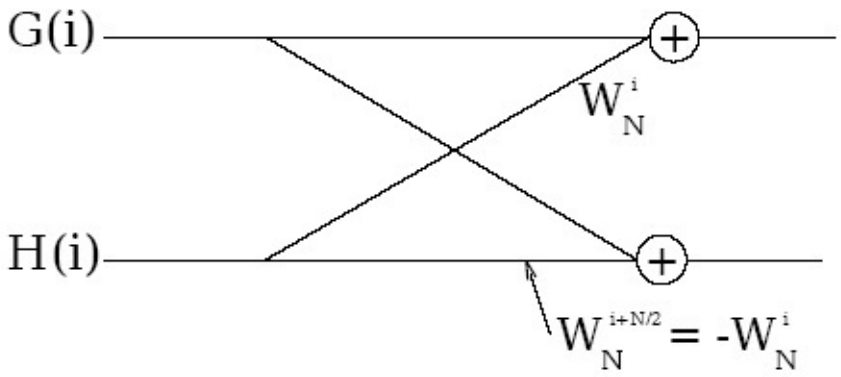
This simple reorganization and reuse has reduced the total computation by almost a factor of two

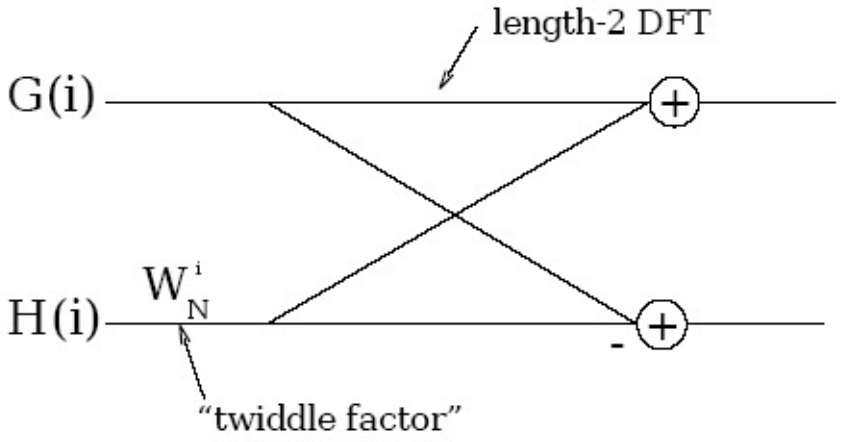
over direct **DFT** computation!

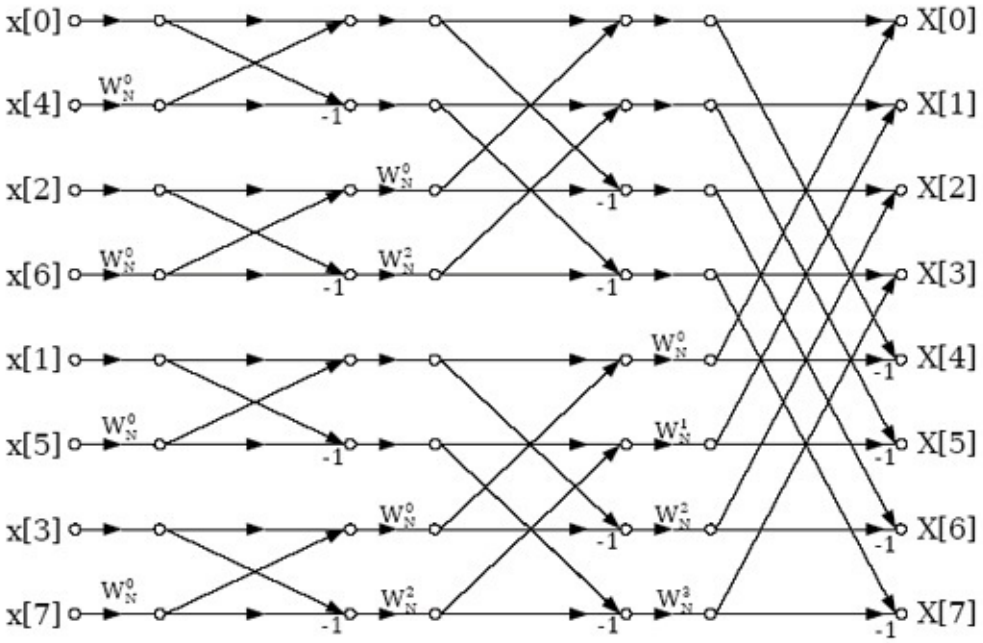
Additional Simplification

A basic **butterfly** operation is shown in Figure 3.28, which requires only **twiddle-factor** multiplies per **stage**. It is worthwhile to note that, after merging the twiddle factors to a single

term on the lower branch, the remaining butterfly is actually a length-2 DFT! The theory of **multi-dimensional index maps** shows that this must be the case, and that FFTs of any factorable length







may consist of successive stages of shorter-length FFTs with twiddle-factor multiplications in between.

(b)

(a)

Figure 3.28.

Radix-2 DIT butterfly simplification: both operations produce the same outputs

Radix-2 decimation-in-time FFT

The same radix-2 decimation in time can be applied recursively to the two length **DFT**s to save computation. When successively applied until the shorter and shorter DFTs reach length-2, the result is the **radix-2 DIT FFT algorithm**.

Figure 3.29.

Radix-2 Decimation-in-Time FFT algorithm for a length-8 signal

The full radix-2 decimation-in-time decomposition illustrated in Figure 3.29 using the **simplified butterflies** involves $M=\log_2 N$ stages, each with butterflies per stage. Each butterfly requires 1 complex multiply and 2 adds per butterfly. The total cost of the algorithm is thus

Computational cost of radix-2 DIT FFT

complex multiplies

N log2N complex adds

This is a remarkable savings over direct computation of the DFT. For example, a length-1024 DFT would require 1048576 complex multiplications and 1047552 complex additions with direct computation, but only 5120 complex multiplications and 10240 complex additions using the radix-2 FFT, a savings by a factor of 100 or more. The relative savings increase with longer FFT lengths, and are less for shorter lengths.

Modest additional reductions in computation can be achieved by noting that certain twiddle factors, namely Using special butterflies for W 0

N ,

,

,

,

, require no multiplications,

or fewer real multiplies than other ones. By implementing special butterflies for these twiddle factors as discussed in **FFT algorithm and programming tricks**, the computational cost of the radix-2 decimation-in-time FFT can be reduced to

$2\ N \log_2 N - 7\ N + 12$ real multiplies

$3\ N \log_2 N - 3\ N + 4$ real additions

In a decimation-in-time radix-2 FFT as illustrated in Figure 3.29, the input is in **bit-reversed**

order (hence "decimation-in-time"). That is, if the time-sample index *n* is written as a binary number, the order is that binary number reversed. The bit-reversal process is illustrated for a length- *N*=8 example below.

Example 3.3. N=8

Table 3.1.

In-order index In-order index in binary Bit-reversed binary Bit-reversed index

0

000

000

0

1

001

100

4

2

010

010

2

3

011

110

6

4

100

001

1

5

101

101

5

6

110

011

3

7

111

111

7

It is important to note that, if the input signal data are placed in bit-reversed order before beginning the FFT computations, the outputs of each butterfly throughout the computation can be placed in the same memory locations from which the inputs were fetched, resulting in an **in-place algorithm** that requires no extra memory to perform the FFT. Most FFT implementations are in-place, and overwrite the input data with the intermediate values and finally the output.

Example FFT Code

The following function, written in the C programming language, implements a radix-2 decimation-in-time FFT. It is designed for computing the DFT of complex-valued inputs to produce complex-valued outputs, with the real and imaginary parts of each number stored in separate double-precision floating-point arrays. It is an in-place algorithm, so the intermediate and final output values are stored in the same array as the input data, which is overwritten. After initializations, the program first bit-reverses the discrete-time samples, as is typical with a

decimation-in-time algorithm (but see **alternate FFT structures** for DIT algorithms with other input orders), then computes the FFT in stages according to the above description.

Ihis **FFT program** uses a standard three-loop structure for the main FFT computation. The outer loop steps through the stages (each column in Figure 3.29); the middle loop steps through "**flights**" (butterflies with the same twiddle factor from each short-length DFT at each stage), and the inner loop steps through the individual butterflies. This ordering minimizes the number of fetches or computations of the twiddle-factor values. Since the bit-reverse of a bit-reversed index is the original index, bit-reversal can be performed fairly simply by swapping pairs of data.

While of *O*(***NlogN***) complexity and thus much faster than a direct DFT, this simple program is optimized for clarity, not for speed. A speed-optimized program making use of additional **efficient FFT algorithm and programming tricks** will compute a DFT several times faster on most machines.

```
/*************************************************************/
/* fft.c */
/* (c) Douglas L. Jones */
/* University of Illinois at Urbana-Champaign */
/* January 19, 1992 */
/* */
/* fft: in-place radix-2 DIT DFT of a complex input */
/* */
/* input: */
/* n: length of FFT: must be a power of two */
/* m: n = 2**m */
/* input/output */
/* x: double array of length n with real part of data */
/* y: double array of length n with imag part of data */
```

```
/* */

/* Permission to copy and use this program is granted */

/* under a Creative Commons "Attribution" license */

/* http://creativecommons.org/licenses/by/1.0/ */

/************************************************************/

fft(n,m,x,y)

int n,m;

double x[],y[];

{

int i,j,k,n1,n2;

double c,s,e,a,t1,t2;

j = 0; /* bit-reverse */

n2 = n/2;

for (i=1; i < n - 1; i++)

{

n1 = n2;

while ( j >= n1 )

{

j = j - n1;

n1 = n1/2;

}

j = j + n1;

if (i < j)

{

t1 = x[i];
```

```
x[i] = x[j];
x[j] = t1;
t1 = y[i];
y[i] = y[j];
y[j] = t1;
}
}
n1 = 0; /* FFT */
n2 = 1;
for (i=0; i < m; i++)
{
n1 = n2;
n2 = n2 + n2;
e = -6.283185307179586/n2;
a = 0.0;
for (j=0; j < n1; j++)
{
c = cos(a);
s = sin(a);
a = a + e;
for (k=j; k < n; k=k+n2)
{
t1 = c*x[k+n1] - s*y[k+n1];
t2 = s*x[k+n1] + c*y[k+n1];
x[k+n1] = x[k] - t1;
```

```
y[k+n1] = y[k] - t2;

x[k] = x[k] + t1;

y[k] = y[k] + t2;

}

}

}

return;

}
```

Decimation-in-Frequency (DIF) Radix-2 FFT*

The radix-2 decimation-in-frequency and **decimation-in-time** fast Fourier transforms (FFTs) are

the simplest **FFT algorithms**. Like all FFTs, they compute the **discrete Fourier transform**

(DFT)

()

where for notational convenience

. FFT algorithms gain their speed by reusing the

results of smaller, intermediate computations to compute multiple DFT frequency outputs.

Decimation in frequency

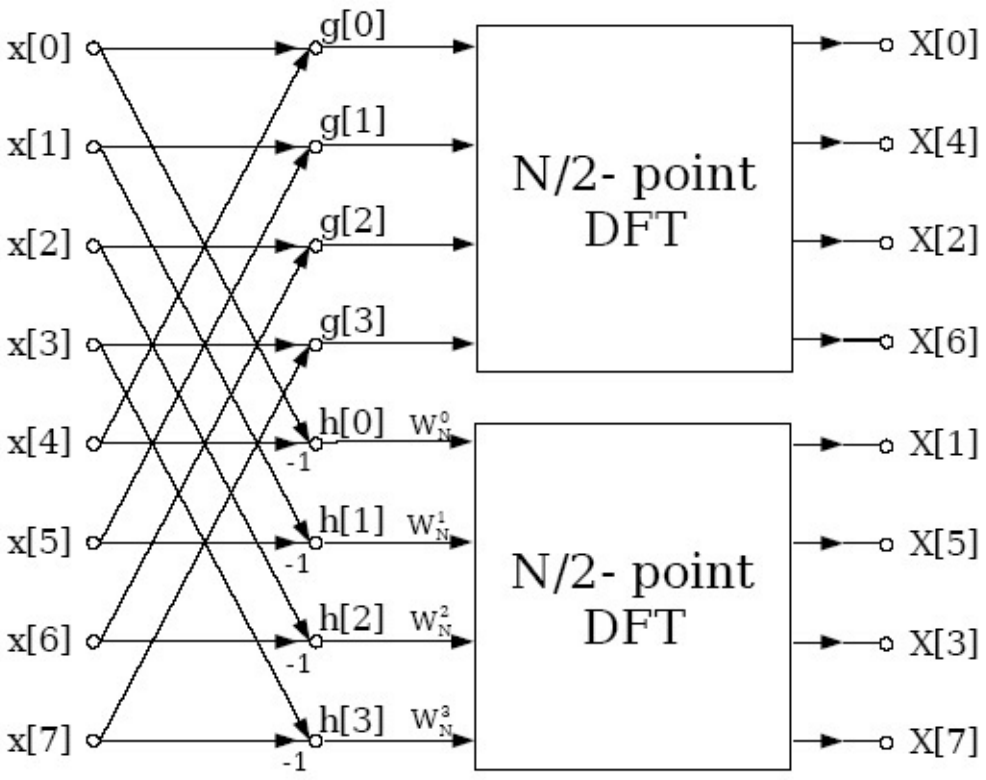
The radix-2 decimation-in-frequency algorithm rearranges the **discrete Fourier transform (DFT)**

equation into two parts: computation of the even-numbered discrete-frequency indices $X(k)$ for

k=[0, 2, 4, …, N−2] (or $X(2r)$ as in Equation) and computation of the odd-numbered indices k=[1, 3, 5, …, N−1] (or $X(2r+1)$ as in Equation)

()

()



The mathematical simplifications in Equation and Equation reveal that both the even-indexed and odd-indexed frequency outputs $X(k)$ can each be computed by a length- DFT. The inputs to these

DFTs are sums or differences of the first and second halves of the input signal, respectively,

where the input to the short DFT producing the odd-indexed frequencies is multiplied by a so-

called **twiddle factor** term

. This is called a **decimation in frequency** because the

frequency samples are computed separately in alternating groups, and a **radix-2** algorithm

because there are two groups. Figure 3.30 graphically illustrates this form of the DFT

computation. This conversion of the full DFT into a series of shorter DFTs with a simple

preprocessing step gives the decimation-in-frequency FFT its computational savings.

Figure 3.30.

Decimation in frequency of a length- N DFT into two length-

DFTs preceded by a preprocessing stage.

Whereas direct computation of all N DFT frequencies according to the **DFT equation** would require N 2 complex multiplies and N 2− N complex additions (for complex-valued data), by

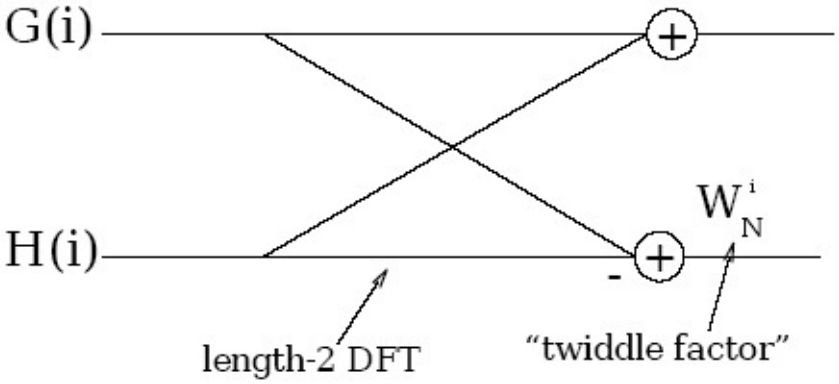
breaking the computation into two short-length DFTs with some preliminary combining of the

data, as illustrated in Figure 3.30, the computational cost is now

New Operation Counts

complex multiplies

complex additions



This simple manipulation has reduced the total computational cost of the DFT by almost a factor of two!

The initial combining operations for both short-length DFTs involve parallel groups of two time samples, $x(n)$ and . One of these so-called **butterfly** operations is illustrated in Figure 3.31.

There are butterflies per **stage**, each requiring a complex addition and subtraction followed by one **twiddle-factor** multiplication by on the lower output branch.

Figure 3.31.

DIF butterfly: twiddle factor after length-2 DFT

It is worthwhile to note that the initial add/subtract part of the DIF butterfly is actually a length-2 DFT! The theory of **multi-dimensional index maps** shows that this must be the case, and that FFTs of any factorable length may consist of successive stages of shorter-length FFTs with twiddle-factor multiplications in between. It is also worth noting that this butterfly differs from the **decimation-in-time radix-2 butterfly** in that the twiddle factor multiplication occurs **after** the combining.

Radix-2 decimation-in-frequency algorithm

The same radix-2 decimation in frequency can be applied recursively to the two length- **DFTs** to save additional computation. When successively applied until the shorter and shorter DFTs reach

length-2, the result is the **radix-2 decimation-in-frequency FFT algorithm**.

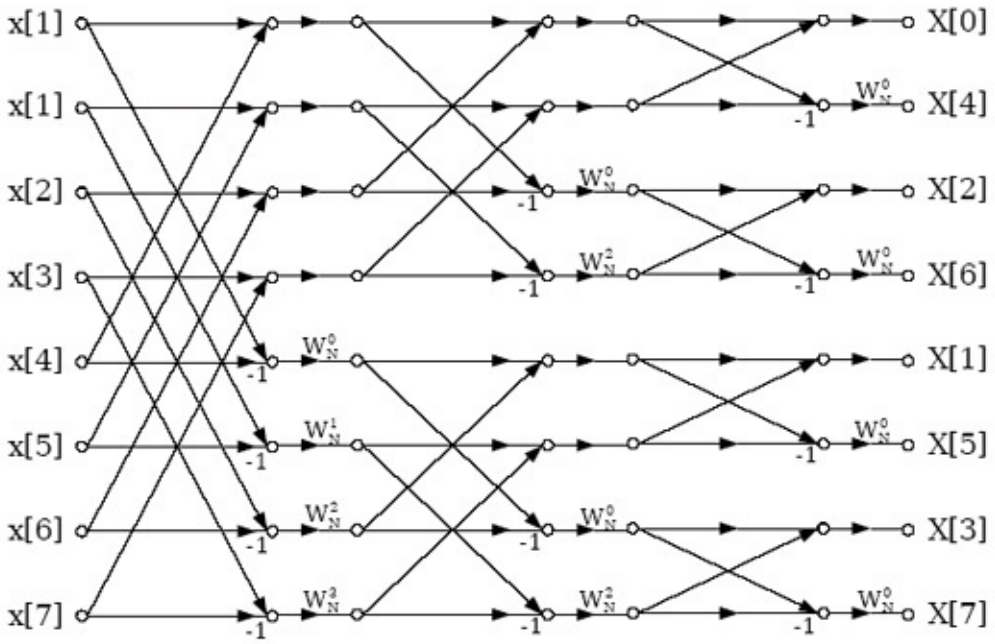


Figure 3.32.

Radix-2 decimation-in-frequency FFT for a length-8 signal

The full radix-2 decimation-in-frequency decomposition illustrated in Figure 3.32 requires $M=\log_2 N$ stages, each with butterflies per stage. Each butterfly requires 1 complex multiply and 2 adds per butterfly. The total cost of the algorithm is thus

Computational cost of radix-2 DIF FFT

complex multiplies

$N \log_2 N$ complex adds

This is a remarkable savings over direct computation of the DFT. For example, a length-1024 DFT would require 1048576 complex multiplications and 1047552 complex additions with direct computation, but only 5120 complex multiplications and 10240 complex additions using the radix-2 FFT, a savings by a factor of 100 or more. The relative savings increase with longer FFT lengths, and are less for shorter lengths. Modest additional reductions in computation can be achieved by noting that certain twiddle factors, namely W^0_N,

,

,

, require no

multiplications, or fewer real multiplies than other ones. By implementing special butterflies for these twiddle factors as discussed in **FFT algorithm and programming tricks**, the computational cost of the radix-2 decimation-in-frequency FFT can be reduced to

$2N \log_2 N - 7N + 12$ real multiplies

$3N \log_2 N - 3N + 4$ real additions

The decimation-in-frequency FFT is a flow-graph reversal of the **decimation-in-time** FFT: it has the same twiddle factors (in reverse pattern) and the same operation counts.

In a decimation-in-frequency radix-2 FFT as illustrated in Figure 3.32, the output is in **bit-reversed** order (hence "decimation-in-frequency"). That is, if the frequency-sample index n is written as a binary number, the order is that binary number reversed. The bit-reversal process is illustrated **here**.

It is important to note that, if the input data are in order before beginning the FFT computations, the outputs of each butterfly throughout the computation can be placed in the same memory locations from which the inputs were fetched, resulting in an **in-place algorithm** that requires no extra memory to perform the FFT. Most FFT implementations are in-place, and overwrite the input data with the intermediate values and finally the output.

Alternate FFT Structures*

Bit-reversing the input in **decimation-in-time (DIT) FFTs** or the output in **decimation-in-frequency (DIF) FFTs** can sometimes be inconvenient or inefficient. For such situations, alternate FFT structures have been developed. Such structures involve the same mathematical computations as the standard algorithms, but alter the memory locations in which intermediate values are stored or the order of computation of the **FFT butterflies**.

The structure in Figure 3.33 computes a **decimation-in-frequency FFT**, but remaps the memory usage so that the **input** is **bit-reversed**, and the output is in-order as in the conventional **decimation-in-time FFT**. This alternate structure is still considered a DIF FFT because the **twiddle factors** are applied as in the **DIF FFT**. This structure is useful if for some reason the DIF butterfly is preferred but it is easier to bit-reverse the input.

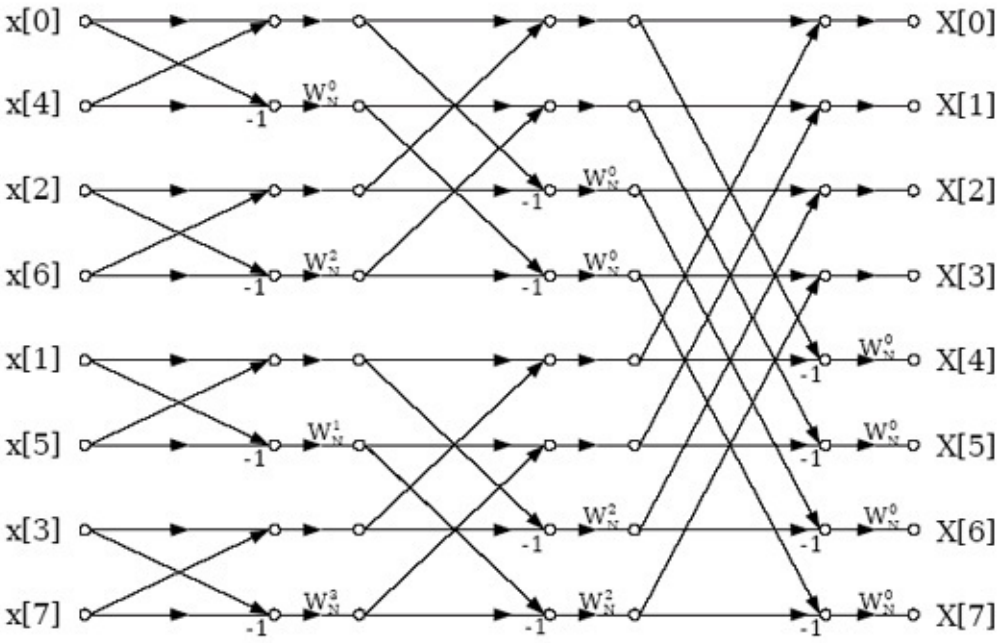


Figure 3.33.

Decimation-in-frequency radix-2 FFT with bit-reversed **input**. This is an **in-place** algorithm in which the same memory can be reused throughout the computation.

There is a similar structure for the **decimation-in-time FFT** with in-order inputs and bit-reversed frequencies. This structure can be useful for **fast convolution** on machines that favor decimation-in-time algorithms because the filter can be stored in bit-reverse order, and then the inverse FFT

returns an in-order result without ever bit-reversing any data. As discussed in **Efficient FFT**

Programming Tricks, this may save several percent of the execution time.

The structure in Figure 3.34 implements a **decimation-in-frequency FFT** that has both input and output in order. It thus avoids the need for bit-reversing altogether. Unfortunately, it destroys the

in-place structure somewhat, making an FFT program more complicated and requiring more

memory; on most machines the resulting cost exceeds the benefits. This structure can be

computed in place if **two** butterflies are computed simultaneously.

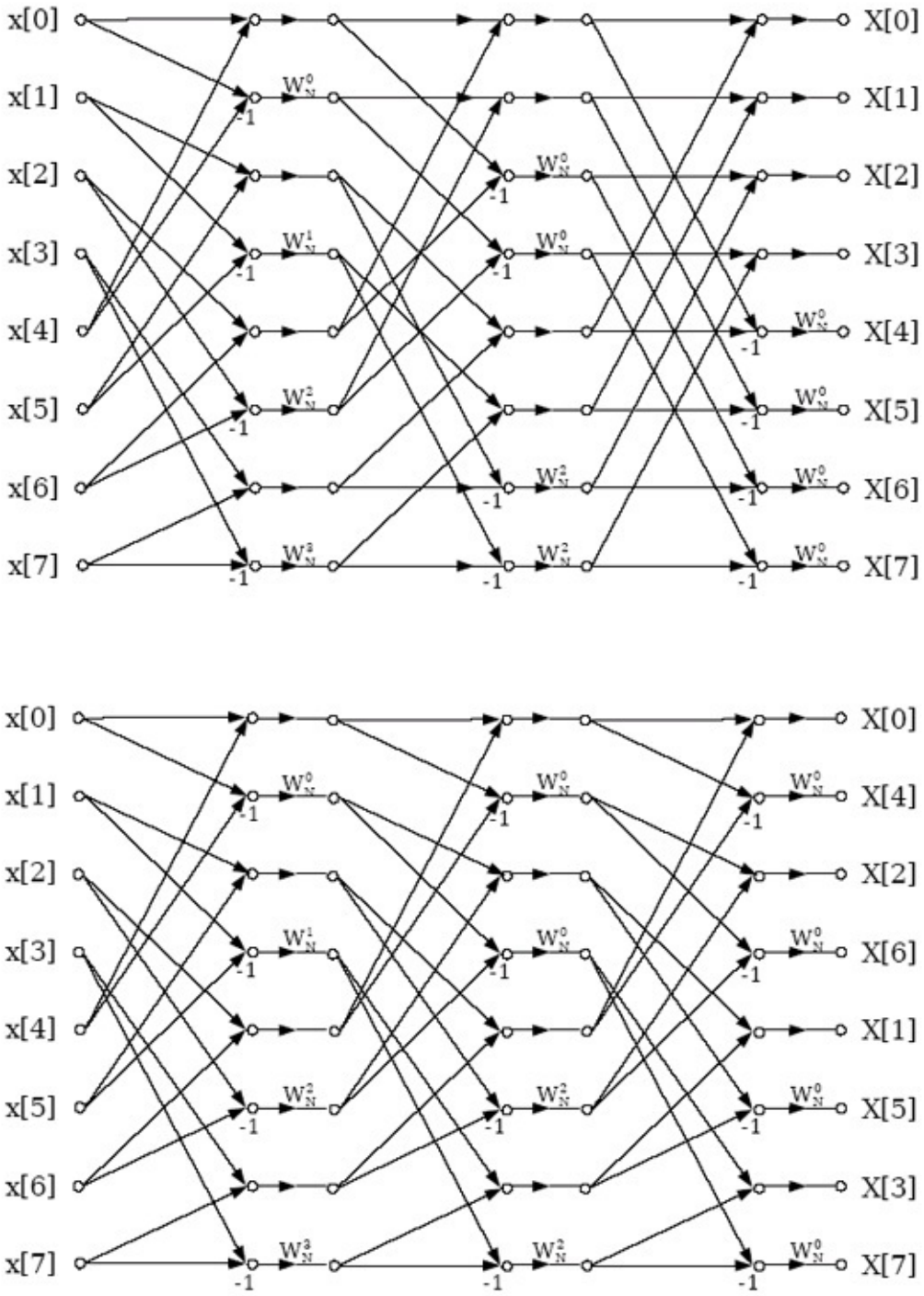


Figure 3.34.

Decimation-in-frequency radix-2 FFT with in-order input and output. It can be computed in-place if two butterflies are computed

simultaneously.

The structure in Figure 3.35 has a constant geometry; the connections between memory locations

are identical in each **FFT stage**. Since it is not in-place and requires bit-reversal, it is inconvenient for software implementation, but can be attractive for a highly parallel hardware implementation

because the connections between stages can be hardwired. An analogous structure exists that has

bit-reversed inputs and in-order outputs.

Figure 3.35.

This constant-geometry structure has the same interconnect pattern from stage to stage. This structure is sometimes useful for special hardware.

Radix-4 FFT Algorithms*

The radix-4 **decimation-in-time** and **decimation-in-frequency** **fast Fourier transforms (FFTs)**

gain their speed by reusing the results of smaller, intermediate computations to compute multiple

DFT frequency outputs. The radix-4 decimation-in-time algorithm rearranges the **discrete**

Fourier transform (DFT) equation into four parts: sums over all groups of every fourth discrete-time index n=[0, 4, 8, …, N−4] , n=[1, 5, 9, …, N−3] , n=[2, 6, 10, …, N−2] and n=[3, 7, 11, …, N−1] as in Equation. (This works out only when the FFT length is a multiple of four.) Just as in the **radix-2 decimation-in-time FFT**, further mathematical manipulation shows that the length- N DFT can be computed as the sum of the outputs of four length- DFTs, of the

even-indexed and odd-indexed discrete-time samples, respectively, where three of them are

multiplied by so-called **twiddle factors**

,

, and

.

()

This is called a **decimation in time** because the time samples are rearranged in alternating groups,

and a **radix-4** algorithm because there are four groups. Figure 3.36 graphically illustrates this form of the DFT computation.

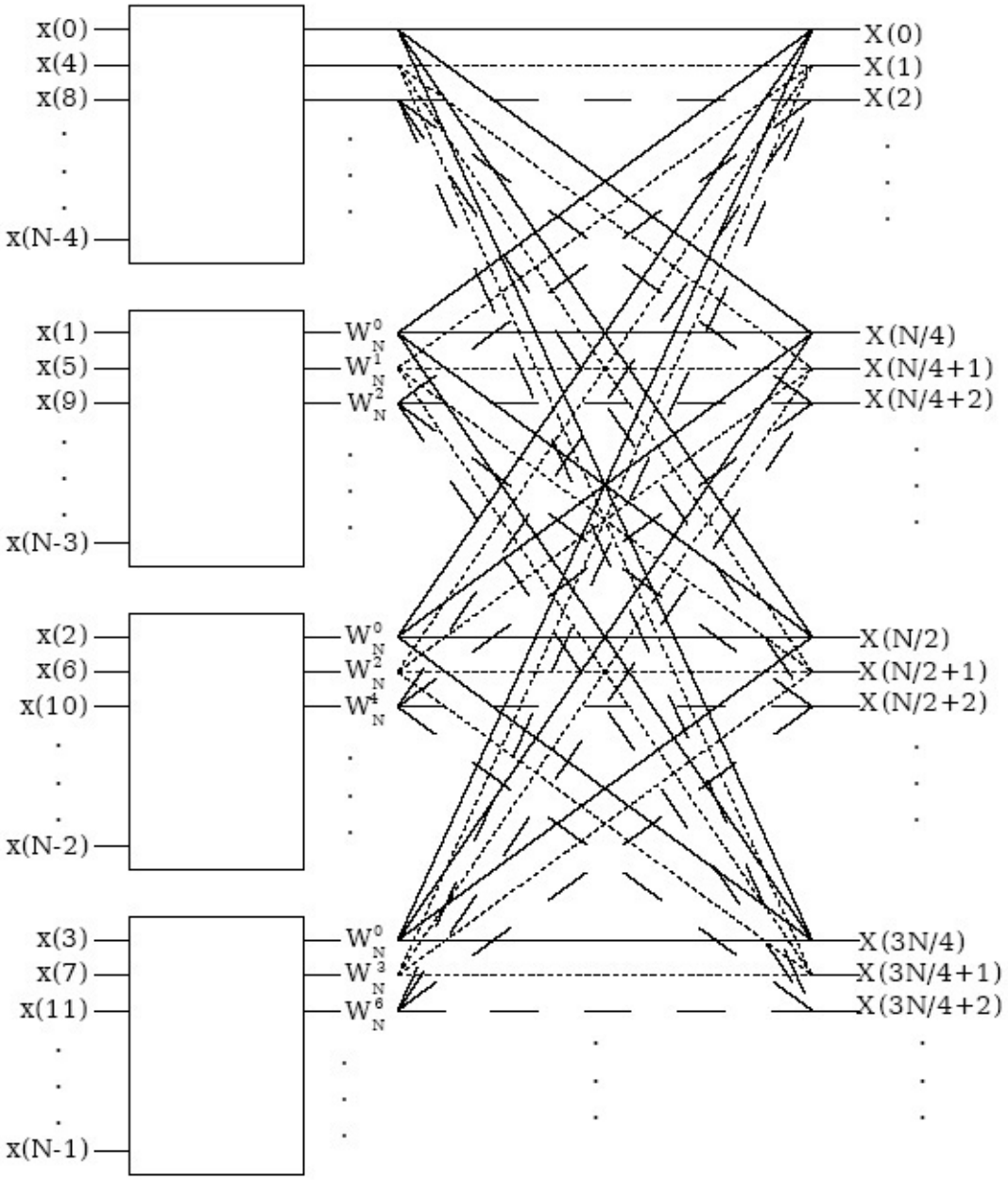


Figure 3.36. Radix-4 DIT structure

Decimation in time of a length- *N* DFT into four length-

DFTs followed by a combining stage.

Due to the periodicity with of the short-length DFTs, their outputs for frequency-sample *k* are

reused to compute *X*(*k*) ,

,

, and

. It is this reuse that gives the radix-4 FFT its

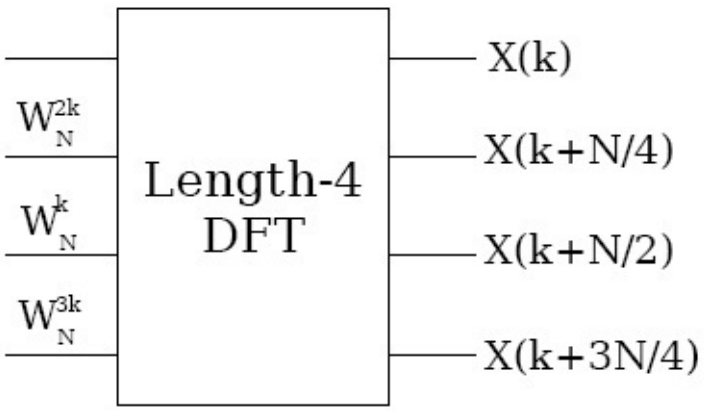
efficiency. The computations involved with each group of four frequency samples constitute the

radix-4 butterfly, which is shown in Figure 3.37. Through further rearrangement, it can be shown that this computation can be simplified to three twiddle-factor multiplies and a length-4 DFT! The

theory of **multi-dimensional index maps** shows that this must be the case, and that FFTs of any factorable length may consist of successive stages of shorter-length FFTs with twiddle-factor

multiplications in between. The length-4 DFT requires no multiplies and only eight complex

additions (this efficient computation can be derived using a **radix-2 FFT**).



(a)

(b)

Figure 3.37.

The radix-4 DIT butterfly can be simplified to a length-4 DFT preceded by three twiddle-factor multiplies.

If the FFT length $N=4M$, the shorter-length DFTs can be further decomposed recursively in the same manner to produce the full **radix-4 decimation-in-time FFT**. As in the **radix-2 decimation-in-time FFT**, each stage of decomposition creates additional savings in computation. To determine the total computational cost of the radix-4 FFT, note that there are

stages, each with butterflies per stage. Each radix-4 butterfly requires 3 complex multiplies and 8 complex additions. The total cost is then

Radix-4 FFT Operation Counts

complex multiplies (75% of a radix-2 FFT)

complex adds (same as a radix-2 FFT)

The radix-4 FFT requires only 75% as many complex multiplies as the **radix-2** FFTs, although it uses the same number of complex additions. These additional savings make it a widely-used FFT algorithm.

The decimation-in-time operation regroups the input samples at each successive stage of decomposition, resulting in a "digit-reversed" input order. That is, if the time-sample index n is written as a base-4 number, the order is that base-4 number reversed. The digit-reversal process is illustrated for a length-$N=64$ example below.

Example 3.4. N = 64 = 4^3

Table 3.2.

Original

Original Digit

Reversed Digit

Digit-Reversed

Number

Order

Order

Number

0

000

000

0

1

001

100

16

2

002

200

32

3

003

300

48

4

010

010

4

5

011

110

20

⋮

⋮

⋮

⋮

It is important to note that, if the input signal data are placed in digit-reversed order before beginning the FFT computations, the outputs of each butterfly throughout the computation can be placed in the same memory locations from which the inputs were fetched, resulting in an **in-place algorithm** that requires no extra memory to perform the FFT. Most FFT implementations are in-place, and overwrite the input data with the intermediate values and finally the output. A slight rearrangement within the radix-4 FFT introduced by *Burrus* **[link]** allows the inputs to be arranged in **bit-reversed** rather than digit-reversed order.

A radix-4 **decimation-in-frequency** FFT can be derived similarly to the **radix-2 DIF FFT,** by separately computing all four groups of every fourth **output** frequency sample. The DIF radix-4 FFT is a flow-graph reversal of the DIT radix-4 FFT, with the same operation counts and twiddle factors in the reversed order. The output ends up in digit-reversed order for an in-place DIF algorithm.

Exercise 5.

How do we derive a radix-4 algorithm when $N=4^M 2$?

Perform a radix-2 decomposition for one stage, then radix-4 decompositions of all subsequent shorter-length DFTs.

References

1. C.S. Burrus. (1988, July). Unscrambling for fast DFT algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *ASSP-36*(7), 1086-1089.

Split-radix FFT Algorithms*

The split-radix algorithm, first clearly described and named by *Duhamel and Hollman* [**link**] in 1984, required fewer total multiply and add operations operations than any previous power-of-two

algorithm. (*Yavne* [**link**] first derived essentially the same algorithm in 1968, but the description was so atypical that the work was largely neglected.) For a time many FFT experts thought it to be

optimal in terms of total complexity, but even more efficient variations have more recently been

discovered by *Johnson and Frigo* [**link**].

The split-radix algorithm can be derived by careful examination of the **radix-2** and **radix-4**

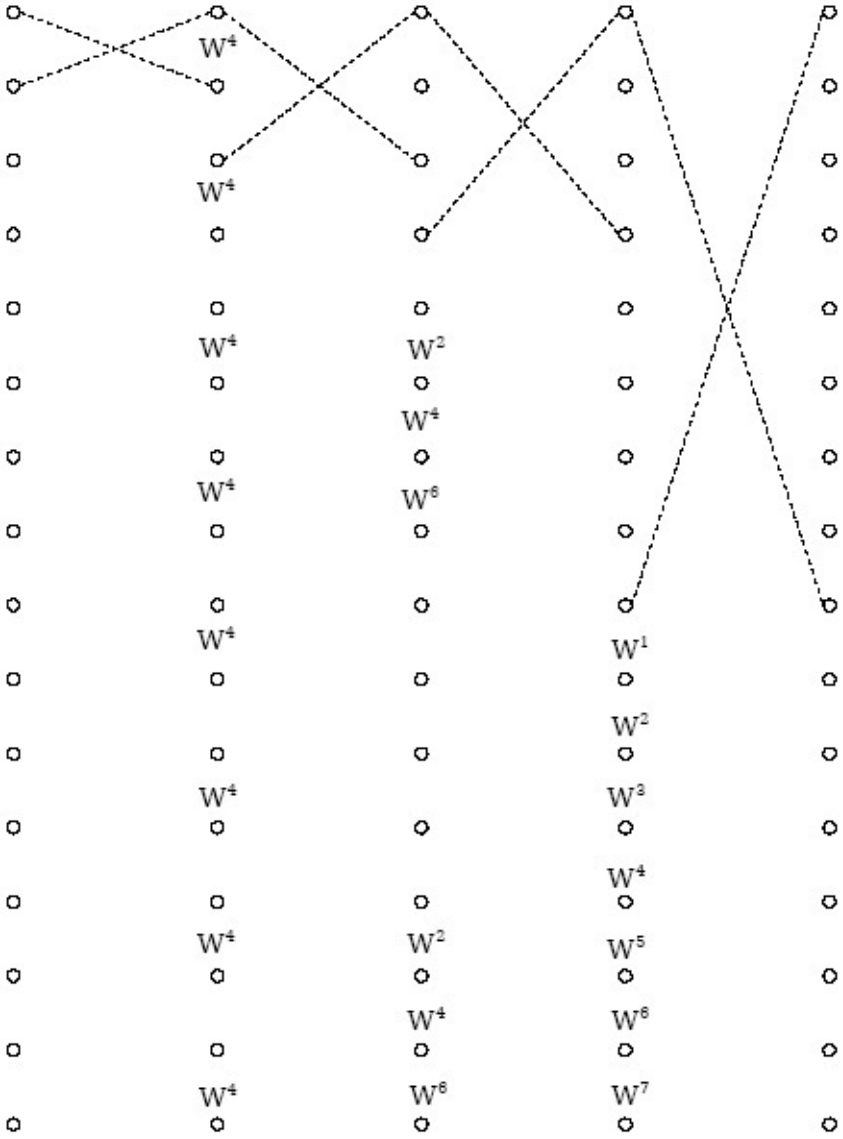
flowgraphs as in Figure 1 below. While in most places the **radix-4** algorithm has fewer nontrivial twiddle factors, in some places the **radix-2** actually lacks twiddle factors present in the **radix-4**

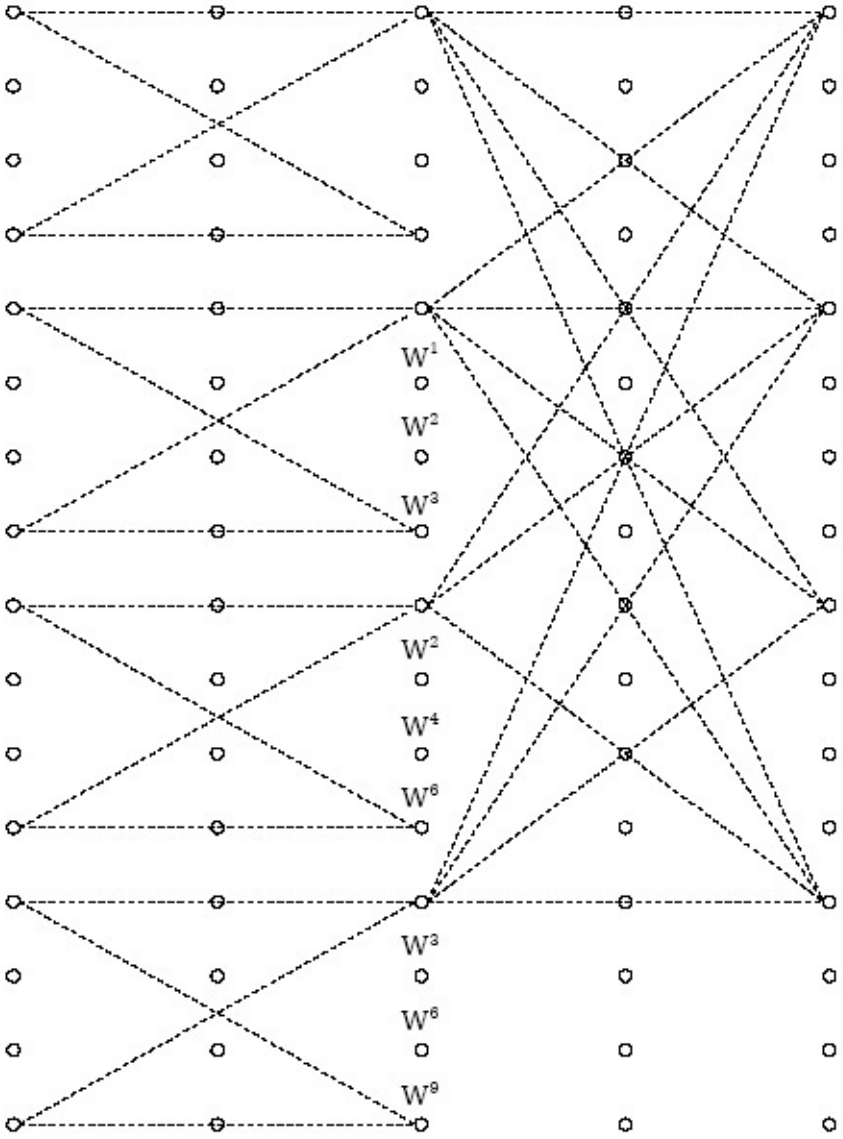
structure or those twiddle factors simplify to multiplication by $-i$, which actually requires only

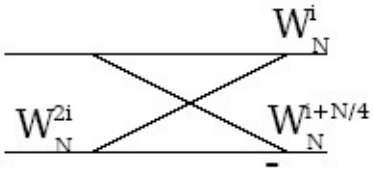
additions. By mixing **radix-2** and **radix-4** computations appropriately, an algorithm of lower complexity than either can be derived.

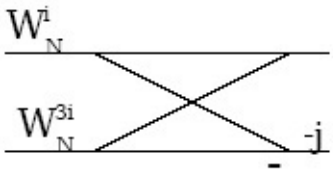
<db:title>radix-2</db:title>

<db:title>radix-4</db:title>


W4
W4
W4
W2
W4
W4
W6
W4
W1
W2
W4
W3
W4
W4
W2
W5
W4
W6
W4
W6
W7







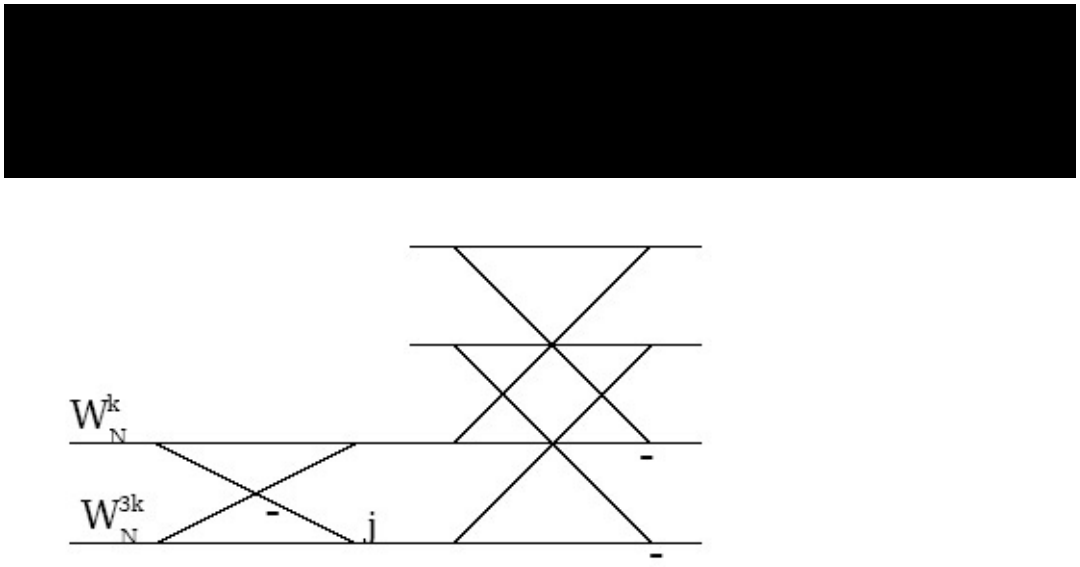
(b)

(a)

Figure 3.38. Motivation for split-radix algorithm

See **Decimation-in-Time (DIT) Radix-2 FFT** and **Radix-4 FFT Algorithms** for more information on these algorithms.

An alternative derivation notes that radix-2 butterflies of the form shown in Figure 2 can merge twiddle factors from two successive stages to eliminate one-third of them; hence, the split-radix algorithm requires only about two-thirds as many multiplications as a radix-2 FFT.



(a)

(b)

Figure 3.39.

Note that these two butterflies are equivalent

The split-radix algorithm can also be derived by mixing the **radix-2** and **radix-4** decompositions.

(3.3)

DIT Split-radix derivation

Figure 3 illustrates the resulting split-radix butterfly.

Figure 3.40. Decimation-in-Time Split-Radix Butterfly

The split-radix butterfly mixes radix-2 and radix-4 decompositions and is L-shaped

Further decomposition of the half- and quarter-length DFTs yields the full split-radix algorithm.

The mix of different-length FFTs in different parts of the flowgraph results in a somewhat

irregular algorithm; *Sorensen et al.* [**link**] show how to adjust the computation such that the data retains the simpler radix-2 bit-reverse order. A decimation-in-frequency split-radix FFT can be

derived analogously.

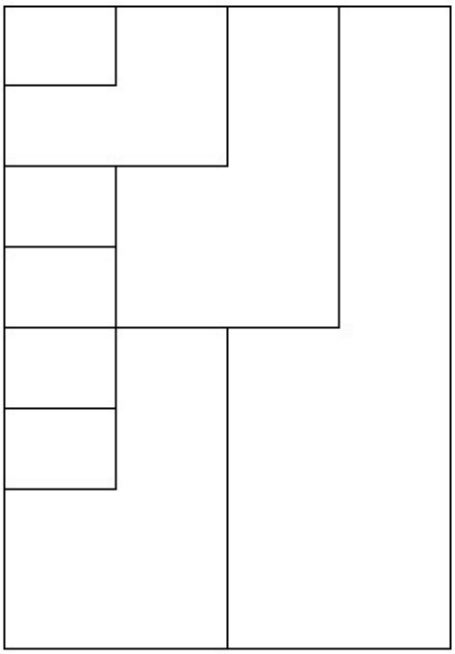


Figure 3.41.

The split-radix transform has L-shaped butterflies

The multiplicative complexity of the split-radix algorithm is only about two-thirds that of the radix-2 FFT, and is better than the radix-4 FFT or any higher power-of-two radix as well. The additions within the complex twiddle-factor multiplies are similarly reduced, but since the underlying butterfly tree remains the same in all power-of-two algorithms, the butterfly additions remain the same and the overall reduction in additions is much less.

Table 3.3. Operation Counts

Complex M/As Real M/As (4/2)

Real M/As (3/3)

Multiplies

$N \log 2\ N - 3\ N + 4$

Additions $O[\ N \log 2\ N]$

$3\ N \log 2\ N - 3\ N + 4$

Comments

The split-radix algorithm has a somewhat irregular structure. Successful progams have been

written (*Sorensen* **[link]**) for uni-processor machines, but it may be difficult to efficiently code the split-radix algorithm for vector or multi-processor machines.

G. Bruun's algorithm [**link**] requires only $N-2$ more operations than the split-radix algorithm and has a regular structure, so it might be better for multi-processor or special-purpose

hardware.

The execution time of FFT programs generally depends more on compiler- or hardware-

The execution time of FFT programs generally depends more on compiler- or hardware-

friendly software design than on the exact computational complexity. See **Efficient FFT**

Algorithm and Programming Tricks for further pointers and links to good code.

References

1. P. Duhamel and H. Hollman. (1984, Jan 5). Split-radix FFT algorithms. *Electronics Letters*,

20, 14-16.

2. R. Yavne. (1968). An economical method for calculating the discrete Fourier transform. *Proc.*

AFIPS Fall Joint Computer Conf., , *33*, 115-125.

3. S.G Johnson and M. Frigo. (2006). A modified split-radix FFT with fewer arithmetic

operations. *IEEE Transactions on Signal Processing*, *54*,

4. H.V. Sorensen, M.T. Heideman, and C.S. Burrus. (1986). On computing the split-radix FFT.

IEEE Transactions on Signal Processing, *34*(1), 152-156.

5. G. Bruun. (1978, February). Z-Transform DFT Filters and FFTs. *IEEE Transactions on Signal Processing*, *26*, 56-63.

Efficient FFT Algorithm and Programming Tricks***

The use of **FFT algorithms** such as the **radix-2 decimation-in-time** or **decimation-in-frequency** methods result in tremendous savings in computations when computing the **discrete Fourier transform**. While most of the speed-up of FFTs comes from this, careful implementation can provide additional savings ranging from a few percent to several-fold increases in program speed.

Precompute twiddle factors

The **twiddle factor**, or , terms that multiply the intermediate data in the **FFT algorithms** consist of cosines and sines that each take the equivalent of several multiplies to compute. However, at most *N* unique twiddle factors can appear in any FFT or DFT algorithm. (For example, in the **radix-2 decimation-in-time FFT**, only twiddle factors are used.) These twiddle factors can be precomputed once and stored in an array in computer memory, and accessed in the FFT algorithm by **table lookup**. This simple technique yields very substantial savings and is almost always used in practice.

Compiler-friendly programming

On most computers, only some of the total computation time of an FFT is spent performing the FFT butterfly computations; determining indices, loading and storing data, computing loop parameters and other operations consume the majority of cycles. Careful programming that allows the compiler to generate efficient code can make a several-fold improvement in the run-time of an FFT. The best choice of radix in terms of program speed may depend more on characteristics of the hardware (such as the number of CPU registers) or compiler than on the exact number of computations. Very often the manufacturer's library codes are carefully crafted by experts who know intimately both the hardware and compiler architecture and how to get the most

performance out of them, so use of well-written FFT libraries is generally recommended. Certain freely available programs and libraries are also very good. Perhaps the best current general-purpose library is the **FFTW** package; information can be found at **http://www.fftw.org**. A paper by *Frigo and Johnson* [**link**] describes many of the key issues in developing compiler-friendly code.

Program in assembly language

While compilers continue to improve, FFT programs written directly in the assembly language of a specific machine are often several times faster than the best compiled code. This is particularly true for DSP microprocessors, which have special instructions for accelerating FFTs that compilers don't use. (I have myself seen differences of up to 26 to 1 in favor of assembly!) Very often, FFTs in the manufacturer's or high-performance third-party libraries are hand-coded in assembly. For DSP microprocessors, the codes developed by *Meyer, Schuessler, and Schwarz* [**link**] are perhaps the best ever developed; while the particular processors are now obsolete, the techniques remain equally relevant today. Most DSP processors provide special instructions and a hardware design favoring the radix-2 decimation-in-time algorithm, which is thus generally fastest on these machines.

Special hardware

Some processors have special hardware accelerators or co-processors specifically designed to accelerate FFT computations. For example, **AMI Semiconductor's Toccata** ultra-low-power DSP microprocessor family, which is widely used in digital hearing aids, have on-chip FFT accelerators; it is always faster and more power-efficient to use such accelerators and whatever radix they prefer.

In a surprising number of applications, almost all of the computations are FFTs. A number of special-purpose chips are designed to specifically compute FFTs, and are used in specialized high-performance applications such as radar systems. Other systems, such as **OFDM**-based communications receivers, have special FFT hardware built into the digital receiver circuit. Such hardware can run many times faster, with much less power consumption, than FFT programs on general-purpose processors.

Effective memory management

Cache misses or excessive data movement between registers and memory can greatly slow down an FFT computation. Efficient programs such as the **FFTW package** are carefully designed to minimize these inefficiences. **In-place algorithms** reuse the data memory throughout the transform, which can reduce cache misses for longer lengths.

Real-valued FFTs

FFTs of real-valued signals require only half as many computations as with complex-valued data. There are several methods for reducing the computation, which are described in more detail in *Sorensen et al.* [**link**]

1. Use **DFT symmetry properties** to do two real-valued DFTs at once with one FFT program

2. Perform one stage of the **radix-2 decimation-in-time** decomposition and compute the two length- DFTs using the above approach.

3. Use a direct real-valued FFT algorithm; see *H.V. Sorensen et.al.* [**link**]

Special cases

Occasionally only certain DFT frequencies are needed, the input signal values are mostly zero, the signal is real-valued (as discussed above), or other special conditions exist for which faster algorithms can be developed. *Sorensen and Burrus* [**link**] describe slightly faster algorithms for **pruned** or **zero-padded** data. **Goertzel's algorithm** is useful when only a few DFT outputs are needed. The **running FFT** can be faster when DFTs of highly overlapped blocks of data are needed, as in a **spectrogram**.

Higher-radix algorithms

Higher-radix algorithms, such as the **radix-4,** radix-8, or **split-radix** FFTs, require fewer computations and can produce modest but worthwhile savings. Even the **split-radix FFT** reduces the multiplications by only 33% and the additions by a much lesser amount relative to the **radix-2 FFTs**; significant improvements in program speed are often due to implicit **loop-unrolling** or other compiler benefits than from the computational reduction itself!

Fast bit-reversal

Bit-reversing the input or output data can consume several percent of the total run-time of an FFT program. Several fast bit-reversal algorithms have been developed that can reduce this to two percent or less, including the method published by *D.M.W. Evans* [**link**].

Trade additions for multiplications

When FFTs first became widely used, hardware multipliers were relatively rare on digital computers, and multiplications generally required many more cycles than additions. Methods to reduce multiplications, even at the expense of a substantial increase in additions, were often beneficial. The **prime factor algorithms** and the **Winograd Fourier transform algorithms**, which required fewer multiplies and considerably more additions than the **power-of-two-length algorithms**, were developed during this period. Current processors generally have high-speed pipelined hardware multipliers, so trading multiplies for additions is often no longer beneficial. In particular, most machines now support single-cycle multiply-accumulate (MAC) operations, so balancing the number of multiplies and adds and combining them into single-cycle MACs generally results in the fastest code. Thus, the prime-factor and Winograd FFTs are rarely used today unless the application requires FFTs of a specific length.

It is possible to implement a complex multiply with 3 real multiplies and 5 real adds rather than the usual 4 real multiplies and 2 real adds: $(C + iS)(X + iY) = CX - SY + i(CY + SX)$ but alernatively $Z = C(X - Y)$ $D = C + S$ $E = C - S$ $CX - SY = EY + Z$ $CY + SX = DX - Z$ In an FFT, D and E come entirely from th twiddle factors, so they can be precomputed and stored in a look-up table. This reduces

the cost of the complex twiddle-factor multiply to 3 real multiplies and 3 real adds, or one less and one more, respectively, than the conventional 4/2 computation.

Special butterflies

Certain twiddle factors, namely $W^0_N = 1$,

,

,

, etc., can be implemented with no additional operations, or with fewer real operations than a general complex multiply. Programs that specially implement such butterflies in the most efficient manner throughout the algorithm can reduce the computational cost by up to several N multiplies and additions in a length-N FFT.

Practical Perspective

When optimizing FFTs for speed, it can be important to maintain perspective on the benefits that can be expected from any given optimization. The following list categorizes the various techniques by potential benefit; these will be somewhat situation- and machine-dependent, but clearly one should begin with the most significant and put the most effort where the pay-off is likely to be largest.

Methods to speed up computation of DFTs

Tremendous Savings:

1. FFT (

savings)

Substantial Savings: (≥ 2)

1. Table lookup of cosine/sine

2. Compiler tricks/good programming

3. Assembly-language programming

4. Special-purpose hardware

5. Real-data FFT for real data (factor of 2)

6. Special cases

Minor Savings:

1. **radix-4, split-radix** (-10% - +30%)

2. special butterflies

3. 3-real-multiplication complex multiply

4. Fast bit-reversal (up to 6%)

Fact

On general-purpose machines, computation is only part of the total run time. Address generation, indexing, data shuffling, and memory access take up much or most of the cycles.

Fact

A well-written **radix-2** program will run much faster than a poorly written **split-radix** program!

References

1. D.M.W. Evans. (1987, August). An improved digit-reversal permutation algorithm for the fast Fourier and Hartley transforms. *IEEE Transactions on Signal Processing*, *35*(8), 1120-1125.

2. M. Frigo and S.G. Johnson. (2005, February). The design and implementation of FFTW3. [redacted]

Proceedings of the IEEE, *93*(2), 216-231.

3. R. Meyer, H.W. Schuessler, and K. Schwarz. (1990). FFT Implmentation on DSP chips - Theory and Practice. *IEEE International Conference on Acoustics, Speech, and Signal Processing*.

4. H.V. Sorensen, D.L Jones, M.T. Heideman, and C.S. Burrus. (1987, June). Real-valued fast Fourier transform algorithms. *IEEE Transactions on Signal Processing*, *35*(6), 849-863.

5. H.V. Sorensen and C.S. Burrus. (1993, March). Efficient computation of the DFT with only a subset of input or output points. *IEEE Transactions on Signal Processing*, *41*(3), 1184-1200.

3.4. Fast Convolution*

Fast Circular Convolution

Since,

$y(n)$ can be computed as

$y(n)=\text{IDFT}[\text{DFT}[x(n)]\text{DFT}[h(n)]]$

Cost

Direct

$N2$ complex multiplies.

$N(N-1)$ complex adds.

Via FFTs

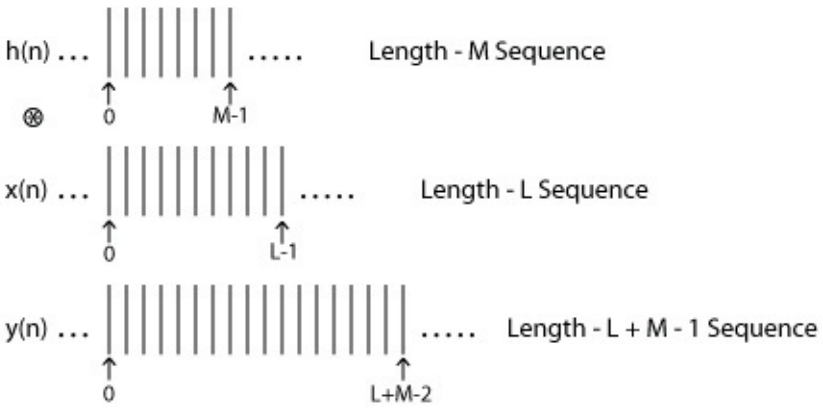
3 FFTs + N multipies.

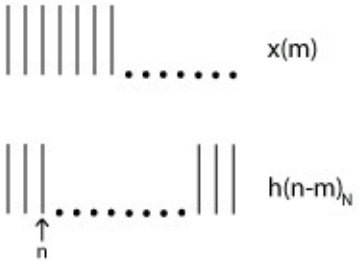
complex multiplies.

$3(N \log 2 N)$ complex adds.

If $H(k)$ can be precomputed, cost is only 2 FFts + N multiplies.

Fast Linear Convolution





[redacted]

DFT produces cicular convolution. For linear convolution, we must zero-pad sequences so that circular wrap-around always wraps over zeros.

Figure 3.42.

To achieve linear convolution using fast circular convolution, we must use zero-padded DFTs of length $N \geq L + M - 1$

Figure 3.43.

Choose shortest convenient N (usually smallest power-of-two greater than or equal to $L + M - 1$)

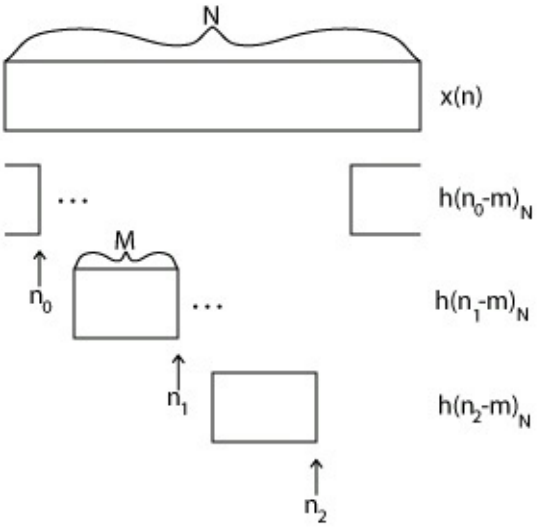
$y(n) = \text{IDFT}_N[\text{DFT}_N[x(n)]\text{DFT}_N[h(n)]]$

There is some inefficiency when compared to circular convolution due to longer zero-padded **DFTs**. Still,

savings over direct computation.

Running Convolution

Suppose $L = \infty$, as in a real time filter application, or ($L \gg M$) . There are efficient block methods for computing fast convolution.



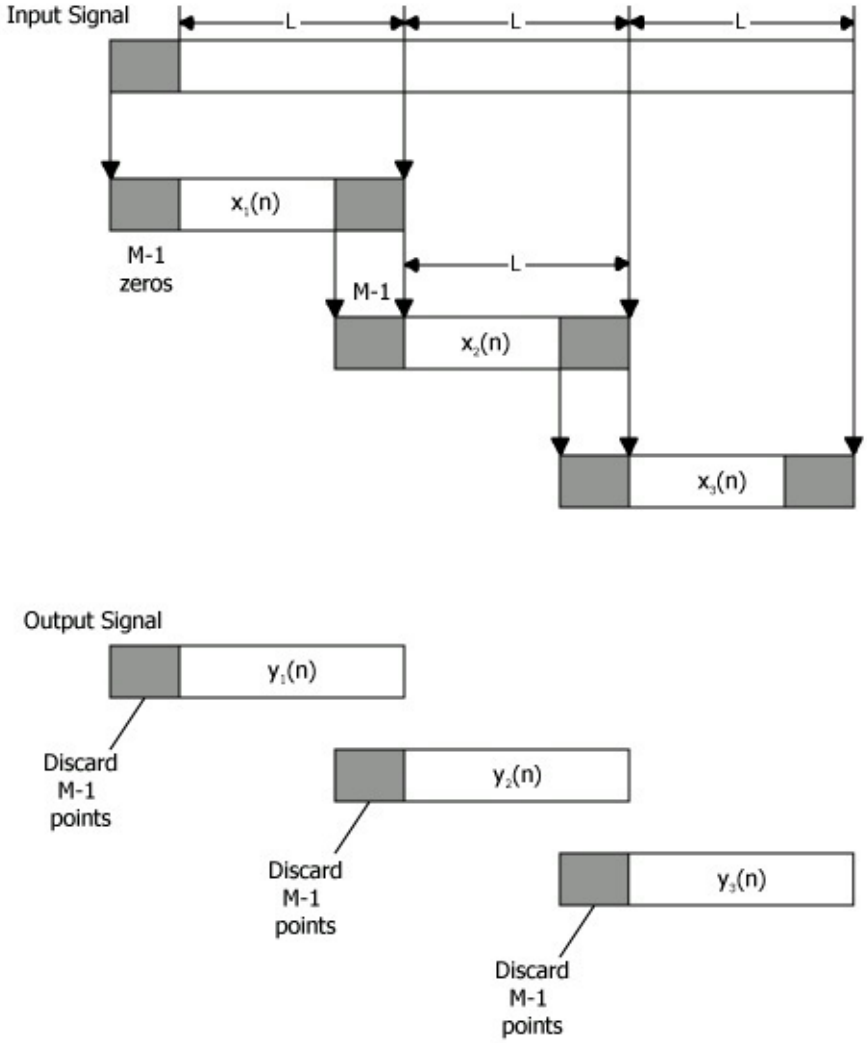
Overlap-Save (OLS) Method

Note that if a length- M filter $h(n)$ is circularly convulved with a length- N segment of a signal $x(n)$,

Figure 3.44.

the first $M-1$ samples are wrapped around and thus is **incorrect**. However, for $M-1 \leq n \leq N-1$, the convolution is linear convolution, so these samples are correct. Thus $N-M+1$ good outputs are produced for each length- N circular convolution.

The Overlap-Save Method: Break long signal into successive blocks of N samples, each block overlapping the previous block by $M-1$ samples. Perform circular convolution of each block with filter $h(m)$. Discard first $M-1$ points in each output block, and concatenate the remaining points to create $y(n)$.



[redacted]

[redacted]

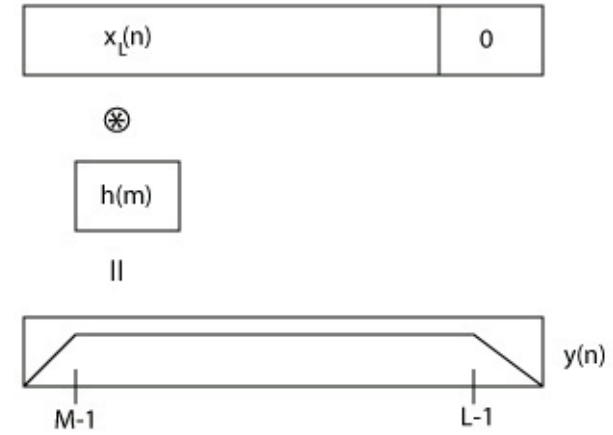
Figure 3.45.

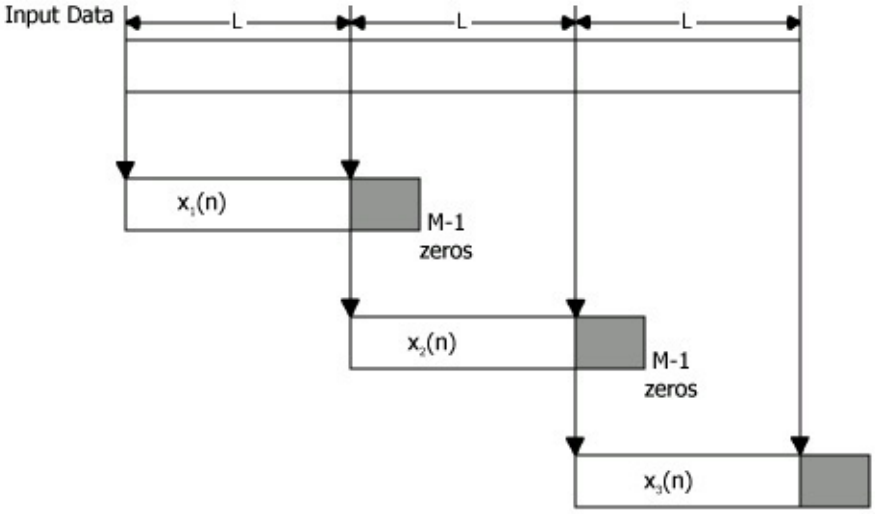
Computation cost for a length- N equals 2 n FFT per output sample is (assuming precomputed $H(k)$) 2 FFTs and N multiplies

Compare to M mults, $M-1$ adds per output point for direct method. For a given M , optimal N can be determined by finding N minimizing operation counts. Usualy, optimal N is $4M \leq N_{opt} \leq 8M$.

Overlap-Add (OLA) Method

Zero-pad length- L blocks by $M-1$ samples.





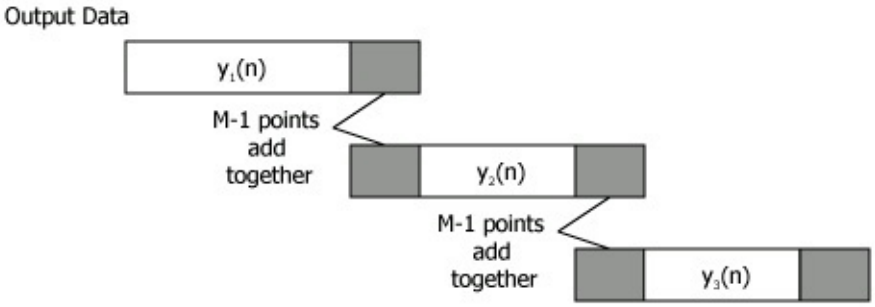
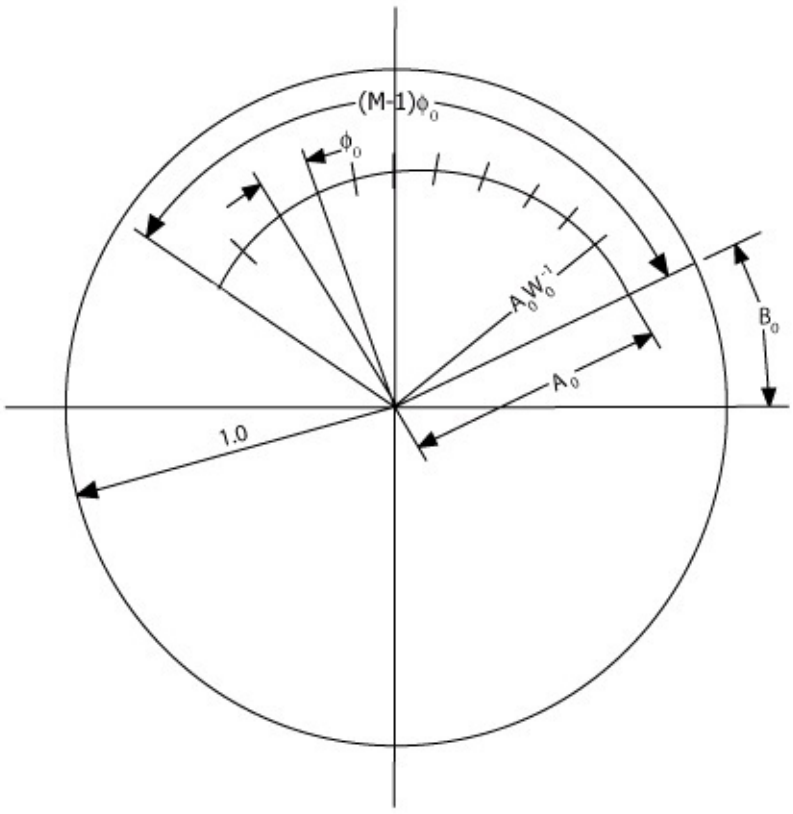


Figure 3.46.

Add successive blocks, overlapped by $M-1$ samples, so that the tails sum to produce the complete linear convolution.

Figure 3.47.

Computational Cost: Two length $N = L + M-1$ FFTs and M mults and $M-1$ adds per L output points; essentially the sames as OLS method.



3.5. Chirp-z Transform*

Let $zk = AW - k$, where $A = Ao e^{i\theta o}$, $W = Wo e^{-(i\varphi o)}$.

We wish to compute M samples, k=[0, 1, 2, …, M−1] of

Figure 3.48.

Note that

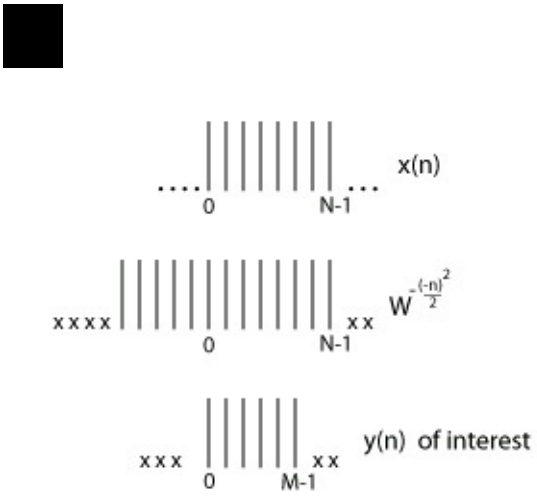
, So

Thus, $X(zk)$ can be compared by

1. Premultiply $x(n)$ by

, $n=[0, 1, \ldots, N-1]$ to make $y(n)$

2. Linearly convolve with



3. Post multiply by to get

to get $X(zk)$.

1. and **3.** require N and M operations respectively. **2.** can be performed efficiently using fast convolution.

Figure 3.49.

is required only for $-((N-1)) \le n \le M-1$, so this linear convolution can be implemented with

$L \ge N+M-1$ FFTs.

Wrap

around L when implementing with circular convolution.

So, a weird-length **DFT** can be implemented relatively efficiently using power-of-two algorithms via the chirp-z transform.

Also useful for "zoom-FFTs".

3.6. FFTs of prime length and Rader's conversion*

The **power-of-two FFT algorithms**, such as the **radix-2** and **radix-4** FFTs, and the **common-factor** and **prime-factor** FFTs, achieve great reductions in computational complexity of the **DFT** when the length, N, is a composite number. DFTs of prime length are sometimes needed, however, particularly for the short-length DFTs in common-factor or prime-factor algorithms. The methods described here, along with the composite-length algorithms, allow fast computation of DFTs of **any** length.

There are two main ways of performing DFTs of prime length:

1. Rader's conversion, which is most efficient, and the

2. **Chirp-z transform**, which is simpler and more general.

Oddly enough, both work by turning prime-length DFTs into convolution! The resulting convolutions can then be computed efficiently by either

1. **fast convolution** via composite-length FFTs (simpler) or by

2. Winograd techniques (more efficient)

Rader's Conversion

Rader's conversion is a one-dimensional **index-mapping** scheme that turns a length- N **DFT** (N prime) into a length-($N-1$) convolution and a few additions. Rader's conversion works **only** for prime-length N.

An **index map** simply rearranges the order of the sum operation in the **DFT definition**. Because addition is a commutative operation, the same mathematical result is produced from any order, as long as all of the same terms are added once and only once. (This is the condition that defines an index map.) Unlike the **multi-dimensional index maps** used in deriving **common factor** and **prime-factor FFTs**, Rader's conversion uses a one-dimensional index map in a finite group of N integers: $k=(rm) \bmod N$

Fact from number theory

If N is prime, there exists an integer " r" called a **primitive root**, such that the index map $k=(rm) \bmod N$, $m=[0, 1, 2, \ldots, N-2]$, uniquely generates all elements $k=[1, 2, 3, \ldots, N-1]$

Example 3.5.

$N=5$, $r=2$ $(20) \bmod 5=1$ $(21) \bmod 5=2$ $(22) \bmod 5=4$ $(23) \bmod 5=3$

Another fact from number theory

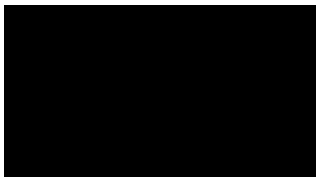
For N prime, the inverse of r (i.e. $(r\text{-}1\ r) \bmod N=1$ is also a primitive root (call it r-1).

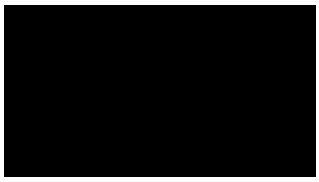
Example 3.6.

[redacted]

[redacted]

[redacted]





N=5 , r=2 r-1=3 (2×3)mod5=1 (30)mod5=1 (31)mod5=3 (32)mod5=4 (33)mod5=2

So why do we care? Because we can use these facts to turn a **DFT** into a convolution!

Rader's Conversion

Let n=(r– m)mod N , m=[0, 1, …, N−2]∧ n∈[1, 2, …, N−1] , k=(rp)mod N , p=[0, 1, …, N−2]∧ k∈[1, 2, …, N−1]

where for convenience

in the DFT equation. For k≠0

()

where l=[0, 1, …, N−2]

Example 3.7.

N=5 , r=2 , r-1=3

where for visibility the matrix

entries represent only the **power**, m of the corresponding DFT term W m

N Note that the 4-by-4

circulant matrix

corresponds to a length-4 circular convolution.

Rader's conversion turns a prime-length **DFT** into a few adds and a **composite-length** (N−1) circular convolution, which can be computed efficiently using either

1. **fast convolution** via FFT and IFFT

2. index-mapped convolution algorithms and short Winograd convolution alogrithms. (Rather complicated, and trades fewer multiplies for many more adds, which may not be worthwile on most modern processors.) See *R.C. Agarwal and J.W. Cooley* **[link]**

Winograd minimum-multiply convolution and DFT algorithms

S. Winograd has proved that a length- *N* circular or linear convolution or **DFT** requires less than 2 *N* multiplies (for real data), or 4 *N* real multiplies for complex data. (This doesn't count multiplies by rational fractions, like 3 or or , which can be computed with additions and one overall scaling factor.) Furthermore, Winograd showed how to construct algorithms achieving these counts. Winograd prime-length DFTs and convolutions have the following characteristics:

1. Extremely efficient for small *N* ($N<20$)

2. The number of adds becomes **huge** for large *N*.

Thus Winograd's minimum-multiply FFT's are useful only for small *N*. They are very important for **Prime-Factor Algorithms**, which generally use Winograd modules to implement the short-length DFTs. Tables giving the multiplies and adds necessary to compute Winograd FFTs for various lengths can be found in *C.S. Burrus (1988)* [**link**]. Tables and FORTRAN and TMS32010 programs for these short-length transforms can be found in *C.S. Burrus and T.W. Parks (1985)* [**link**]. The theory and derivation of these algorithms is quite elegant but requires substantial background in number theory and abstract algebra. Fortunately for the practitioner, all of the short algorithms one is likely to need have already been derived and can simply be looked up without mastering the details of their derivation.

Winograd Fourier Transform Algorithm (WFTA)

The Winograd Fourier Transform Algorithm (WFTA) is a technique that recombines the short Winograd modules in a **prime-factor FFT** into a composite- *N* structure with fewer multiplies but more adds. While theoretically interesting, WFTAs are complicated and different for every length, and on modern processors with hardware multipliers the trade of multiplies for many more adds is

very rarely useful in practice today.

References

1. R.C. Agarwal and J.W. Cooley. (1977, Oct). New Algorithms for Digital Convolution. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, *25*, 392-410.

2. C.S. Burrus. (1988). Efficient Fourier Transform and Convolution Algorithms. In J.S. Lin and A.V. Oppenheim (Eds.), *Advanced Topics in Signal Processing*. Prentice-Hall.

3. C.S. Burrus and T.W. Parks. (1985). *DFT/FFT and Convolution Algorithms*. Wiley-Interscience.

3.7. Choosing the Best FFT Algorithm*

3.7. Choosing the Best FFT Algorithm*

Choosing an FFT length

The most commonly used FFT algorithms **by far** are the **power-of-two-length FFT** algorithms. The **Prime Factor Algorithm (PFA)** and **Winograd Fourier Transform Algorithm (WFTA)** require somewhat fewer multiplies, but the overall difference usually isn't sufficient to warrant the extra difficulty. This is particularly true now that most processors have single-cycle pipelined hardware multipliers, so the total operation count is more relevant. As can be seen from the following table, for similar lengths the split-radix algorithm is comparable in total operations to the Prime Factor Algorithm, and is considerably better than the WFTA, although the PFA and WTFA require fewer multiplications and more additions. Many processors now support single cycle multiply-accumulate (MAC) operations; in the power-of-two algorithms all multiplies can be combined with adds in MACs, so the number of additions is the most relevant indicator of computational cost.

Table 3.4. Representative FFT Operation Counts

FFT length Multiplies (real) Adds(real) Mults + Adds

Radix 2

1024

10248

30728

40976

Split Radix

1024

7172

27652

34824

Prime Factor Alg 1008

5804

29100

34904

Winograd FT Alg 1008

3548

34416

37964

The **Winograd Fourier Transform Algorithm** is particularly difficult to program and is rarely used in practice. For applications in which the transform length is somewhat arbitrary (such as fast convolution or general spectrum analysis), the length is usually chosen to be a power of two. When a particular length is required (for example, in the USA each carrier has exactly 416 frequency channels in each band in the **AMPS** cellular telephone standard), a **Prime Factor Algorithm** for all the relatively prime terms is preferred, with a **Common Factor Algorithm** for other non-prime lengths. **Winograd's short-length modules** should be used for the prime-length factors that are not powers of two. The **chirp z-transform** offers a universal way to compute any length **DFT** (for example, **Matlab** reportedly uses this method for lengths other than a power of two), at a few times higher cost than that of a CFA or PFA optimized for that specific length. The

chirp z-transform, along with **Rader's conversion**, assure us that algorithms of $O(N\log N)$ complexity exist for **any** DFT length N .

Selecting a power-of-two-length algorithm

The choice of a power-of-two algorithm may not just depend on computational complexity. The latest extensions of the **split-radix algorithm** offer the lowest known power-of-two FFT operation counts, but the 10%-30% difference may not make up for other factors such as regularity of structure or data flow, **FFT programming tricks**, or special hardware features. For example, the **decimation-in-time radix-2 FFT** is the fastest FFT on **Texas Instruments'** TMS320C54x DSP microprocessors, because this processor family has special assembly-language instructions that accelerate this particular algorithm. On other hardware, **radix-4 algorithms** may be more efficient. Some devices, such as **AMI Semiconductor's** **Toccata** ultra-low-power DSP microprocessor family, have on-chip FFT accelerators; it is always faster and more power-efficient to use these accelerators and whatever radix they prefer. For **fast convolution**, the **decimation-in-frequency** algorithms may be preferred because the bit-reversing can be bypassed; however, most DSP microprocessors provide zero-overhead bit-reversed indexing hardware and prefer decimation-in-time algorithms, so this may not be true for such machines. Good, compiler- or hardware-friendly programming always matters more than modest differences in raw operation counts, so manufacturers' or good third-party FFT libraries are often the best choice. The module **FFT programming tricks** references some good, free FFT software (including the **FFTW** package) that is carefully coded to be compiler-friendly; such codes are likely to be considerably faster than codes written by the casual programmer.

Multi-dimensional FFTs

Multi-dimensional FFTs pose additional possibilities and problems. The orthogonality and separability of multi-dimensional DFTs allows them to be efficiently computed by a series of one-dimensional FFTs along each dimension. (For example, a two-dimensional DFT can quickly be computed by performing FFTs of each row of the data matrix followed by FFTs of all columns, or vice-versa.) **Vector-radix FFTs** have been developed with higher efficiency per sample than row-

column algorithms. Multi-dimensional datasets, however, are often large and frequently exceed the cache size of the processor, and excessive cache misses may increase the computational time greatly, thus overwhelming any minor complexity reduction from a vector-radix algorithm. Either vector-radix FFTs must be carefully programmed to match the cache limitations of a specific processor, or a row-column approach should be used with matrix transposition in between to ensure data locality for high cache utilization throughout the computation.

Few time or frequency samples

FFT algorithms gain their efficiency through intermediate computations that can be reused to compute many DFT frequency samples at once. Some applications require only a handful of frequency samples to be computed; when that number is of order less than $O(\boldsymbol{logN})$, direct computation of those values via **Goertzel's algorithm** is faster. This has the additional advantage that any frequency, not just the equally-spaced DFT frequency samples, can be selected. *Sorensen and Burrus* [**link]** developed algorithms for when most input samples are zero or only a block of DFT frequencies are needed, but the computational cost is of the same order.

Some applications, such as time-frequency analysis via the **short-time Fourier transform** or **spectrogram,** require DFTs of overlapped blocks of discrete-time samples. When the step-size between blocks is less than $O(\boldsymbol{logN})$, the **running FFT** will be most efficient. (Note that any **window** must be applied via frequency-domain convolution, which is quite efficient for sinusoidal windows such as the **Hamming window.**) For step-sizes of $O(\boldsymbol{logN})$ or greater, computation of the DFT of each successive block via an FFT is faster.

References

1. H.V. Sorensen and C.S. Burrus. (1993). Efficient computation of the DFT with only a subset of input or output points. *IEEE Transactions on Signal Processing*, *41*(3), 1184-1200.

Solutions

[redacted]

[redacted]

▪

Chapter 4. Wavelets

4.1. Time Frequency Analysis and Continuous Wavelet Transform

Why Transforms? *

In the field of signal processing we frequently encounter the use of **transforms**. Transforms are named such because they take a signal and **transform** it into another signal, hopefully one which is easier to process or analyze than the original. Essentially, transforms are used to manipulate signals such that their most important characteristics are made plainly evident. To isolate a signal's important characteristics, however, one must employ a transform that is well matched to that signal. For example, the Fourier transform, while well matched to certain classes of signal, does not efficiently extract information about signals in other classes. This latter fact motivates our development of the wavelet transform.

Limitations of Fourier Analysis*

Let's consider the Continuous-Time Fourier Transform (CTFT) pair:

The Fourier transform pair supplies us with our notion of "frequency." In other words, all of our intuitions regarding the relationship between the time domain and the frequency domain can be traced to this particular transform pair.

It will be useful to view the CTFT in terms of basis elements. The inverse CTFT equation above says that the time-domain signal $x(t)$ can be expressed as a weighted **summation** of basis elements , where $b_\Omega(t) = e^{i\Omega t}$ is the basis element corresponding to frequency Ω. In other words, the basis elements are parameterized by the variable Ω that we call **frequency**. Finally,

$X(\Omega)$ specifies the weighting coefficient for $b_\Omega(t)$. In the case of the CTFT, the number of basis elements is uncountably infinite, and thus we need an integral to express the summation.

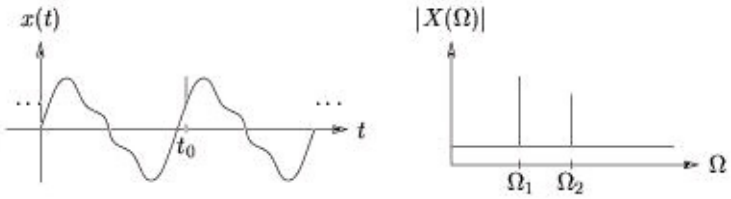
The Fourier Series (FS) can be considered as a special sub-case of the CTFT that applies when the time-domain signal is periodic. Recall that if $x(t)$ is periodic with period T, then it can be expressed as a weighted summation of basis elements

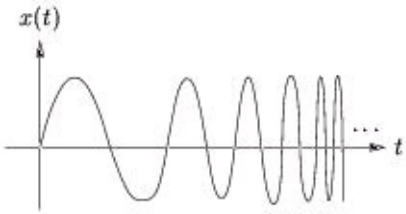
, where

:

[redacted]

[redacted]





Here the basis elements comes from a countably-infinite set, parameterized by the frequency index $k \in \mathbb{Z}$. The coefficients specify the strength of the corresponding basis elements within signal $x(t)$.

Though quite popular, Fourier analysis is not always the best tool to analyze a signal whose characteristics vary with time. For example, consider a signal composed of a periodic component plus a sharp "glitch" at time t 0 , illustrated in time- and frequency-domains, Figure 4.1.

Figure 4.1.

Fourier analysis is successful in reducing the complicated-looking periodic component into a few simple parameters: the frequencies and their corresponding magnitudes/phases. The glitch

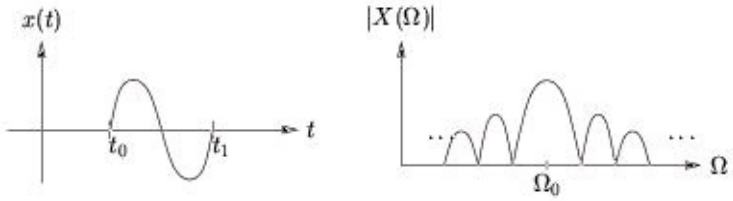
component, described compactly in terms of the time-domain location t 0 and amplitude, however, is not described efficiently in the frequency domain since it produces a wide spread of frequency components. Thus, neither time- nor frequency-domain representations alone give an efficient description of the glitched periodic signal: each representation distills only certain aspects of the signal.

As another example, consider the **linear chirp** $x(t)=\sin(\Omega t\ 2)$ illustrated in Figure 4.2.

Figure 4.2.

Though written using the sin(·) function, the chirp is not described by a single Fourier frequency.

We might try to be clever and write $\sin(\Omega t\ 2)=\sin(\boldsymbol{\Omega t \cdot t})=\sin(\Omega(t)\cdot t)$ where it now seems that signal has an **instantaneous frequency** $\Omega(t)=\boldsymbol{\Omega t}$ which grows linearly in time. But here we must be cautious! Our newly-defined instantaneous frequency $\Omega(t)$ is **not** consistent with the Fourier notion of frequency. Recall that the CTFT says that a signal can be constructed as a superposition of fixed-frequency basis elements $e^{j\boldsymbol{\Omega t}}$ with time support from $-\infty$ to $+\infty$; these elements are evenly spread out over all time, and so there is noting instantaneous about Fourier frequency! So,



[redacted]

[redacted]

[redacted]

while **instantaneous frequency** gives a compact description of the linear chirp, Fourier analysis is not capable of uncovering this simple structure.

As a third example, consider a sinusoid of frequency Ω 0 that is rectangularly windowed to extract only one period (Figure 4.3).

Figure 4.3.

Instantaneous-frequency arguments would claim that

where

$\Omega(t)$ takes on exactly two distinct "frequency" values. In contrast, Fourier theory says that

rectangular windowing induces a frequency-domain spreading by a

profile, resulting in a

continuum of Fourier frequency components. Here again we see that Fourier analysis does not

efficiently decompose signals whose "instantaneous frequency" varies with time.

Time-Frequency Uncertainty Principle*

Recall that Fourier basis elements $b_\Omega(t) = e^{i\Omega t}$ exhibit poor time localization abilities - a

consequence of the fact that $b_\Omega(t)$ is evenly spread over all $t \in (-\infty, \infty)$. By **time localization** we

mean the ability to clearly identify signal events which manifest during a short time interval, such

as the "glitch" described in **an earlier example**.

At the opposite extreme, a basis composed of shifted Dirac deltas $b_\tau(t) = \Delta(t - \tau)$ would have excellent time localization but terrible "frequency localization," since every Dirac basis element is

evenly spread over all Fourier frequencies $\Omega \in [-\infty, \infty]$. This can be seen via

$\forall\ \Omega$, regardless of τ. By **frequency localization** we mean the ability to

clearly identify signal components which are concentrated at particular Fourier frequencies, such

as sinusoids.

These observations motivate the question: does there exist a basis that provides both excellent

frequency localization **and** excellent time localization? The answer is "not really": there is a

fundamental tradeoff between the time localization and frequency localization of any basis

element. This idea is made concrete below.

Let us consider an arbitrary waveform, or basis element, $b(t)$. Its CTFT will be denoted by $B(\Omega)$.

Define the energy of the waveform to be E, so that (by Parseval's theorem)

[redacted]

Next, define the temporal and spectral centers [1] as

and the temporal and spectral widths [2] as

If the waveform is well-localized in time, then $b(t)$ will be

concentrated at the point tc and Δt will be small. If the waveform is well-localized in frequency, then $B(\Omega)$ will be concentrated at the point Ωc and $\Delta\Omega$ will be small. If the waveform is well-localized in both time **and** frequency, then $\Delta t \Delta\Omega$ will be small. The quantity $\Delta t \Delta\Omega$ is known as the **time-bandwidth product**.

From the definitions above one can derive the fundamental properties below. When interpreting the properties, it helps to think of the waveform $b(t)$ as a prototype that can be used to generate an entire basis set. For example, the Fourier basis

can be generated by frequency shifts

of $b(t)=1$, while the Dirac basis

can be generated by time shifts of $b(t) = \delta(t)$

1. Δt and $\Delta\Omega$ are invariant to time and frequency [3] shifts.

$\Delta t(b(t)) = \Delta t(b(t - t_0))$, $t_0 \in \mathbb{R}$ $\Delta\Omega(B(\Omega)) = \Delta\Omega(B(\Omega - \Omega_0))$, $\Omega_0 \in \mathbb{R}$ This implies that all basis elements constructed from time and/or frequency shifts of a prototype waveform $b(t)$ will

inherit the temporal and spectral widths of $b(t)$.

2. The **time-bandwidth product** $\Delta t \Delta\Omega$ is invariant to time-scaling. [4]

$\Delta\Omega(b(\boldsymbol{at})) = |a| \Delta\Omega(b(t))$ The above two equations imply

$\Delta t \Delta\Omega(b(\boldsymbol{at})) = \Delta t \Delta\Omega(b(t))$, $a \in \mathbb{R}$ Observe that time-domain expansion (*i.e.* , $|a| < 1$) increases the temporal width but decreases the spectral width, while time-domain contraction (*i.e.* ,

$|a| > 1$) does the opposite. This suggests that time-scaling might be a useful tool for the design

of a basis element with a particular tradeoff between time and frequency resolution. On the

other hand, scaling cannot simultaneously increase both time **and** frequency resolution.

3. No waveform can have time-bandwidth product less than .

This is known as the

time-frequency uncertainty principle.

4. The Gaussian pulse $g(t)$ achieves the minimum time-bandwidth product

.

Note that this waveform is neither bandlimited nor time-limited, but reasonable

concentrated in both domains (around the points $t_c = 0$ and $\Omega_c = 0$).

Properties 1 and 2 can be easily verified using the definitions above. Properties 3 and 4 follow

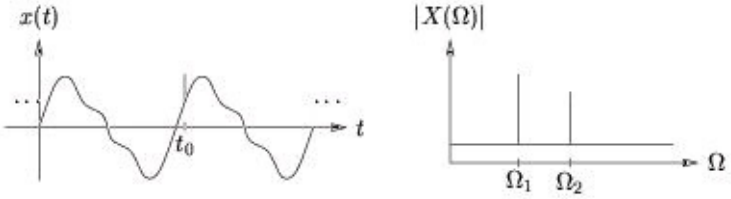
from the **Cauchy-Schwarz inequality**.

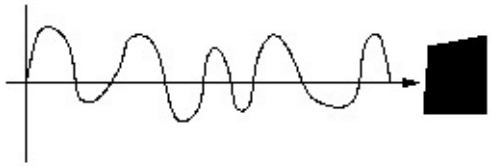
Since the Gaussian pulse $g(t)$ achieves the minimum time-bandwidth product, it makes for a

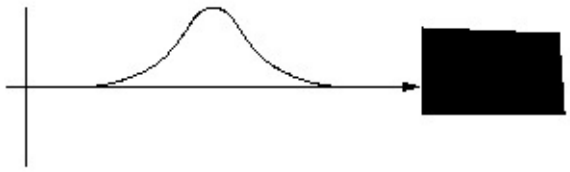
theoretically good prototype waveform. In other words, we might consider constructing a basis

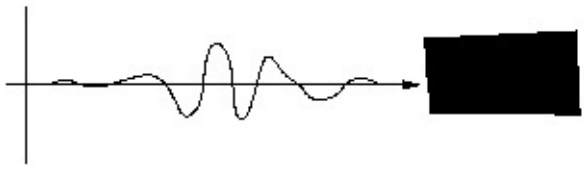
from time shifted, frequency shifted, time scaled, or frequency scaled versions of $g(t)$ to give a

range of spectral/temporal centers and spectral/temporal resolutions. Since the Gaussian pulse has









doubly-infinite time-support, though, other windows are used in practice. Basis construction from a prototype waveform is the main concept behind **Short-Time Fourier Analysis** and the **continuous Wavelet transform** discussed later.

Short-time Fourier Transform*

We saw earlier that Fourier analysis is not well suited to describing local changes in "frequency content" because the frequency components defined by the Fourier transform have infinite (*i.e.* , global) time support. For example, if we have a signal with periodic components plus a glitch at time *t* 0 , we might want accurate knowledge of both the periodic component frequencies **and** the glitch time (Figure 4.4).

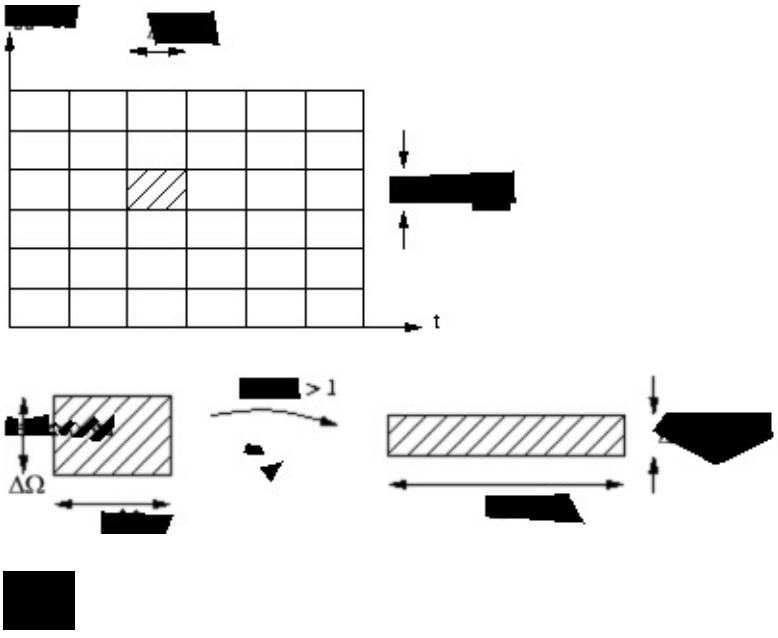
Figure 4.4.

The Short-Time Fourier Transform (STFT) provides a means of joint time-frequency analysis. The STFT pair can be written

assuming real-valued $w(t)$ for which $\int(|w(t)|)2dt=1$. The STFT can be interpreted as a "sliding window CTFT": to calculate X STFT(Ω, τ) , slide the center of window $w(t)$ to time τ, window the input signal, and compute the CTFT of the result (Figure 4.5).

Figure 4.5. "Sliding Window CTFT"

The idea is to isolate the signal in the vicinity of time τ, then perform a CTFT analysis in order to



estimate the "local" frequency content at time τ.

Essentially, the STFT uses the basis elements $b\ \Omega\ ,\ \tau\ (\ t)= w(\ t- \tau)\ e^{i\Omega t}$ over the range $t\in(-\infty, \infty)$ and $\Omega\in(-\infty, \infty)$. This can be understood as time and frequency shifts of the window function $w(\ t)$.

The STFT basis is often illustrated by a tiling of the time-frequency plane, where each tile represents a particular basis element (Figure 4.6):

Figure 4.6.

The height and width of a tile represent the spectral and temporal widths of the basis element, respectively, and the position of a tile represents the spectral and temporal centers of the basis element. Note that, while the **tiling diagram** suggests that the STFT uses a discrete set of time/frequency shifts, the STFT basis is really constructed from a continuum of time/frequency shifts.

Note that we can decrease spectral width $\Delta\Omega$ at the cost of increased temporal width Δt by stretching basis waveforms in time, although the time-bandwidth product $\Delta t\Delta\Omega$ (*i.e.* , the area of each tile) will remain constant (Figure 4.7).

Figure 4.7.

Our observations can be summarized as follows:

the time resolutions and frequency resolutions of every STFT basis element will equal those of the window $w(t)$. (All STFT tiles have the same shape.)

the use of a wide window will give good frequency resolution but poor time resolution, while the use of a narrow window will give good time resolution but poor frequency resolution. (When tiles are stretched in one direction they shrink in the other.)

The combined time-frequency resolution of the basis, proportional to

, is determined not

[redacted]

[redacted]

by window width but by window shape. Of all shapes, the Gaussian [5] gives the highest time-frequency resolution, although its infinite time-support makes it impossible to implement. (The Gaussian window results in tiles with minimum area.)

Finally, it is interesting to note that the STFT implies a particular definition of **instantaneous frequency**. Consider the linear chirp $x(t)=\sin(\Omega_0 t^2)$. From casual observation, we might expect an instantaneous frequency of $\Omega_0 \tau$ at time τ since $\sin(\Omega_0 t^2)=\sin(\Omega_0 \tau t)$, $t=\tau$ The STFT, however, will indicate a time- τ instantaneous frequency of

Caution

The phase-derivative interpretation of instantaneous frequency only makes sense for signals containing exactly **one** sinusoid, though! In summary, always remember that the traditional notion of "frequency" applies only to the CTFT; we must be very careful when bending the notion to include, *e.g.* , "instantaneous frequency", as the results may be unexpected!

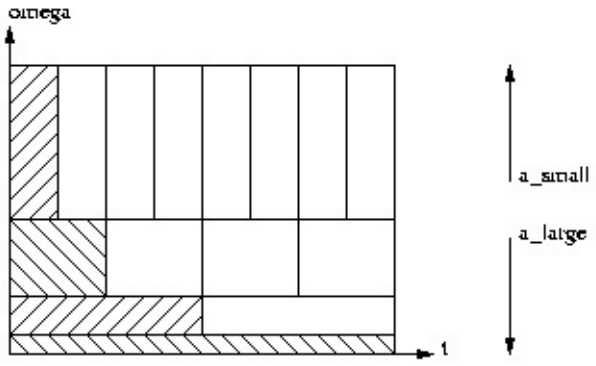
Continuous Wavelet Transform*

The STFT provided a means of (joint) time-frequency analysis with the property that

spectral/temporal widths (or resolutions) were the same for all basis elements. Let's now take a closer look at the implications of uniform resolution.

Consider two signals composed of sinusoids with frequency 1 Hz and 1.001 Hz, respectively. It may be difficult to distinguish between these two signals in the presence of background noise unless many cycles are observed, implying the need for a many-second observation. Now consider two signals with pure frequencies of 1000 Hz and 1001 Hz-again, a 0.1% difference. Here it should be possible to distinguish the two signals in an interval of much less than one second. In other words, good frequency resolution requires longer observation times as frequency decreases. Thus, it might be more convenient to construct a basis whose elements have larger temporal width at low frequencies.

The previous example motivates a multi-resolution time-frequency tiling of the form (Figure 4.8):



[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

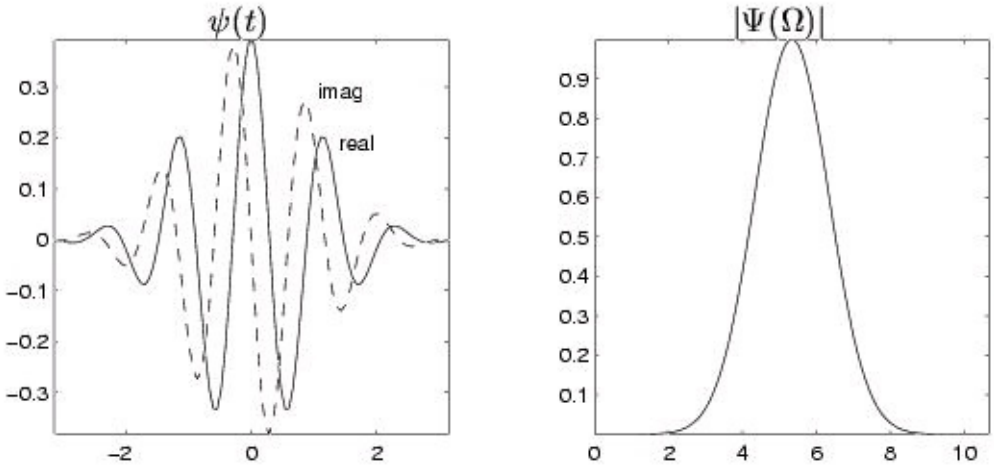


Figure 4.8.

The Continuous Wavelet Transform (CWT) accomplishes the above multi-resolution tiling by time-scaling and time-shifting a prototype function $\psi(t)$, often called the **mother wavelet**. The a-scaled and τ-shifted basis elements is given by

where $a \wedge \tau \in \mathbb{R}$

The conditions above imply that $\psi(t)$ is bandpass and sufficiently smooth.

Assuming that $\|\psi(t)\|=1$, the definition above ensures that $\|\psi_{a,\tau}(t)\|=1$ for all a and τ. The CWT is then defined by the transform pair

In

basis terms, the CWT says that a waveform can be decomposed into a collection of shifted and stretched versions of the mother wavelet $\psi(t)$. As such, it is usually said that wavelets perform a "time-scale" analysis rather than a time-frequency analysis.

The **Morlet wavelet** is a classic example of the CWT. It employs a windowed complex exponential as the mother wavelet:

where it is typical to select

. (See **illustration**.) While this wavelet does not exactly satisfy the conditions

established earlier, since $\Psi(0) \approx 7 \times 10^{-7} \neq 0$, it can be corrected, though in practice the correction is negligible and usually ignored.

Figure 4.9.

While the CWT discussed above is an interesting theoretical and pedagogical tool, the discrete wavelet transform (DWT) is much more practical. Before shifting our focus to the DWT, we take a step back and review some of the basic concepts from the branch of mathematics known as Hilbert Space theory (**Vector Space**, **Normed Vector Space**, **Inner Product Space**, **Hilbert Space**, **Projection Theorem**). These concepts will be essential in our development of the DWT.

4.2. Hilbert Space Theory

Hilbert Space Theory*

Hilbert spaces provide the mathematical foundation for signal processing theory. In this section we attempt to clearly define some key Hilbert space concepts like **vectors**, **norms**, **inner products**, **subspaces**, **orthogonality**, **orthonormal bases**, and **projections**. The intent is not to bury you in mathematics, but to familiarize you with the terminology, provide intuition, and leave you with a "lookup table" for future reference.

Vector Space*

A **vector space** consists of the following four elements:

1. A set of vectors V,

2. A field of scalars (where, for our purposes, is either $\mathbb{R}$ or $\mathbb{C}$),

3. The operations of vector addition "+" (*i.e.* , $+ : V \times V \rightarrow V$)

4. The operation of scalar multiplication "$\cdot$"(*i.e.* , $\cdot : \times V \rightarrow V$)

for which the following properties hold. (Assume $\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{z} \in V$ and $\alpha \wedge \beta \in$.)

Table 4.1.

Properties

Examples

commutativity

$\mathbf{x}+\mathbf{y}=\mathbf{y}+\mathbf{x}$

[redacted]

[redacted]

associativity

$(\mathbf{x}+\mathbf{y})+\mathbf{z}=\mathbf{x}+(\mathbf{y}+\mathbf{z})$

$(\ \boldsymbol{\alpha\beta})\mathbf{x}=\alpha(\ \boldsymbol{\beta x})$

$\alpha\cdot(\mathbf{x}+\mathbf{y})=(\ \alpha\cdot\mathbf{x})+(\ \alpha\cdot\mathbf{y})$

distributivity

$(\ \alpha+\ \beta)\mathbf{x}=\boldsymbol{\alpha x}+\boldsymbol{\beta x}$

additive identity

$\exists 0,0\in V:(\mathbf{x}+0=\mathbf{x})\ ,\ \mathbf{x}\in V$

additive inverse

$\exists -\mathbf{x},-\mathbf{x}\in V:(\mathbf{x}+(-\mathbf{x})=0)\ ,\ \mathbf{x}\in V$

multiplicative identity

$(1\cdot\mathbf{x})=\mathbf{x}\ ,\ \mathbf{x}\in V$

Important examples of vector spaces include

Table 4.2.

Properties

Examples

real N-vectors

$V=\mathbb{R}\ N\ ,\ =\mathbb{R}$

complex N-vectors

V=ℂ N , =ℂ

,

sequences in " lp "

=ℂ

functions in " $\mathcal{L}p$ "

, =ℂ

where we have assumed the usual definitions of addition and multiplication. From now on, we will denote the arbitrary vector space (V, , +, ·) by the shorthand V and assume the usual selection of (, +, ·). We will also suppress the "·" in scalar multiplication, so that ($\alpha \cdot x$) becomes $\boldsymbol{\alpha x}$.

A **subspace** of V is a subset $M \subset V$ for which

1. $\mathbf{x}+\mathbf{y} \in M$, $\mathbf{x} \in M \wedge \mathbf{y} \in M$

2. $\boldsymbol{\alpha x} \in M$, $\mathbf{x} \in M \wedge \alpha \in$

Note that every subspace must contain 0, and that V is a subspace of itself.

The **span** of set $S \subset V$ is the subspace of V containing all linear combinations of vectors in S.

When

,

A subset of linearly-independent vectors

is called a **basis for V** when its span

equals V. In such a case, we say that V has **dimension** N. We say that V is **infinite-dimensional**

[6] if it contains an infinite number of linearly independent vectors.

V is a **direct sum** of two subspaces M and N, written $V=(M \oplus N)$, iff every $\mathbf{x} \in V$ has a **unique** representation $\mathbf{x}=\mathbf{m}+\mathbf{n}$ for $\mathbf{m} \in M$ and $\mathbf{n} \in N$.

Note that this requires $M \bigcap N=\{0\}$

Normed Vector Space*

Now we equip a vector space V with a notion of "size".

A **norm** is a function ($(\|\cdot\| : V \rightarrow \mathbb{R})$) such that the following properties hold (

, $\mathbf{x} \in V \wedge \mathbf{y} \in V$ and , $\alpha \in$):

1. $\|\mathbf{x}\| \geq 0$ with equality iff $\mathbf{x}=0$

2. $\| \alpha \mathbf{x}\|=|\alpha| \cdot \|\mathbf{x}\|$

3. $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\|+\|\mathbf{y}\|$, (the **triangle inequality**).

In simple terms, the norm measures the size of a vector. Adding the norm operation to a vector

space yields a **normed vector space**. Important example include:

1. $V=\mathbb{R}$ N ,

2. $V=\mathbb{C}$ N ,

3. $V=$ lp ,

4. $V=$ $\mathcal{L}p$,

[redacted]

[redacted]

[redacted]

[redacted]

Inner Product Space*

Next we equip a normed vector space V with a notion of "direction".

An **inner product** is a function ($(<\cdot,\cdot> : V\times V)\to\mathbb{C}$) such that the following properties hold (, $\mathbf{x}\in V\wedge\mathbf{y}\in V\wedge\mathbf{z}\in V$ and , $\alpha\in$):

1.

2. $<\mathbf{x}, \boldsymbol{\alpha y}>=\alpha<\mathbf{x}, \mathbf{y}>$ ...implying that

3. $<\mathbf{x}, \mathbf{y}+\mathbf{z}>=<\mathbf{x}, \mathbf{y}>+<\mathbf{x}, \mathbf{z}>$

4. $<\mathbf{x}, \mathbf{x}>\geq 0$ with equality iff $\mathbf{x}=0$

In simple terms, the inner product measures the relative alignment between two vectors.

Adding an inner product operation to a vector space yields an **inner product space**. Important examples include:

1. $V=\mathbb{R}\ N$, $(<\mathbf{x}, \mathbf{y}> \coloneqq \mathbf{x}\mathrm{T}\mathbf{y})$

2. $V=\mathbb{C}\ N$, $(<\mathbf{x}, \mathbf{y}> \coloneqq \mathbf{x}\mathrm{H}\mathbf{y})$

3. $V=l\ 2$,

4. $V=\mathcal{L}\ 2$,

The inner products above are the "usual" choices for those spaces.

The inner product naturally defines a norm:

though not every norm can be

defined from an inner product. [7] Thus, an inner product space can be considered as a normed vector space with additional structure. Assume, from now on, that we adopt the inner-product norm when given a choice.

The **Cauchy-Schwarz inequality** says

$|<\mathbf{x}, \mathbf{y}>|\leq\|\mathbf{x}\|\|\mathbf{y}\|$ with equality iff $\exists, \alpha\in :(\mathbf{x}=\boldsymbol{\alpha y})$.

When $< \mathbf{x}, \mathbf{y} > \in \mathbb{R}$, the inner product can be used to define an "angle" between vectors:

Vectors **x** and **y** are said to be **orthogonal**, denoted as $(\mathbf{x} \perp \mathbf{y})$, when $< \mathbf{x}, \mathbf{y} >= 0$. The **Pythagorean theorem** says: $(\|\mathbf{x}+\mathbf{y}\|)2=(\|\mathbf{x}\|)2+(\|\mathbf{y}\|)2$, $(\mathbf{x} \perp \mathbf{y})$ Vectors **x** and **y** are said to

[redacted]

be **orthonormal** when $(\mathbf{x} \perp \mathbf{y})$ and $\|\mathbf{x}\|=\|\mathbf{y}\|=1$.

$(\mathbf{x} \perp S)$ means $(\mathbf{x} \perp \mathbf{y})$ for all $\mathbf{y} \in S$. S is an **orthogonal set** if $(\mathbf{x} \perp \mathbf{y})$ for all $\mathbf{x} \wedge \mathbf{y} \in S$ s.t. $\mathbf{x} \neq \mathbf{y}$. An orthogonal set S is an **orthonormal set** if $\|\mathbf{x}\|=1$ for all $\mathbf{x} \in S$. Some examples of orthonormal

sets are

1. $\mathbb{R}3$:

2. $\mathbb{C}\ N$: Subsets of columns from unitary matrices

3. $l\ 2$: Subsets of shifted Kronecker delta functions $S \subset \{\{\ \delta[\ n-\ k]\}|\ k \in \mathbb{Z}\}$

4. $\mathcal{L}\ 2$:

for unit pulse $f(\ t)= u(\ t)- u(\ t- T)$, unit step $u(\ t)$

where in each case we assume the usual inner product.

Hilbert Spaces*

Now we consider inner product spaces with nice convergence properties that allow us to define

countably-infinite orthonormal bases.

A **Hilbert space** is a **complete** inner product space. A **complete** [8] space is one where all **Cauchy** sequences **converge** to some vector within the space. For sequence $\{\ xn\}$ to be **Cauchy**,

the distance between its elements must eventually become arbitrarily small:

For a sequence $\{\ xn\}$ to be **convergent to x**, the distance

between its elements and **x** must eventually become arbitrarily small:

Examples are listed below (assuming the usual inner products):

1. $V=\mathbb{R}^N$

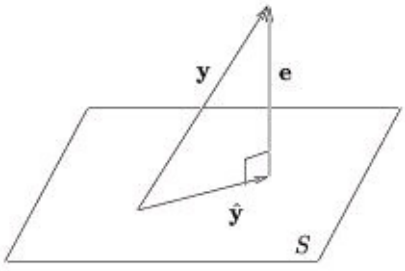
2. $V=\mathbb{C}^N$

3. $V=l^2$ (*i.e.* , square summable sequences)

4. $V=\mathcal{L}^2$ (*i.e.* , square integrable functions)

We will always deal with **separable** Hilbert spaces, which are those that have a countable [9] orthonormal (ON) basis. A countable **orthonormal basis** for V is a countable orthonormal set such that every vector in V can be represented as a linear combination of elements in S:

Due to the orthonormality of S, the basis coefficients are given by



We can see this via:

where

(where the second equality invokes the continuity of the inner product). In finite *n*-dimensional spaces (*e.g.* , $\mathbb{R}n$ or $\mathbb{C}n$), any *n*-element ON set constitutes an ON basis. In infinite-dimensional spaces, we have the following **equivalences**:

1.

is an ON basis

2. If

for all *i*, then **y**=0

3.

(Parseval's theorem)

4. Every $\mathbf{y} \in V$ is a limit of a sequence of vectors in

Examples of countable ON bases for various Hilbert spaces include:

1. $\mathbb{R}n$:

for

with "1" in the *i* th position

2. $\mathbb{C}n$: same as $\mathbb{R}n$

3. $l2$:

, for ($\{\delta i[n]\} \coloneqq \{\delta[n-i]\}$) (all shifts of the Kronecker sequence)

4. $\mathcal{L}2$: to be constructed using wavelets ...

Say *S* is a subspace of Hilbert space *V*. The **orthogonal complement of S in V**, denoted $S\perp$, is

the subspace defined by the set $\{\mathbf{x} \in V | (\mathbf{x} \perp S)\}$. When S is closed, we can write $V = (S \oplus S\perp)$ The **orthogonal projection of y onto S**, where S is a closed subspace of V, is

s.t.

is an ON basis for S. Orthogonal projection yields the best approximation of **y** in S:

The approximation error

obeys the **orthogonality principle**:

($\mathbf{e} \perp S$) We illustrate this concept using $V = \mathbb{R}3$ (Figure 4.10) but stress that the same geometrical interpretation applies to any Hilbert space.

Figure 4.10.

A proof of the orthogonality principle is:

()

4.3. Discrete Wavelet Transform

Discrete Wavelet Transform: Main Concepts*

Main Concepts

The **discrete wavelet transform** (DWT) is a representation of a signal $x(t) \in \mathcal{L}2$ using an

orthonormal basis consisting of a countably-infinite set of **wavelets**. Denoting the wavelet basis as

, the DWT transform pair is

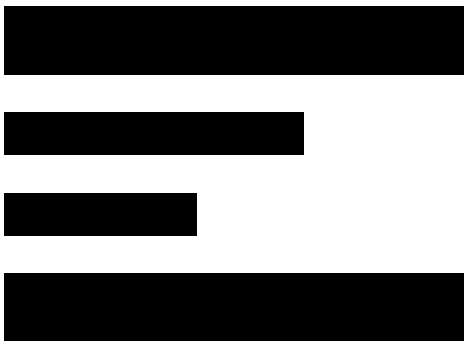
()

()

where { $d\,k$, n } are the wavelet coefficients. Note the relationship to Fourier series and to the sampling theorem: in both cases we can perfectly describe a continuous-time signal $x(\,t)$ using a countably-infinite (*i.e.* , discrete) set of coefficients. Specifically, Fourier series enabled us to describe **periodic** signals using Fourier coefficients $\{\,X[\,k]|\,k\in\mathbb{Z}\}$, while the sampling theorem enabled us to describe **bandlimited** signals using signal samples $\{\,x[\,n]|\,n\in\mathbb{Z}\}$. In both cases, signals within a limited class are represented using a coefficient set with a single countable index. The DWT can describe **any** signal in $\mathcal{L}\,2$ using a coefficient set parameterized by two countable indices:

.

Wavelets are orthonormal functions in $\mathcal{L}\,2$ obtained by shifting and stretching a **mother wavelet**, $\psi(\,t)\in\mathcal{L}\,2$. For example,



()

defines a family of wavelets

related by power-of-two stretches. As k increases,

the wavelet stretches by a factor of two; as n increases, the wavelet shifts right.

When $\|\,\psi(\,t)\|=1$, the normalization ensures that $\|\,\psi\,k\,,\,n\,(\,t)\|=1$ for all $k\in\mathbb{Z}$, $n\in\mathbb{Z}$.

Power-of-two stretching is a convenient, though somewhat arbitrary, choice. In our treatment of

the discrete wavelet transform, however, we will focus on this choice. Even with power-of two stretches, there are various possibilities for $\psi(t)$, each giving a different flavor of DWT.

Wavelets are constructed so that

(*i.e.* , the set of all shifted wavelets at fixed scale k),

describes a particular level of 'detail' in the signal. As k becomes smaller (*i.e.* , closer to $-\infty$), the wavelets become more "fine grained" and the level of detail increases. In this way, the DWT can give a **multi-resolution** description of a signal, very useful in analyzing "real-world" signals.

Essentially, the DWT gives us a **discrete multi-resolution description of a continuous-time signal in** $\mathcal{L}2$.

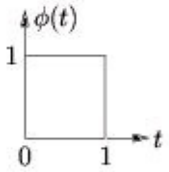
In the modules that follow, these DWT concepts will be developed "from scratch" using Hilbert space principles. To aid the development, we make use of the so-called **scaling function** $\varphi(t) \in \mathcal{L}2$, which will be used to approximate the signal **up to a particular level of detail**. Like with wavelets, a family of scaling functions can be constructed via shifts and power-of-two stretches

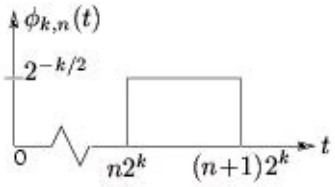
()

given mother scaling function $\varphi(t)$. The relationships between wavelets and scaling functions will be elaborated upon later via **theory** and **example**.

The inner-product expression for $d\ k\ ,\ n$, Equation is written for the general complex-valued case. In our treatment of the discrete wavelet transform, however, we will assume real-valued signals and wavelets. For this reason, we omit the complex conjugations in the remainder of our DWT discussions

The Haar System as an Example of DWT*





The Haar basis is perhaps the simplest example of a DWT basis, and we will frequently refer to it in our DWT development. Keep in mind, however, that **the Haar basis is only an example**; there are many other ways of constructing a DWT decomposition.

For the Haar case, the mother **scaling function** is defined by Equation and Figure 4.11.

()

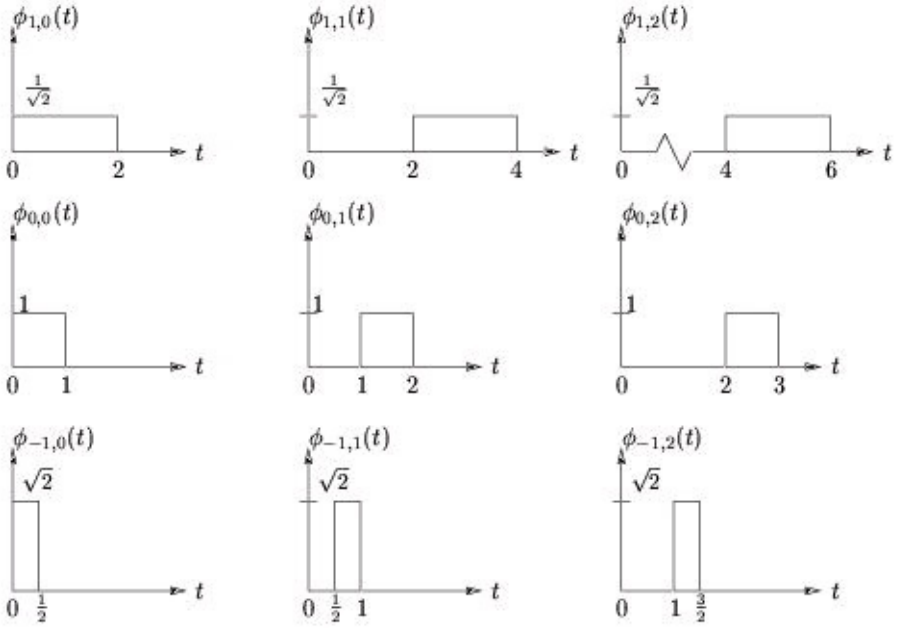
Figure 4.11.

From the mother scaling function, we define a family of shifted and stretched scaling functions $\{ \varphi k , n (t)\}$ according to Equation and Figure 4.12

()

Figure 4.12.

which are illustrated in Figure 4.13 for various *k* and *n*. Equation makes clear the principle that incrementing *n* by one shifts the pulse one place to the right. Observe from Figure 4.13 that is orthonormal for each *k* (*i.e.* , along each row of figures).



[redacted]

[redacted]

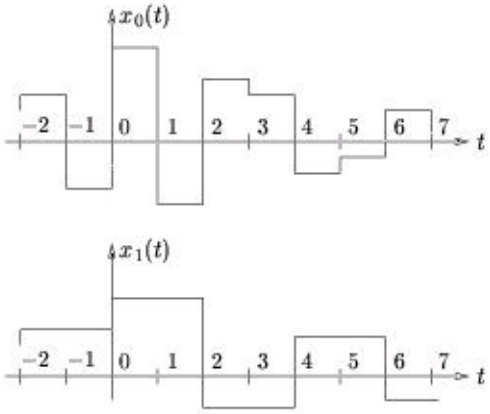


Figure 4.13.

A Hierarchy of Detail in the Haar System*

Given a mother scaling function $\varphi(t) \in \mathcal{L}2$ — the choice of which will be discussed later — let us construct scaling functions at "coarseness-level-k" and "shift- n" as follows:

Let

us then use Vk to denote the subspace defined by linear combinations of scaling functions at the k th level:

In the Haar system, for example, $V0$ and $V1$ consist of signals with

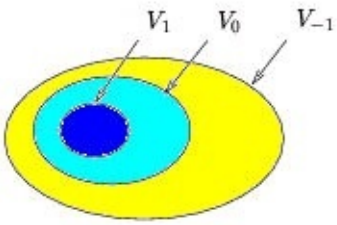
the characteristics of $x0(t)$ and $x1(t)$ illustrated in Figure 4.14.

Figure 4.14.

We will be careful to choose a scaling function $\varphi(t)$ which ensures that the following nesting

property is satisfied: …⊂ V 2⊂ V 1⊂ V 0⊂ V-1⊂ V-2⊂ … coarse detailed In other words, any signal in Vk can be constructed as a linear combination of **more detailed** signals in $V\,k-1$. (The Haar

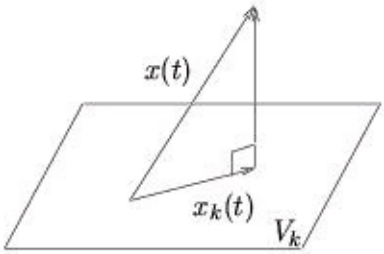
system gives proof that at least one such $\varphi(t)$ exists.)



[redacted]

[redacted]

[redacted]



The nesting property can be depicted using the set-theoretic diagram, Figure 4.15, where $V-1$ is represented by the contents of the largest egg (which includes the smaller two eggs), V 0 is

represented by the contents of the medium-sized egg (which includes the smallest egg), and V 1 is

represented by the contents of the smallest egg.

Figure 4.15.

Going further, we will assume that $\varphi(t)$ is designed to yield the following three important

properties:

1.

constitutes an orthonormal basis for Vk ,

2. $V\infty$={0} (contains no signals). [10]

3. $V - \infty = \mathcal{L}\ 2$ (contains all signals).

Because

is an orthonormal basis, the best (in $\mathcal{L}\ 2$ norm) approximation of $x(\ t) \in \mathcal{L}\ 2$ at

coarseness-level- k is given by the orthogonal projection, Figure 4.16

()

()

$c\ k\ ,\ n = < \varphi\ k\ ,\ n\ (\ t),\ x(\ t) >$

Figure 4.16.

We will soon derive conditions on the scaling function $\varphi(\ t)$ which ensure that the properties above

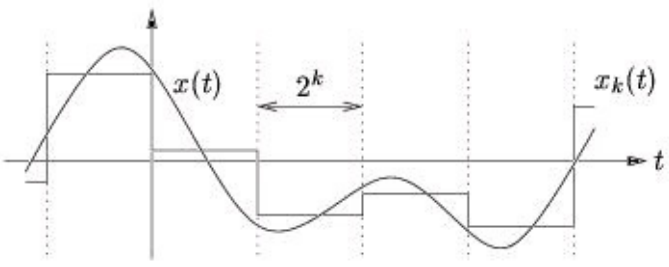
are satisfied.

[redacted]

[redacted]

[redacted]

[redacted]



[redacted]

[redacted]

Haar Approximation at the kth Coarseness Level*

It is instructive to consider the approximation of signal $x(t) \in \mathcal{L}2$ at coarseness-level- k of the Haar system. For the Haar case, projection of $x(t) \in \mathcal{L}2$ onto Vk is accomplished using the basis

coefficients

()

giving the approximation

()

where

This corresponds to taking the

average value of the signal in each interval of width 2 k and approximating the function by a

constant over that interval (see Figure 4.17).

Figure 4.17.

The Scaling Equation*

Consider the level-1 subspace and its orthonormal basis:

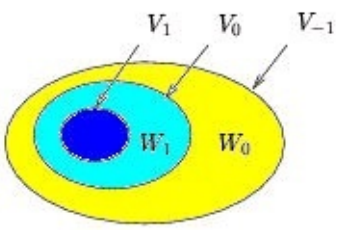
()

()

[redacted]

[redacted]

[redacted]



[redacted]

Since $V\,1 \subset V\,0$ (*i.e.* , $V\,0$ is more detailed than $V\,1$) and since $\varphi\,1\,,\,0\,(\,t) \in V\,0$, there must exist coefficients $\{\,h[\,n]\,|\,n \in \mathbb{Z}\}$ such that

()

()

(4.1)

Scaling Equation

To be a valid scaling function, $\varphi(\,t)$ must obey the scaling equation for some coefficient set

$\{\,h[\,n]\}$.

The Wavelet Scaling Equation*

The **difference** in detail between Vk and $V\,k-1$ will be described using Wk , the orthogonal complement of Vk in $V\,k-1$:

()

$V\,k-1 = (\,Vk \oplus Wk)$

At times it will be convenient to write W

$\perp$

$k = Vk$. This concept is illustrated in the set-theoretic

diagram, Figure 4.18.

Figure 4.18.

Suppose now that, for each $k \in \mathbb{Z}$, we construct an orthonormal basis for Wk and denote it by

. It turns out that, because every Vk has a basis constructed from shifts and stretches of

a mother scaling function (*i.e.* ,

, every Wk has a basis that can be constructed

from shifts and stretches of a "mother wavelet" $\psi(\,t) \in \mathcal{L}\,2$:

The Haar system

will soon provide us with a concrete example .

Let's focus, for the moment, on the specific case $k=1$. Since $W\,1 \subset V\,0$, there must exist

$\{\, g[\, n]\,|\, n \in \mathbb{Z}\}$ such that:

()

(4.2)

Wavelet Scaling Equation

To be a valid scaling-function/wavelet pair, $\varphi(\, t)$ and $\psi(\, t)$ must obey the wavelet scaling equation for some coefficient set $\{\, g[\, n]\}$.

Conditions on h[n] and g[n]*

Here we derive sufficient conditions on the coefficients used in the scaling equation and wavelet

scaling equation that ensure, for every $k \in \mathbb{Z}$, that the sets

and

have the

orthonormality properties described in **The Scaling Equation** and **The Wavelet Scaling Equation**.

For

to be orthonormal at all k, we certainly need orthonormality when $k=1$. This is

equivalent to

()

where $\delta[\, n-\ell+2\, m]=<\varphi(\, t-\, n),\, \varphi(\, t-\, \ell-2\, m)>$ **()**

There is an interesting frequency-domain interpretation of the previous condition. If we define

()

then we see that our condition is equivalent to $p[2\, m]=\delta[\, m]$. In the z-domain, this yields the pair of conditions

(4.3)

Power-Symmetry Property

$P(\, z)=H(\, z)\, H(\, z\text{-}1)$

Putting these together,

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

()

2= $H(z\ 1/2)\ H(z{-}1/2){+}\ H({-}(z\ 1/2))\ H({-}(z{-}1/2))$

⇔ 2= $H(z)\ H(z\text{-}1){+}\ H({-}z)\ H({-}(z\text{-}1))$ ⇔ 2=(| $H(e^{i\omega})$|)2+(| $H(e^{i(\pi-\omega)})$|)2 where the last property invokes the fact that $h[n]\in\mathbb{R}$ and that real-valued impulse responses yield conjugate-symmetric DTFTs.

Thus we find that $h[n]$ are the impulse response coefficients of a power-symmetric filter. Recall

that this property was also shared by the analysis filters in an orthogonal perfect-reconstruction

FIR filterbank.

Given orthonormality at level k=0 , we have now derived a condition on $h[n]$ which is necessary

and sufficient for orthonormality at level k=1 . Yet the same condition is necessary and sufficient

for orthonormality at level k=2 :

()

where $\delta[n-\ell+2m]$=< $\varphi\ 1\ ,\ n\ (t)$, $\varphi\ 1\ ,\ \ell+2\ m\ (t)$> . Using induction, we conclude that the previous condition will be necessary and sufficient for orthonormality of

for all $k\in\mathbb{Z}$.

To find conditions on { $g[n]$} ensuring that the set

is orthonormal at every k, we can

repeat the steps above but with $g[n]$ replacing $h[n]$, $\psi\ k\ ,\ n\ (t)$ replacing $\varphi\ k\ ,\ n\ (t)$, and the wavelet-scaling equation replacing the scaling equation. This yields

()

⇔ 2= $G(z)\ G(z\text{-}1){+}\ G({-}z)\ G({-}(z\text{-}1))$ Next derive a condition which guarantees that ($Wk\perp Vk$) , as required by our definition W

⊥

k= Vk , for all $k\in\mathbb{Z}$. Note that, for any $k\in\mathbb{Z}$, ($Wk\perp Vk$) is guaranteed

by

which is equivalent to

()

for all m where $\delta[\, n-\ell+2\, m]=<\varphi\, k\, ,\, n\, (\, t),\, \varphi\, k\, ,\, \ell\, +\, 2\, m\, (\, t)>$. In other words, a 2-downsampled version of $g[\, n]^{*}\, h[-\, n]$ must consist only of zeros. This necessary and sufficient condition can be restated in the frequency domain as

()

$\Leftrightarrow 0=G(\, z\, 1/2)\, H(\, z-1/2)+\, G(-(\, z\, 1/2))\, H(-(\, z-1/2))\Leftrightarrow 0=G(\, z)\, H(\, z\text{-}1)+\, G(-\, z)\, H(-(\, z\text{-}1))$ The choice **()**

$G(\, z)=\pm(\, z-\, PH((-\, z)\text{-}1))$

satisfies our condition, since

$G(\, z)\, H(\, z\text{-}1)+\, G(-\, z)\, H(-(\, z\text{-}1))=(\pm\, z-\, PH((-\, z)\text{-}1)\, H(\, z\text{-}1)\, \mp\, z-\, PH(\, z\text{-}1)\, H((-\, z)\text{-}1))=0$ In the time domain, the condition on $G(\, z)$ and $H(\, z)$ can be expressed

()

$g[\, n]=\pm(\text{-}1\, nh[\, P-\, n])$.

Recall that this property was satisfied by the analysis filters in an orthogonal perfect

reconstruction FIR filterbank.

Note that the two conditions $G(z)=\pm(z-PH((-z)-1))\ 2=H(z)\,H(z-1)+H(-z)\,H(-(z-1))$ are sufficient to ensure that both

and

are orthonormal for all k and that ($Wk \perp Vk$) for all

k, since they satisfy the condition $2=G(z)\,G(z-1)+G(-z)\,G(-(z-1))$ automatically.

Values of g[n] and h[n] for the Haar System*

The coefficients { $h[n]$} were originally introduced at describe $\varphi\ 1\ ,\ 0\ (t)$ in terms of the basis for $V\ 0$:

From the previous equation we find that

()

where $\delta[n-m]=<\varphi\ 0\ ,\ m\ (t),\ \varphi\ 0\ ,\ n\ (t)>$, which gives a way to calculate the coefficients { $h[m]$} when we know $\varphi\ k\ ,\ n\ (t)$.

In the Haar case

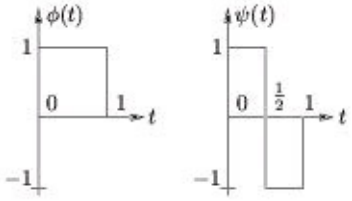
()

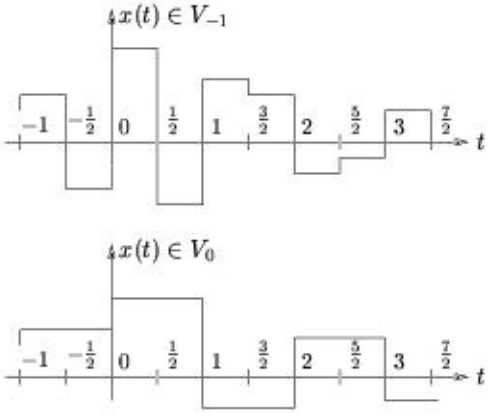
since

in the interval [0, 2) and zero otherwise. Then choosing $P=1$ in $g[n]=-1\ nh(P-n)$,

[redacted]

[redacted]





[redacted]

we find that

for the Haar system. From the wavelet scaling equation

we can see that the Haar mother wavelet and scaling function

look like in Figure 4.19:

Figure 4.19.

It is now easy to see, in the Haar case, how integer shifts of the mother wavelet describe the

differences between signals in $V - 1$ and $V 0$ (Figure 4.20):

Figure 4.20.

We expect this because $V - 1 = (V 0 \oplus W 0)$.

Wavelets: A Countable Orthonormal Basis for the Space of

Square-Integrable Functions*

Recall that $Vk = (W k + 1 \oplus V k + 1)$ and that $V k + 1 = (W k + 2 \oplus V k + 2)$. Putting these together and extending the idea yields

()

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

If we take the limit as $k \to -\infty$, we find that

()

Moreover,

()

$$(W1 \perp V1) \wedge W k \geq 2 \subset V1 \Rightarrow (W1 \perp W k \geq 2)$$

()

$$(W2 \perp V2) \wedge W k \geq 3 \subset V2 \Rightarrow (W2 \perp W k \geq 3)$$

from which it follows that

()

$$(Wk \perp W j \neq k)$$

or, in other words, all subspaces Wk are orthogonal to one another. Since the functions

form an orthonormal basis for Wk , the results above imply that

()

This implies that, for any $f(t) \in \mathcal{L}2$, we can write

()

()

$$dk[m] = < \psi k, m(t), f(t) >$$

This is the key idea behind the orthogonal wavelet system that we have been developing!

Filterbanks Interpretation of the Discrete Wavelet Transfor[m*]

Assume that we start with a signal $x(t) \in \mathcal{L}2$. Denote the best approximation at the 0th level of coarseness by $x0(t)$. (Recall that $x0(t)$ is the orthogonal projection of $x(t)$ onto $V0$.) Our goal, for

the moment, is to decompose $x\,0(\,t)$ into scaling coefficients and wavelet coefficients at higher

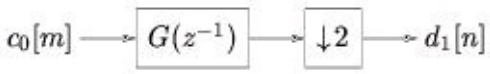
levels. Since $x\,0(\,t)\in V\,0$ and $V\,0=(\,V\,1\oplus W\,1)$, there exist coefficients $\{\,c\,0[\,n]\}$, $\{\,c\,1[\,n]\}$, and

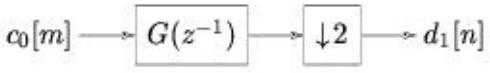
$\{\,d\,1[\,n]\}$ such that

()

Using the fact that

is an orthonormal basis for $V\,1$, in conjunction with the scaling





equation,

()

where $\delta[\,t-\ell-2\,n]=<\varphi(\,t-m),\varphi(\,t-\ell-2\,n)>$. The previous expression (Equation) indicates that

$\{\,c\,1[\,n]\}$ results from convolving $\{\,c\,0[\,m]\}$ with a time-reversed version of $h[\,m]$ then downsampling by factor two (Figure 4.21).

Figure 4.21.

Using the fact that

is an orthonormal basis for $W\,1$, in conjunction with the wavelet

scaling equation,

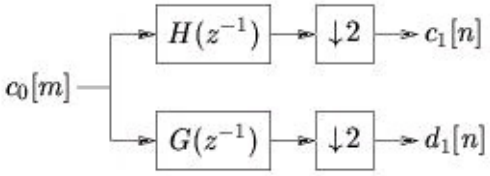
()

where $\delta[\,t-\ell-2\,n]=<\varphi(\,t-\,m),\,\varphi(\,t-\,\ell-2\,n)>$.

The previous expression (Equation) indicates that $\{\,d\,1[\,n]\}$ results from convolving $\{\,c\,0[\,m]\}$ with a time-reversed version of $g[\,m]$ then downsampling by factor two (Figure 4.22).

Figure 4.22.

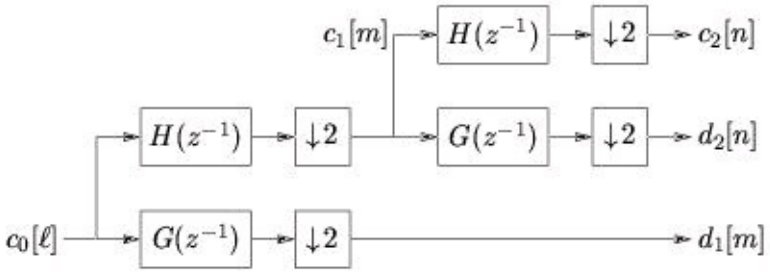
Putting these two operations together, we arrive at what looks like the analysis portion of an FIR



[redacted]

[redacted]

[redacted]



filterbank (Figure 4.23):

Figure 4.23.

We can repeat this process at the next higher level. Since $V\,1=(\,W\,2\oplus\,V\,2)$, there exist coefficients

{ *c* 2[*n*]} and { *d* 2[*n*]} such that

()

Using the same steps as before we find that

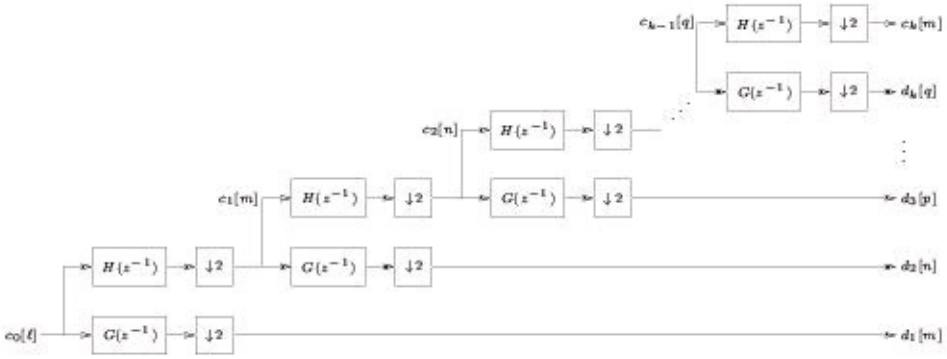
()

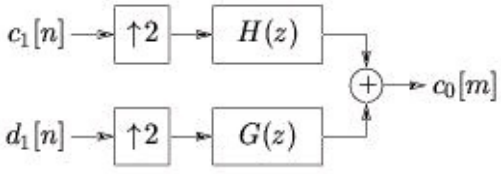
()

which gives a cascaded analysis filterbank (Figure 4.24):

Figure 4.24.

If we use *V* 0=(*W* 1⊕ *W* 2⊕ *W* 3⊕⋯⊕ *Wk*⊕ *Vk*) to repeat this process up to the *k* th level, we get the iterated analysis filterbank (Figure 4.25).





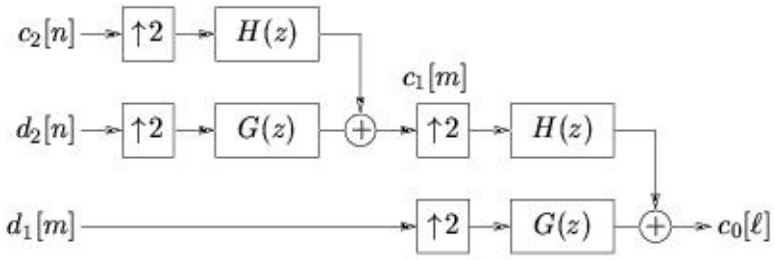


Figure 4.25.

As we might expect, signal reconstruction can be accomplished using cascaded two-channel synthesis filterbanks. Using the same assumptions as before, we have:

()

where $h[m-2n]=<\varphi_{1,n}(t), \varphi_{0,m}(t)>$ and $g[m-2n]=<\psi_{1,n}(t), \varphi_{0,m}(t)>$ which can be implemented using the block diagram in Figure 4.26.

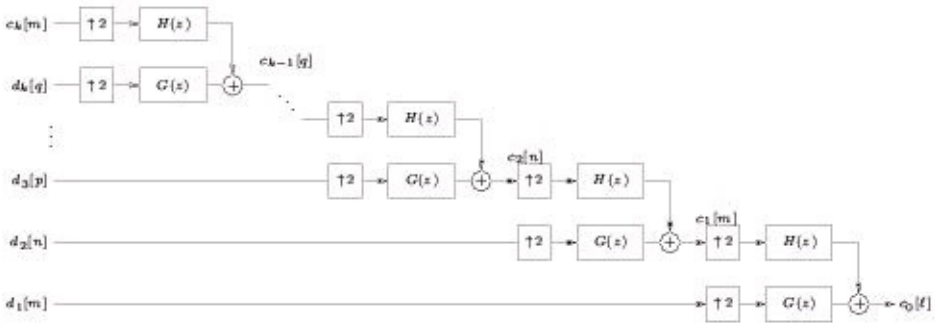
Figure 4.26.

The same procedure can be used to derive

()

from which we get the diagram in Figure 4.27.

Figure 4.27.



To reconstruct from the k th level, we can use the iterated synthesis filterbank (Figure 4.28).

Figure 4.28.

The **table** makes a direct comparison between wavelets and the two-channel orthogonal PR-FIR filterbanks.

Table 4.3.

2-Channel Orthogonal PR-FIR

Discrete Wavelet Transform

Filterbank

Analysis-LPF

$H(z^{-1})$

$H\ 0(\ z)$

Power

$H(\ z)\ H(\ z\text{-}1)+\ H(-\ z)\ H(-(\ z\text{-}1))=2$

H

Symmetry

$0(\ z)\ H\ 0(\ z\text{-}1)+\ H\ 0(-\ z)\ H\ 0(-(\ z\text{-}1))=1$

Analysis HPF

$G(\ z\text{-}1)$

$H\ 1(\ z)$

Spectral

$G(\ z)=\pm(\ z-\ PH(-(\ z\text{-}1)))$, P is odd H

Reverse

$1(\ z)=\pm(\ z-((\ N-1))\ H\ 0(-(\ z\text{-}1)))$, N is even

Synthesis LPF $H(\ z)$

$G\ 0(\ z)=2\ z-((\ N-1))\ H\ 0(\ z\text{-}1)$

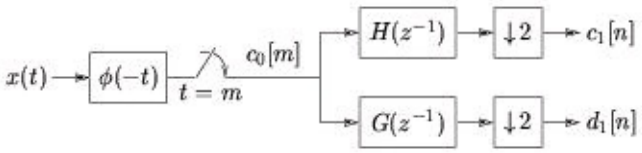
Synthesis HPF $G(\ z)$

$G\ 1(\ z)=2\ z-((\ N-1))\ H\ 1(\ z\text{-}1)$

From the table, we see that the discrete wavelet transform that we have been developing is identical to two-channel orthogonal PR-FIR filterbanks in all but a couple details.

1. Orthogonal PR-FIR filterbanks employ synthesis filters with twice the gain of the analysis filters, whereas in the DWT the gains are equal.

2. Orthogonal PR-FIR filterbanks employ causal filters of length N, whereas the DWT filters are not constrained to be causal.



For convenience, however, the wavelet filters $H(z)$ and $G(z)$ are usually chosen to be causal. For both to have even impulse response length N, we require that $P = N-1$.

Initialization of the Wavelet Transform*

The filterbanks developed in the module on the **filterbanks interpretation of the DWT** start with the signal representation

and break the representation down into wavelet coefficients

and scaling coefficients at lower resolutions (*i.e.* , higher levels k). The question remains: how do we get the initial coefficients $\{ c_0[n] \}$?

From their definition, we see that the scaling coefficients can be written using a convolution:

()

which suggests that the proper initialization of wavelet transform is accomplished by passing the continuous-time input $x(t)$ through an analog filter with impulse response $\varphi(-t)$ and sampling its output at integer times (Figure 4.29).

Figure 4.29.

Practically speaking, however, it is very difficult to build an analog filter with impulse response $\varphi(-t)$ for typical choices of scaling function.

The most often-used approximation is to set $c_0[n] = x[n]$. The sampling theorem implies that this would be exact if

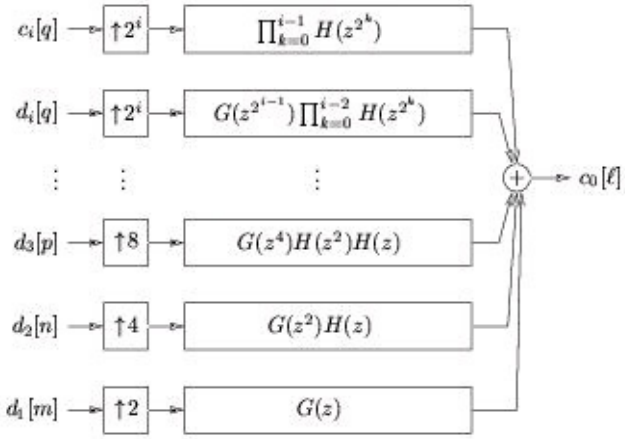
, though clearly this is not **correct** for general $\varphi(t)$. Still, this

technique is somewhat justified if we adopt the view that the principle advantage of the wavelet

transform comes from the multi-resolution capabilities implied by an iterated perfect-reconstruction filterbank (with **good** filters).

Regularity Conditions, Compact Support, and Daubechies' Wavelets*

Here we give a quick description of what is probably the most popular family of filter coefficients $h[n]$ and $g[n]$ — those proposed by Daubechies.



[redacted]

Recall the iterated synthesis filterbank. Applying the Noble identities, we can move the up-samplers before the filters, as illustrated in Figure 4.30.

Figure 4.30.

The properties of the *i*-stage cascaded lowpass filter

()

in the limit $i \to \infty$ give an important characterization of the wavelet system. But how do we know

that

converges to a response in $\mathcal{L}\,2$? In fact, there are some rather strict conditions on

$H(\,e^{i\omega})$ that must be satisfied for this convergence to occur. Without such convergence, we might

have a finite-stage perfect reconstruction filterbank, but we will **not** have a countable wavelet

basis for $\mathcal{L}\,2$. Below we present some "regularity conditions" on $H(\,e^{i\omega})$ that ensure convergence of the iterated synthesis lowpass filter.

The convergence of the lowpass filter implies convergence of all other filters in the bank.

Let us denote the impulse response of $H\,(\,i\,)\,(\,z)$ by $h\,(\,i\,)\,[\,n]$. Writing

$H\,(\,i\,)\,(\,z) = H(\,z\,2\,i-1)\,H\,(\,i-1\,)\,(\,z)$ in the time domain, we have Now define

the function

where $\mathcal{J}\,[\,a\,,\,b\,)\,(\,t)$ denotes the indicator function over

the interval $[\,a, b)$:

The definition of $\varphi\,(\,i\,)\,(\,t)$ implies

()

()

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

and plugging the two previous expressions into the equation for $h(i)[n]$ yields

()

Thus, if $\varphi(i)(t)$ converges pointwise to a continuous function, then it must satisfy the scaling

equation, so that

. *Daubechies* [**link**] showed that, for pointwise convergence of

$\varphi(i)(t)$ to a continuous function in $\mathcal{L}2$, it is sufficient that $H(e^{i\omega})$ can be factored as **()**

for $R(e^{i\omega})$ such that

()

Here P denotes the number of zeros that $H(e^{i\omega})$ has at $\omega=\pi$. Such conditions are called **regularity** conditions because they ensure the regularity, or **smoothness** of $\varphi(t)$. In fact, if we make the

previous condition stronger:

()

then

for $\varphi(t)$ that is ℓ-times continuously differentiable.

There is an interesting and important by-product of the preceding analysis. If $h[n]$ is a causal

length-N filter, it can be shown that $h(i)[n]$ is causal with length $Ni=(2i-1)(N-1)+1$. By construction, then, $\varphi(i)[t]$ will be zero outside the interval

. Assuming that the

regularity conditions are satisfied so that

, it follows that $\varphi(t)$ must be zero outside

the interval $[0, N-1]$. In this case we say that $\varphi(t)$ has **compact support**. Finally, the wavelet

scaling equation implies that, when $\varphi(t)$ is compactly supported on $[0, N-1]$ and $g[n]$ is length N, $\psi(t)$ will also be compactly supported on the interval $[0, N-1]$.

Daubechies constructed a family of $H(z)$ with impulse response lengths $N\in\{4, 6, 8, 10, \ldots\}$ which

satisfy the regularity conditions. Moreover, her filters have the maximum possible number of

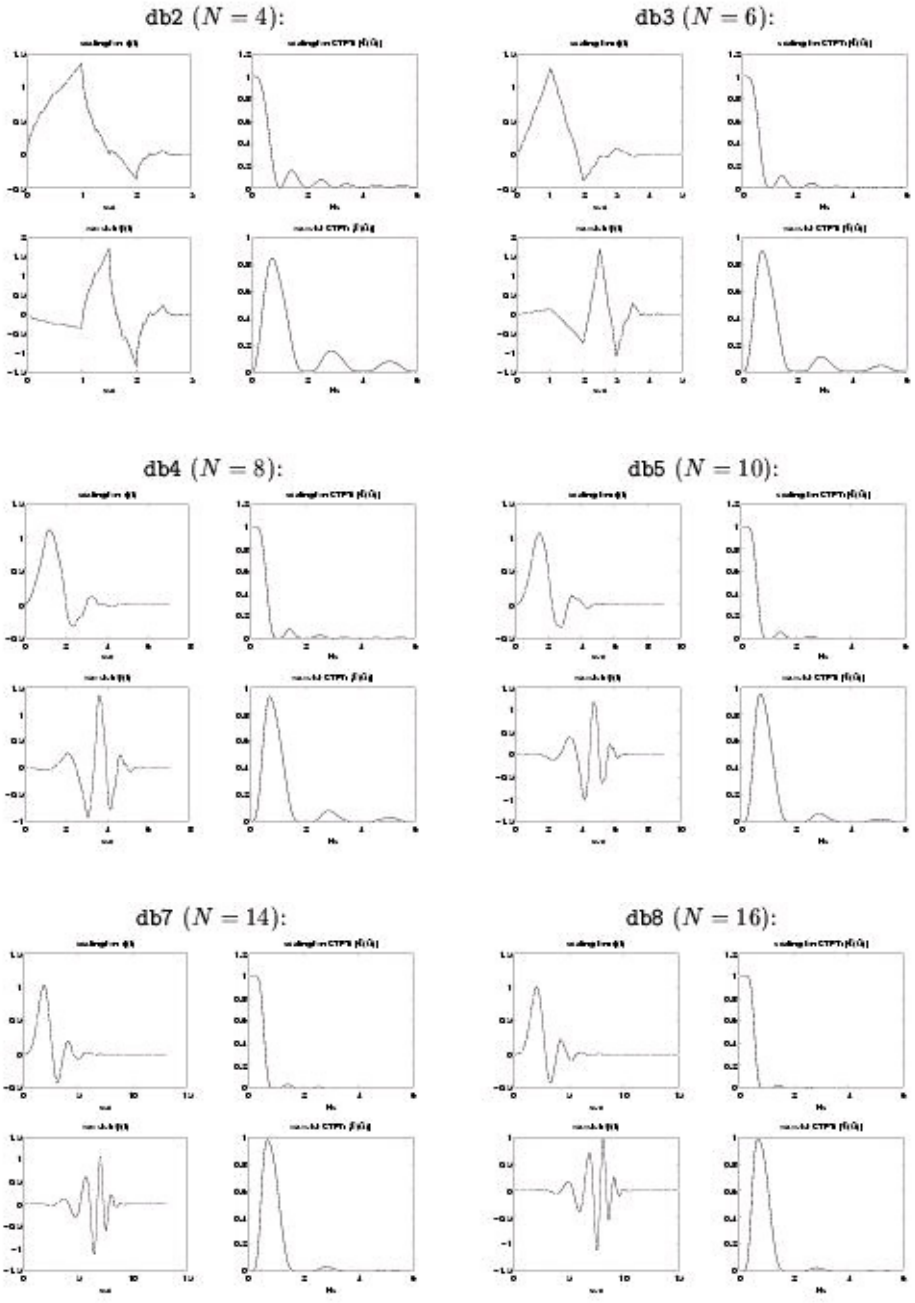
zeros at $\omega=\pi$, and thus are maximally regular (i.e., they yield the smoothest possible $\varphi(t)$ for a given

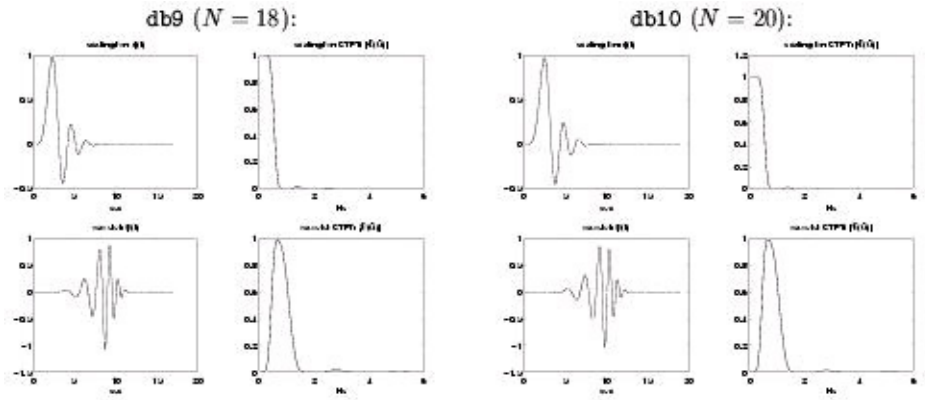
support interval). It turns out that these filters are the **maximally flat** filters derived by

Herrmann [**link**] long before filterbanks and wavelets were in vogue. In Figure 4.31 and

Figure 4.32 we show $\varphi(t)$, $\Phi(\Omega)$, $\psi(t)$, and $\Psi(\Omega)$ for various members of the Daubechies' wavelet system.

See *Vetterli and Kovacivić* [**link**] for a more complete discussion of these matters.





(a)

(b)

Figure 4.31.

(a)

(b)

[redacted]

Figure 4.32.

References

1. I. Daubechies. (1988, Nov). Orthonormal bases of compactly supported wavelets. *Commun. on Pure and Applied Math*, *41*, 909-996.

2. O. Herrmann. (1971). On the approximation problem in nonrecursive digital filter design.

IEEE Trans. on Circuit Theory, *18*, 411-413.

3. M. Vetterli and J. Kovacivić. (1995). *Wavelets and Subband Coding*. Englewood Cliffs, NJ:

Prentice Hall.

Computing the Scaling Function: The Cascade Algorithm*

Given coefficients { $h[n]$} that satisfy the regularity conditions, we can iteratively calculate

samples of $\varphi(t)$ on a fine grid of points { t} using the **cascade algorithm**. Once we have obtained

$\varphi(t)$, the wavelet scaling equation can be used to construct $\psi(t)$.

In this discussion we assume that $H(z)$ is causal with impulse response length N. Recall, from our

discussion of the regularity conditions, that this implies $\varphi(t)$ will have compact support on the interval $[0, N-1]$. The cascade algorithm is described below.

1. Consider the scaling function at integer times $t = m \in \{0, \ldots, N-1\}$:

Knowing that $\varphi(t)=0$ for $t \notin [0, N-1]$, the previous equation can be written using an N x N

matrix. In the case where $N=4$, we have

()

The matrix H is structured as a **row-decimated convolution matrix**.

From the matrix equation above (Equation), we see that $(\varphi(0), \varphi(1), \varphi(2), \varphi(3))^T$ must be (some scaled version of) the eigenvector of H corresponding to eigenvalue

. In general,

the nonzero values of $\{ \varphi(n) | n \in \mathbb{Z}\}$, *i.e.* , $(\varphi(0), \varphi(1), \ldots, \varphi(N-1))^T$, can be calculated by appropriately scaling the eigenvector of the N x N row-decimated convolution matrix H

corresponding to the eigenvalue

. It can be shown that this eigenvector must be scaled so

that

.

2. Given $\{ \varphi(n) | n \in \mathbb{Z}\}$, we can use the scaling equation to determine

:

2. Given $\{\ \varphi(\ n)|\ n\in\mathbb{Z}\}$, we can use the scaling equation to determine

:

()

This produces the $2\ N-1$ non-zero samples $\{\ \varphi(0),\ \varphi(1/2),\ \varphi(1),\ \varphi(3/2),\ \ldots,\ \varphi(\ N-1)\}$.

3. Given

, the scaling equation can be used to find

:

()

where $h \uparrow 2[p]$ denotes the impulse response of $H(z^2)$, *i.e.*, a 2-upsampled version of $h[n]$, and where

. Note that

is the result of convolving $h \uparrow 2[n]$ with

.

4. Given

, another convolution yields

:

()

where $h \uparrow 4[n]$ is a 4-upsampled version of $h[n]$ and where

.

5. At the ℓ th stage,

is calculated by convolving the result of the $\ell - 1$ th stage with a $2^{\ell-1}$ -

upsampled version of $h[n]$:

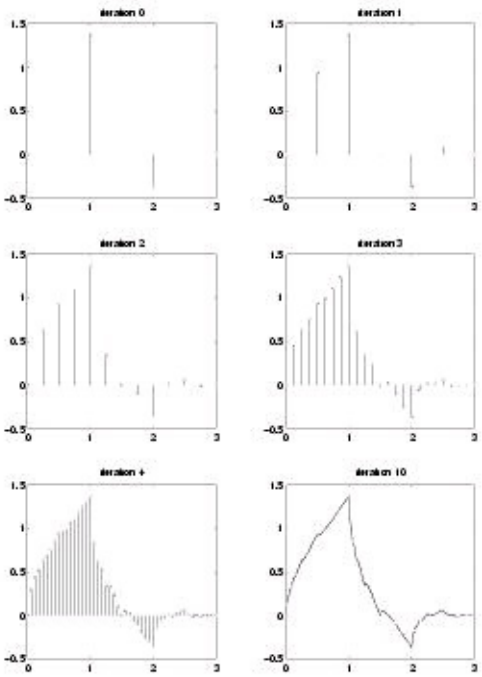
()

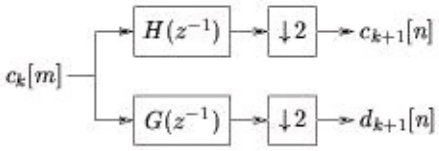
For $\ell \approx 10$, this gives a very good approximation of $\varphi(t)$. At this point, you could verify the key

properties of $\varphi(t)$, such as orthonormality and the satisfaction of the scaling equation.

In Figure 4.33 we show steps 1 through 5 of the cascade algorithm, as well as step 10, using

Daubechies' db2 coefficients (for which $N=4$).





[redacted]

[redacted]

[redacted]

[redacted]

Figure 4.33.

Finite-Length Sequences and the DWT Matrix*

The wavelet transform, viewed from a filterbank perspective, consists of iterated 2-channel

analysis stages like the one in Figure 4.34.

Figure 4.34.

First consider a very long (i.e., practically infinite-length) sequence

. For every pair of

input samples { *ck*[2 *n*], *ck*[2 *n*−1]} that enter the *k* th filterbank stage, exactly one pair of output samples { *c k* + 1 [*n*], *d k* + 1 [*n*]} are generated. In other words, the number of output equals the number of input during a fixed time interval. This property is convenient from a real-time

processing perspective.

For a short sequence

, however, linear convolution requires that we make an

assumption about the tails of our finite-length sequence. One assumption could be

()

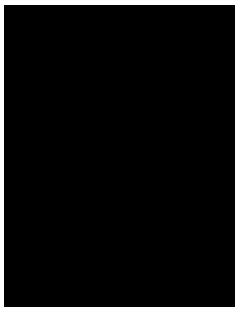
$c_k[m]=0\ ,\ m\in\{0,\ldots,M-1\}$

In this case, the linear convolution implies that M nonzero inputs yield

outputs from each

branch, for a total of

outputs. Here we have assumed that both $H(z\text{-}1)$ and



$G(z^{-1})$ have impulse response lengths of $N>2$, and that M and N are both even. The fact that each filterbank stage produces more outputs than inputs is very disadvantageous in many applications.

A more convenient assumption regarding the tails of

is that the data outside

of the time window $\{0, \ldots, M-1\}$ is a cyclic extension of data inside the time window. In other words, given a length- M sequence, the points **outside the sequence** are related to points **inside the sequences** via

()

$$c_k[m] = c_k[m+M]$$

Recall that a linear convolution with an M-cyclic input is equivalent to a circular convolution with one M-sample period of the input sequences. Furthermore, the output of this circular convolution is itself M-cyclic, implying our 2-downsampled branch outputs are cyclic with period . Thus, given an M-length input sequence, the total filterbank output consists of exactly M values.

It is instructive to write the circular-convolution analysis fiterbank operation in matrix form. In

Equation we give an example for filter length $N=4$, sequence length $N=8$, and causal synthesis filters $H(z)$ and $G(z)$.

()

where

The matrices *HM* and *GM* have

interesting properties. For example, the conditions

g[*n*]=-1 *nh*[*N*−1− *n*] imply

that

where *IM* denotes the *M* x *M* identity matrix. Thus, it makes sense to

define the *M* x *M* **DWT matrix** as

()

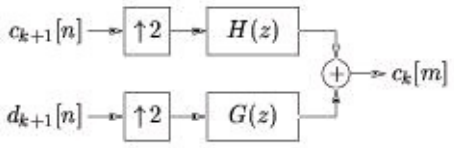
whose transpose constitutes the *M* x *M* **inverse DWT matrix**:

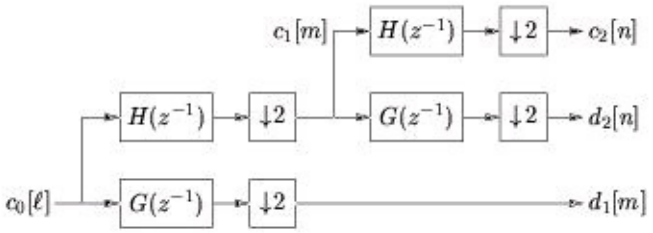
-1

T

()

TM = *TM*





Since the synthesis filterbank (Figure 4.35)

Figure 4.35.

gives perfect reconstruction, and since the cascade of matrix operations T T

M TM also corresponds

to perfect reconstruction, we expect that the matrix operation T T

M describes the action of the

synthesis filterbank. This is readily confirmed by writing the upsampled circular convolutions in

matrix form:

()

where

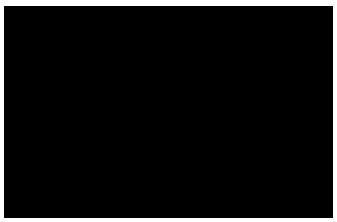
So far we have concentrated on one stage in the

wavelet decomposition; a two-stage decomposition is illustrated in Figure 4.36.

Figure 4.36.

The two-stage analysis operation (assuming circular convolution) can be expressed in matrix form

as



()

Similarly, a three-stage analysis could be implemented via

()

It should now be evident how to extend this procedure to >3 stages. As noted earlier, the corresponding synthesis operations are accomplished by transposing the matrix products used in the analysis.

DWT Implementation using FFTs*

Finally, we say a few words about DWT implementation. Here we focus on a single DWT stage and assume circular convolution, yielding an M x M DWT matrix TM . In the general case, M x M matrix multiplication requires M 2 multiplications. The DWT matrices, however, have a circular-convolution structure which allows us to implement them using significantly less multiplies. Below we present some simple and reasonably efficient approaches for the implementation of

T

T

M and TM .

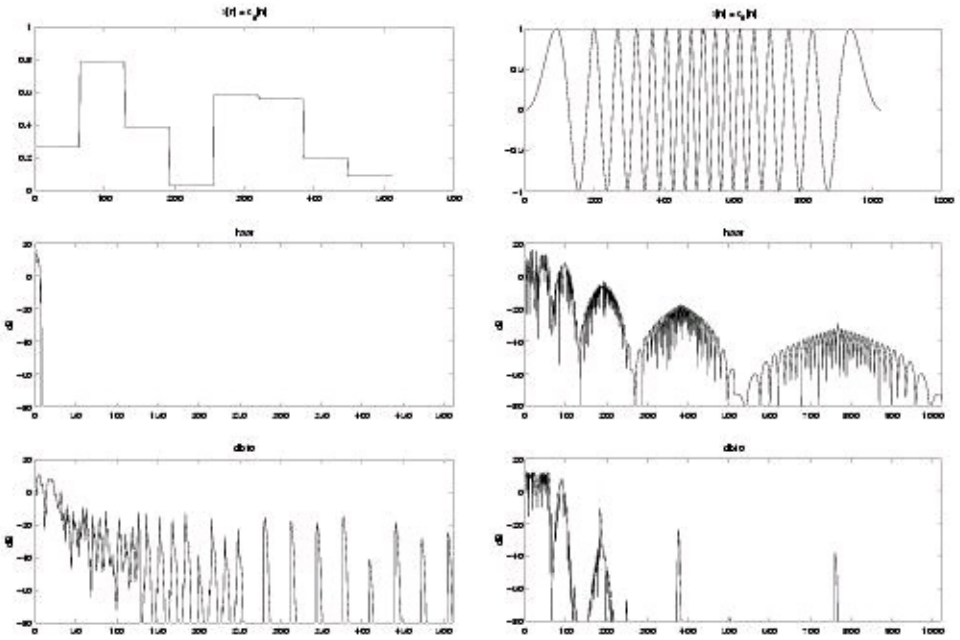
We treat the inverse DWT first. Recall that in the lowpass synthesis branch, we upsample the input before circularly convolving with $H(z)$. Denoting the upsampled coefficient sequence by $a[n]$, fast circular convolution $a[n]* h[n]$ can be described as follows (using Matlab notation) ifft(fft(a).*fft(h,length(a)))

where we have assumed that length(a) ≥ length(h). [11] The highpass branch is handled similarly using $G(z)$, after which the two branch outputs are summed.

Next we treat the forward DWT. Recall that in the lowpass analysis branch, we circularly convolve the input with $H(z\text{-}1)$ and then downsample the result. The fast circular convolution $a[n]* h[-n]$ can be implemented using

wshift('1', ifft(fft(a).*fft(flipud(h),length(a))), length(h)-1)

where wshift accomplishes a circular shift of the ifft output that makes up for the unwanted



delay of length(h)-1 samples imposed by the flipud operation. The highpass branch is handled similarly but with filter $G(z\text{-}1)$. Finally, each branch is downsampled by factor two.

We note that the proposed approach is not totally efficient because downsampling is performed after circular convolution (and upsampling before circular convolution). Still, we have outlined this approach because it is easy to understand and still results in major saving when M is large: it converts the $O(M\ 2)$ matrix multiply into an $O(M \log 2\ M)$ operation.

DWT Applications - Choice of phi(t)*

Transforms are signal processing tools that are used to give a clear view of essential signal characteristics. Fourier transforms are ideal for infinite-duration signals that contain a relatively small number of sinusoids: one can completely describe the signal using only a few coefficients. Fourier transforms, however, are not well-suited to signals of a non-sinusoidal nature (as discussed earlier in the context of **time-frequency analysis)**. The multi-resolution DWT is a more general transform that is well-suited to a larger class of signals. For the DWT to give an efficient description of the signal, however, we must choose a wavelet $\psi(t)$ from which the signal can be constructed (to a good approximation) using only a few stretched and shifted copies.

We illustrate this concept in Figure 4.37 using two examples. On the left, we analyze a step-like waveform, while on the right we analyze a chirp-like waveform. In both cases, we try DWTs based

on the Haar and Daubechies db10 wavelets and plot the log magnitudes of the transform coefficients [c T

T

T

T

T

k , dk , $d\,k-1$, $d\,k-2$, …, d 1] .

Figure 4.37.

Observe that the Haar DWT yields an extremely efficient representation of the step-waveform: only a few of the transform coefficients are nonzero. The db10 DWT does not give an efficient representation: many coefficients are sizable. This makes sense because the Haar scaling function is well matched to the step-like nature of the time-domain signal. In contrast, the Haar DWT does not give an efficient representation of the chirp-like waveform, while the db10 DWT does better. This makes sense because the sharp edges of the Haar scaling function do not match the smooth chirp signal, while the smoothness of the db10 wavelet yields a better match.

DWT Application - De-noising*

Say that the DWT for a particular choice of wavelet yields an efficient representation of a particular signal class. In other words, signals in the class are well-described using a few large transform coefficients.

Now consider unstructured **noise**, which cannot be eifficiently represented by any transform, including the DWT. Due to the orthogonality of the DWT, such noise sequences make, on average, equal contributions to all transform coefficients. Any given noise sequence is expected to yield many small-valued transform coefficients.

Together, these two ideas suggest a means of **de-noising** a signal. Say that we perform a DWT on

a signal from our **well-matched** signal class that has been corrupted by additive noise. We expect that large transform coefficients are composed mostly of signal content, while small transform coefficients should be composed mostly of noise content. Hence, throwing away the transform coefficients whose magnitude is less than some small threshold should improve the signal-to-noise ratio. The de-noising procedure is illustrated in Figure 4.38.

Figure 4.38.

Now we give an example of denoising a step-like waveform using the Haar DWT. In Figure 4.39, the top right subplot shows the noisy signal and the top left shows it DWT coefficients. Note the presence of a few large DWT coefficients, expected to contain mostly signal components, as well as the presence of many small-valued coefficients, expected to contain noise. (The bottom left subplot shows the DWT for the original signal before any noise was added, which confirms that all signal energy is contained within a few large coefficients.) If we throw away all DWT coefficients whose magnitude is less than 0.1, we are left with only the large coefficients (shown in the middle left plot) which correspond to the de-noised time-domain signal shown in the middle right plot. The difference between the de-noised signal and the original noiseless signal is shown in the bottom right. Non-zero error results from noise contributions to the large coefficients; there is no way of distinguishing these noise components from signal components.

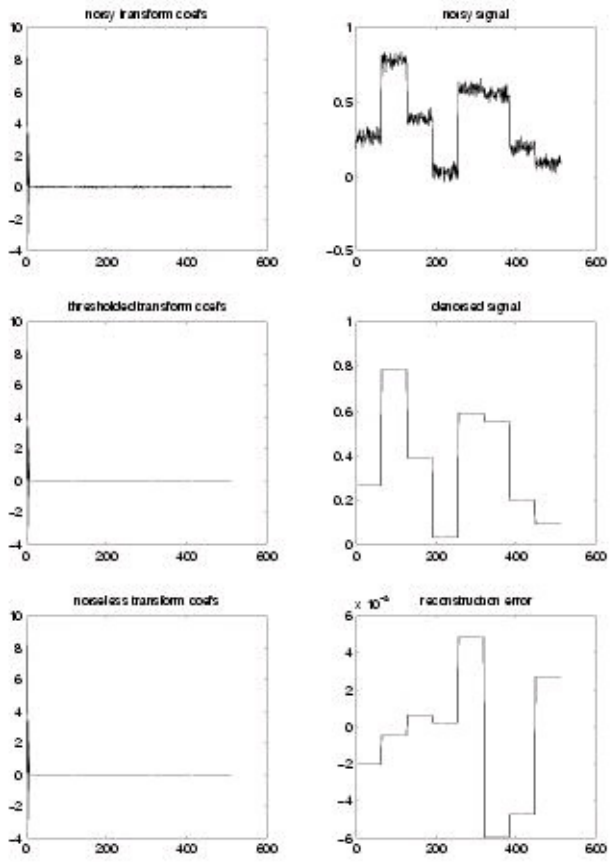


Figure 4.39.

Solutions

Chapter 5. Multirate Signal Processing

5.1. Overview of Multirate Signal Processing*

Digital transformation of the sampling rate of signals, or signal processing with different sampling rates in the system.

Applications

1. **Sampling-rate conversion:** CD to DAT format change, for example.

2. **Improved D/A, A/D conversion:** oversampling converters; which reduce performance requirements on anti-aliasing or reconstruction filters

3. **FDM channel modulation and processing:** bandwidth of individual channels is much less than the overall bandwidth

4. **Subband coding of speech and images:** Eyes and ears are not as sensitive to errors in higher frequency bands, so many coding schemes split signals into different frequency bands and

quantize higher-frequency bands with much less precision.

Outline of Multirate DSP material

1. **General Rate-changing System**

2. **Integer-factor Interpolation and Decimation and Rational-factor Rate Changing**

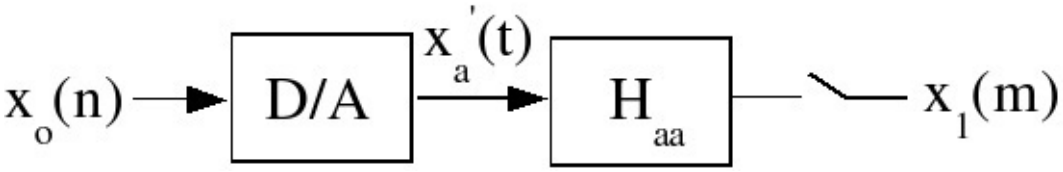
3. **Efficient Multirate Filter Structures**

4. **Optimal Filter Design for Multirate Systems**

5. **Multi-stage Multirate Systems**

6. **Oversampling D/As**

7. **Perfect-Reconstruction Filter Banks and Quadrature Mirror Filters**



General Rate-Changing Procedure

This procedure is motivated by an analog-based method: one conceptually simple method to

change the sampling rate is to simply convert a digital signal to an analog signal and resample it!

(Figure 5.1)

Figure 5.1.

Recall the ideal D/A:

()

The problems with this scheme are:

1. A/D, D/A, filters cost money

2. imperfections in these devices introduce errors

Digital implementation of rate-changing according to this formula has three problems:

1. Infinite sum: The solution is to truncate. Consider sinc $t \approx 0$ for $t < t_1$, $t > t_2$: Then $mT_1 - nT_0 < t_1$ and $mT_1 - nT_0 > t_2$ which implies

This is essentially lowpass filter design using a boxcar window: other finite-length filter

design methods could be used for this.

2. Lack of **causality**: The solution is to delay by $\max\{|N|\}$ samples. The mathematics of the analog portions of this system can be implemented digitally.

()

()

So we have an all-digital formula for **exact** digital-to-digital rate changing!

3. Cost of computing sinc *T*′(*mT* 1− *nT* 0) : The solution is to precompute the table of sinc(*t*) values.

However, if

is not a rational fraction, an infinite number of samples will be needed, so

some approximation will have to be tolerated.

Rate transformation of any rate to any other rate can be accomplished digitally with

arbitrary precision (if some delay is acceptable). This method is used in practice in many

cases. We will examine a number of special cases and computational improvements, but in

some sense everything that follows are details; the above idea is the central idea in

multirate signal processing.

Useful references for the traditional material (everything except PRFBs) are *Crochiere and*

Rabiner, 1981 [**link**] and *Crochiere and Rabiner, 1983* [**link**]. A more recent tutorial is *Vaidyanathan* [**link**]; see also *Rioul and Vetterli* [**link**]. References to most of the original papers can be found in these tutorials.

References

1. R.E. Crochiere and L.R. Rabiner. (1981, March). Interpolation and Decimation of Digital

Signals: A Tutorial Review. *Proc. IEEE, 69*(3), 300-331.

2. R.E. Crochiere and L.R. Rabiner. (1983). *Multirate Digital Signal Processing*. Englewood

Cliffs, NJ: Prentice-Hall.

3. P.P Vaidyanathan. (1990, January). Multirate Digital Filters, Filter Banks, Polyphase

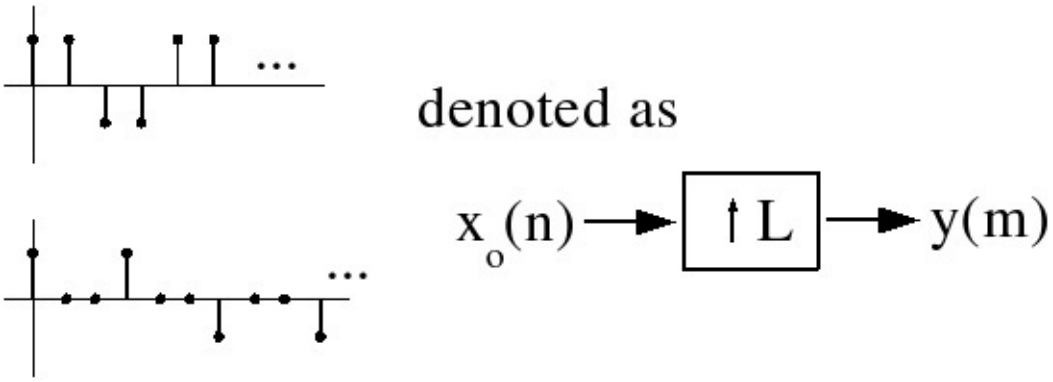
Networks, and Applications: A Tutorial. *Proc. IEEE, 78*(1), 56-93.

4. O. Rioul and M. Vetterli. (1991, October). Wavelets and Signal Processing. *IEEE Signal Processing Magazine, 8*(4), 14-38.

5.2. Interpolation, Decimation, and Rate Changing by

[redacted]

[redacted]



[redacted]

[redacted]

Integer Fractions*

Interpolation: by an integer factor L

Interpolation means increasing the sampling rate, or filling in in-between samples. Equivalent to sampling a bandlimited analog signal *L* times faster. For the ideal interpolator,

()

We wish to accomplish this digitally. Consider Equation and Figure 5.2.

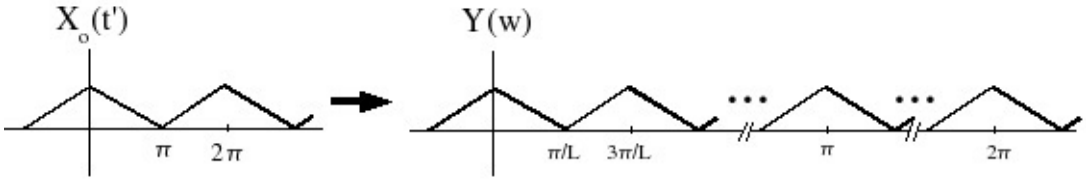
()

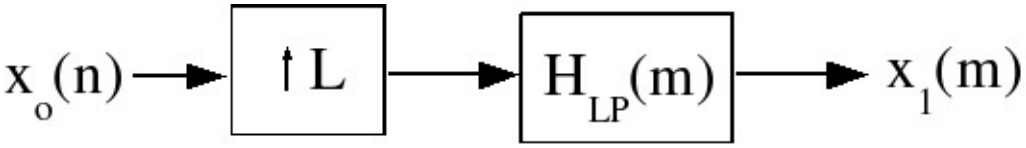
Figure 5.2.

The DTFT of *y*(*m*) is

()

Since $X\,0(\,\omega')$ is periodic with a period of $2\,\pi$, $X\,0(\,\boldsymbol{L\omega})= Y(\,\omega)$ is periodic with a period of (see Figure 5.3).





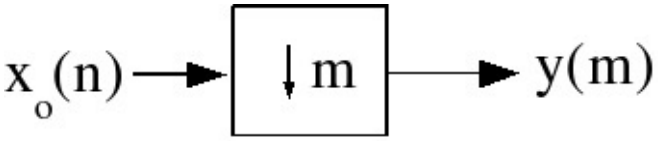


Figure 5.3.

By inserting zero samples between the samples of $x\,0(\,n)$, we obtain a signal with a scaled frequency response that simply replicates $X\,0(\,\omega')$ L times over a $2\,\pi$ interval!

Obviously, the desired $x\,1(\,m)$ can be obtained simply by lowpass filtering $y(\,m)$ to remove the replicas.

()

$x\,1(\,m)= y(\,m)* hL(\,m)$

Given

In practice, a finite-length lowpass filter is designed using any of the methods studied so far (Figure 5.4).

Figure 5.4. Interpolator Block Diagram

Decimation: sampling rate reduction (by an integer factor M)

Let $y(\,m)= x\,0(\,\boldsymbol{Lm})$ (Figure 5.5)

Figure 5.5.

That is, keep only every L th sample (Figure 5.6)

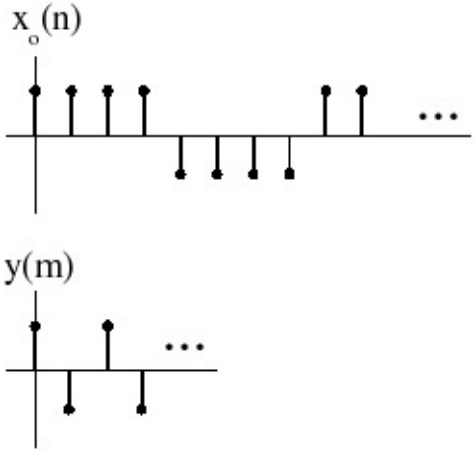


Figure 5.6.

In frequency (DTFT):

()

Now

for | ω|< π as shown in homework #1 , where $X(\ k)$ is the

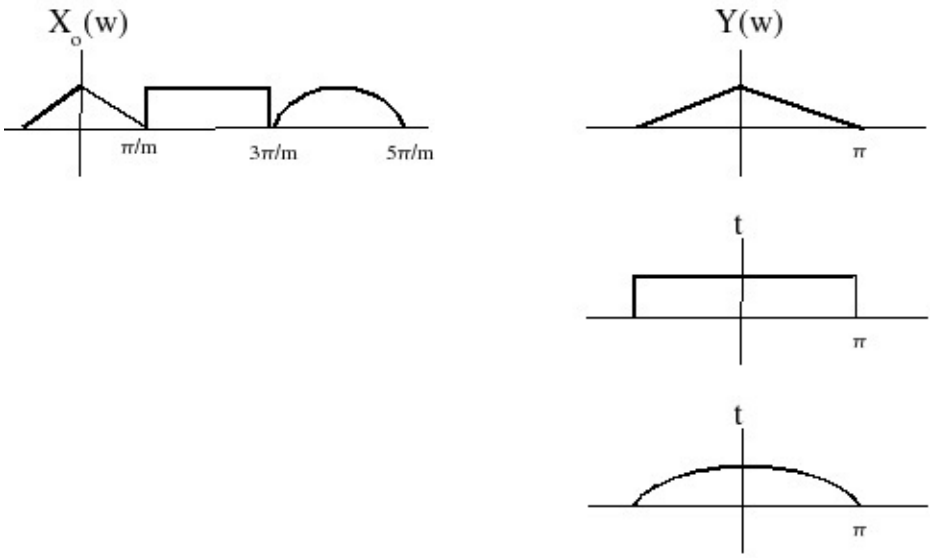
DFT of one period of the periodic sequence. In this case, $X(k)=1$ for $k \in \{0, 1, \ldots, M-1\}$ and

.

()

so

i.e. , we get **digital aliasing**.(Figure 5.7)



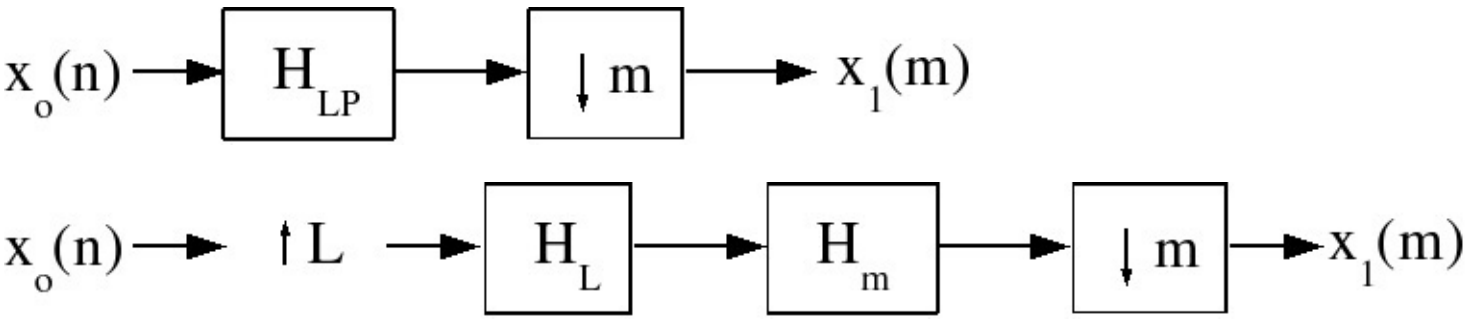


Figure 5.7.

Usually, we prefer not to have aliasing, so the downsampler is preceded by a lowpass filter to remove all frequency components above

(Figure 5.8).

Figure 5.8.

Rate-Changing by a Rational Fraction L/M

This is easily accomplished by interpolating by a factor of *L*, then decimating by a factor of *M* (Figure 5.9).

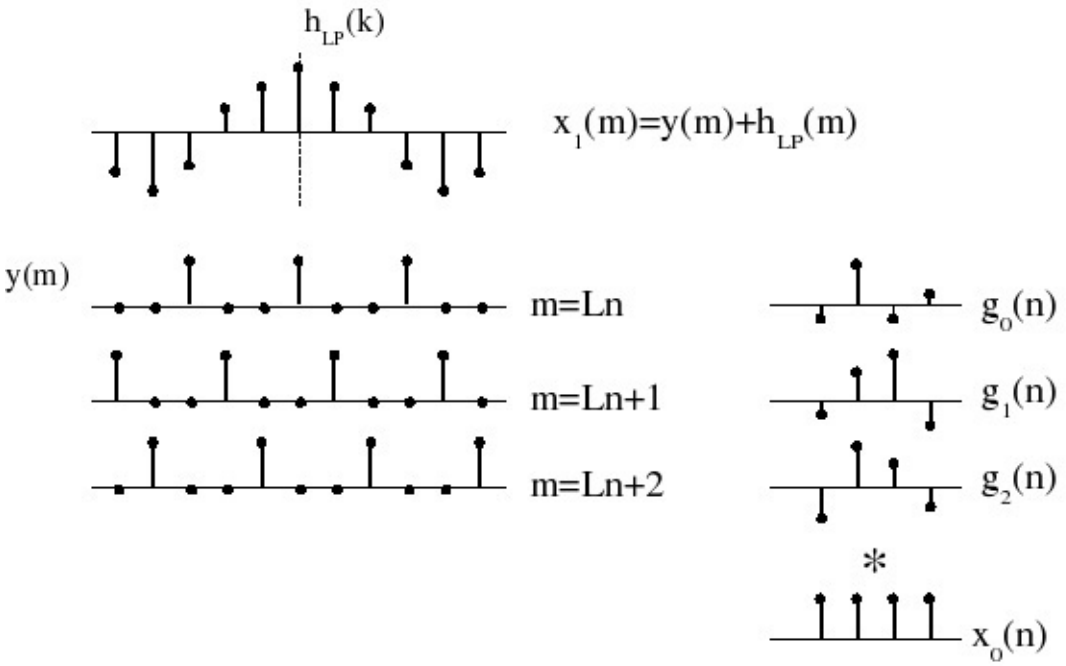
Figure 5.9.

The two lowpass filters can be combined into one LP filter with the lower cutoff,

Obviously, the computational complexity and simplicity of

implementation will depend on : 2/3 will be easier to implement than 1061/1060 !

5.3. Efficient Multirate Filter Structures*



[redacted]

[redacted]

Rate-changing appears expensive computationally, since for both decimation and interpolation the lowpass filter is implemented at the higher rate. However, this is not necessary.

Interpolation

For the interpolator, most of the samples in the upsampled signal are zero, and thus require no

computation. (Figure 5.10)

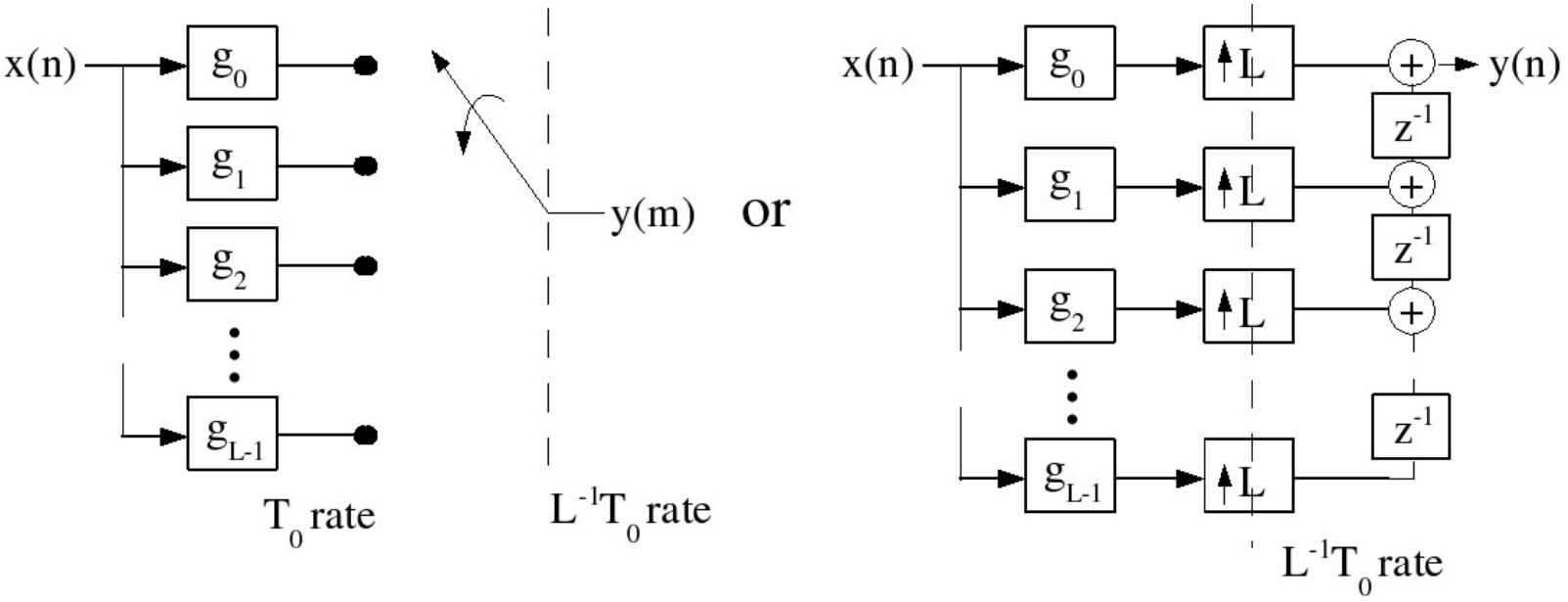
Figure 5.10.

For

and $p = m \bmod L$,

()

$gp(n) = h(\boldsymbol{Ln} + p)$ Pictorially, this can be represented as in Figure 5.11.



[redacted]

[redacted]

[redacted]

[redacted]

Figure 5.11.

These are called **polyphase structures**, and the $gp(n)$ are called **polyphase filters**.

Computational cost

If $h(m)$ is a length- N filter:

No simplification:

Polyphase structure:

where L is the number of filters, is the taps/filter,

and

is the rate.

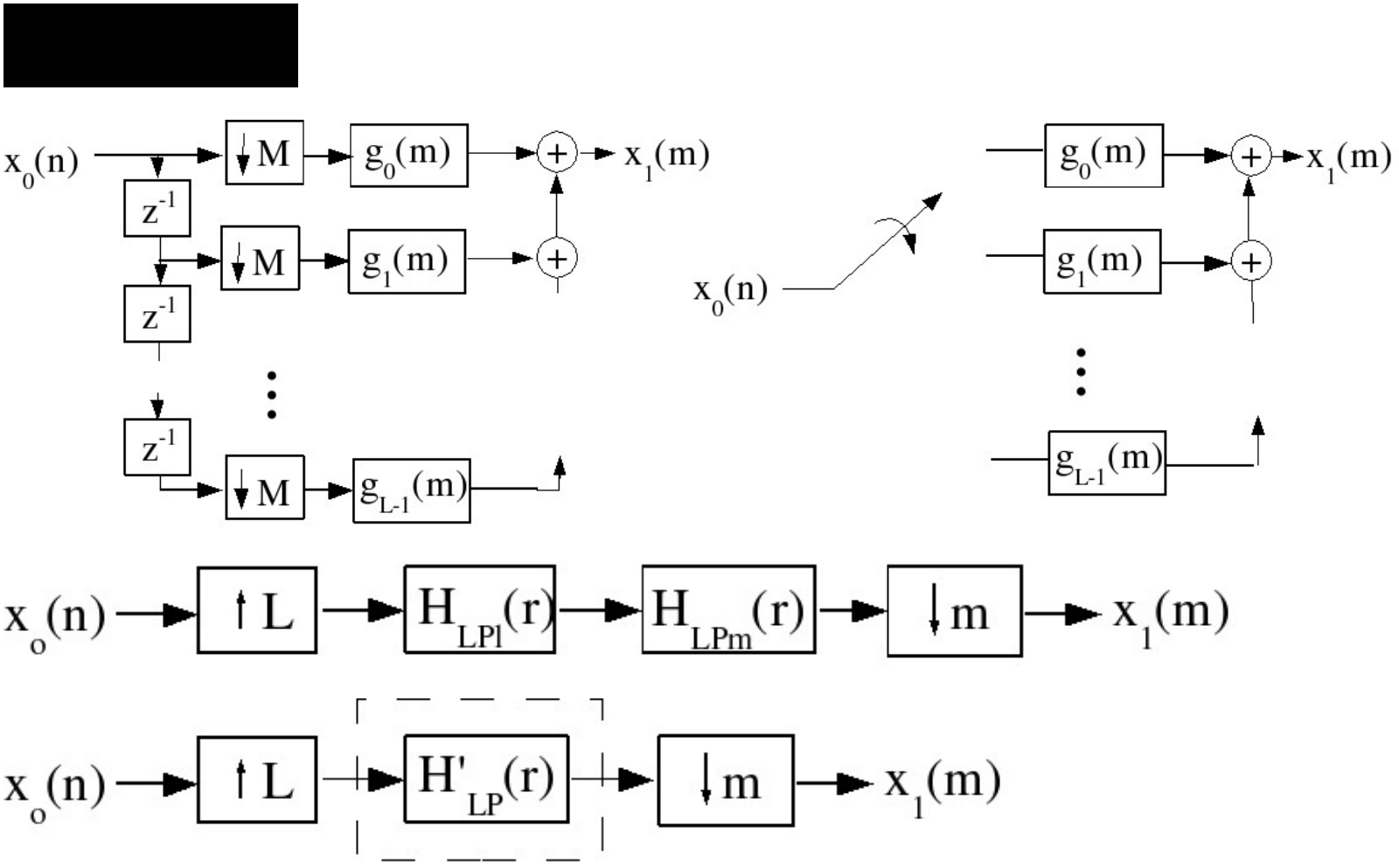
Thus we save a factor of *L* by not being dumb.

For a given precision, *N* is proportional to *L*, (why?), so the computational cost does increase with the interpolation rate.

Question

Can similar computational savings be obtained with IIR structures?

Efficient Decimation Structures



We only want every *M* th output, so we compute only the outputs of interest. (Figure 5.12)

Figure 5.12. Polyphase Decimation Structure

The decimation structures are flow-graph reversals of the interpolation structure. Although direct implementation of the full filter for every *M* th sample is obvious and straightforward, these polyphase structures give some idea as to how one might evenly partition the computation over *M*

cycles.

Efficient L/M rate changers

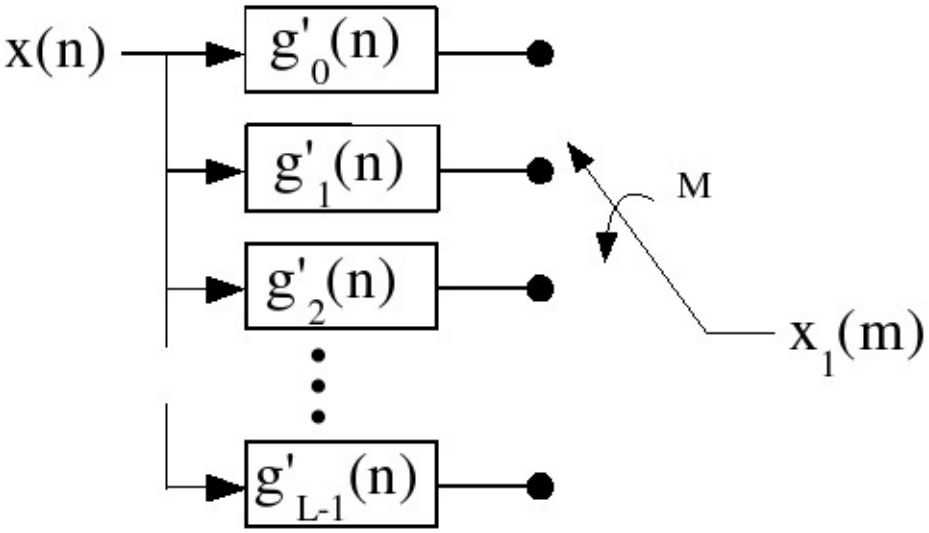
Interpolate by *L* and decimate by *M* (Figure 5.13).

Figure 5.13.

Combine the lowpass filters (Figure 5.14).

Figure 5.14.

We can couple the lowpass filter either to the interpolator or the decimator to implement it efficiently (Figure 5.15).



[redacted]

[redacted]

[redacted]

Figure 5.15.

Of course we only compute the polyphase filter output selected by the decimator.

Computational Cost

Every

, compute one polyphase filter of length , or

However,

note that *N* is proportional to max{ *L*, *M*} .

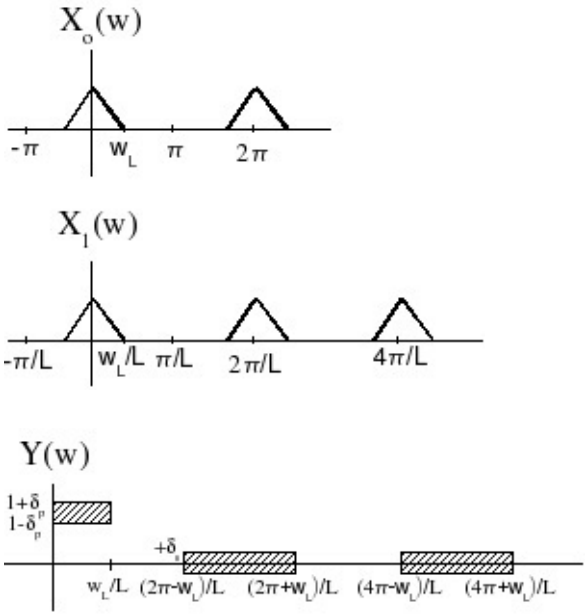
5.4. Filter Design for Multirate Systems*

The **filter design techniques** learned earlier can be applied to the design of filters in multirate systems, with a few twists.

Example 5.1.

Design a factor-of- *L* interpolator for use in a CD player, we might wish that the out-of-band error be below the least significant bit, or 96dB down, and < 0.05 % error in the passband, so these specifications could be used for optimal *L*∞ filter design.

In a CD player, the sampling rate is 44.1kHz, corresponding to a Nyquist frequency of 22.05kHz, but the sampled signal is bandlimited to 20kHz. This leaves a small transition band, from 20kHz to 24.1kHz. However, note that in any case where the signal spectrum is zero over some band, this introduces **other** zero bands in the scaled, replicated spectrum (Figure 5.16).



[redacted]

[redacted]

[redacted]

[redacted]

Figure 5.16.

So we need only control the filter response in the stopbands over the frequency regions with

nonzero energy. (Figure 5.17)

Figure 5.17.

The extra "don't care" bands allow a given set of specifications to be satisfied with a shorter-

length filter.

Direct polyphase filter design

Note that in an integer-factor interpolator, each set of output samples $x\ 1(\ \boldsymbol{Ln}+\ p)$,

$p=\{0,\ 1,\ \ldots,\ L-1\}$, is created by a different polyphase filter $gp(\ n)$, which has no interaction with the other polyphase filters except in that they each interpolate the same signal. We can thus treat

the design of each polyphase filter **independently**, as an -length filter design problem.

(Figure 5.18)

Figure 5.18.

Each $gp(\ n)$ produces samples

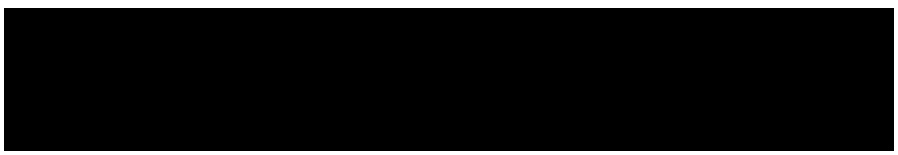
, where

is not an integer. That is, $gp(\ n)$ is to

produce an output signal (at a $T\ 0$ rate) that is $x\ 0(\ n)$ time-advanced by a non-integer advance .

The desired response of this polyphase filter is thus

for $|\ \omega|\leq\ \pi$, an all-pass filter with a



[redacted]

linear, non-integer, phase. Each polyphase filter can be designed independently to approximate this response according to any of the design criteria developed so far.

Exercise 1.

What should the polyphase filter for p=0 be?

A delta function: $h\,0(\,n)=\delta(\,n')$

Example 5.2. Least-squares Polyphase Filter Design

Deterministic x(n): Minimize

Given $x(\,n)=x(\,n)*\,h(\,n)$ and $xd(\,n)=x(\,n)*\,hd(\,n)$.

Using Parseval's theorem, this becomes

()

This is simply weighted least squares design, with (|$X(\,\omega)$|)2 as the weighting function.

stochastic X(ω):

()

$\boldsymbol{Sxx}(\,\omega)$ is the power spectral density of x. $\boldsymbol{Sxx}(\,\omega)$=DTFT[$\boldsymbol{rxx}(\,k)$]

Again, a

weighted least squares filter design problem.

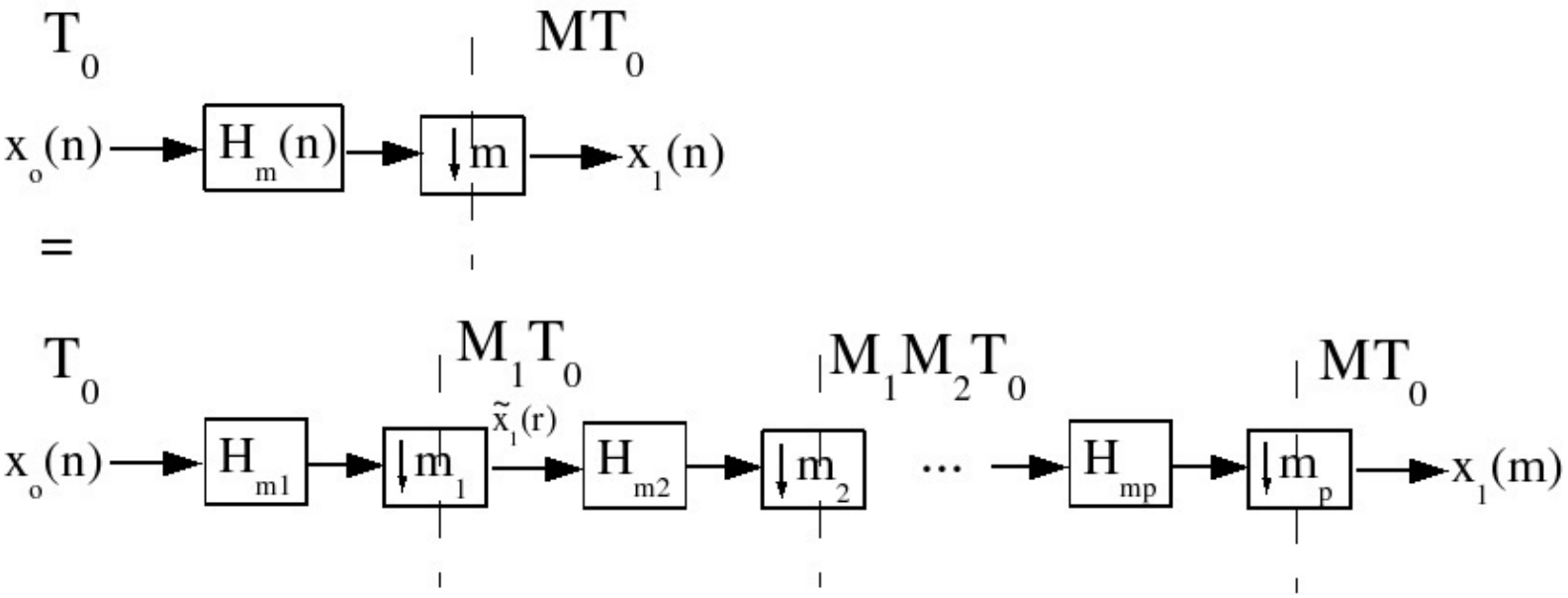
Problem

Is it feasible to use IIR polyphase filters?

The recursive feedback of previous outputs means that portions of each IIR polyphase filter must be computed for every input sample; this usually makes IIR filters more expensive than FIR implementations.

5.5. Multistage Multirate Systems*

[redacted]



Multistage multirate systems are often more efficient. Suppose one wishes to decimate a signal by an integer factor *M*, where *M* is a composite integer

. A decimator can be

implemented in a multistage fashion by first decimating by a factor *M* 1 , then decimating this signal by a factor *M* 2 , *etc.* (Figure 5.19)

Figure 5.19. Multistage decimator

Multistage implementations are of significant practical interest only if they offer significant computational savings. In fact, they often do!

The computational cost of a **single-stage** interpolator is:

The computational cost of a

multistage interpolator is:

The first term is the most significant, since the

rate is highest. Since ($Ni \propto Mi$) for a lowpass filter, it is not immediately clear that a multistage system should require less computation. However, the multistage structure relaxes the requirements on the filters, which reduces their length and makes the overall computation less.

Filter design for Multi-stage Structures

Ostensibly, the first-stage filter must be a lowpass filter with a cutoff at

, to prevent aliasing

after the downsampler. However, note that aliasing outside the **final overall** passband

is of

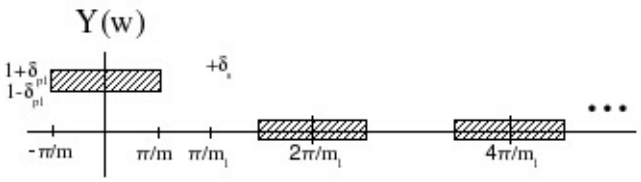
no concern, since it will be removed by later stages. We only need prevent aliasing into the band

; thus we need only specify the passband over the interval

, and the stopband over the

intervals

, for $k \in \{1, \ldots, M-1\}$. (Figure 5.20)



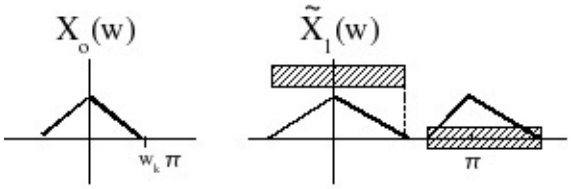


Figure 5.20.

Of course, we don't want **gain** in the transition bands, since this would need to be suppressed later, but otherwise we don't care about the response in those regions. Since the transition bands are so large, the required filter turns out to be quite short. The final stage has no "don't care" regions; however, it is operating at a low rate, so it is relatively unimportant if the final filter turns out to be rather long!

L-infinity Tolerances on the Pass and Stopbands

The overall response is a cascade of multiple filters, so the worst-case overall passband deviation, assuming all the peaks just happen to line up, is

So one could

choose all filters to have equal specifications and require for each-stage filter. For ($\delta p \ll 1$) ,

ov

Alternatively, one can design later stages (at lower

computation rates) to compensate for the passband ripple in earlier stages to achieve exceptionally accurate passband response.

δs remains essentially unchanged, since the worst-case scenario is for the error to alias into the passband and undergo no further suppression in subsequent stages.

Interpolation

Interpolation is the flow-graph reversal of the multi-stage decimator. The first stage has a cutoff at (Figure 5.21):

Figure 5.21.

However, all subsequent stages have large bands without signal energy, due to the earlier stages

(Figure 5.22):

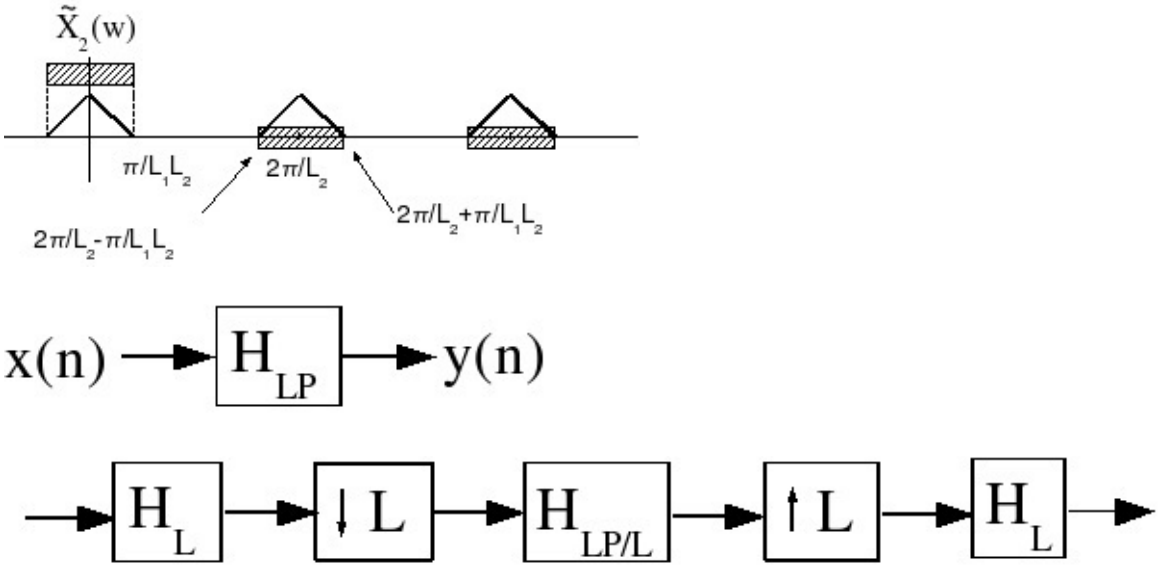


Figure 5.22.

The order of the filters is reversed, but otherwise the filters are identical to the decimator filters.

Efficient Narrowband Lowpass Filtering

A very narrow lowpass filter requires a very long FIR filter to achieve reasonable resolution in the frequency response. However, were the input sampled at a lower rate, the cutoff frequency would be correspondingly higher, and the filter could be shorter!

The transition band is also broader, which helps as well. Thus, Figure 5.23 can be implemented as Figure 5.24.

Figure 5.23.

Figure 5.24.

and in practice the inner lowpass filter can be coupled to the decimator or interpolator filters. If the decimator and interpolator are implemented as multistage structures, the overall algorithm can be dramatically more efficient than direct implementation!

5.6. DFT-Based Filterbanks*

One common application of multirate processing arises in multirate, multi-channel filter banks (Figure 5.25).

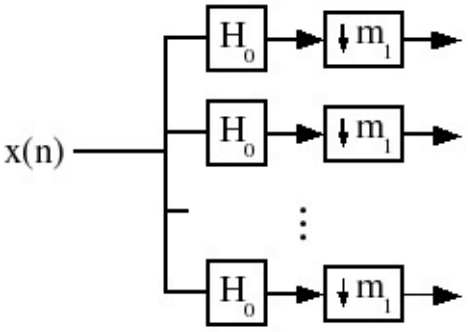


Figure 5.25.

One application is separating frequency-division-multiplexed channels. If the filters are narrowband, the output channels can be decimated without significant aliasing.

Such structures are especially attractive when they can be implemented efficiently. For example, if the filters are simply frequency modulated (by

) versions of each other, they can be

efficiently implemented using FFTs!

Furthermore, there are classes of filters called **perfect reconstruction filters** which are of finite length but from which the signal can be reconstructed exactly (using all *M* channels), even though the output of each channel experiences aliasing in the decimation step. These types of filterbanks have received a lot of research attention, culminating in wavelet theory and techniques.

Uniform DFT Filter Banks

Suppose we wish to split a digital input signal into *N* frequency bands, uniformly spaced at center

frequencies

, for $0 \le k \le N-1$. Consider also a lowpass filter $h(n)$,

. Bandpass

filters can be constructed which have the frequency response

from

The output of the k th bandpass filter is simply (assume $h(n)$ are FIR)

()

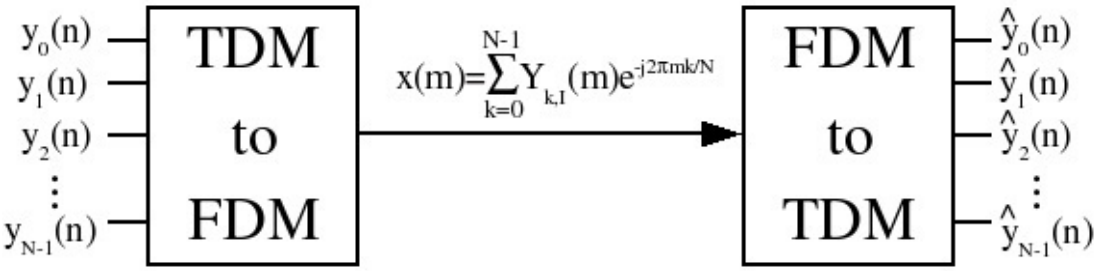
This looks suspiciously like a DFT, except that $M \neq N$, in general. However, if we fix $M = N$, then

we can compute **all** $yk(n)$ outputs simultaneously using an FFT of $x(n-m)\,h(m)$: The kth FFT frequency output= $yk(n)$! So the cost of computing all of these filter banks outputs is

$O[\,N\mathbf{log}N]$, rather than N^2 , per a given n. This is very useful for efficient implementation of **transmultiplexors** (FDM to TDM).

Exercise 2.

How would we implement this efficiently if we wanted to decimate the individual channels



$yk(n)$ by a factor of N, to their approximate Nyquist bandwidth?

Simply step by N time samples between FFTs.

Exercise 3.

Do you expect significant aliasing? If so, how do you propose to combat it? Efficiently?

Aliasing should be expected. There are two ways to reduce it:

1. Decimate by less ("oversample" the individual channels) such as decimating by a factor of .

This is efficiently done by time-stepping by the appropriate factor.

2. Design better (and thus longer) filters, say of length ***LN*** . These can be efficiently computed by producing only *N* (every *L* th) FFT outputs using simplified FFTs.

Exercise 4.

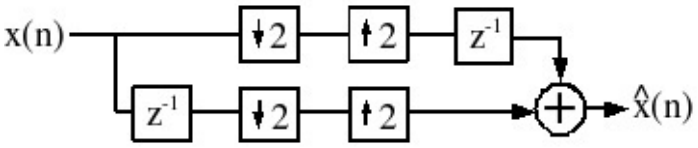
How might one convert from *N* input channels into an FDM signal efficiently? (Figure 5.25)

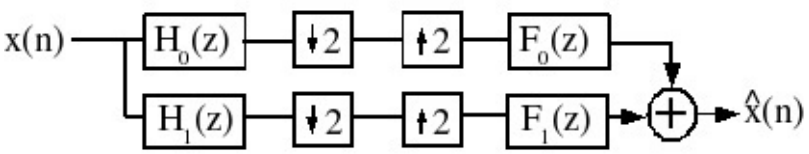
Figure 5.25.

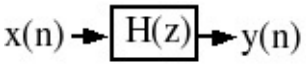
Such systems are used throughout the telephone system, satellite communication links, *etc.*

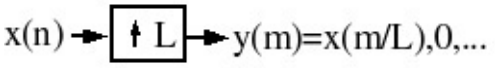
Use an FFT and an inverse FFT for the modulation (TDM to FDM) and demodulation (FDM to TDM), respectively.

5.7. Quadrature Mirror Filterbanks (QMF)*









Although the DFT filterbanks are widely used, there is a problem with aliasing in the decimated channels. At first glance, one might think that this is an insurmountable problem and must simply be accepted. Clearly, with FIR filters and maximal decimation, aliasing will occur. However, a simple example will show that it is possible to **exactly cancel out aliasing** under certain conditions!!!

Consider the following trivial filterbank system, with two channels. (Figure 5.26)

Figure 5.26.

Note

with no error whatsoever, although clearly aliasing occurs in both channels! Note

that the overall data rate is still the Nyquist rate, so there are clearly enough degrees of freedom

available to reconstruct the data, if the filterbank is designed carefully. However, this isn't

splitting the data into separate frequency bands, so one questions whether something other than

this trivial example could work.

Let's consider a general two-channel filterbank, and try to determine conditions under which

aliasing can be cancelled, and the signal can be reconstructed perfectly (Figure 5.27).

Figure 5.27.

Let's derive

, using z-transforms, in terms of the components of this system. Recall

(Figure 5.28) is equivalent to $Y(z) = H(z)\,X(z)$ $Y(\omega) = H(\omega)\,X(\omega)$ Figure 5.28.

and note that (Figure 5.29) is equivalent to

$Y(\omega) = X(\boldsymbol{L\omega})$

Figure 5.29.

[redacted]

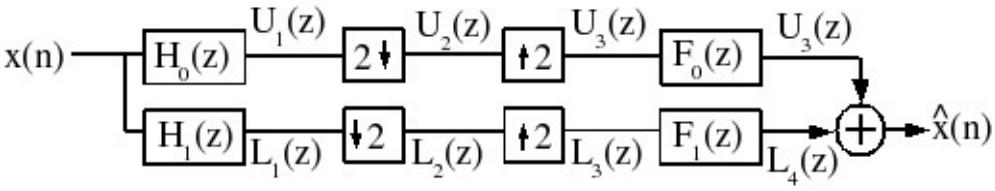
[redacted]

x(n) → ↓M → y(m)=x(Mm)

[redacted]

[redacted]

[redacted]



and (Figure 5.30) is equivalent to

Figure 5.30.

Y(*z*) is derived in the downsampler as follows:

Let *n*= ***Mm*** and

, then

Now

()

so

()

Armed with these results, let's determine

. (Figure 5.31)

Figure 5.31.

Note $U\,1(\,z)= X(\,z)\,H\,0(\,z)$

and

Finally then,

()

Note that the ($X(-\,z)\ \rightarrow\ X(\,\omega+\,\pi)$) corresponds to the aliasing terms!

There are four things we would like to have:

1. No aliasing distortion

2. No phase distortion (overall linear phase → simple time delay)

3. No amplitude distortion

4. FIR filters

No aliasing distortion

By insisting that $H\,0(-\,z)\,F\,0(\,z)+\,H\,1(-\,z)\,F\,1(\,z)=0$, the $X(-\,z)$ component of can be removed, and all

aliasing will be eliminated! There may be many choices for $H\ 0$, $H\ 1$, $F\ 0$, $F\ 1$ that eliminate

aliasing, but most research has focused on the choice $F\ 0(\ z)= H\ 1(– z)$: $F\ 1(\ z)=–(\ H\ 0(– z))$ We will consider only this choice in the following discussion.

Phase distortion

The transfer function of the filter bank, with aliasing cancelled, becomes

,

which with the above choice becomes

. We would like $T(\ z)$ to correspond

to a linear-phase filter to eliminate phase distortion: Call $P(\ z)= H\ 0(\ z)\ H\ 1(– z)$ Note that Note that ($P(– z) \Leftrightarrow (–1)\ np(\ n)$) , and that if $p(\ n)$ is a linear-phase filter, $(–1)\ np(\ n)$ is also (perhaps of the opposite symmetry). Also note that the sum of two linear-phase filters of the

same symmetry (*i.e.* , length of $p(\ n)$ must be **odd**) is also linear phase, so if $p(\ n)$ is an odd-length linear-phase filter, there will be no phase distortion. Also note that

means $p(\ n)=0$, when n is even. If we choose $h\ 0(\ n)$ and

$h\ 1(\ n)$ to be linear phase, $p(\ n)$ will also be linear phase. Thus by choosing $h\ 0(\ n)$ and $h\ 1(\ n)$ to be FIR

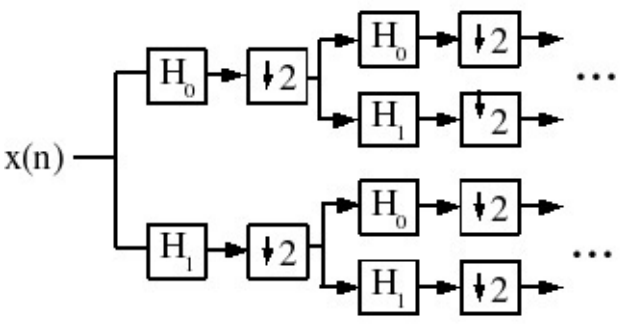
linear phase, we eliminate phase distortion and get FIR filters as well (condition 4).

Amplitude distortion

Assuming aliasing cancellation and elimination of phase distortion, we might also desire no

amplitude distortion ($|\ T(\ \omega)|=1$). All of these conditions require

where c is some constant and D is a linear phase delay. $c=1$ for $|\ T(\ \omega)|=1$. It can be shown by considering that the following can be satisfied!



Thus we require

Any factorization of a $P(z)$ of this form, $P(z)= A(z) B(z)$ can

lead to a Perfect Reconstruction filter bank of the form $H0(z)= A(z)$ $H1(–z)= B(z)$ [This result is attributed to Vetterli.] A well-known special case (Smith and Barnwell)

$H1(z)=–(z–(2D)+1$ $H0(–(z-1)))$ Design techniques exist for optimally choosing the coefficients of these filters, under all of these constraints.

(5.1)

Quadrature Mirror Filters

(H

*

$1(z)= H0(–z) \Leftrightarrow H1(\omega)= H0(\pi+\omega)= H0(\pi-\omega))$ for real-valued filters. The frequency response is "mirrored" around

. This choice leads to

$T(z)= H2$

2

$0(z)- H0(–z)$: it can be shown that this can be a perfect reconstruction system only if

$H0(z)= c0\, z–(2\, n\, 0)+ c1\, z–(2\, n\, 1)$ which isn't a very flexible choice of filters, and not a very good lowpass!

The Smith and Barnwell approach is more commonly used today.

5.8. M-Channel Filter Banks*

The theory of M-band QMFBs and PRFBs has been investigated recently. Some results are

available.

Tree-structured filter banks

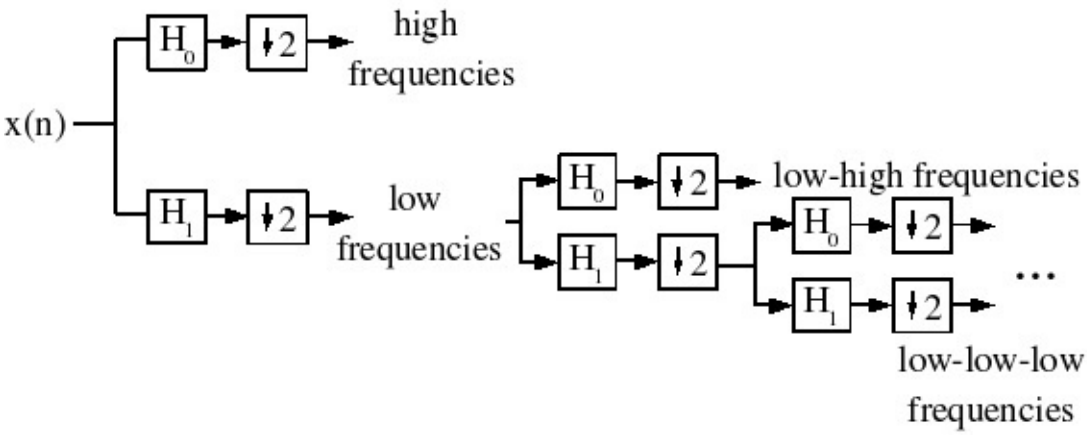
Once we have a two-band PRFB, we can continue to split the subbands with similar systems!

(Figure 5.32)

Figure 5.32.

Thus we can recursively decompose a signal into $2p$ bands, each sampled at $2p$ th the rate of the

original signal, and reconstruct exactly! Due to the tree structure, this can be quite efficient, and in



fact close to the efficiency of an FFT filter bank, which does **not** have perfect reconstruction.

Wavelet decomposition

We need not split both the upper-frequency and lower-frequency bands identically. (Figure 5.33)

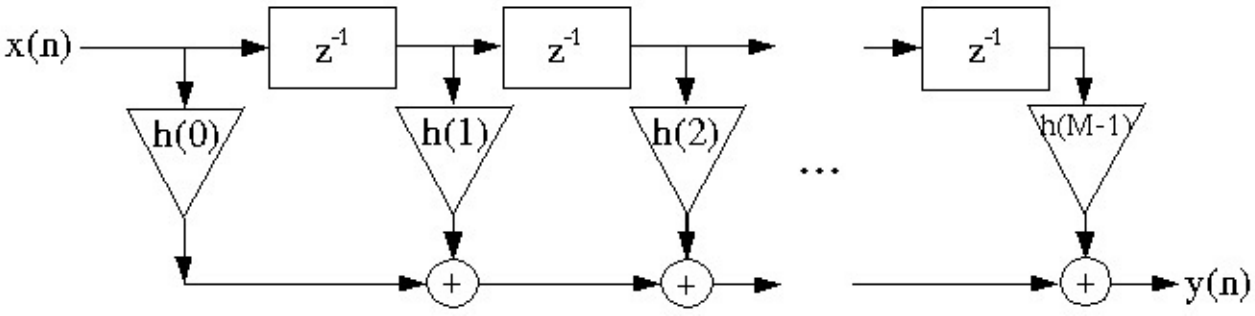
Figure 5.33.

This is good for image coding, because the energy tends to be distributed such that after a wavelet decomposition, each band has roughly equal energy.

Solutions

[redacted]

[redacted]



Chapter 6. Digital Filter Structures and Quantization

Error Analysis

6.1. Filter Structures

Filter Structures*

A realizable filter must require only a finite number of computations per output sample. For linear, causal, time-Invariant filters, this restricts one to rational transfer functions of the form

Assuming no pole-zero cancellations, $H(z)$ is FIR if $ai=0$, $i>0$, and

IIR otherwise. Filter structures usually implement rational transfer functions as difference equations.

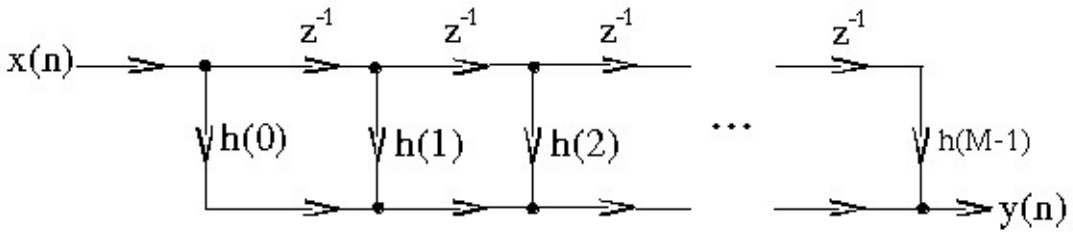
Whether FIR or IIR, a given transfer function can be implemented with many different filter structures. With infinite-precision data, coefficients, and arithmetic, all filter structures implementing the same transfer function produce the same output. However, different filter strucures may produce very different errors with quantized data and finite-precision or fixed-point arithmetic. The computational expense and memory usage may also differ greatly. Knowledge of different filter structures allows DSP engineers to trade off these factors to create the best implementation.

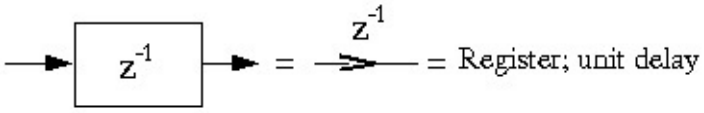
FIR Filter Structures*

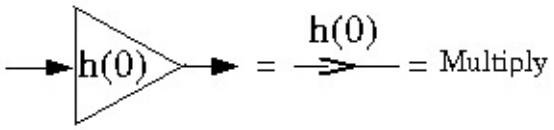
Consider causal FIR filters:

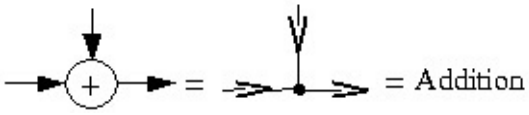
; this can be realized using the following structure

Figure 6.1.









[redacted]

[redacted]

or in a different notation

Figure 6.2.

(a)

(b)

(c)

Figure 6.3.

This is called the **direct-form FIR filter structure**.

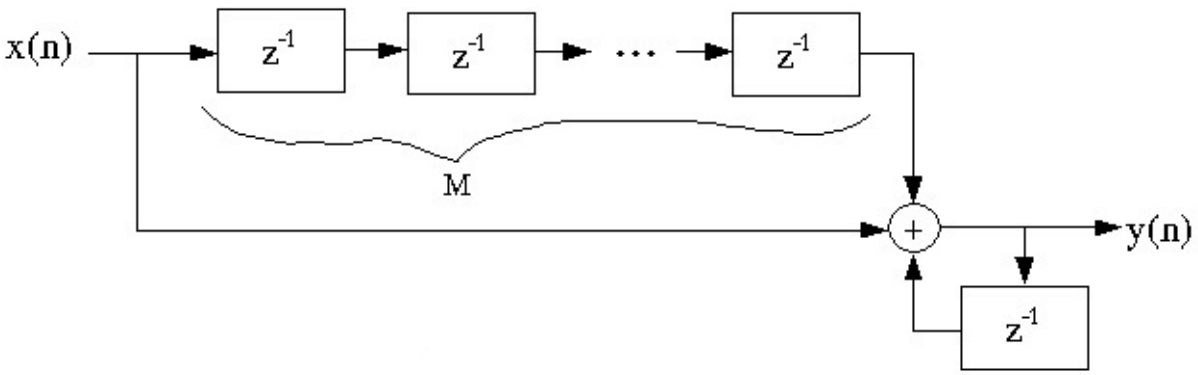
There are no closed loops (no feedback) in this structure, so it is called a **non-recursive structure**. Since any FIR filter can be implemented using the direct-form, non-recursive structure, it is always possible to implement an FIR filter non-recursively. However, it is also possible to implement an FIR filter **recursively**, and for some special sets of FIR filter coefficients this is much more efficient.

Example 6.1.

where

But note that

$y(n) = y(n-1) + x(n) - x(n-M)$ This can be implemented as



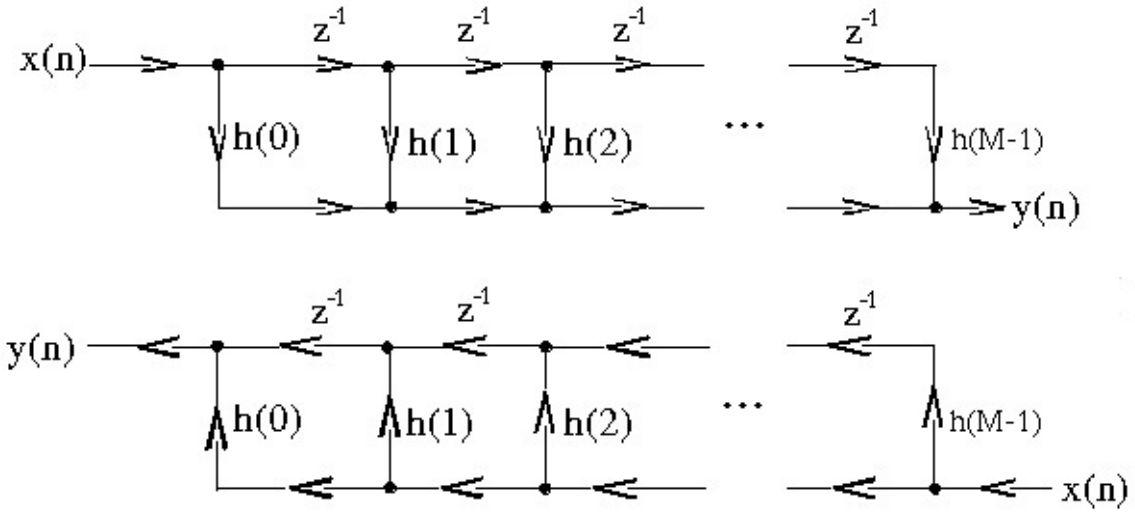


Figure 6.4.

Instead of costing $M-1$ adds/output point, this comb filter costs only two adds/output.

Exercise 1.

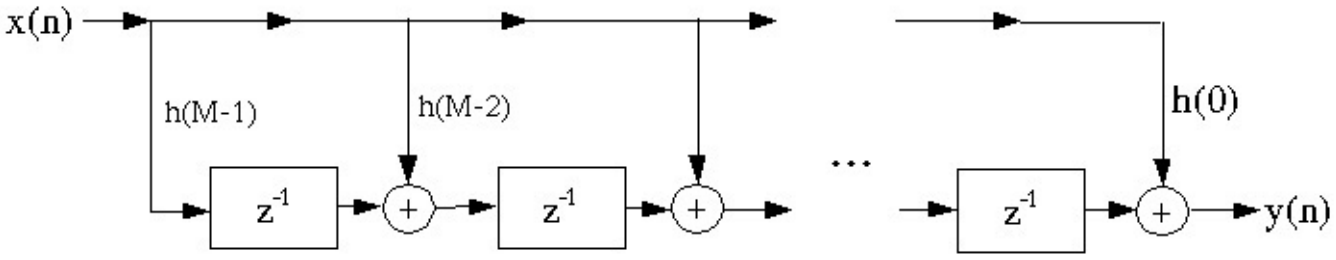
Is this stable, and if not, how can it be made so?

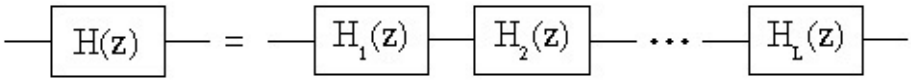
IIR filters must be implemented with a **recursive** structure, since that's the only way a finite number of elements can generate an infinite-length impulse response in a linear, time-invariant (LTI) system. Recursive structures have the advantages of being able to implement IIR systems, and sometimes greater computational efficiency, but the disadvantages of possible instability, limit cycles, and other deletorious effects that we will study shortly.

Transpose-form FIR filter structures

The **flow-graph-reversal theorem** says that if one changes the directions of all the arrows, and inputs at the output and takes the output from the input of a reversed flow-graph, the new system has an identical input-output relationship to the original flow-graph.

Figure 6.5. Direct-form FIR structure





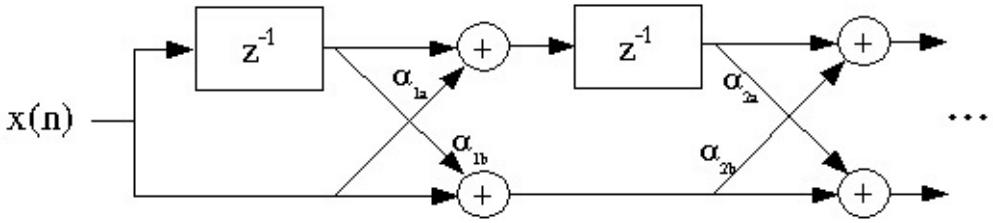


Figure 6.6. reverse = transpose-form FIR filter structure

Figure 6.7. or redrawn

Cascade structures

The z-transform of an FIR filter can be factored into a cascade of short-length filters

$b_0 + b_1 z^{-1} + b_2 z^{-3} + \ldots + b_m z^{-m} = b_0(1 - z_1 z^{-1})(1 - z_2 z^{-1}) \ldots (1 - z_m z^{-1})$ where the z_i are the zeros of this polynomial. Since the coefficients of the polynomial are usually real, the roots are usually

complex-conjugate pairs, so we generally combine

into one quadratic (length-2)

section with **real** coefficients

The overall filter can then be

implemented in a **cascade** structure.

Figure 6.8.

This is occasionally done in FIR filter implementation when one or more of the short-length

filters can be implemented efficiently.

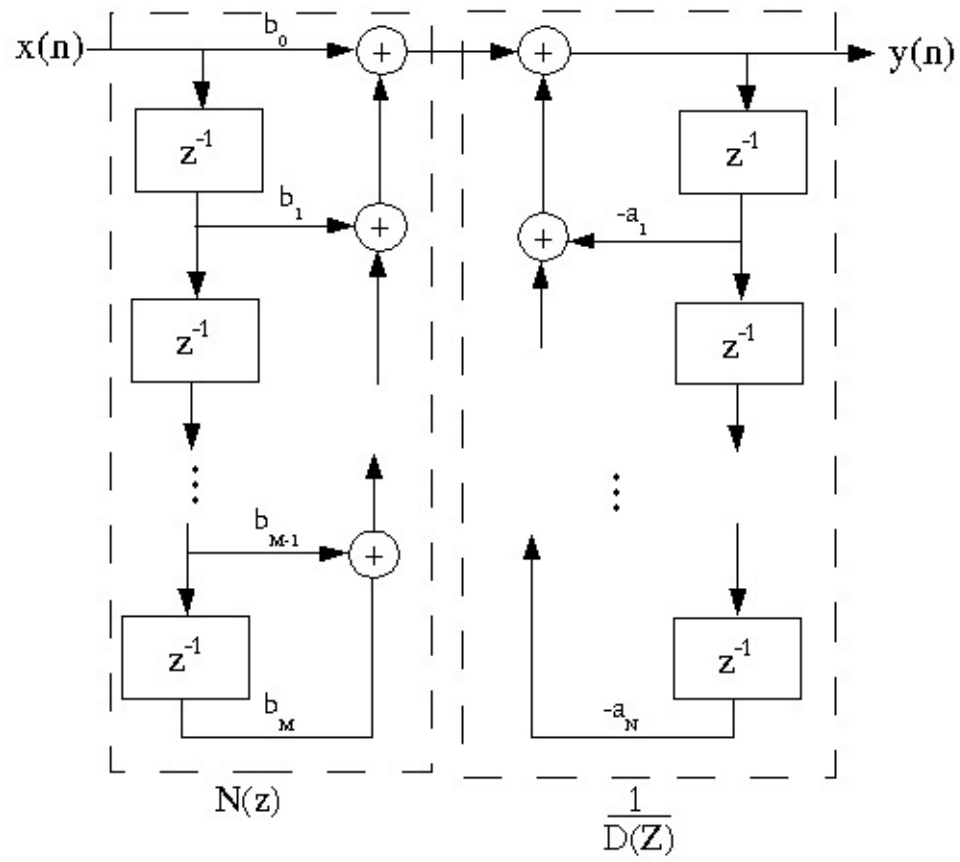
Lattice Structure

It is also possible to implement FIR filters in a lattice structure: this is sometimes used in

adaptive filtering

Figure 6.9.

IIR Filter Structures*



IIR (Infinite Impulse Response) filter structures must be recursive (use feedback); an infinite number of coefficients could not otherwise be realized with a finite number of computations per sample.

The corresponding time-domain difference equation is

$y(n) = -(a_1 y(n-1)) - a_2 y(n-2) + \ldots - a_N y(n-N) + b_0 x(0) + b_1 x(n-1) + \ldots + b_M x(n-M)$ **Direct-form I IIR Filter Structure**

The difference equation above is implemented directly as written by the Direct-Form I IIR Filter Structure.

Figure 6.10.

Note that this is a cascade of two systems, $N(z)$ and

. If we reverse the order of the filters, the

overall system is unchanged: The memory elements appear in the middle and store identical

values, so they can be combined, to form the Direct-Form II IIR Filter Structure.

Direct-Form II IIR Filter Structure

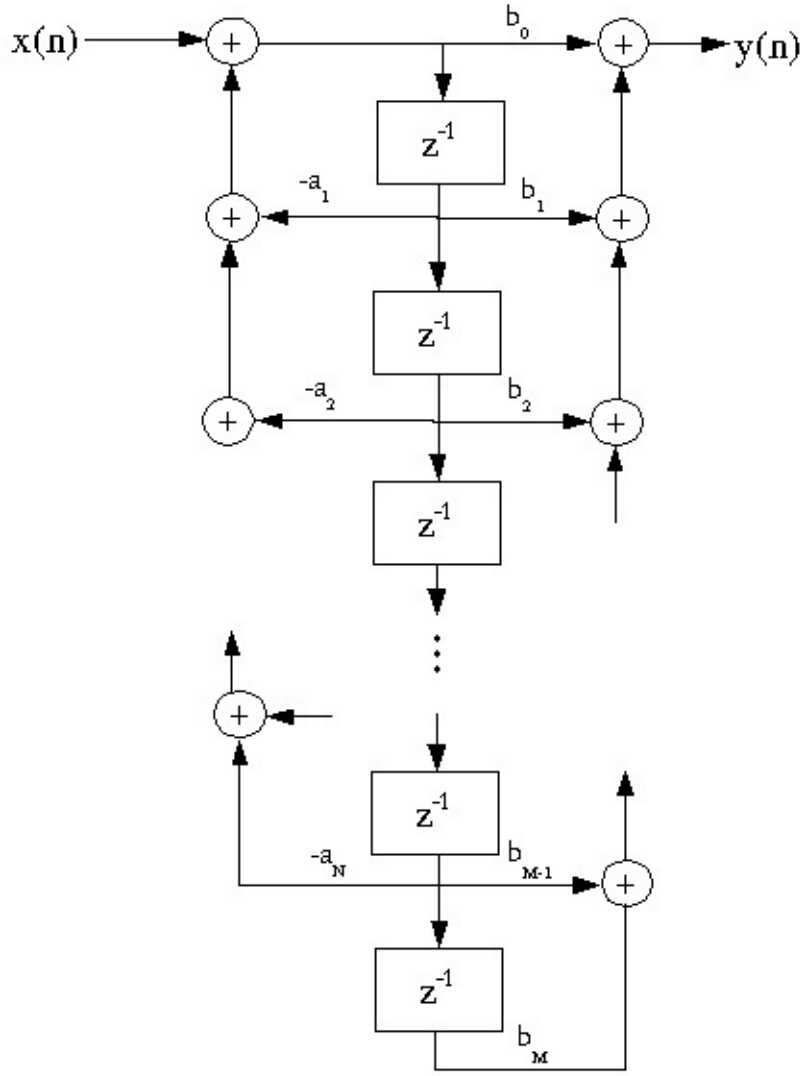


Figure 6.11.

This structure is **canonic**: (i.e., it requires the minimum number of memory elements).

Flowgraph reversal gives the

Transpose-Form IIR Filter Structure

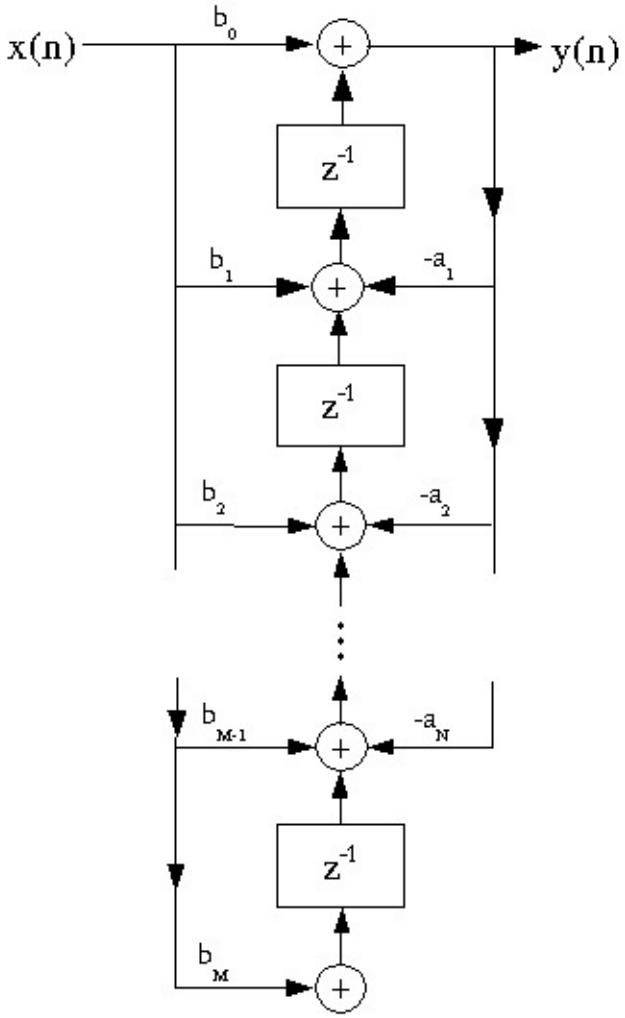


Figure 6.12.

Usually we design IIR filters with $N = M$, but not always.

Obviously, since all these structures have identical frequency response, filter structures are not unique. We consider many different structures because

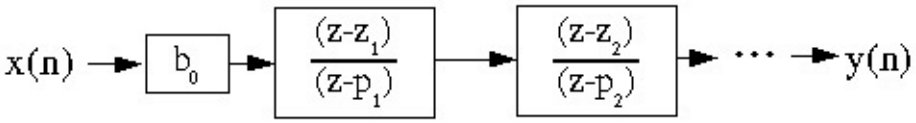
1. Depending on the technology or application, one might be more convenient than another

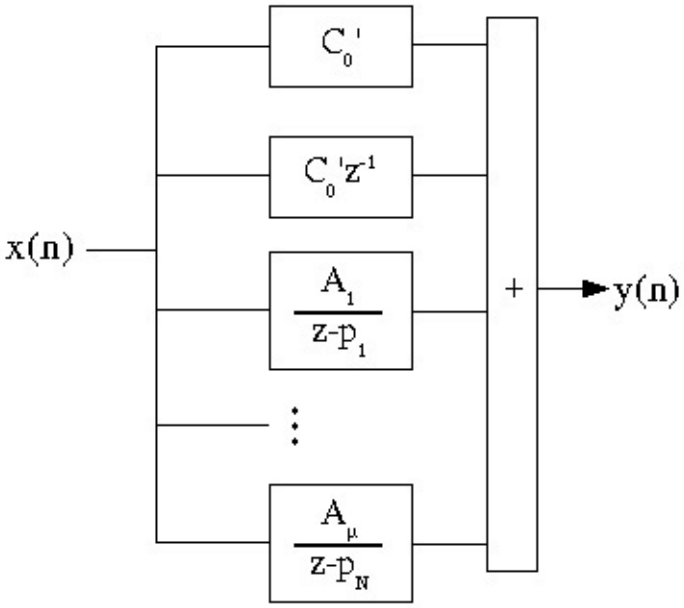
2. The response in a practical realization, in which the data and coefficients must be **quantized**, may differ substantially, and some structures behave much better than others with quantization.

The Cascade-Form IIR filter structure is one of the least sensitive to quantization, which is why it is the most commonly used IIR filter structure.

IIR Cascade Form

The numerator and denominator polynomials can be factored





and implemented as a cascade of short IIR filters.

Figure 6.13.

Since the filter coefficients are usually real yet the roots are mostly complex, we actually implement these as second-order sections, where comple-conjugate pole and zero pairs are combined into second-order sections with real coefficients. The second-order sections are usually implemented with either the Direct-Form II or Transpose-Form structure.

Parallel form

A rational transfer function can also be written as

which by linearity can be implemented as

Figure 6.14.

As before, we combine complex-conjugate pole pairs into second-order sections with real

coefficients.

The cascade and parallel forms are of interest because they are much less sensitive to coefficient quantization than higher-order structures, as analyzed in later modules in this course.

Other forms

There are many other structures for IIR filters, such as wave digital filter structures, lattice-ladder, all-pass-based forms, and so forth. These are the result of extensive research to find structures which are computationally efficient **and** insensitive to quantization error. They all represent various tradeoffs; the best choice in a given context is not yet fully understood, and may never be.

State-Variable Representation of Discrete-Time Systems*

State and the State-Variable Representation

Definition: State

the minimum additional information at time *n*, which, along with all current and future input values, is necessary to compute all future outputs.

Essentially, the state of a system is the information held in the delay registers in a filter structure or signal flow graph.

Fact

Any LTI (linear, time-invariant) system of finite order *M* can be represented by a state-variable description $\mathbf{x}(n+1)=A\mathbf{x}(n)+\mathbf{B}u(n)$ $\mathbf{y}(n)=\mathbf{C}\mathbf{x}(n)+Du(n)$ where **x** is an (*M* x 1) "state vector," $u(n)$ is the input at time *n*, $y(n)$ is the output at time *n*; *A* is an (*M* x *M*) matrix, **B** is an (*M* x 1) vector, **C** is a (1 x *M*) vector, and *D* is a (1 x 1) scalar.

One can always obtain a state-variable description of a signal flow graph.

Example 6.2. 3rd-Order IIR

$y(n)=-(a_1 y(n-1))-a_2 y(n-2)-a_3 y(n-3)+b_0 x(n)+b_1 x(n-1)+b_2 x(n-2)+b_3 x(n-3)$

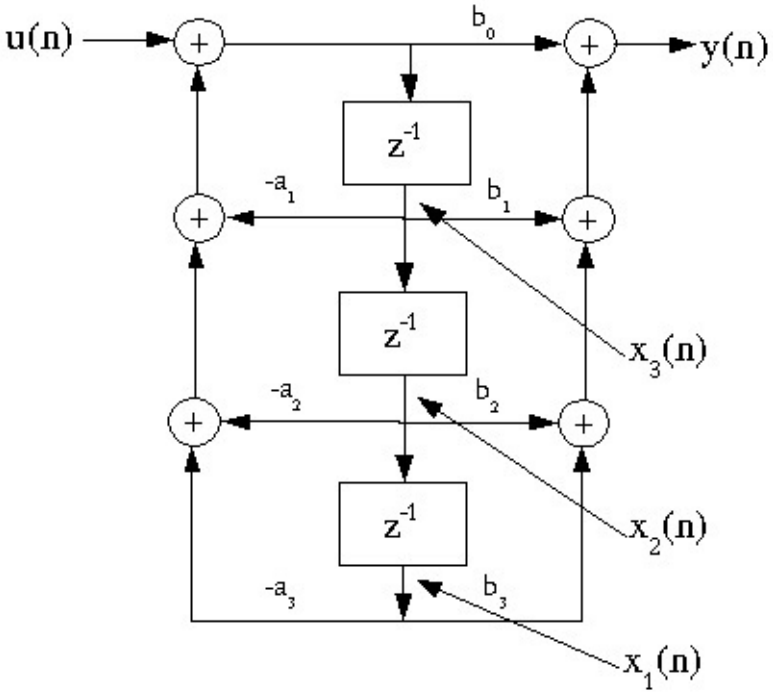


Figure 6.15.

Exercise 2.

Is the state-variable description of a filter $H(z)$ unique?

Exercise 3.

Does the state-variable description fully describe the signal flow graph?

State-Variable Transformation

Suppose we wish to define a new set of state variables, related to the old set by a linear

transformation: $\mathbf{q}(n) = T\mathbf{x}(n)$, where T is a nonsingular (M x M) matrix, and $\mathbf{q}(n)$ is the new state vector. We wish the overall system to remain the same. Note that $\mathbf{x}(n) = T\text{-}1\mathbf{q}(n)$, and thus

$(\mathbf{x}(n+1) = A\mathbf{x}(n) + Bu(n) \Rightarrow T\text{-}1\mathbf{q}(n) = AT\text{-}1\mathbf{q}(n) + Bu(n) \Rightarrow \mathbf{q}(n) = TAT\text{-}1\mathbf{q}(n) + TBu(n))$ ($y(n) = C\mathbf{x}(n) + Du($

This defines a new state system with an input-output behavior identical to the old system, but with

different internal memory contents (states) and state matrices.

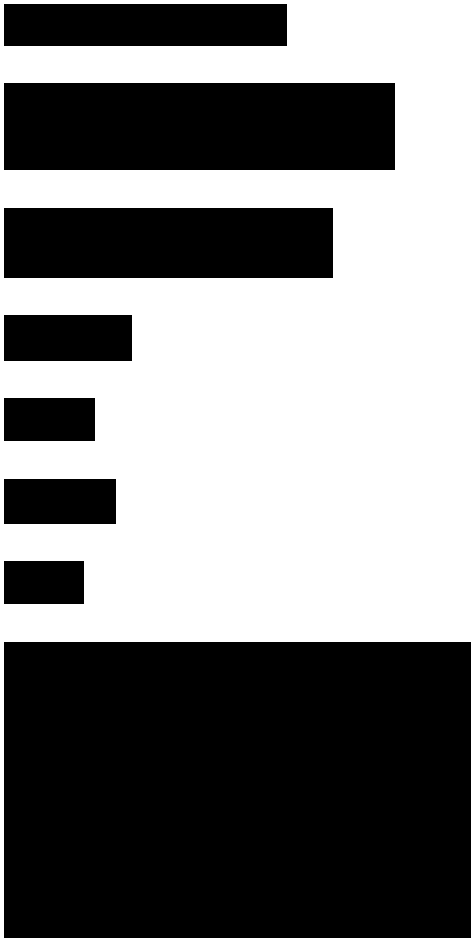
,

,

,

These transformations can be used to generate a wide variety of alternative stuctures or

implementations of a filter.



Transfer Function and the State-Variable Description

Taking the z transform of the state equations

$Z[\mathbf{x}(n+1)] = Z[A\mathbf{x}(n) + Bu(n)]$ $Z[y(n)] = Z[C\mathbf{x}(n) + Du(n)] \Downarrow z\mathbf{X}(z) = A\mathbf{X}(z) + BU(z)$ $\mathbf{X}(z)$ z

Y(*z*)= *C***X**(*n*)+ *DU*(*n*) ((***zI***− *A*)**X**(*z*)= *BU*(*z*) ⇒ **X**(*z*)=(***zI***− *A*)-1 *BU*(*z*)) so **()**

and thus *H*(*z*)= *C*(***zI***− *A*)-1 *B*+ *D* Note that since

, this transfer function is an

M th-order rational fraction in *z*. The denominator polynomial is *D*(*z*)=det(***zI***− *A*) . A discrete-time state system is thus stable if the *M* roots of det(***zI***− *A*) (i.e., the poles of the digital filter) are all inside the unit circle.

Consider the transformed state system with

,

,

,

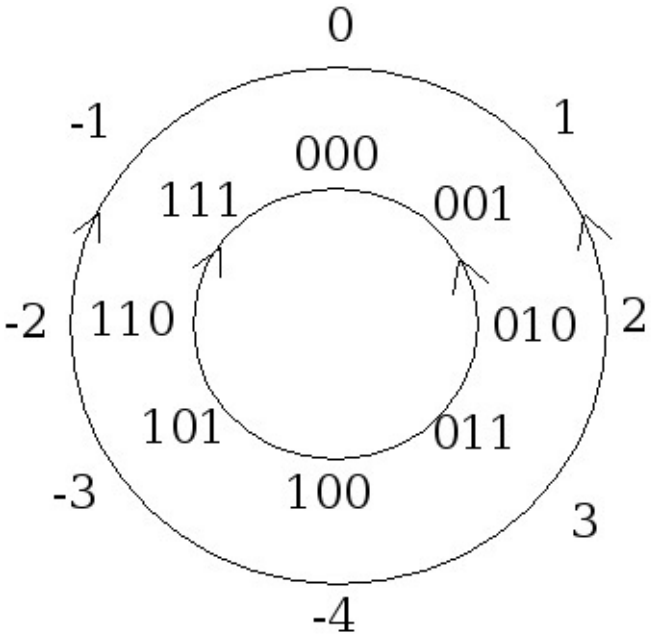
:

()

This proves that state-variable transformation doesn't change the transfer function of the underlying system. However, it can provide alternate forms that are less sensitive to coefficient quantization or easier to analyze, understand, or implement.

State-variable descriptions of systems are useful because they provide a fairly general tool for analyzing all systems; they provide a more detailed description of a signal flow graph than does the transfer function (although not a full description); and they suggest a large class of alternative implementations. They are even more useful in control theory, which is largely based on state descriptions of systems.

6.2. Fixed-Point Numbers



Fixed-Point Number Representation*

Fixed-point arithmetic is generally used when hardware cost, speed, or complexity is important. Finite-precision quantization issues usually arise in fixed-point systems, so we concentrate on fixed-point quantization and error analysis in the remainder of this course. For basic signal processing computations such as digital filters and FFTs, the magnitude of the data, the internal states, and the output can usually be scaled to obtain good performance with a fixed-point implementation.

Two's-Complement Integer Representation

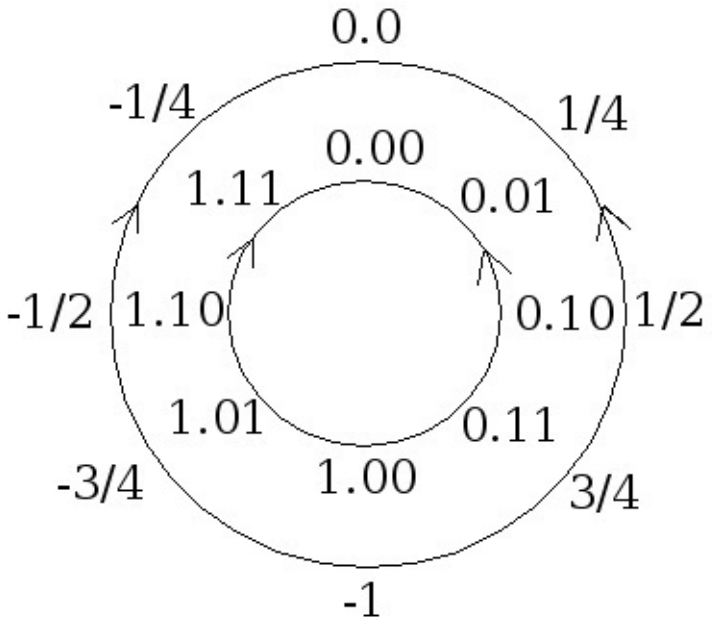
As far as the hardware is concerned, fixed-point number systems represent data as *B*-bit integers. The two's-complement number system is usually used:

Figure 6.16.

The most significant bit is known at the **sign bit**; it is 0 when the number is non-negative; 1 when the number is negative.

Fractional Fixed-Point Number Representation

For the purposes of signal processing, we often regard the fixed-point numbers as binary fractions between [-1, 1) , by implicitly placing a decimal point after the sign bit.



[redacted]

	Fractional Interpretation	Integer Interpretation
0.10	1/2	2
x 0.11	x 3/4	x 3
. 010		
.010		
0.00		
0.0110	3/8	6

Figure 6.17.

or

This interpretation makes it clearer how to implement digital filters in fixed-

point, at least when the coefficients have a magnitude less than 1.

Truncation Error

Consider the multiplication of two binary fractions

Figure 6.18.

Note that full-precision multiplication almost doubles the number of bits; if we wish to return the

product to a B-bit representation, we must truncate the $B-1$ least significant bits. However, this introduces **truncation error** (also known as **quantization error**, or **roundoff error** if the number

	Fractional Interpretation	Integer Interpretation
0.10	1/2	2
+ 0.11	+ 3/4	+ 3
1.01	5/4 = -1/4	5 = -1

is rounded to the nearest B-bit fractional value rather than truncated). Note that this occurs after **multiplication**.

Overflow Error

Consider the addition of two binary fractions;

Figure 6.19.

Note the occurence of wraparound **overflow**; this only happens with **addition**. Obviously, it can be a bad problem.

There are thus two types of fixed-point error: roundoff error, associated with data quantization and multiplication, and overflow error, associated with data quantization and additions. In fixed-point systems, one must strike a balance between these two error sources; by scaling down the data, the occurence of overflow errors is reduced, but the relative size of the roundoff error is increased.

Since multiplies require a number of additions, they are especially expensive in terms of hardware (with a complexity proportional to $BxBh$, where Bx is the number of bits in the data, and Bh is the number of bits in the filter coefficients). Designers try to minimize both Bx and Bh , and often choose $Bx \neq Bh$!

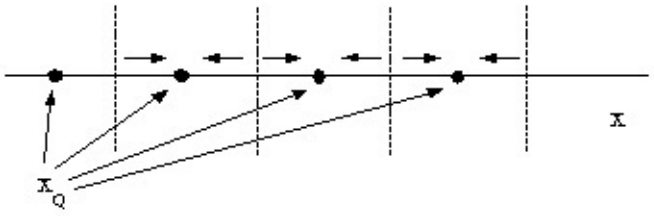
Fixed-Point Quantization*

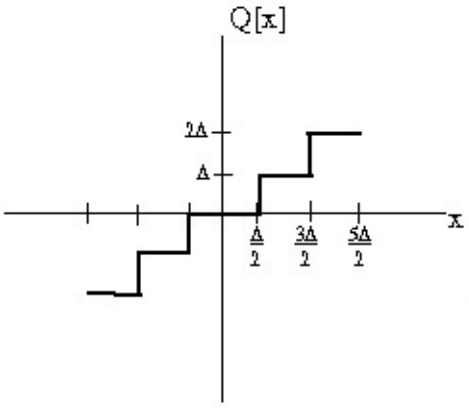
The fractional B-bit two's complement number representation evenly distributes 2^B quantization

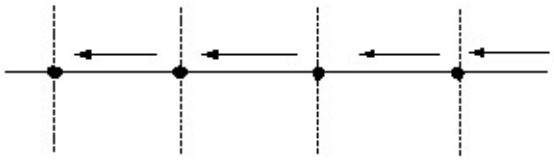
levels between -1 and $1-2-((B-1))$. The spacing between quantization levels is then

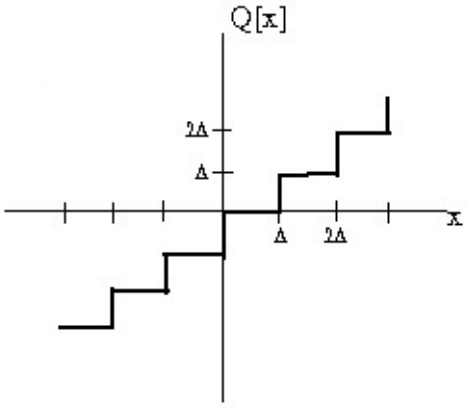
Any signal value falling between two levels is assigned to one of the two levels.

$XQ = Q[x]$ is our notation for quantization. $e = Q[x] - x$ is then the quantization error.









One method of quantization is **rounding**, which assigns the signal value to the **nearest** level. The maximum error is thus

.

(a)

(b)

Figure 6.20.

Another common scheme, which is often easier to implement in hardware, is **truncation**.

$Q[\ x]$ assigns x to the next lowest level.

(a)

(b)

Figure 6.21.

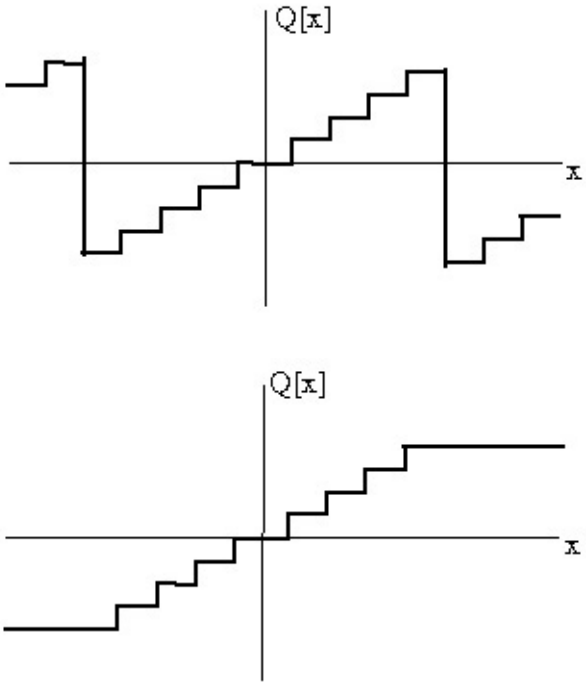
The worst-case error with truncation is $\Delta=2-((\ B-1))$, which is twice as large as with rounding.

Also, the error is always negative, so on average it may have a non-zero mean (i.e., a bias component).

Overflow is the other problem. There are two common types: two's complement (or **wraparound**) overflow, or **saturation** overflow.

<db:title>wraparound</db:title>

<db:title>saturation</db:title>



(a)

(b)

Figure 6.22.

Obviously, overflow errors are bad because they are typically **large**; two's complement (or wraparound) overflow introduces more error than saturation, but is easier to implement in hardware. It also has the advantage that if the **sum** of several numbers is between [-1, 1) , the final answer will be correct even if intermediate sums overflow! However, wraparound overflow leaves IIR systems susceptible to zero-input large-scale limit cycles, as discussed in another module. As usual, there are many tradeoffs to evaluate, and no one right answer for all applications.

6.3. Quantization Error Analysis

Finite-Precision Error Analysis*

Fundamental Assumptions in finite-precision error analysis

Quantization is a highly nonlinear process and is very difficult to analyze precisely.

Approximations and assumptions are made to make analysis tractable.

Assumption #1

The roundoff or truncation errors at any point in a system at each time are **random**, **stationary**, and **statistically independent** (white and independent of all other quantizers in a system).

That is, the error autocorrelation function is r

2

$e[k] = E[ene n + k] = \sigma q \delta[k]$. Intuitively, and

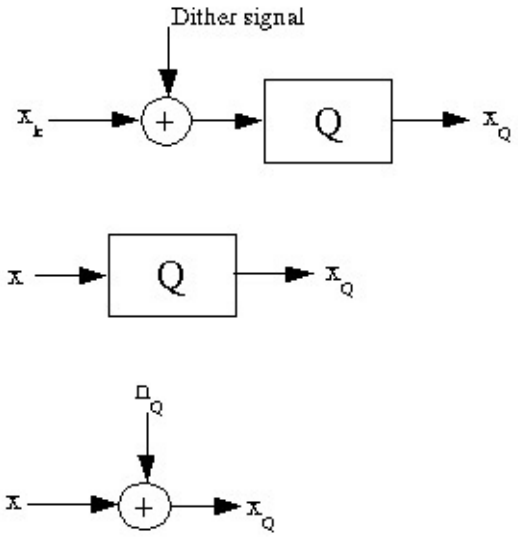
confirmed experimentally in some (but not all!) cases, one expects the quantization error to have a uniform distribution over the interval

for rounding, or $(-\Delta, 0]$ for truncation.

[redacted]

[redacted]

[redacted]



In this case, rounding has zero mean and variance $E[\ Q[\ xn]-\ xn]=0$

and truncation

has the statistics

Please note that the independence assumption may be very bad (for example, when quantizing a sinusoid with an integer period N). There is another quantizing scheme called **dithering**, in which the values are randomly assigned to nearby quantization levels. This can be (and often is) implemented by adding a small (one- or two-bit) random input to the signal before a truncation or rounding quantizer.

Figure 6.23.

This is used extensively in practice. Altough the overall error is somewhat higher, it is spread evenly over all frequencies, rather than being concentrated in spectral lines. This is very important when quantizing sinusoidal or other periodic signals, for example.

Assumption #2

Pretend that the quantization error is really additive **Gaussian** noise with the same mean and variance as the uniform quantizer. That is, model

(a)

<db:title>as</db:title>

(b)

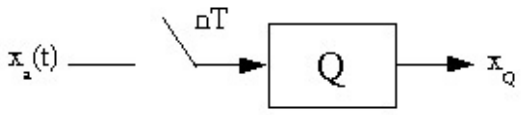
Figure 6.24.

This model is a **linear** system, which our standard theory can handle easily. We model the noise as Gaussian because it remains Gaussian after passing through filters, so analysis in a system context is tractable.

Summary of Useful Statistical Facts

[redacted]

[redacted]



[redacted]

correlation function: ($rx[k] \doteq E[xnx\ n+k]$)

power spectral density: ($Sx(w) \doteq$ DTFT[$rx[n]$]])

Note

(r xy[k] $\doteq E[x^*[n]\ y[n+k]]$)

cross-spectral density: S xy(w)=DTFT[r xy[n]]

For $y= h^* x$: S yx(w)= $H(w)\ Sx(w)$ S yy(w)=(| $H(w)$|)2 $Sx(w)$ Note that the **output** noise level after filtering a noise sequence is

so

postfiltering quantization noise alters the noise power spectrum and may change its variance!

For x 1 , x 2 statistically independent r x

[k]+ r [k] S

(w)+ S (w)

1 + x 2 [k]= rx 1

x 2

x 1 + *x* 2 (*w*)= *Sx* 1

x 2

For independent random variables *σ*

2

2

2

x

= *σ*

+ *σ*

1 + *x* 2

x 1

x 2

Input Quantization Noise Analysis*

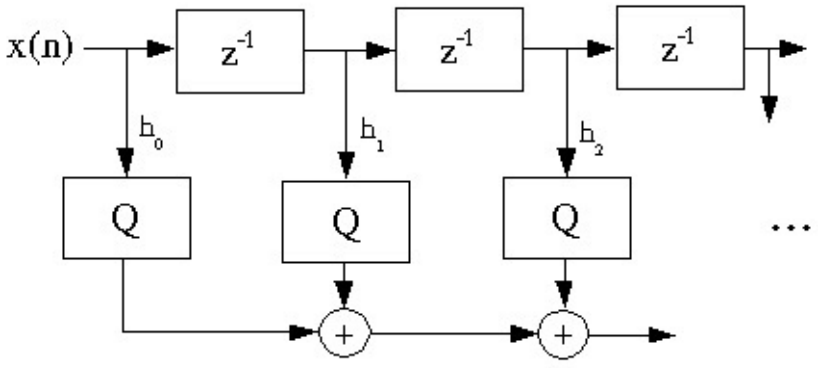
All practical analog-to-digital converters (A/D) must quantize the input data. This can be modeled as an ideal sampler followed by a *B*-bit quantizer.

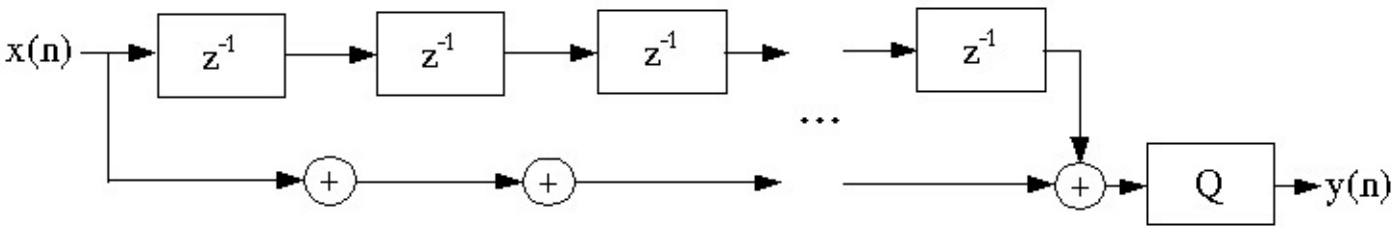
Figure 6.25.

The signal-to-noise ratio (SNR) of an A/D is

()

where *Px* is the power in the signal and *Pn* is the power of the quantization noise, which equals its variance if it has a zero mean. The SNR increases by 6dB with each additional bit.





Quantization Error in FIR Filters*

In digital filters, both the data at various places in the filter, which are continually varying, and the coefficients, which are fixed, must be quantized. The effects of quantization on data and coefficients are quite different, so they are analyzed separately.

Data Quantization

Typically, the input and output in a digital filter are quantized by the analog-to-digital and digital-to-analog converters, respectively. Quantization also occurs at various points in a filter structure, usually after a multiply, since multiplies increase the number of bits.

Direct-form Structures

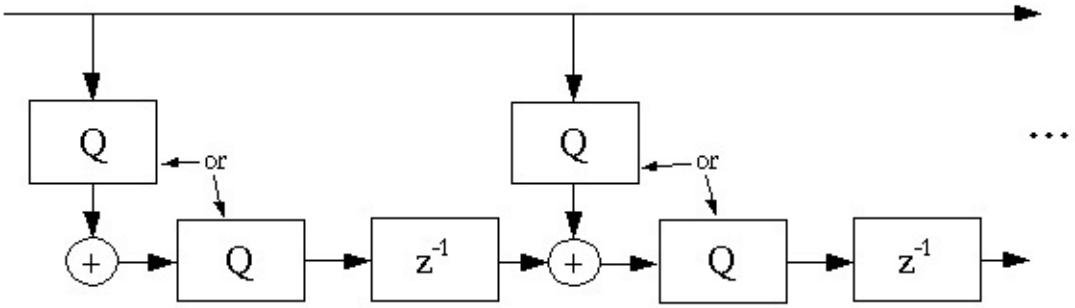
There are two common possibilities for quantization in a direct-form FIR filter structure: after each multiply, or only once at the end.

(a) Single-precision accumulate; total variance

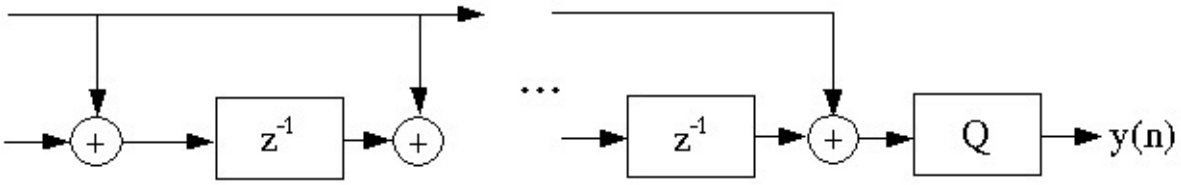
(b) Double-precision accumulate; variance

Figure 6.26.

In the latter structure, a double-length accumulator adds all 2 $B-1$ bits of each product into the accumulating sum, and truncates only at the end. Obviously, this is much preferred, and should **always** be used wherever possible. All DSP microprocessors and most general-pupose computers support double-precision accumulation.



[redacted]



[redacted]

[redacted]

Transpose-form

Similarly, the transpose-form FIR filter structure presents two common options for quantization: after each multiply, or once at the end.

(a) Quantize at each stage before storing intermediate sum. Output variance

<db:title>or</db:title>

(b) Store double-precision partial sums. Costs more memory, but variance

Figure 6.27.

The transpose form is not as convenient in terms of supporting double-precision accumulation, which is a significant disadvantage of this structure.

Coefficient Quantization

Since a quantized coefficient is fixed for all time, we treat it differently than data quantization. The fundamental question is: how much does the quantization affect the frequency response of the filter?

The quantized filter frequency response is

Assuming the

quantization model is correct, $He(w)$ should be fairly random and white, with the error spread fairly equally over all frequencies $w \in [-\pi, \pi)$; however, the randomness of this error destroys any equiripple property or any infinite-precision optimality of a filter.

Exercise 4.

What quantization scheme minimizes the L 2 quantization error in frequency (minimizes

)? On average, how big is this error?

Ideally, if one knows the coefficients are to be quantized to B bits, one should incorporate this directly into the filter design problem, and find the M B-bit binary fractional coefficients minimizing the maximum deviation ($L\infty$ error). This can be done, but it is an integer program, which is known to be np-hard (i.e., requires almost a brute-force search). This is so expensive computationally that it's rarely done. There are some sub-optimal methods that are much more efficient and usually produce pretty good results.

Data Quantization in IIR Filters*

Finite-precision effects are much more of a concern with IIR filters than with FIR filters, since the effects are more difficult to analyze and minimize, coefficient quantization errors can cause the filters to become unstable, and disastrous things like large-scale limit cycles can occur.

Roundoff noise analysis in IIR filters

Suppose there are several quantization points in an IIR filter structure. By our simplifying

assumptions about quantization error and Parseval's theorem, the quantization noise variance

σ 2

y,i at the output of the filter from the *i* th quantizer is

()

where σ 2

n is the variance of the quantization error at the *i* th quantizer, *S* (*w*) is the power spectral

i

ni

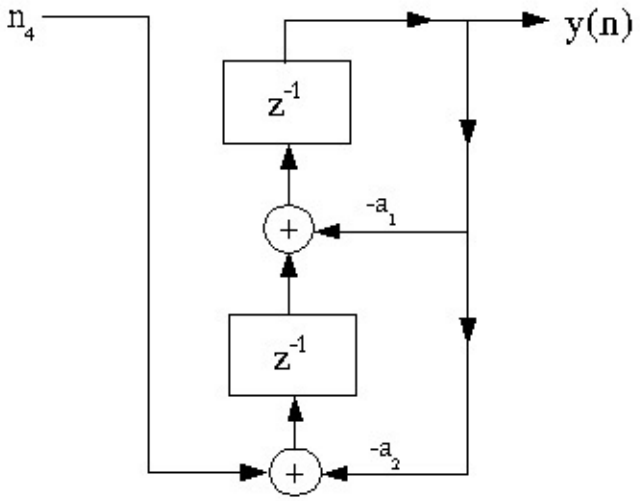
density of that quantization error, and *Hi*(*w*) is the transfer function from the *i* th quantizer to the output point. Thus for *P* independent quantizers in the structure, the total quantization noise

variance is

Note that in general, each *Hi*(*w*) , and thus the variance at the

output due to each quantizer, is different; for example, the system as seen by a quantizer at the

input to the first delay state in the Direct-Form II IIR filter structure to the output, call it *n* 4 , is



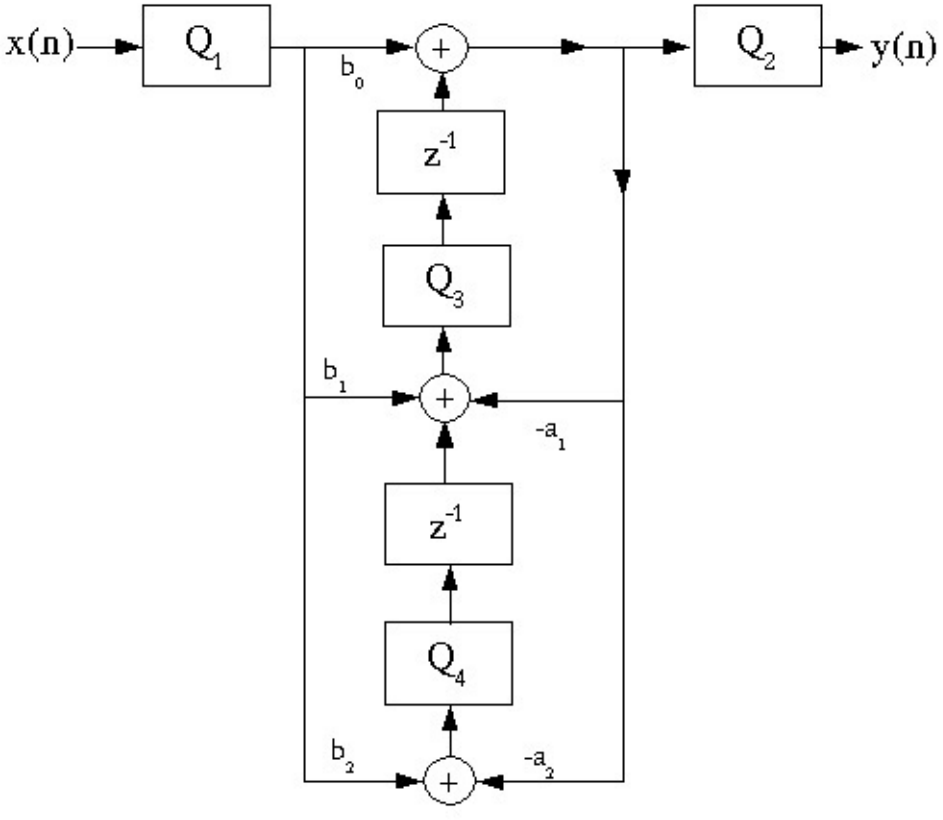


Figure 6.28.

with a transfer function

which can be evaluated at *z*= *eiw* to obtain the frequency

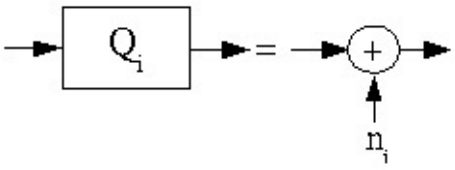
response.

A general approach to find *Hi*(*w*) is to write state equations for the equivalent structure as seen by *ni* , and to determine the transfer function according to *H*(*z*)= *C*(***zI***− *A*)-1 *B*+ *d* .

Figure 6.29.

Exercise 5.

The above figure illustrates the quantization points in a typical implementation of a Direct-Form



The above figure illustrates the quantization points in a typical implementation of a Direct-Form

II IIR second-order section. What is the total variance of the output error due to all of the quantizers in the system?

By making the assumption that each *Qi* represents a noise source that is white, independent of the

other sources, and additive,

Figure 6.30.

the variance at the output is the sum of the variances at the output due to each noise source:

The variance due to each noise source at the output can be determined from

; note that S

2

$n\ (\ w)= \sigma$

by our assumptions, and H

i

ni

$i(\ w)$ is the transfer function

from the noise source to the output.

IIR Coefficient Quantization Analysis*

Coefficient quantization is an important concern with IIR filters, since straigthforward quantization often yields poor results, and because quantization can produce unstable filters.

Sensitivity analysis

The performance and stability of an IIR filter depends on the pole locations, so it is important to know how quantization of the filter coefficients ak affects the pole locations pj . The denominator polynomial is

We wish to know

, which, for small deviations,

will tell us that a δ change in ak yields an

change in the pole location.

is the

sensitivity of the pole location to quantization of ak . We can find

using the chain rule.

⇓

which is

()

Note that as the poles get closer together, the sensitivity increases greatly. So as the filter order increases and more poles get stuffed closer together inside the unit circle, the error introduced by coefficient quantization in the pole locations grows rapidly.

How can we reduce this high sensitivity to IIR filter coefficient quantization?

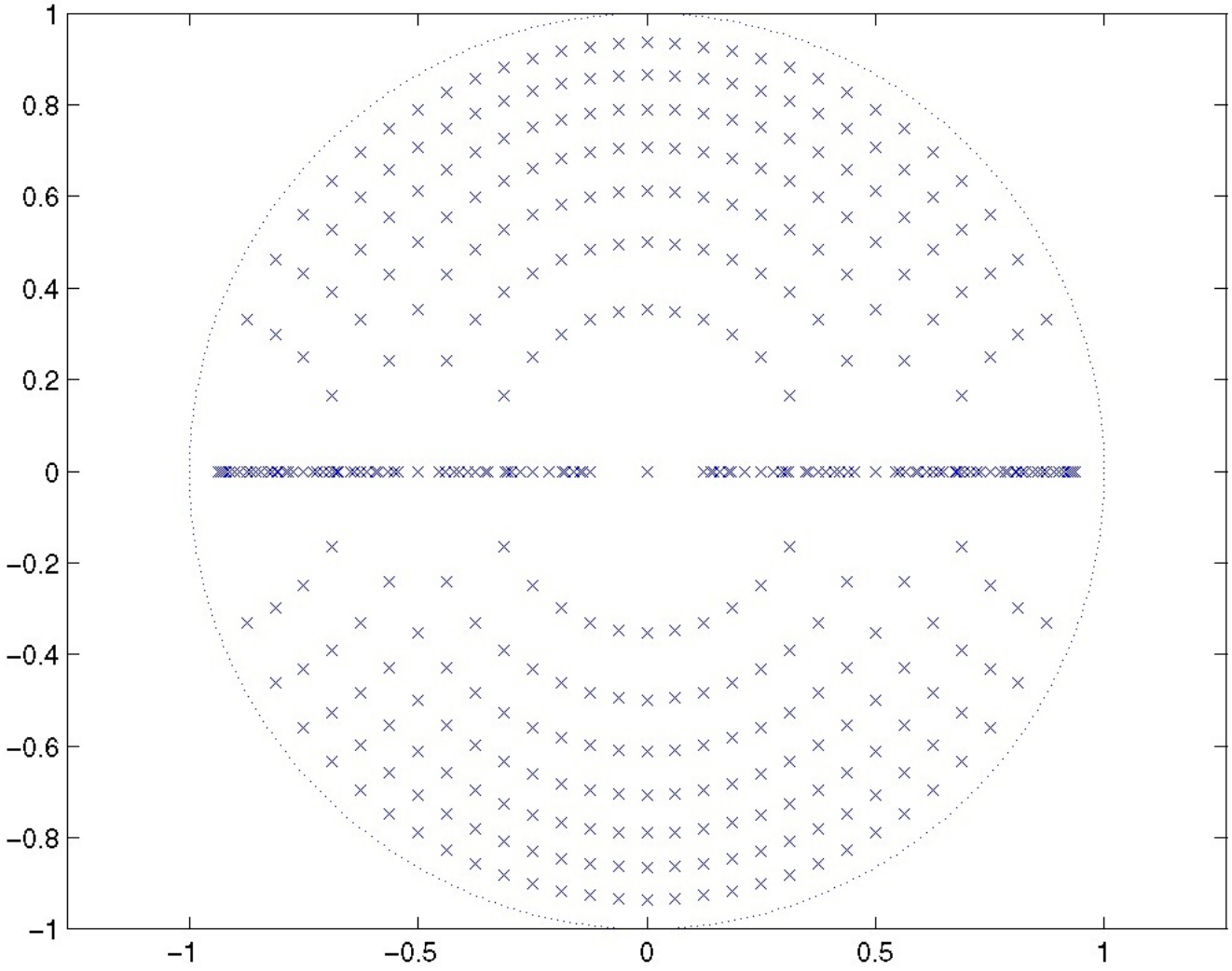
Solution

Cascade or **parallel form** implementations! The numerator and denominator polynomials can be factored off-line at very high precision and grouped into second-order sections, which are then quantized section by section. The sensitivity of the quantization is thus that of second-order, rather than *N*-th order, polynomials. This yields major improvements in the frequency response of the overall filter, and is almost always done in practice.

Note that the numerator polynomial faces the same sensitivity issues; the **cascade** form also improves the sensitivity of the zeros, because they are also factored into second-order terms. However, in the **parallel** form, the zeros are globally distributed across the sections, so they suffer from quantization of all the blocks. Thus the **cascade** form preserves zero locations much better than the parallel form, which typically means that the stopband behavior is better in the cascade form, so it is most often used in practice.

Note on FIR Filters

On the basis of the preceding analysis, it would seem important to use cascade structures in FIR filter implementations. However, most FIR filters are linear-phase and thus symmetric or anti-symmetric. As long as the quantization is implemented such that the filter coefficients retain symmetry, the filter retains linear phase. Furthermore, since all zeros off the unit circle must appear in groups of four for symmetric linear-phase filters, zero pairs can leave the unit circle only by joining with another pair. This requires relatively severe quantizations (enough to completely remove or change the sign of a ripple in the amplitude response). This "reluctance" of pole pairs to leave the unit circle tends to keep quantization from damaging the frequency response as much as might be expected, enough so that cascade structures are rarely used for FIR filters.



Exercise 6.

What is the worst-case pole pair in an IIR digital filter?

The pole pair closest to the real axis in the z-plane, since the complex-conjugate poles will be

closest together and thus have the highest sensitivity to quantization.

Quantized Pole Locations

In a **direct-form** or **transpose-form** implementation of a second-order section, the filter coefficients are quantized versions of the polynomial coefficients.

$p = re^{i\theta}$ $D(z) = z^2 - 2r\cos(\theta) + r^2$ So $a_1 = -(2r\cos(\theta))$ $a_2 = r^2$ Thus the quantization of a_1 and a_2 to B bits restricts the radius r to

, and $a_1 = -(2\mathrm{Re}(p)) = k\Delta B$ The following figure

shows all stable pole locations after four-bit two's-complement quantization.

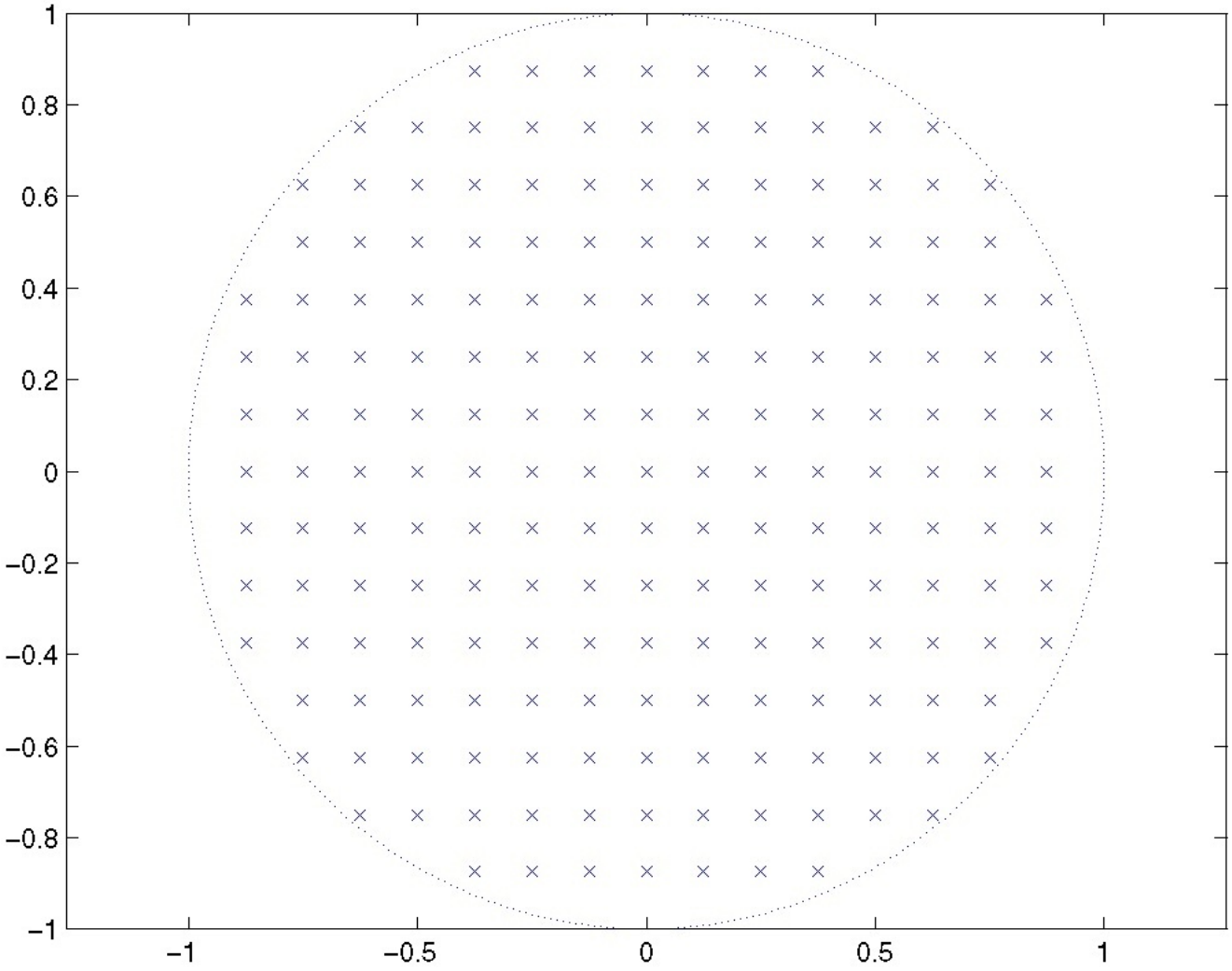


Figure 6.31.

Note the nonuniform distribution of possible pole locations. This might be **good** for poles near

$r=1$,

, but not so good for poles near the origin or the Nyquist frequency.

In the "normal-form" structures, a **state-variable** based realization, the poles are uniformly spaced.

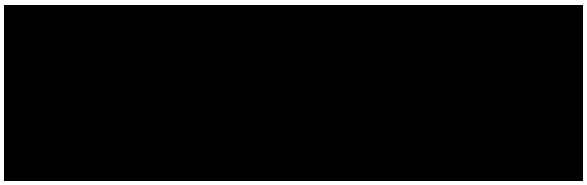
Figure 6.32.

This can only be accomplished if the coefficients to be quantized equal the real and imaginary

parts of the pole location; that is, $\alpha 1= r \cos(\theta)=\mathrm{Re}(r)$ $\alpha 2= r \sin(\theta)=\mathrm{Im}(p)$ This is the case for a 2nd-

order system with the **state matrix**

: The denominator polynomial is



()

Given any second-order filter coefficient set, we can write it as a **state-space system**, find a

transformation matrix *T* such that

is in normal form, and then implement the second-

order section using a structure corresponding to the state equations.

The normal form has a number of other advantages; both eigenvalues are equal, so it minimizes the norm of ***Ax*** , which makes overflow less likely, and it minimizes the output variance due to quantization of the state values. It is sometimes used when minimization of finite-precision effects is critical.

Exercise 7.

What is the disadvantage of the normal form?

It requires more computation. The general **state-variable equation** requires nine multiplies, rather than the five used by the **Direct-Form II** or **Transpose-Form** structures.

6.4. Overflow Problems and Solutions

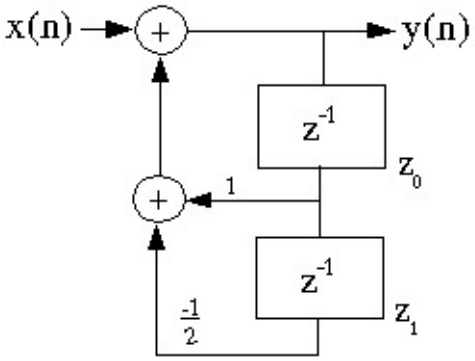
Limit Cycles*

Large-scale limit cycles

When overflow occurs, even otherwise stable filters may get stuck in a **large-scale limit cycle**, which is a short-period, almost full-scale persistent filter output caused by overflow.

Example 6.3.

Consider the second-order system

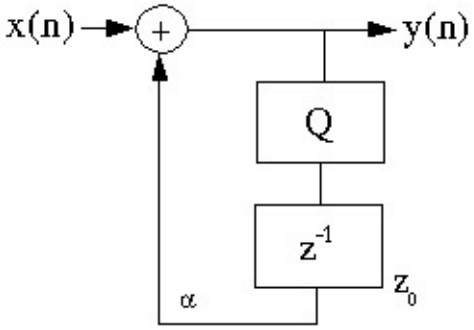


[redacted]

[redacted]

[redacted]

[redacted]



[redacted]

[redacted]

Figure 6.33.

with zero input and initial state values $z\,0[0]=0.8$, $z\,1[0]=-0.8$. Note $y[\,n]= z\,0[\,n+1]$.

The filter is obviously stable, since the magnitude of the poles is

, which is well inside

the unit circle. However, with wraparound overflow, note that

, and

that

, so

even with zero input.

Clearly, such behavior is intolerable and must be prevented. Saturation arithmetic has been proved to prevent **zero-input limit cycles**, which is one reason why all DSP microprocessors support this feature. In many applications, this is considered sufficient protection. Scaling to prevent overflow is another solution, if as well the inital state values are never initialized to limit-cycle-producing values. The normal-form structure also reduces the chance of overflow.

Small-scale limit cycles

Small-scale limit cycles are caused by quantization. Consider the system

Figure 6.34.

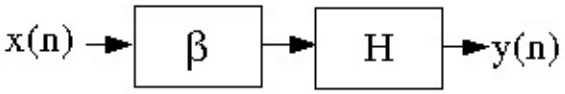
Note that when

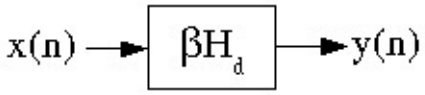
, rounding will quantize the output to the current level (with zero input),

so the output will remain at this level forever. Note that the maximum amplitude of this "small-scale limit cycle" is achieved when

In a higher-order system, the

small-scale limit cycles are oscillatory in nature. Any quantization scheme that never increases





[redacted]

[redacted]

[redacted]

the magnitude of any quantized value prevents small-scale limit cycles.

Two's-complement truncation does **not** do this; it increases the magnitude of negative numbers.

However, this introduces greater error and bias. Since the level of the limit cycles is proportional to ΔB , they can be reduced by increasing the number of bits. Poles close to the unit circle increase the magnitude and likelihood of small-scale limit cycles.

Scaling*

Overflow is clearly a serious problem, since the errors it introduces are very large. As we shall see, it is also responsible for large-scale limit cycles, which cannot be tolerated. One way to prevent overflow, or to render it acceptably unlikely, is to **scale** the input to a filter such that overflow cannot (or is sufficiently unlikely to) occur.

Figure 6.35.

In a fixed-point system, the range of the input signal is limited by the fractional fixed-point number representation to $|x[n]| \leq 1$. If we scale the input by multiplying it by a value β, $0 < \beta < 1$, then $|\beta x[n]| \leq \beta$.

Another option is to incorporate the scaling directly into the filter coefficients.

Figure 6.36.

FIR Filter Scaling

What value of β is required so that the output of an FIR filter cannot overflow ($|y(n)| \leq 1$, $|x(n)| \leq 1$)?

⇓

Alternatively, we can

incorporate the scaling directly into the filter, and require that

to prevent overflow.

IIR Filter Scaling

To prevent the output from overflowing in an IIR filter, the condition above still holds: ($M=\infty$)

so an initial scaling factor

can be used, or the filter itself can be scaled.

However, it is also necessary to prevent the **states** from overflowing, and to prevent overflow at any point in the signal flow graph where the arithmetic hardware would thereby produce errors. To prevent the states from overflowing, we determine the transfer function from the input to all states i, and scale the filter such that

Although this method of scaling guarantees no overflows, it is often too conservative. Note that a worst-case signal is $x(n)=\text{sign}(h(-n))$; this input may be extremely unlikely. In the relatively common situation in which the input is expected to be mainly a single-frequency sinusoid of unknown frequency and amplitude less than 1, a scaling condition of $|H(w)|\leq 1$ is sufficient to guarantee no overflow. This scaling condition is often used. If there are several potential overflow locations i in the digital filter structure, the scaling conditions are $|Hi(w)|\leq 1$ where $Hi(w)$ is the frequency response from the input to location i in the filter.

Even this condition may be excessively conservative, for example if the input is more-or-less random, or if occasional overflow can be tolerated. In practice, experimentation and simulation are often the best ways to optimize the scaling factors in a given application.

For filters implemented in the cascade form, rather than scaling for the entire filter at the beginning, (which introduces lots of quantization of the input) the filter is usually scaled so that each stage is just prevented from overflowing. This is best in terms of reducing the quantization noise. The scaling factors are incorporated either into the previous or the next stage, whichever is most convenient.

Some heurisitc rules for grouping poles and zeros in a cascade implementation are:

1. Order the poles in terms of decreasing radius. Take the pole pair closest to the unit circle and group it with the zero pair closest to that pole pair (to minimize the gain in that section). Keep doing this with all remaining poles and zeros.

2. Order the section with those with highest gain (argmax| *Hi*(*w*)|) in the middle, and those with lower gain on the ends.

Leland B. Jackson [**link**] has an excellent intuitive discussion of finite-precision problems in digital filters. The book by *Roberts and Mullis* [**link**] is one of the most thorough in terms of detail.

References

1. Leland B. Jackson. (1989). *Digital Filters and Signal Processing*. (2nd Edition). Kluwer Academic Publishers.

2. Richard A. Roberts and Clifford T. Mullis. (1987). *Digital Signal Processing*. Prentice Hall.

Glossary

Definition: State

the minimum additional information at time simplemath *mathml-miitalicsn* n, which, along with all current and future input values, is necessary to compute all future outputs.

Solutions

Index

A

B

C

D

P

Q

R

S

T

U

V

Attributions

Collection: **DSPA**

Edited by: Janko Calic

Edited by: Catherine Elder and Catherine Elder

URL: http://cnx.org/content/col10599/1.5/

Module: **Preface for Digital Signal Processing: A User's Guide**

By: Douglas Jones

URL: http://cnx.org/content/m13782/1.1/

Module: **Discrete-Time Signals and Systems**

By: Don Johnson

URL: http://cnx.org/content/m10342/2.15/

Module: **Systems in the Time-Domain**

By: Don Johnson

URL: http://cnx.org/content/m0508/2.7/

Module: **Discrete Time Convolution**

By: Ricardo Radaelli-Sanchez, Richard Baraniuk, Stephen Kruzick, and Catherine Elder

Edited by: Catherine Elder

URL: http://cnx.org/content/m10087/2.27/

Module: **Introduction to Fourier Analysis**

By: Richard Baraniuk

URL: http://cnx.org/content/m10096/2.12/

Module: **Continuous Time Fourier Transform (CTFT)**

By: Richard Baraniuk and Melissa Selik

URL: http://cnx.org/content/m10098/2.16/

Copyright: Richard Baraniuk and Melissa Selik

License: http://creativecommons.org/licenses/by/3.0/

Module: **Discrete-Time Fourier Transform (DTFT)**

By: Don Johnson

URL: http://cnx.org/content/m10247/2.31/

Copyright: Don Johnson

License: http://creativecommons.org/licenses/by/1.0

Module: **DFT as a Matrix Operation**

By: Robert Nowak

URL: http://cnx.org/content/m10962/2.5/

Copyright: Robert Nowak

License: http://creativecommons.org/licenses/by/1.0

Module: **Introduction**

By: Anders Gjendemsjø

URL: http://cnx.org/content/m11419/1.29/

Copyright: Anders Gjendemsjø

License: http://creativecommons.org/licenses/by/1.0

Module: **Proof**

By: Anders Gjendemsjø

URL: http://cnx.org/content/m11423/1.27/

Copyright: Anders Gjendemsjø

License: http://creativecommons.org/licenses/by/1.0

Module: **Illustrations**

By: Anders Gjendemsjø

URL: http://cnx.org/content/m11443/1.33/

Copyright: Anders Gjendemsjø

License: http://creativecommons.org/licenses/by/1.0

Module: **Systems view of sampling and reconstruction**

By: Anders Gjendemsjø

URL: http://cnx.org/content/m11465/1.20/

Copyright: Anders Gjendemsjø

License: http://creativecommons.org/licenses/by/1.0

Module: **Sampling CT Signals: A Frequency Domain Perspective**

By: Robert Nowak

URL: http://cnx.org/content/m10994/2.2/

Copyright: Robert Nowak

License: http://creativecommons.org/licenses/by/1.0

Module: **The DFT: Frequency Domain with a Computer Analysis**

By: Robert Nowak

URL: http://cnx.org/content/m10992/2.3/

Copyright: Robert Nowak

License: http://creativecommons.org/licenses/by/1.0

Module: **Discrete-Time Processing of CT Signals**

By: Robert Nowak

URL: http://cnx.org/content/m10993/2.2/

Copyright: Robert Nowak

License: http://creativecommons.org/licenses/by/1.0

Module: **Difference Equation**

By: Michael Haag

URL: http://cnx.org/content/m10595/2.6/

Copyright: Michael Haag

License: http://creativecommons.org/licenses/by/1.0

Module: **The Z Transform: Definition**

By: Benjamin Fite

URL: http://cnx.org/content/m10549/2.10/

Copyright: Benjamin Fite

License: http://creativecommons.org/licenses/by/1.0

Module: **Table of Common z-Transforms**

By: Melissa Selik and Richard Baraniuk

URL: http://cnx.org/content/m10119/2.14/

Copyright: Melissa Selik and Richard Baraniuk

License: http://creativecommons.org/licenses/by/1.0

Module: **Understanding Pole/Zero Plots on the Z-Plane**

By: Michael Haag

URL: http://cnx.org/content/m10556/2.12/

Copyright: Michael Haag

License: http://creativecommons.org/licenses/by/3.0/

Module: **Overview of Digital Filter Design**

By: Douglas Jones

URL: http://cnx.org/content/m12776/1.2/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **Linear Phase Filters**

By: Douglas Jones

URL: http://cnx.org/content/m12802/1.2/

Module: **Window Design Method**

By: Douglas Jones

URL: http://cnx.org/content/m12790/1.2/

Module: **Frequency Sampling Design Method for FIR filters**

By: Douglas Jones

URL: http://cnx.org/content/m12789/1.2/

Module: **Parks-McClellan FIR Filter Design**

By: Douglas Jones

URL: http://cnx.org/content/m12799/1.3/

Module: **Overview of IIR Filter Design**

By: Douglas Jones

URL: http://cnx.org/content/m12758/1.2/

Module: **Prototype Analog Filter Design**

By: Douglas Jones

URL: http://cnx.org/content/m12763/1.2/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **IIR Digital Filter Design via the Bilinear Transform**

By: Douglas Jones

URL: http://cnx.org/content/m12757/1.2/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **Impulse-Invariant Design**

By: Douglas Jones

URL: http://cnx.org/content/m12760/1.2/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **Digital-to-Digital Frequency Transformations**

By: Douglas Jones

URL: http://cnx.org/content/m12759/1.2/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **Prony's Method**

By: Douglas Jones

URL: http://cnx.org/content/m12762/1.2/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **Linear Prediction**

By: Douglas Jones

URL: http://cnx.org/content/m12761/1.2/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **DFT Definition and Properties**

By: Douglas Jones

URL: http://cnx.org/content/m12019/1.5/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **Spectrum Analysis Using the Discrete Fourier Transform**

By: Douglas Jones

URL: http://cnx.org/content/m12032/1.6/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **Classical Statistical Spectral Estimation**

By: Douglas Jones

URL: http://cnx.org/content/m12014/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **Short Time Fourier Transform**

By: Ivan Selesnick

URL: http://cnx.org/content/m10570/2.4/

Copyright: Ivan Selesnick

License: http://creativecommons.org/licenses/by/1.0

Module: **Overview of Fast Fourier Transform (FFT) Algorithms**

By: Douglas Jones

URL: http://cnx.org/content/m12026/1.3/

Module: **Running FFT**

By: Douglas Jones

URL: http://cnx.org/content/m12029/1.5/

Module: **Goertzel's Algorithm**

By: Douglas Jones

URL: http://cnx.org/content/m12024/1.5/

Module: **Power-of-two FFTs**

By: Douglas Jones

URL: http://cnx.org/content/m12059/1.2/

Module: **Decimation-in-time (DIT) Radix-2 FFT**

By: Douglas Jones

URL: http://cnx.org/content/m12016/1.7/

Module: **Decimation-in-Frequency (DIF) Radix-2 FFT**

By: Douglas Jones

URL: http://cnx.org/content/m12018/1.6/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **Alternate FFT Structures**

By: Douglas Jones

URL: http://cnx.org/content/m12012/1.6/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **Radix-4 FFT Algorithms**

By: Douglas Jones

URL: http://cnx.org/content/m12027/1.4/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **Split-radix FFT Algorithms**

By: Douglas Jones

URL: http://cnx.org/content/m12031/1.5/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **Efficient FFT Algorithm and Programming Tricks**

By: Douglas Jones

URL: http://cnx.org/content/m12021/1.6/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **Fast Convolution**

By: Douglas Jones

URL: http://cnx.org/content/m12022/1.5/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **Chirp-z Transform**

By: Douglas Jones

URL: http://cnx.org/content/m12013/1.4/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **FFTs of prime length and Rader's conversion**

By: Douglas Jones

URL: http://cnx.org/content/m12023/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **Choosing the Best FFT Algorithm**

By: Douglas Jones

URL: http://cnx.org/content/m12060/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **Why Transforms?**

By: Phil Schniter

URL: http://cnx.org/content/m11001/2.1/

Copyright: Phil Schniter

License: http://creativecommons.org/licenses/by/1.0

Module: **Limitations of Fourier Analysis**

By: Phil Schniter

URL: http://cnx.org/content/m11003/2.1/

Module: **Time-Frequency Uncertainty Principle**

By: Phil Schniter

URL: http://cnx.org/content/m10416/2.18/

Module: **Short-time Fourier Transform**

By: Phil Schniter

URL: http://cnx.org/content/m10417/2.14/

Module: **Continuous Wavelet Transform**

By: Phil Schniter

URL: http://cnx.org/content/m10418/2.14/

Module: **Hilbert Space Theory**

By: Phil Schniter

URL: http://cnx.org/content/m11007/2.1/

Module: **Vector Space**

By: Phil Schniter

URL: http://cnx.org/content/m10419/2.13/

Module: **Normed Vector Space**

By: Phil Schniter

URL: http://cnx.org/content/m10428/2.14/

Module: **Inner Product Space**

By: Phil Schniter

URL: http://cnx.org/content/m10430/2.13/

Module: **Hilbert Spaces**

By: Phil Schniter

URL: http://cnx.org/content/m10434/2.11/

Module: **Discrete Wavelet Transform: Main Concepts**

By: Phil Schniter

URL: http://cnx.org/content/m10436/2.12/

Module: **The Haar System as an Example of DWT**

By: Phil Schniter

URL: http://cnx.org/content/m10437/2.10/

Module: **A Hierarchy of Detail in the Haar System**

By: Phil Schniter

URL: http://cnx.org/content/m11012/2.3/

Module: **Haar Approximation at the kth Coarseness Level**

By: Phil Schniter

URL: http://cnx.org/content/m11013/2.2/

Module: **The Scaling Equation**

By: Phil Schniter

URL: http://cnx.org/content/m10476/2.7/

Module: **The Wavelet Scaling Equation**

By: Phil Schniter

URL: http://cnx.org/content/m11014/2.2/

Module: **Conditions on h[n] and g[n]**

By: Phil Schniter

URL: http://cnx.org/content/m11015/2.2/

Module: **Values of g[n] and h[n] for the Haar System**

By: Phil Schniter

URL: http://cnx.org/content/m11016/2.2/

Module: **Wavelets: A Countable Orthonormal Basis for the Space of Square-Integrable Functions**

By: Phil Schniter

URL: http://cnx.org/content/m11017/2.2/

Module: **Filterbanks Interpretation of the Discrete Wavelet Transform**

By: Phil Schniter

URL: http://cnx.org/content/m10474/2.6/

Module: **Initialization of the Wavelet Transform**

By: Phil Schniter

URL: http://cnx.org/content/m11018/2.2/

Module: **Regularity Conditions, Compact Support, and Daubechies' Wavelets**

By: Phil Schniter

URL: http://cnx.org/content/m10495/2.8/

Copyright: Phil Schniter

License: http://creativecommons.org/licenses/by/1.0

Module: **Computing the Scaling Function: The Cascade Algorithm**

By: Phil Schniter

URL: http://cnx.org/content/m10486/2.6/

Copyright: Phil Schniter

License: http://creativecommons.org/licenses/by/1.0

Module: **Finite-Length Sequences and the DWT Matrix**

By: Phil Schniter

URL: http://cnx.org/content/m10459/2.6/

Copyright: Phil Schniter

License: http://creativecommons.org/licenses/by/1.0

Module: **DWT Implementation using FFTs**

By: Phil Schniter

URL: http://cnx.org/content/m10999/2.1/

Copyright: Phil Schniter

License: http://creativecommons.org/licenses/by/1.0

Module: **DWT Applications - Choice of phi(t)**

By: Phil Schniter

URL: http://cnx.org/content/m11004/2.1/

Copyright: Phil Schniter

License: http://creativecommons.org/licenses/by/1.0

Module: **DWT Application - De-noising**

By: Phil Schniter

URL: http://cnx.org/content/m11000/2.1/

Copyright: Phil Schniter

License: http://creativecommons.org/licenses/by/1.0

Module: **Overview of Multirate Signal Processing**

By: Douglas Jones

URL: http://cnx.org/content/m12777/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **Interpolation, Decimation, and Rate Changing by Integer Fractions**

By: Douglas Jones

URL: http://cnx.org/content/m12801/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **Efficient Multirate Filter Structures**

By: Douglas Jones

URL: http://cnx.org/content/m12800/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **Filter Design for Multirate Systems**

By: Douglas Jones

URL: http://cnx.org/content/m12773/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **Multistage Multirate Systems**

By: Douglas Jones

URL: http://cnx.org/content/m12803/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **DFT-Based Filterbanks**

By: Douglas Jones

URL: http://cnx.org/content/m12771/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **Quadrature Mirror Filterbanks (QMF)**

By: Douglas Jones

URL: http://cnx.org/content/m12770/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **M-Channel Filter Banks**

By: Douglas Jones

URL: http://cnx.org/content/m12775/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/2.0/

Module: **Filter Structures**

By: Douglas Jones

URL: http://cnx.org/content/m11917/1.3/

Copyright: Douglas Jones

License: http://creativecommons.org/licenses/by/1.0

Module: **FIR Filter Structures**

By: Douglas Jones

URL: http://cnx.org/content/m11918/1.2/

Module: **IIR Filter Structures**

By: Douglas Jones

URL: http://cnx.org/content/m11919/1.2/

Module: **State-Variable Representation of Discrete-Time Systems**

By: Douglas Jones

URL: http://cnx.org/content/m11920/1.2/

Module: **Fixed-Point Number Representation**

By: Douglas Jones

URL: http://cnx.org/content/m11930/1.2/

Module: **Fixed-Point Quantization**

By: Douglas Jones

URL: http://cnx.org/content/m11921/1.2/

Module: **Finite-Precision Error Analysis**

By: Douglas Jones

URL: http://cnx.org/content/m11922/1.2/

Module: **Input Quantization Noise Analysis**

By: Douglas Jones

URL: http://cnx.org/content/m11923/1.2/

Module: **Quantization Error in FIR Filters**

By: Douglas Jones

URL: http://cnx.org/content/m11924/1.2/

Module: **Data Quantization in IIR Filters**

By: Douglas Jones

URL: http://cnx.org/content/m11925/1.2/

Module: **IIR Coefficient Quantization Analysis**

By: Douglas Jones

URL: http://cnx.org/content/m11926/1.2/

Module: **Limit Cycles**

About Connexions

Since 1999, Connexions has been pioneering a global system where anyone can create course materials and make them fully accessible and easily reusable free of charge. We are a Web-based authoring, teaching and learning environment open to anyone interested in education, including students, teachers, professors and lifelong learners. We connect ideas and facilitate educational communities. Connexions's modular, interactive courses are in use worldwide by universities, community colleges, K-12 schools, distance learners, and lifelong learners. Connexions materials are in many languages, including English, Spanish, Chinese, Japanese, Italian, Vietnamese, French, Portuguese, and Thai.

Document Outline