Signal Computing:

Digital Signals in the Software Domain

Spring 2010

Prof. Michael Stiber
Computing and Software Systems Program
University of Washington, Bothell
18115 Campus Way NE, Box 358534
Bothell, Washington 98011-8246

Dr. Bilin Zhang Stiber
S-Squared Technical Consulting
15220 80th Ave. NE
Kenmore, WA 98028-5605

Contents

List of Figures

List of Tables

List of Algorithms

Preface

One of the fastest growing application areas for computers is the processing of *digital signals*: multimedia (sound, images, and video) and other measurements of the physical world (in medical instruments, for example). Digital signals place great demands on processing power, network bandwidth, storage capacity, I/O speed, and software design. In this book, you will learn how digital signals are captured, represented, processed, communicated, and stored in computers. The specific topics we will cover include: physical properties of physical source information (sound, images), devices for information capture (microphones, cameras), digitization, compression, digital signal representation (JPEG, MPEG), digital signal processing, and network communication. By the end of this book, you should understand the problems and solutions facing multi/hypermedia systems development in the areas of user interfaces, information retrieval, data structures and algorithms, and communications.

Objectives

By the end of this book, you should know:

- What digital signals are like in the "real" world and how their properties affect how we perceive them.
- How these signals are digitized and the tradeoffs among sampling speed, levels of quantization, and file size.
- How to perform simple signal filtering to remove noise, emphasize important features, etc. You should be well-prepared to work with electrical engineers in the design of more advanced signal processing systems.
- How multimedia file sizes can be reduced by compression, and the tradeoffs among compression, processing overhead, and media quality.

Prerequisites

It is expected that you have already taken an algorithms and data structures class, or at least are beyond the point where you need to focus most of your attention on getting code working. While this course will not involve writing large quantities of code, I will assume that adapting to a new programming environment will be no problem for you. For certain assignments, you may be asked to write small programs in your choice of a "lower level" language (such as C, C++, or Java). The actual amount of coding involved will be quite small, however. This course covers much of the mathematical foundations for understanding signals and signal processing; however, it is assumed that you are familiar with topics such as complex numbers, trigonometry, derivatives, vectors, the basic idea of integrals, infinite series, and basic physics (mass, acceleration, force, etc). Figure 0.1 shows these prerequisites in the context of this book's chapters.

Figure 0.1 also outlines how each chapter's concepts are used by subsequent ones and what the major themes are. You'll notice that there's not much in the way of programming indicated. While there *is* an expectation that you understand the basics of computational complexity and can understand, appreciate, and analyze algorithms, this is *not* a programming book. Instead, this is a book that makes concrete many of the previously abstract mathematical concepts that you are familiar with. It shows how these concepts relate to *real* applications that produce tangible changes on digital signals — it connects *mathematics* to *bits*.

About This Book

This book is divided into nine chapters. Chapters include self-test exercises scattered throughout, so I suggest that you go through all the material sequentially (or, at least, do the self-test exercises in each section). Each chapter concludes with written or programming assignments and pointers to additional readings.

If you read the PDF version of this book in Acrobat Reader, you will find that it is extensively hyper-linked. This includes links from exercises to their answers, links to resources on the course web site, and links from the table of contents, list of figures, index, etc. to their respective locations.

Note that this book is still in testing; consider it a late beta version. Please let me know if you find any errors, omissions, or lack of clarity.

Typographical Conventions

There are no real "standards" for much of the notation that is used in digital signal processing; depending on which textbook you read, you will encounter different typographical conventions. In this textbook, I have chosen the following:

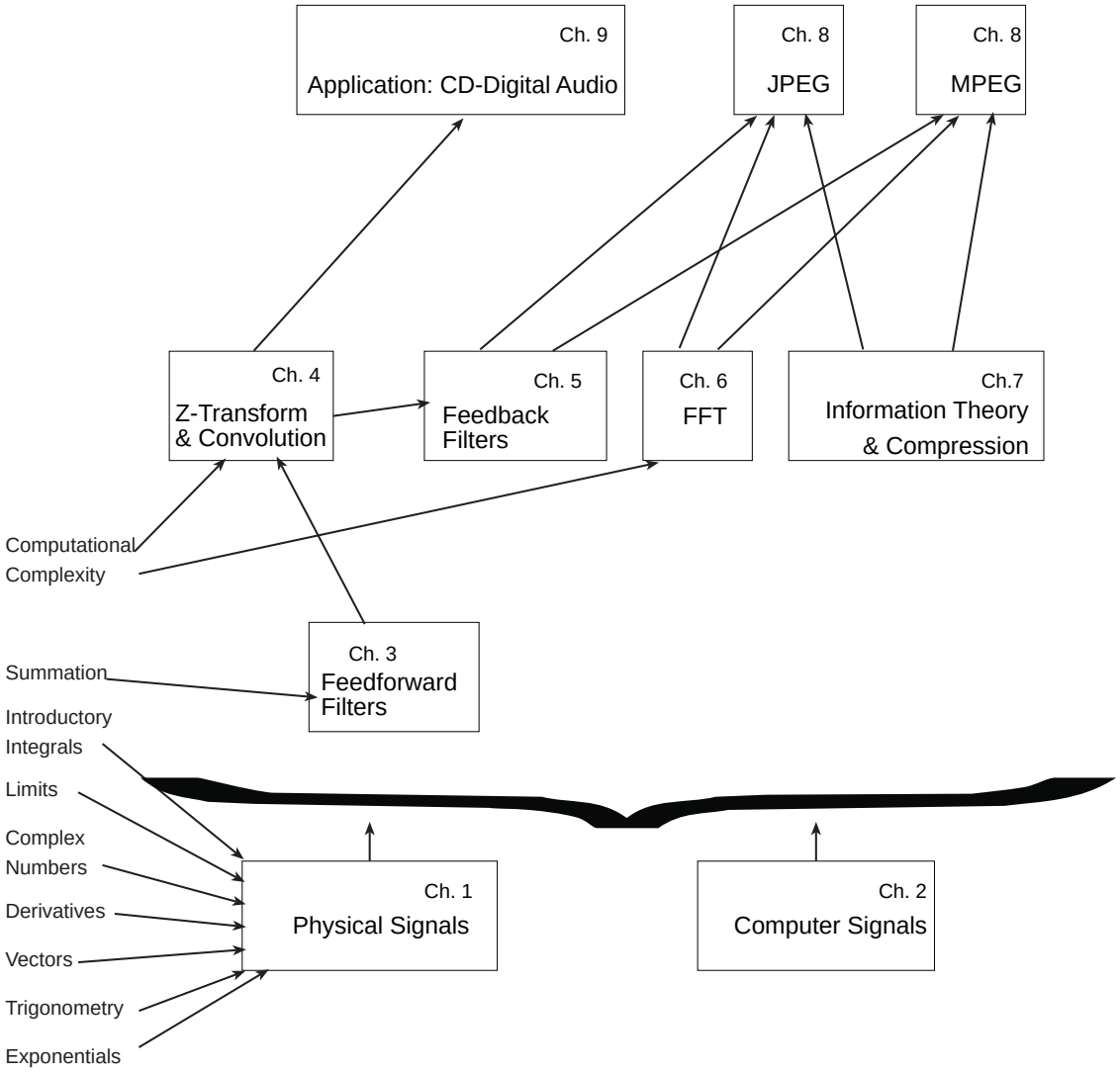


Figure 0.1: Course conceptual road map and background knowledge.

j	$\sqrt{-1}$
t	continuous time; units typically seconds
$x(t)$	a real-valued function of continuous time (physical signal)
f	continuous frequency; units cycles/second or Hertz (Hz)
T	an interval of time; often $T = 1/f$, meaning the period of a signal
f_0	a particular frequency; subscript may vary depending on use
ω	continuous angular frequency; units radians/second
ω_0	a particular angular frequency; subscript may vary depending on use
n	discrete time or sample number; dimensionless, but you can think of the units as being "samples"
$x[n]$	a function of discrete time; may be real-valued (sampled signal) or discrete valued (quantized/digital signal)
$\hat{f}$	discrete frequency; units cycles/sample
$\hat{f}_0$	a particular discrete frequency; subscript may vary depending on use
$\hat{\omega}$	discrete angular frequency; units radians/sample
$\hat{\omega}_0$	a particular discrete angular frequency; subscript may vary depending on use

Further Reading

An excellent introduction to digital signal processing that is targeted at beginning electrical engineering students is , *DSP First: A Multimedia Approach*, by James H McClellan, Ronald W. Schafer, and Mark A. Yoder (Prentice Hall, 1998). It is important to note that that book's target is the design of low-level signal processing components (i.e., filters) and more generally linear, time-invariant systems, rather than the role such components play in overall software systems.

1 Signals in the Physical World

This lesson introduces the fundamental concepts of multimedia from an "outside the computer" perspective. I define the term "multimedia," discuss why it is an important issue, how multimedia signals (sound, images) are generated in the "real world" (by both natural and artificial means) and how we perceive them. I also introduce a set of convenient mathematical tools for describing signals. When you are done with this chapter, I hope you will appreciate the complexity and importance of multimedia communications and will be eager to see how signals can be processed by computer.

1.1 Multimedia and Sensation

> "What can give us surer knowledge than our senses? With what else can we better distinguish the true from the false?" *Lucretius*

Humans, like any other organisms, depend on their senses to provide information critical to their development and survival. Over time, life has evolved sophisticated systems for capturing information about the physical world. The first senses were almost certainly chemical — what we would call taste or smell. These senses have a number of drawbacks: they typically require close physical presence or even direct contact, they have poor temporal resolution (you can tell that someone was cooking onions, but not *when*), and they have poor spatial resolution (from where is that bad smell coming?). Other senses — hearing, seeing, electrical senses that fish have, magnetic senses that birds have — allow us to gather information remotely from the information source.

Given all this, it is perhaps not surprising that people find computer systems difficult to use, uninspiring, or even objects of distaste. Computers by and large produce a very impoverished set of stimuli, composed almost entirely of patterns of light that we must interpret as text. It is only very recently that computers have done much more than this, as processing power and hardware speed has enabled *multi*media capabilities (though these are almost always used to imitate other appliances, like televisions, telephones, photo albums, etc., rather than to produce new things). The goal of making computers more "human friendly," then, depends on our ability to process multimedia information digitally, and to do so in a *useful* fashion. But what is useful? The answer to this question depends on the user, and central to the user is how his or her sensory system works.

1.2 Sensation and Perception

Important Terms:

color constancy our ability to perceive object color independent of light source qualities.

cone a *photoreceptor* that supports color vision.

luminance measure of visible light strength.

photoreceptors sensory cells in the *retina*.

retina Layer of sensory and other neurons at the back of the eye.

rod a *photoreceptor* that supports low-light-level vision.

trichromatic vision construction of color sensation from measurement of three primary colors.

Regardless of the sensory *modality*, all senses have one thing in common: they are composed of cells which are responsive to some property of the physical world and whose responses are *interpreted* by a nervous system to *construct a model* of the world. In other words, the world does not "look" like our perception. Our perception is a *construct*; an interpretation.

For example, consider sight. Our environment is constantly bathed in a sea of radiation, which we can think of as sinusoidal waves of varying wavelength, ranging from very long (radio waves) to very short (X-rays, gamma rays), and amplitude. Some of this radiation is absorbed by objects and some is reflected, depending on the objects' preoperties. The "goal" of our visual system is *not* to provide us with information about reflected energy. Our visual system has evolved in tandem with the rest of our nervous system for the purpose of extracting useful information about objects' properties based on reflected radiation. In this respect, it is much like a remote-sensing satellite and ground data analysis system: we want to know about crop conditions, we gather reflected light data at a number of wavelengths, and then we analyze the data to produce a report on crop conditions.

Likewise, when we "see" a chair, what we perceive is no more a chair than the word "chair" is a chair. It is an *internal representation* of a chair — a useful representation, which can allow us to recognize the object's purpose, location, size, composition, etc., but a representation nonetheless. What we perceive about the world is at least as much a psychological issue as a data acquisition one. For brevity's sake, however, I will confine myself in this section to discuss the basics of "low-level" sensory organs, rather than high-level perceptual issues.

Our eyes are made up of an optical system and a sensory system. The sensory system — the *retina* — is composed of a number of different kind of nerve cells arranged in well-structured layers. I won't go into the details of the eye's structure, but instead concentrate on one class of cells: the *photoreceptors*.

Photoreceptors are specialized cells that contain pigments that absorb light and respond electrically (to make a complex matter simple). Each pigment is responsive to a fairly narrow range of light wavelengths, and in total, we can see a range of *visible light* with wavelengths between roughly 400 nanometers (400×10^{-9} meters) to 700nm — this from an electromagnetic spectrum that covers 1×10^{-6}nm to thousands of kilometers.

Our eyes have photoreceptors specialized for color vision at high (daytime) light levels (called *cones*) and photoreceptors specialized for low (nighttime) light levels (called *rods*). There are three cone types, each with a pigment sensitive to a different range of light wave-

lengths. These are usually termed red, green, and blue, though the correspondences to these colors is not exact. The fact that our visual system combines measurements of three "colors" of light to form our perception of color is known as *trichromatic vision*.

We do *not* experience light as a set of intensity (*luminance*) measurements at a variety of light wavelengths, however. We experience light as having *color*. Color is a complex psychological issue, and it is still not understood exactly how this is achieved. A simple proof-by-contradiction of the idea that the color we see is the "color" of the light hitting our eyes is the observation that we see objects as having the same color under a wide range of lighting conditions (morning sunlight, afternoon sunlight, incandescent lighting). In each of these circumstances, the light reflected from the objects around us is very different because the illuminating light is different. The fact that our perception of color is more characteristic of the objects' surface properties than the illuminating light is called *color constancy*. Again, our perceptual system acts to give us information about *things*, not measurements of incoming data.

Similar observations can be made of our sense of hearing. Sound is made of moving waves of varying air pressure, in a manner analogous to ripples on a pond. These waves are eventually transduced by auditory neurons, each sensitive to a relatively small range of frequencies (low pitched sounds, high pitched sounds — the full range of audible sounds). Yet, we do not perceive a set of signal strengths, just as we don't for light.

Even more extraordinarily, though our sensors for sound and light are distinct and very different, we don't have separate sensations of hearing and vision. Instead, we perceive the world as a unified whole, with sound and appearance being properties of the objects around us. Our brains *fuse* the incoming sensory stimuli and *construct* a *synthetic* experience.

All of the foregoing is of course a great simplification of how these systems actually work. However, these issues have great impact on the design of computer multimedia systems, because the main point of such systems is delivering a particular user experience. In the following sections, I will outline first how computer systems can generate images, then delve deeper into how we can develop a more precise, mathematical view of sound. This mathematical view will be essential to our understanding of computer representation and manipulation of multimedia information.

1.3 Computer Video Displays

The two most prevalent computer displays are *cathode ray tubes* (CRTs) and *liquid crystal displays* (LCDs); we'll just discuss CRTs here. A simplified CRT is sketched in figure 1.1. It is composed of an *electron gun*, which generates *cathode rays* (aka, "electrons"). In this gun, a metal cathode is heated to cause electrons to "boil off;" they are accelerated toward the screen by a positively-charged anode (or possibly by charging the screen itself), which attracts the negatively-charged electrons. In between the anode and the cathode is a negatively-charged control grid. By varying the charge on the control grid, the intensity of the beam can be varied in a manner analogous to the way that a faucet valve controls the flow of water.

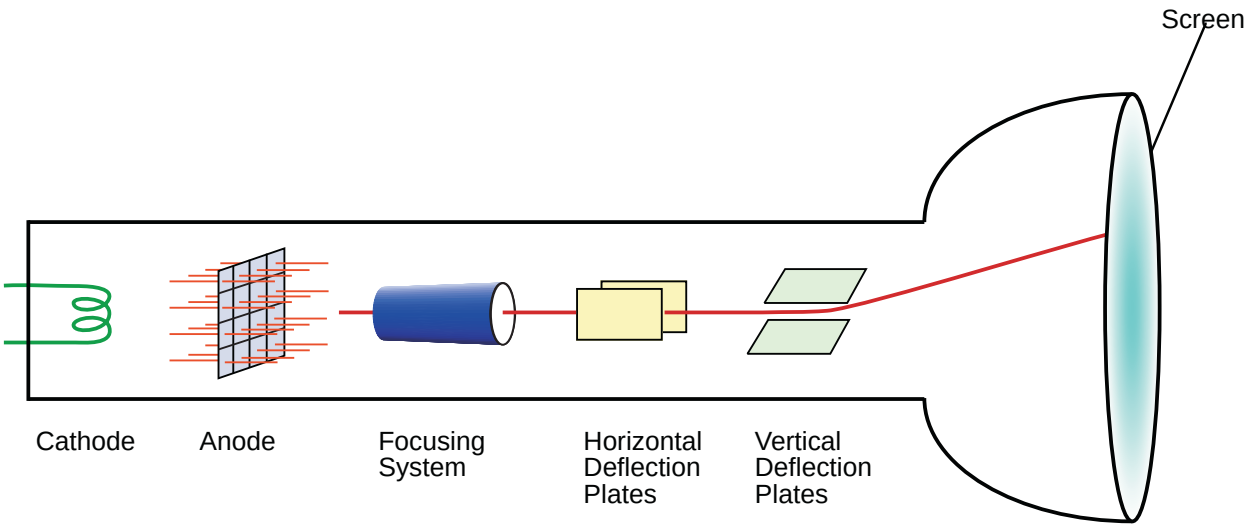


Figure 1.1: Schematic diagram of a CRT.

After leaving the electron gun, the flow of elecrons is focused into a narrow beam by an electrostatic or magnetic focusing system. The focused beam then passes by deflection plates, which can "steer" the beam magnetically to hit any part of the screen.

The screen itself is coated with one (for B&W) or more (for color) *phosphors*: chemical which glow (emit visible light) when struck with electrons. Different phosphors emit different light wavelengths, and so multiple phosphors can be used to produce the perception of color images (typically by combining red, green, and blue phosphors). Another property of phosphors is *persistence*: how long it will glow after an impinging electron beam is removed. Low persistence phosphors will require that the screen image be *refreshed* more often (by repainting the screen with the electron beam); otherwise, the display will flicker. On the other hand, a high persistence phosphor will glow longer, and so will produce blurry moving images.

So, how can we "paint" a two-dimensional image with an electron beam that only creates a single spot on the screen? We do this by *raster scanning* the beam, as shown in figure 1.2. The horizontal and vertical deflection systems work in concert with the control grid to move the beam across the screen while varying its intensity. When the beam moves across the

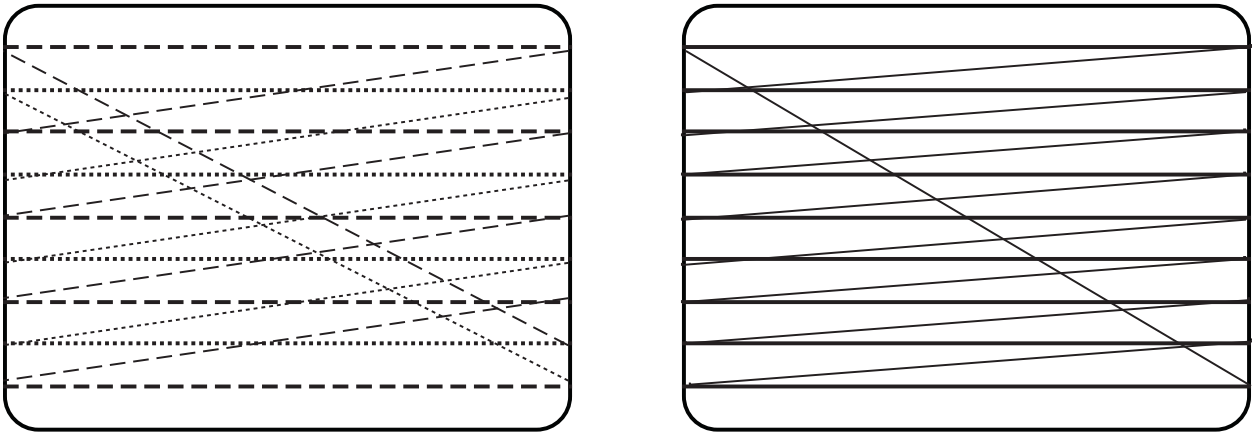


Figure 1.2: Interlaced (left) and non-interlaced (right) raster scanning.

screen horizontally, its intensity is varied in proportion to the desired light intensity. While the deflection system resets to the beginning of the next horizontal line, the beam is turned off. This is called *horizontal retrace*.

Subsequent lines are painted on the display in a similar manner until the bottom of the screen is reached — this is one *frame*. At that point, the beam is turned off and the deflection system resets itself to the beginning of the top screen line; a process called *vertical retrace*.

For computer displays, a frame is usually displayed in between 1/60 and 1/80 of a second (though some displays have higher refresh rates). Typically, such displays are non-interlaced (figure 1.2, right). Analog televisions, on the other hand, have interlaced displays (left). Interlaced raster scanning involves painting all of the odd scan lines in about 1/60 of a second, then a vertical retrace, and a painting of the even lines. This allows the TV 1/30 of a second to display an entire frame and yet still presents a flicker-free image (because every other line is refreshed every 1/60 of a second).

This explains how a greyscale display works. But, what about color? A color display requires three different phosphors, usually considered to be red, green, and blue (RGB). In color monitors, three electron guns are used — one dedicated to each color. The guns are aligned so that their beams strike slightly different spots on the screen. The three phosphors are laid down in a pattern so that each color gun's beam will hit the appropriate phosphor

for each pixel. By varying the intensity of each beam appropriately, control of the relative mix of the three primary colors (and thus the perceived color) and the overall brightness is possible.

1.4 Multimedia System Operation

Multimedia systems need to support four basic functions:

1. They must support appropriate user *perception* of the media — provide output media and represent (code) media in such a way that the output seems natural.

2. They must be able to *retain* media — store it and provide efficient access (perhaps in a database).

3. They should allow users to *reason* about media information. If the goal is greater than a digital television or radio, then users must be able to manipulate the media.

4. They must be capable of *transmitting* media between machines. This can have significant impact on networking, resource allocation, and synchronization of activity across geographically distributed computer systems.

All of these place a high demand on computer power. For example, let's assume that we encode audio information as 8 bits per sample at 8000 samples per second (this would be considered very low quality). How many bits per second is that (answer in A.1 #1)? Perhaps that doesn't sound too bad, so let's consider video. Let's say we encode each frame in a video stream as 1000x1000 pixels, 24 bits/pixel, and 30 frames/second. How many bits per second is that (answer in A.1 #2)? Even if we were to achieve an extremely high compression ration of 1/500, this would still mean more than 1 million bits/second.

On the one hand, we can attack this problem through faster hardware. But you should be well aware by this time that the difference between a good algorithm and a bad one can be the difference between an application which places very little load on a modest CPU and one which takes all the power of a supercomputer. That is what we will be focusing on here: developing the mathematical underpinning for developing efficient multimedia signal processing algorithms.

1.5 Vibrations and Sound

For much of this book, we will focus on sound when talking about signals. The reason we so this is that sound is simpler than video. Sound is a one-dimensional function of time (amplitude verus time), while video is at least a three-dimensional function of time (intensity at each x and y pixel location as a function of time for greyscale; R, G, B values at each x and y pixel location as a function of time for color).

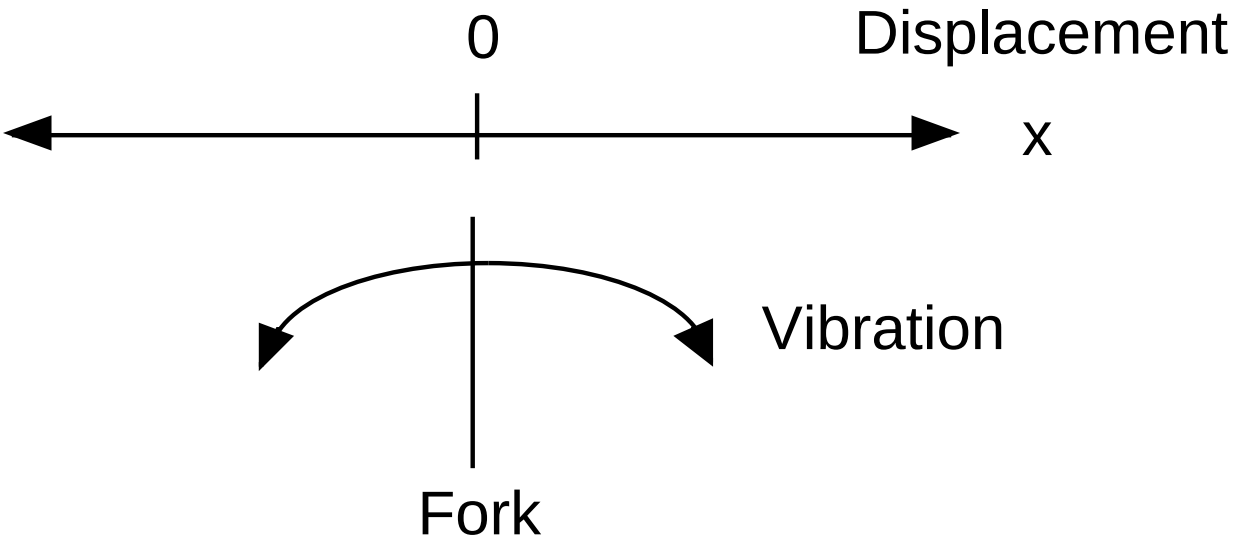


Figure 1.3: Vibration of an idealized tuning fork.

As I said earlier, sound is carried by pressure waves in the air. such pressure waves can be generated by vibrating a physical object. For example, a speaker generates sound by vibrating a flexible cone. Let's start with a simple vibrating system: a tuning fork producing a pure tone. And, to make things even simpler, we'll consider only a single-tine fork (a tuning chopstick?).

There are three concepts central to our understanding of how the idealized tuning fork in figure 1.3 vibrates:

deformation The fork is assumed to be fixed at one end and free at the other, which swings back and forth as the fork vibrates. So, the material the fork is made of is *bent* as the end swings.

inertia An object (such as the end of the vibrating fork) exerts a force proportional to its mass when one tries to accelerate or decelerate it. We can write the equation for this force as $F_i = ma$, where F_i is the inertial force, m is the object's mass, and a is the acceleration.

elastic forces Deforming the fork requires application of force. In the case of our tuning fork, the force results from its inertia. Let's make the simplifying assumption that the fork acts like an ideal spring. That means that as the fork is bent from the vertical ($x = 0$ in the figure), the elastic force increases proportionately to x. So, $F_e = -kx$, where F_e is the elastic force, k is the spring constant (how stiff the fork is), and the minus sign indicates that the direction of the force is opposite the directio of the bend (always towards $x = 0$).

So, we pull the fork and release it. The elastic force results in an acceleration towards $x = 0$ and the tip speed builds up. When the tip passes $x = 0$, the elastic force decelerates the

tip, it stops, and the whole thing repeats. If we ignore troublesome issues like air resistance that would slow down the vibration, then $F_i = F_e$. Solving for a we obtain:

$$F_i = ma = -kx = F_e \tag{1-1}$$

$$a = -\frac{k}{m}x \tag{1-2}$$

Acceleration is the change in velocity over time, or its derivative, $a = \mathrm{d}v/\mathrm{d}t$; velocity in turn is the derivative of distance, $v = \mathrm{d}x/\mathrm{d}t$. So, acceleration is the second derivative of x:

$$a = \frac{\mathrm{d}v}{\mathrm{d}t} = \frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -\frac{k}{m}x \tag{1-3}$$

As an aside, this kind of equation — which relates a derivative of something to itself — is called a *differential equation*. I'll bet you didn't know they were that simple! To *solve* a differential equation, we look for a function of x that, when plugged in for x, will satisfy the equation. This is analogous to solving an algebraic equation, where we look for a number that satisfies. The only difference is that numbers fall along a one-dimensional number line (unless we're talking about complex numbers), while there is no such neat ordering of functions. So, we need to find a function of x whose second derivative is proportional to itself. Can you think of one or two (answer in A.1 #3)?

Luckily, people have been working with functions and differential equations for a while, and so finding such functions is not difficult. For example, consider sinusoids. The first and second derivatives of $\sin \omega t$ are:

$$\frac{\mathrm{d}}{\mathrm{d}t} \sin \omega t = \omega \cos \omega t \tag{1-4}$$

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2} \sin \omega t = \frac{\mathrm{d}}{\mathrm{d}t} \omega \cos \omega t \ = -\omega^2 \sin \omega t \tag{1-5}$$

Therefore, differential equation (1-3) describes *simple harmonic motion*: sinusoidal vibration of the tuning fork's tip.

Self-Test Exercises

See A.1 #4–5 for answers.

1. Solve for ω in terms of k and m.

2. Fork stiffness (k) clearly affects vibration frequency. As the fork get stiffer (k increases), does the vibration frequency go up or down?

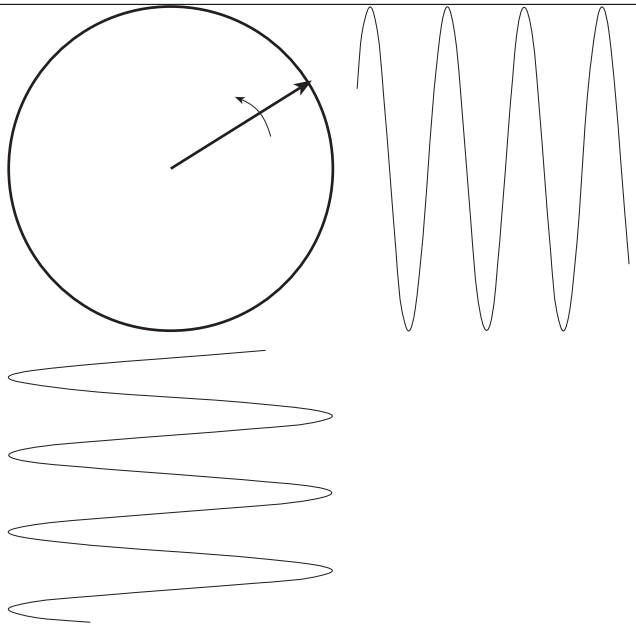


Figure 1.4: Components of a phasor are sinusoids.

Sinusoid Phase

The same argument can be used for $\cos \omega t$. In fact, cosine is just a delayed version of sine, as $\cos \omega t = \sin(\omega t + \pi/2)$. So, a more general solution to (1-3) would be:

$$x(t) = \sin(\omega t + \phi) \tag{1-6}$$

In (1-6), ϕ is called the sinusoid's *phase*: the time closest to $t = 0$ when $x(t) = 0$ (for the case of a sine; it would be when $x(t) = 1$ for a cosine). Depending on *when* we hit the tuning fork (and our time reference; the time that we call $t = 0$), we will get sinusoids of differing phase.

What would happen if we hit two identical tuning forks but at different times? They would vibrate at the same frequency, but would have differing phases and amplitudes (loudnesses). If the two forks were close together, we would heard the sum of their tones. What would that sum be (answer in A.1 #6)? Simplifying this sum would be involved; we would have to remember all sorts of trigonometric identities and then crank out the algebra. Luckily, we can come up with another representation of a sinusoid which simplifies this problem considerably. In fact, much of the new math we will cover in this course is primarily focused on making the paper-and-pencil work easier.

1.6 Phasors

Instead of thinking of a sinusoid as a function which oscillates up and down, let's think of it as something that goes round and round: a rotating vector of length a (where a is the

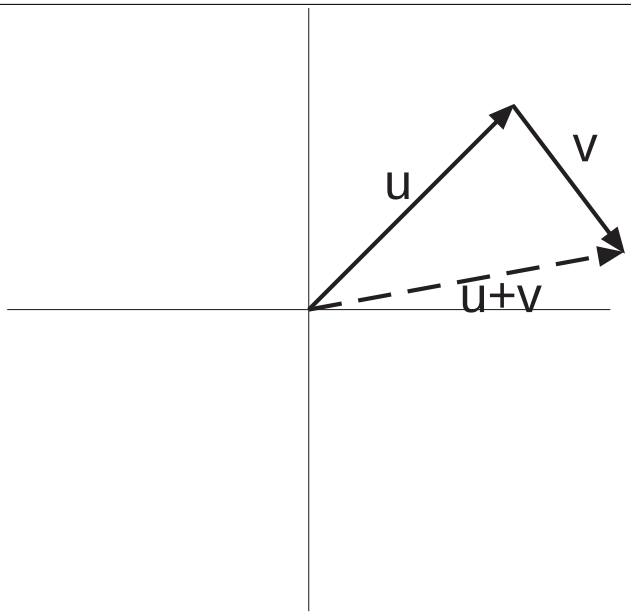


Figure 1.5: Graphical summation of vectors.

sinusoid's amplitude). At any point in time, the vector makes an angle $\theta(t)$ with the positive x axis. The phase $\phi = \theta(0)$ is the angle at $t = 0$. As figure 1.4 shows, the sine and cosine functions are the y and x *components* of this vector.

If we have two vectors $\mathbf{u}$ and $\mathbf{v}$, we can add them by adding their components. This is done graphically in figure 1.5 by placing the "tail" of one vector on the "head" of the other, and the sum $\mathbf{u} + \mathbf{v}$ is the vector from the origin to the "head" of the second vector. Of course, that's just the sum at one point in time. What does $\mathbf{u} + \mathbf{v}$ look like if $\mathbf{u}$ and $\mathbf{v}$ are rotating (answer in A.1 #7)?

Complex Exponential Representation

While we used the x and y axes above as frames of reference (for considering components of the vectors), we didn't actually define what the plane was. Let's imagine that our rotating vectors are in the *complex* plane, where x is the real axis and y is the imaginary axis. At any point in time, the location of a vector is therefore a complex number:

$$\mathbf{v} = x + jy \tag{1-7}$$

I use j here for $\sqrt{-1}$, rather than i as you've probably seen before, to adhere to the convention used by electrical engineers (who use i for current). This will allow us to reduce sinusoidal operations to algebra with complex numbers.

Note that this is the *rectangular* representation of complex numbers; we can also represent them in polar form (magnitude and angle), written as $R\angle\theta$.

Review Exercises: Complex Numbers

1. What is the sum of the two complex numbers $x + jy$ and $v + jw$ (answer in A.1 #8)?
2. What is the product of the two complex numbers $x + jy$ and $v + jw$ (answer in A.1 #9)?
3. Convert the complex number $z = x + jy$ to polar form, $R\angle\theta$ (answer in A.1 #10).
4. Multiply the two polar-form complex numbers $R_1\angle\theta_1$ and $R_2\angle\theta_2$ (answer in A.1 #11).

So, we have an easier-to-manipulate representation of a sinusoid *at one point in time*. But, we want to represent a rotating vector. Let's talk about a "standard" sinusoid — one with unit amplitude — and let's call this $E(\theta)$. We know from our first discussion of the vector representation of sinusoids that sine and cosine are the y and x axis projections of the vector, so let's rewrite our rectangular form as:

$$E(\theta) = \cos\theta + j\sin\theta \tag{1-8}$$

Can we rewrite the polar form in a similar way? We need a function of θ that satisfies the same requirements as the rectangular form of equation (1-8). The derivative of (1-8) is:

$$\frac{\mathrm{d}}{\mathrm{d}\theta}E(\theta) = -\sin\theta + j\cos\theta \tag{1-9}$$

We note that (1-9) is also just $jE(\theta)$, which is the original function times a constant. What function satisfies the condition that its derivative is proportional to itself? This should bring back fond memories of our tuning fork:

$$E(\theta) = e^{j\theta} \tag{1-10}$$

$$\frac{\mathrm{d}}{\mathrm{d}\theta}E(\theta) = je^{j\theta} \tag{1-11}$$

This is the *complex sinusoid* representation. It has the property that $e^{j\theta} = \cos\theta + j\sin\theta$; also called *Euler's formula* ("Euler" is pronounced "oiler"). It will make our work much simpler. For example, *raising the complex sinusoid to a power is equivalent to rotating the vector*:

$$E(\theta)^k = \left(e^{j\theta}\right)^k = e^{j\theta}e^{j\theta}\cdots e^{j\theta} = e^{jk\theta} \tag{1-12}$$

As you can see in (1-12), raising $E(\theta)$ to the k^{th} power is equivalent to multiplying the angle θ by k in the complex sinusoid.

The complex sinusoid in (1-11) is one with unit amplitude. The representation for a general vector $R\angle\theta$ must therefore be $Re^{j\theta}$. So, we see that the since and cosine representation is the rectangular form, and the complex exponential is the polar form.

Self-Test Exercises

1. Multiply the two complex sinusoids z_1 and z_2 (answer in A.1 #12).

2. The complex conjugate is indicated by z^*. If $z = x + jy$, $z^* = x - jy$. What is the complex conjugate of the complex sinusoid, $z = Re^{j\theta}$ (answer in A.1 #13)?

3. Answer the following for $z = x + jy$ (answers in A.1 #14):

   (a) What is $z + z^*$?

   (b) What is $z - z^*$?

   (c) What is zz^*?

Going back to (1-12), we can now write an expression for a complex sinusoid as a function of time (a rotating vector). The angle of the sinusoid is a function of time, $\theta(t) = \omega t$, where ω is its frequency (just as for our original sinusoids):

$$e^{j\theta(t)} = e^{j\omega t} \tag{1-13}$$

This rotating complex sinusoid is called a *phasor* (with apologies to Captain Kirk).

Tuning a Guitar

When a person tunes a guitar, they usually tune just the lowest string to a reference (a pitch pipe, for example). Then, they finger that string to produce the same note that the next higher string should. If the second string does produce the same tone, they can hear that. If the second string is out of tune, it will produce a slightly different tone, and they can hear *beating*. Beating is what results when two sinusoids at different frequencies are added.

Previously, we added two sinusoids at the same frequency, and saw that the result is another sinusoid at that same frequency. That is the case when the second string is in tune. Let's now add two complex sinusoids with slightly different frequencies, ω and $\omega + \delta$, where the difference between the two frequencies is $\delta \ll \omega$:

$$\begin{aligned} a_1 e^{j\omega t} + a_2 e^{j(\omega+\delta)t} &= a_1 e^{j\omega t} + a_2 e^{j\omega t} e^{j\delta t} \\ &= e^{j\omega t}(a_1 + a_2 e^{j\delta t}) \end{aligned} \tag{1-14}$$

This has the form of a complex sinusoid of frequency ω and amplitude that is a function of time: $a_1 + a_2 e^{j\delta t}$. Figure 1.6 presents the graphical version of this, which looks like a top view of the amusement park "octopus" ride. There are times when the $e^{j\delta t}$ adds a factor of up to a_2 to the first sinusoid and times when it subtracts a factor of up to a_2. The result is the beating waveform seen in figure 1.7 (in this case, the result of adding 10Hz and a 12Hz sinusoids, which can be seen to produce a 2Hz "beat frequency").

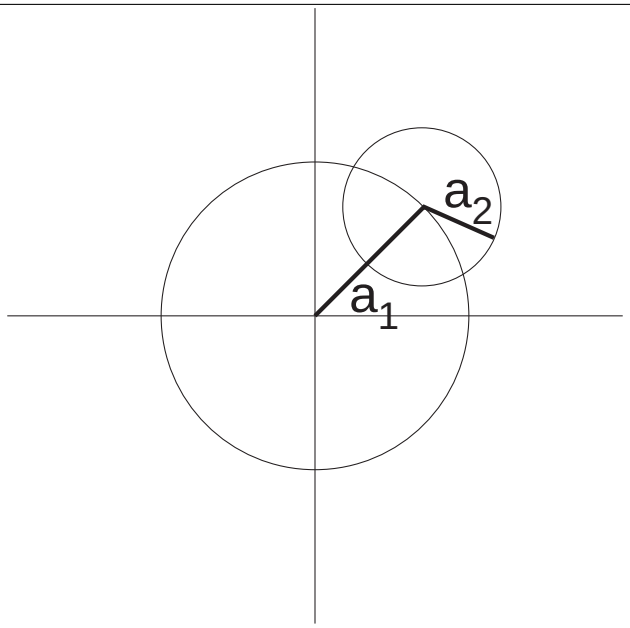


Figure 1.6: Phasor representation of beating.

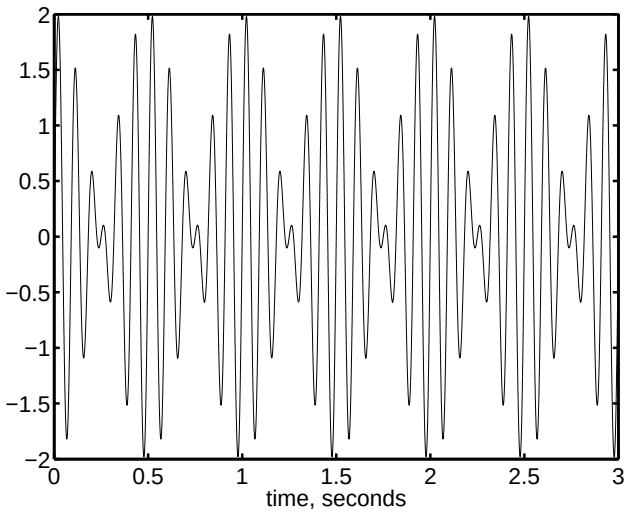


Figure 1.7: Amplitude vs. time representation of beating.

MATLAB and Sound Files

- A MATLAB .m file for demonstrating beating is available at http://courses.washington.edu/css457/ebook/beating.m.
- Audio files to demonstrate beating: a 1000Hz sine wave at http://courses.washington.edu/css457/ebook/1000Hz.au, a 1005Hz sine wave at http://courses.washington.edu/css457/ebook/1000Hz.au, and their sum, at http://courses.

washington.edu/css457/ebook/beating.au.

Frequency and Period

Until now, we've been using ω as the symbol for the speed of rotation of a phasor. If t is in seconds (which we will assume is always the case), then the phasor rotates ω times in one second, or once in $1/\omega$ seconds. Remembering that our exponential representation corresponds $\cos \omega t + j \sin \omega t$, we see that each of these rotations corresponds to 2π radians in the oscillation of a sinusoid, and so its units must be *radians/second*. We will call this *angular frequency*, as it is the rate of change of the phasor's angle per second. However, when we speak of periodic physical signals, it is more convenient to speak of the number of times that the signal repeats per second: *cycles/second* or *Hertz*. We will use f to refer to frequencies expressed as Hertz. Since one cycle is 2π radians, we can convert between these two units as $\omega = 2\pi f$. In this book, we will use whichever units are most convenient for each topic; you'll need to note which is in use.

If ω and f are the number of radians or cycles of a sinusoid per second, then their reciprocals are the number of seconds per radian or cycle of rotation. Seconds per radian is only infrequently used, so we only define a symbol for seconds per cycle, $T = 1/f = 2\pi/\omega$. This is the *period* of a signal in seconds.

1.7 Spectra

What if our two sinusoids are not of slightly different frequencies, but, rather, one is a *harmonic* (integer multiple) of the other? Let's call the first sinusoid's frequency ω_0 and the second's $\omega_1 = m\omega_0$, $m = 2, 3, \ldots$. Their sum is $a_0 e^{j\omega_0 t} + a_1 e^{jm\omega_0 t}$, and there's nothing we can do to simplify this equation. This "failure to simplify" is not a result of our lack of mathematical sophistication — there truly is no simpler representation. Another way of saying this is that sinusoids at different frequencies are independent or *orthogonal* to each other. In general, if we add a large number of harmonics together, their sum is:

$$\sum_{k=0}^{N} a_k e^{jk\omega_0 t} \tag{1-15}$$

You'll note that the $k = 0$ term of this summation has the value a_0 for all time; this is often called the *DC component* of the signal. All other components have frequencies that are integer multiples (harmonics) of ω_0, the *fundamental frequency*. Note that a sinusoid with frequency $k\omega_0$ has a period of $2\pi/(k\omega_0) = T_0/k$ — it repeats k times in T_0 seconds. This means it also repeats every T_0 seconds, and since each term in the summation repeats every T_0 seconds, the whole summation repeats that often, and therefore the period of the signal represented by the summation is T_0. I won't prove it here, but such a summation can in fact be used to represent *any* periodic function of time. This *Fourier series* is one of the most important and fundamental principles of signal processing.

You should be familiar already with the term "spectrum" as used to describe the continuous band of colors that white light (sunlight) can be broken into — a rainbow. From physics, you know that each color corresponds to a specific frequency of the visible spectrum. Hence, the decomposition of white light into colors is actually a form of frequency analysis. *Frequency analysis* of a signal involves the resolution of the signal into its frequency (sinusoidal) components; this is also called *spectral analysis*. The multimedia signal waveforms we are concerned with here are basically functions of time.

At this point, you should recognize that the Fourier series represents the spectrum of a periodic signal. It (and other tools) decomposes signals into spectra defined in terms of sinusoidal (or complex exponential) components. With such a decomposition, a signal is said to be represented in the *frequency domain*; otherwise, usually it is in the *time domain*. For periodic signals, such a decomposition is the Fourier Series. For infinite signals (with finite energy), the decomposition is called a *Fourier Transform*. For finite, discrete signals, the decomposition is the *Discrete Fourier Transform*. Recombining the sinusoidal components to reconstruct the original signal is basically a Fourier synthesis problem or *inverse Fourier analysis*.

1.7.1 Interlude: Vectors

Here is a list of vector operations, most of which you should already know. Let $\mathbf{v}$ and $\mathbf{w}$ be arbitrary vectors and $\vec{\mathbf{x}}, \vec{\mathbf{y}}$ and $\vec{\mathbf{z}}$ be an orthogonal *basis* (the component vectors that define the coordinate system, which for 3D Cartesian coordinates would be unit vectors parallel to the X, Y, and Z axes). We can then define the following:

1. Projection of a vector onto the basis vectors (you may know these as the *components* of the vector)

$$v_x = \langle \mathbf{v}, \vec{\mathbf{x}} \rangle$$
$$v_y = \langle \mathbf{v}, \vec{\mathbf{y}} \rangle \tag{1-16}$$
$$v_z = \langle \mathbf{v}, \vec{\mathbf{z}} \rangle$$
$$\tag{1-17}$$

   and

$$w_x = \langle \mathbf{w}, \vec{\mathbf{x}} \rangle$$
$$w_y = \langle \mathbf{w}, \vec{\mathbf{y}} \rangle \tag{1-18}$$
$$w_z = \langle \mathbf{w}, \vec{\mathbf{z}} \rangle$$
$$\tag{1-19}$$

2. Expressing a vector using the basis (i.e., as the sum of its components)

$$\mathbf{v} = v_x\vec{\mathbf{x}} + v_y\vec{\mathbf{y}} + v_z\vec{\mathbf{z}} \tag{1-20}$$
$$\mathbf{w} = w_x\vec{\mathbf{x}} + w_y\vec{\mathbf{y}} + w_z\vec{\mathbf{z}} \tag{1-21}$$

3. Inner product of two vectors

$$\langle \mathbf{v}, \mathbf{w} \rangle = v_x w_x + v_y w_y + v_z w_z \tag{1-22}$$
$$\langle \mathbf{v}, \mathbf{v} \rangle = v_x^2 + v_y^2 + v_z^2 \tag{1-23}$$

You may be familiar with the inner product of a vector with itself as being its length squared, $|\mathbf{v}|^2$. When the vector is complex

$$\langle \mathbf{v}, \mathbf{w}^* \rangle = v_x w_x^* + v_y w_y^* + v_z w_z^* \tag{1-24}$$
$$\langle \mathbf{v}, \mathbf{v}^* \rangle = |v_x|^2 + |v_x|^2 + |v_x|^2 \tag{1-25}$$

where * denotes complex conjugate.

We can also consider arbitrary functions to be more generalized forms of vectors. If $f(t)$ and $g(t)$ are periodic functions of a continuous time variable t with period T,

$$\langle f, g \rangle = \frac{1}{T} \int_0^T f(t) g^*(t) dt \tag{1-26}$$

which is normalized to one by T.

4. Vector sum

$$\mathbf{v} + \mathbf{w} = (v_x + w_x)\vec{\mathbf{x}} + (v_y + w_y)\vec{\mathbf{y}} + (v_z + w_z)\vec{\mathbf{z}} \tag{1-27}$$

5. Distributive and commutative properties of inner products

$$\langle \mathbf{v} + \mathbf{u}, \mathbf{w} \rangle = \langle \mathbf{v}, \mathbf{w} \rangle + \langle \mathbf{u}, \mathbf{w} \rangle \tag{1-28}$$
$$\langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{v}, \mathbf{u} \rangle \tag{1-29}$$

6. $\mathbf{v}$ is *orthogonal* to $\mathbf{w}$

$$\langle \mathbf{v}, \mathbf{w} \rangle = 0 \tag{1-30}$$

7. Unit vector

$$\langle \mathbf{v}, \mathbf{v} \rangle = v_x^2 + v_y^2 + v_z^2 = 1 \tag{1-31}$$

1.7.2 Derivation of the Fourier Series

Here I present frequency analysis tools for continuous time periodic signals. When I say that a signal is periodic, I mean

$$f(t) = f(t + T) \tag{1-32}$$

where t is a continuous time variable and T is the signal's period.

The basic mathematical representation of periodic signals is the Fourier Series, which as we have seen is a weighted sum of *harmonically related* sinusoids (sinusoids whose frequencies

are all multiples of a single, *fundamental* one) or complex exponentials. This was first developed in the nineteenth century by the French mathematician Jean Baptiste Joseph Fourier to describe heat conduction. Now it has extensive applications in a variety of problems encompassing many different fields.

The *Fourier Series* description of a function $f(t)$ is defined as

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{jk\omega_0 t} \tag{1-33}$$

We may think of the exponential signals or phasors

$$e^{jk\omega_0 t}, \quad k = \ldots, -2, -1, 0, 1, 2, \ldots \tag{1-34}$$

as a set of basis vectors, where $\omega_0 = 2\pi/T$, the repetition rate in radians/second. By our earlier definition, ω_0 is the *fundmental frequency*. In a Fourier series, the periodic function $f(t)$ is expressed in terms of the basis and c_k is the projection of $f(t)$ onto the basis $e^{jk\omega_0 t}$, just like projection v_x of a vector $\mathbf{v}$ to $\vec{\mathbf{x}}$. From equation 1-26, the inner product of $f(t)$ and $e^{jk\omega_0 t}$ is

$$c_k = \langle f(t), e^{jk\omega_0 t} \rangle = \frac{1}{T} \int_0^T f(t) e^{-jk\omega_0 t} dt. \tag{1-35}$$

where the *Fourier coefficients* c_k are called the *frequency content* or *spectrum* of $f(t)$, $k = 0, \pm 1, \pm 2, \ldots, \pm\infty$. This spectrum is just like the familiar shorthand of writing a vector as a column or row of numbers, with the understanding that each is the coefficient of one of the basis vectors (i.e., when we write $\mathbf{v} = [123]$, what we actually mean is $\mathbf{v} = 1\vec{\mathbf{x}} + 2\vec{\mathbf{y}} + 3\vec{\mathbf{z}}$). This is the signal's representation in the *frequency domain*, where frequency is the coordinate (in other words, the signal expressed as a function of frequency). Similar, $f(t)$ is the signal's *time domain* representation, where time is the coordinate. Notice that the Fourier series transforms a finite (periodic), continuous signal in the time domain into an infinite, discrete spectrum in the frequency domain.

Alternative Derivation of the Fourier Series [Optional]

Another way to derive the formula for the c_k is as follows. We first multiply both sides of (1-33) by the phasor $e^{-jl\omega_0 t}$ ($l = \ldots, -2, -1, 0, 1, 2, \ldots$) and then integrate both sides of the resulting equation over a single period [0, T],

$$\begin{aligned} \frac{1}{T} \int_0^T f(t) e^{-jl\omega_0 t} dt &= \frac{1}{T} \int_0^T \sum_{k=-\infty}^{\infty} c_k e^{jk\omega_0 t} e^{-jl\omega_0 t} dt \\ &= \sum_{k=-\infty}^{\infty} c_k \frac{1}{T} \int_0^T e^{j(k-l)\omega_0 t} dt \end{aligned} \tag{1-36}$$

Note that

$$\frac{1}{T}\int_0^T e^{j(k-l)\omega_0 t}dt = \frac{1}{T}\left.\frac{e^{j(k-l)\omega_0 t}}{j(k-l)\omega_0}\right|_0^T$$
$$= \frac{e^{j(k-l)\omega_0 T}-1}{j(k-l)\omega_0 T} \tag{1-37}$$

Equation (1-37) has two different solutions, depending on whether $k = l$ or not. Let's make the substitution $u = k-l$. So, when $k = l$, $u = 0$, and the numerator and denominator of (1-37) are both zero. We therefore use L'Hôpital's rule to find the limit:

$$\lim_{u\to 0}\frac{e^{ju\omega_0 T}-1}{ju\omega_0 T} = \left.\frac{\frac{\mathrm{d}}{\mathrm{d}u}\left(e^{ju\omega_0 T}-1\right)}{\frac{\mathrm{d}}{\mathrm{d}u}ju\omega_0 T}\right|_{u=0}$$
$$= \left.\frac{j\omega_0 T e^{ju\omega_0 T}}{j\omega_0 T}\right|_{u=0}$$
$$= \frac{j\omega_0 T}{j\omega_0 T}$$
$$= 1 \tag{1-38}$$

For the case of $k \neq l$, let's rewrite the numerator of (1-37) as $\cos(k-l)\omega_0 T + j\sin(k-l)\omega_0 T - 1$. Remember that $\omega_0 = 2\pi/T$, and so $\omega_0 T = 2\pi$. Since both k and l are integers and $k \neq l$, their difference is an integer; let's call that integer m. The numerator then becomes $\cos 2\pi m + j\sin 2\pi m - 1$. We know that $\cos 2\pi m = 1$ and $\sin 2\pi m = 0$, and so the numerator is zero. On the other hand, the denominator of (1-37) is $j2\pi m \neq 0$, and so (1-37) is zero.

In summary,

$$\frac{1}{T}\int_0^T e^{j(k-l)\omega_0 t}dt = \begin{cases} 1 & k = l \\ 0 & k \neq l \end{cases} \tag{1-39}$$

From the review at the beginning of this chapter (more specifically, the definition of the inner product of two periodic function in (1-26)), this yields the interesting revelation that complex sinusoids at different frequencies are orthogonal (and so, by extension, can serve as an orthogonal basis).

Using (1-39), equation (1-36) becomes

$$\frac{1}{T}\int_0^T f(t)e^{-jl\omega_0 t}dt = c_l, \quad l = \ldots, -2, -1, 0, 1, 2, \ldots \tag{1-40}$$

We got the same result as (1-35).

Real-Valued Signals

In general, the Fourier coefficients c_k are complex valued. If the periodic signal $f(t)$ is real valued (the kind of signals we're concerned with here), the c_k satisfy the condition,

$$c_k = c_{-k}^* \tag{1-41}$$

Consequently,

$$|c_k|^2 = |c_{-k}^*|^2 \tag{1-42}$$

Since c_k is the spectrum of $f(t)$, $|c_k|^2$ is the *power spectrum*. Equation (1-42) tell us that the power spectrum is a symmetric (or "even") function of frequency. Also when $f(t)$ is real, (1-33) becomes

$$\begin{aligned}
f(t) &= \sum_{k=-\infty}^{\infty} c_k e^{jk\omega_0 t} \\
&= c_0 + \sum_{k=1}^{\infty} c_k e^{jk\omega_0 t} + \sum_{k=-1}^{-\infty} c_k e^{jk\omega_0 t} && \text{(split sum into parts)} \\
&= c_0 + \sum_{k=1}^{\infty} c_k e^{jk\omega_0 t} + \sum_{k=1}^{\infty} c_{-k} e^{-jk\omega_0 t} && \text{(distribute -1 into sum)} \\
&= c_0 + \sum_{k=1}^{\infty} [c_k e^{jk\omega_0 t} + c_{-k} e^{-jk\omega_0 t}] && \text{(combine common sum)} \\
&= c_0 + \sum_{k=1}^{\infty} [c_k e^{jk\omega_0 t} + (c_{-k}^* e^{jk\omega_0 t})^*] && \text{(double conjugate)} \\
&= c_0 + \sum_{k=1}^{\infty} [c_k e^{jk\omega_0 t} + (c_k e^{jk\omega_0 t})^*] && \text{(real signal: } c_k = c_{-k}^*\text{)} \\
&= c_0 + 2\sum_{k=1}^{\infty} \mathrm{Re}[c_k e^{jk\omega_0 t}] && (a + a^* = 2\,\mathrm{Re}(a))
\end{aligned} \tag{1-43}$$

With your help in the next self-test exercise, we get

$$f(t) = c_0 + 2\sum_{k=1}^{\infty} \mathrm{Re}(c_k)\cos k\omega_0 t - 2\sum_{k=1}^{\infty} \mathrm{Im}(c_k)\sin k\omega_0 t \tag{1-44}$$

where

$$\mathrm{Re}(c_k) = \frac{1}{T}\int_0^T f(t)\cos(k\omega_0 t)dt \tag{1-45}$$

$$\mathrm{Im}(c_k) = -\frac{1}{T}\int_0^T f(t)\sin(k\omega_0 t)dt \tag{1-46}$$

from (1-35) and Euler's formula.

Examples of periodic signals that are frequently encountered in practice are square waves, rectangular waves (both being used as timing signals in electronics), triangle waves, and of course sinusoids and complex exponentials. In the next example, we will use the Fourier series to analyze some of these.

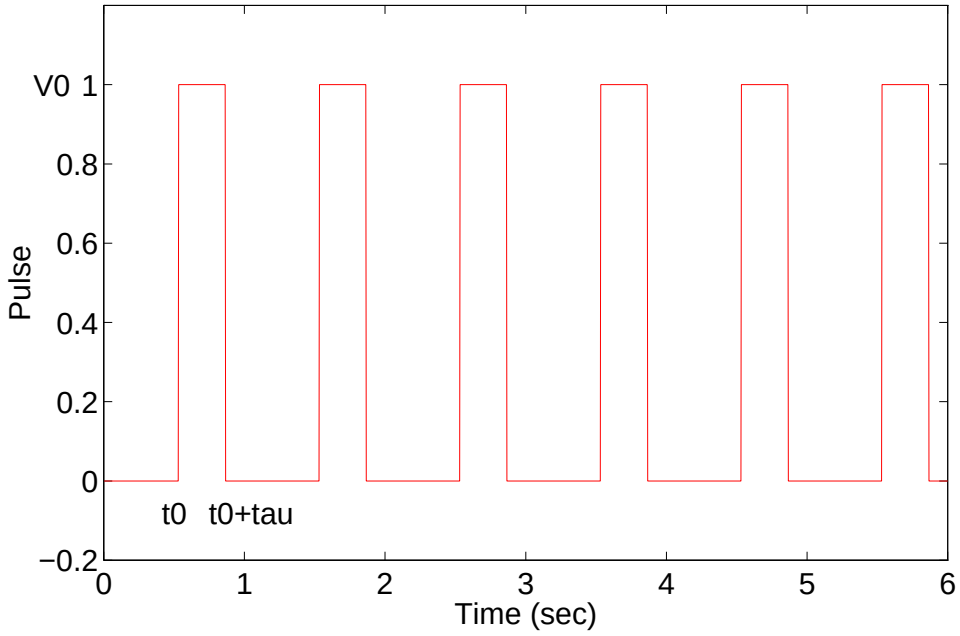


Figure 1.8: A train of periodic rectangular pulses with period $T = 1$, pulse width $\tau = 0.3333$, and initial pulse start at $t_0 = 0.5$.

Self-Test Exercises

See A.1 #15–16 for answers.

1. Prove the relationship in (1-41).

2. From equation (1-43), derive (1-44).

Example: Fourier series of a general square wave

The simplest type of square wave has a 50% duty cycle: that is, 50% of the time $f(t)$ takes on the value of 1 and 50% of the time it has the value -1. Let's do a bit more general of a case. I will consider a train of rectangular pulses (a kind of square wave, but shifted upward and scaled so that it has a minimum of zero and a maximum of V_0) with pulse duration τ and period T seconds. This waveform can be expressed as

$$f(t) = \begin{cases} 0 & nT \le t < nT + t_0 \\ V_0 & nT + t_0 \le t < nT + t_0 + \tau \\ 0 & nT + t_0 + \tau \le t < (n+1)T \end{cases} \qquad n = 0, 1, 2, \ldots \tag{1-47}$$

The waveform is plotted in figure 1.8.

There are a few general observations we can make of this signal: the first, that the pulse duration τ is a variable, so it can be wide or narrow, thus changing the duty cycle. When it

is really narrow, the signal becomes a train of impulses; when it is on the order of $\tau = T/2$, it is called "50% duty cycle". The second observation is that t_0 changes the pulse's phase.

According to (1-35), the Fourier coefficients c_k are

$$c_k = \frac{1}{T}\int_0^{t_0} 0e^{-jk\omega_0 t}dt + \frac{1}{T}\int_{t_0}^{t_0+\tau} V_0 e^{-jk\omega_0 t}dt + \frac{1}{T}\int_{t_0+\tau}^{T} 0e^{-jk\omega_0 t}dt \tag{1-48}$$

$$= \frac{1}{T}\int_{t_0}^{t_0+\tau} V_0 e^{-jk\omega_0 t}dt \tag{1-49}$$

$$= \frac{V_0}{T}\left[\frac{e^{-jk\omega_0 t}}{-jk\omega_0}\right]_{t_0}^{t_0+\tau} \tag{1-50}$$

$$= \frac{V_0}{-jk\omega_0 T}[e^{-jk\omega_0(t_0+\tau)} - e^{-jk\omega_0(t_0)}] \tag{1-51}$$

In equation (1-48), the integral defining the Fourier coefficients in (1-35) has been broken into three parts for the three "segments" of the square wave: $f(t) = 0$ (when $nT \le t < nT + t_0$), $f(t) = V_0$ (when $nT + t_0 \le t < nT + t_0 + \tau$), and $f(t) = 0$ (when $nT + t_0 + \tau \le t < (n+1)T$). The first and third terms are equal to zero, leaving only the middle term, which is just the integral of an exponential. If we remember that the derivative of an exponential is $\frac{de^{at}}{dt}$ is ae^{at}, then the anti-derivative from (1-49) to (1-50) should make sense. Finally, we just need to evaluate the expression in (1-50) between the two limits t_0 and $t_0 + \tau$ to yield (1-51).

I leave the middle steps for your homework and skip to the result:

$$c_k = \underbrace{\frac{V_0\tau}{T}\frac{\sin(k\omega_0\tau/2)}{k\omega_0\tau/2}}_{\text{magnitude}}\underbrace{e^{-jk\omega_0(t_0+\tau/2)}}_{\text{phase}} \tag{1-52}$$

This is just the polar representation of a complex valued $c_k = Re^{j\theta}$. The phase θ or angle of c_k is

$$\theta = -k\omega_0\left(t_0 + \frac{\tau}{2}\right) \tag{1-53}$$

Let's define $\alpha = k\omega_0\tau/2$, and take a look at the trigonometric factor in (1-52). This occurs frequently enough in modern communication theory to be given a name: the sampling function, or *sinc*. We define

$$\operatorname{sinc}\alpha = \frac{\sin\alpha}{\alpha} \tag{1-54}$$

We note that $\operatorname{sinc}\alpha$ is zero whenever α is an integral multiple of π; that is,

$$\operatorname{sinc} n\pi = 0, \quad n = 1, 2, 3, \ldots \tag{1-55}$$

When α is zero, the function is indeterminate, but it is easy to show that its value is unity:

$$\operatorname{sinc} 0 = 1 \tag{1-56}$$

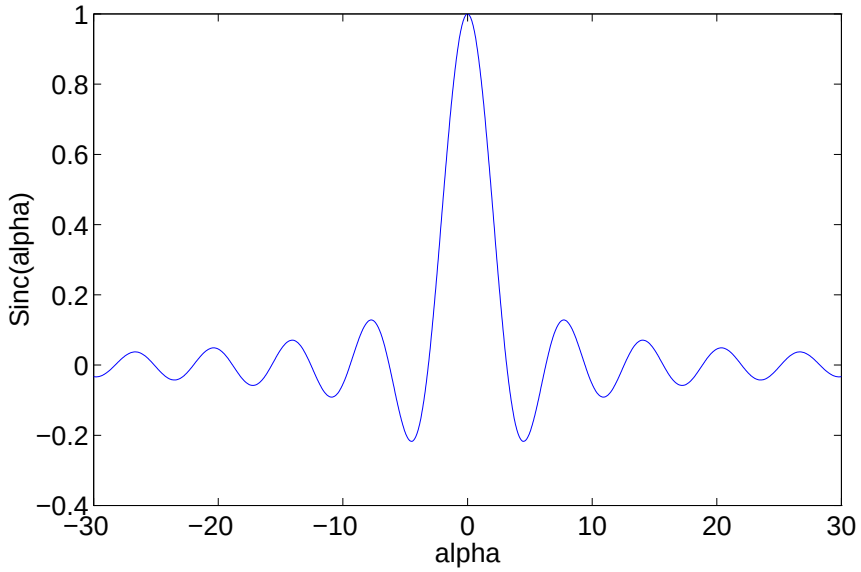


Figure 1.9: The sinc function.

The magnitude of $\operatorname{sinc}\alpha$ therefore decreases from unity at $\alpha = 0$ to zero at $\alpha = \pi$. As α increases from π to 2π, $\operatorname{sinc}\alpha$ increases from zero to a maximum less than unity. As α continues to increase, the successive maxima become smaller because the numerator of $\operatorname{sinc}\alpha$ can not exceed unity while the denominator increases. Also, $\operatorname{sinc}\alpha$ shows even symmetry. Figure 1.9 shows the sinc function.

From (1-56), we know that c_0 is

$$c_0 = \frac{V_0\tau}{T} \tag{1-57}$$

Substituting c_0 and $c_k, k = 1, 2, 3, \ldots$ into (1-43), the Fourier series of $f(t)$ is

$$f(t) = \frac{V_0\tau}{T} + 2\,\mathrm{Re}\left\{\sum_{k=1}^{\infty} \frac{V_0\tau}{T}\frac{\sin(k\omega_0\tau/2)}{k\omega_0\tau/2} e^{-jk\omega_0(t_0+\tau/2)} e^{jk\omega_0 t}\right\} \tag{1-58}$$

$$= \frac{V_0\tau}{T} + 2\,\mathrm{Re}\left\{\sum_{k=1}^{\infty} \frac{V_0\tau}{T}\frac{\sin(k\omega_0\tau/2)}{k\omega_0\tau/2} e^{jk\omega_0(t-t_0-\tau/2)}\right\} \tag{1-59}$$

$$= \underbrace{\frac{V_0\tau}{T}}_{c_0} + \underbrace{\frac{2V_0\tau}{T}\sum_{k=1}^{\infty} \operatorname{sinc}(k\omega_0\tau/2)\,\underbrace{\cos[k\omega_0(t-t_0-\tau/2)]}_{\text{harmonically related sinusoids}}}_{c_k} \tag{1-60}$$

In (1-58), we have used the knowledge of this being a real-valued signal. We collected the exponents to get (1-59), then factored out the $V_0\tau/T$, applied Euler's formula to the exponential, and finally retained only the real part of each term of the summation to yield (1-60).

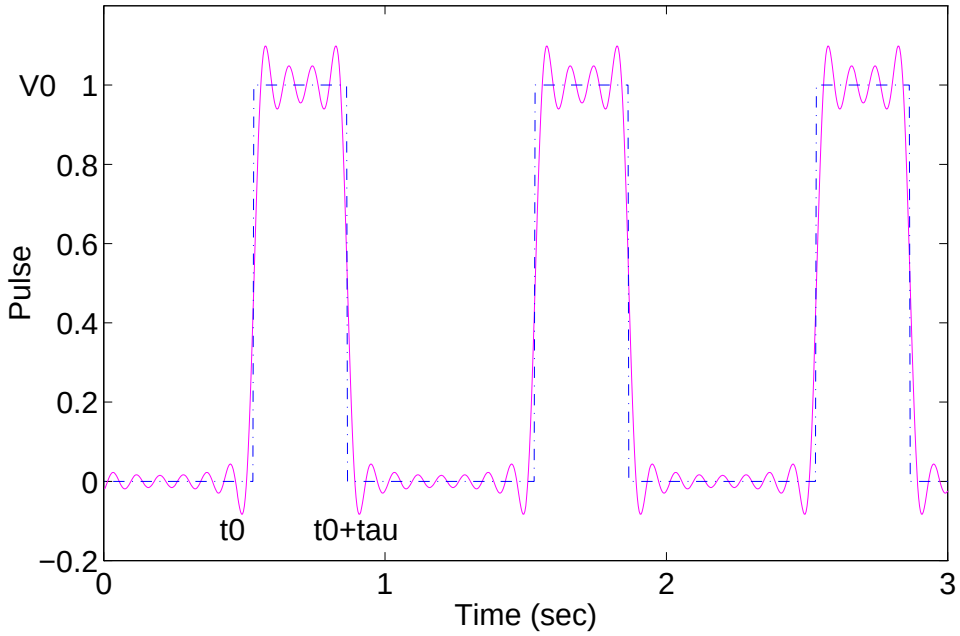


Figure 1.10: The Fourier series for a periodic sequence of rectangular pulses. The first 12 harmonic terms in summation (1-60) are included. The ideal pulse train is shown as dashed lines. The period is $T = 1$, the initial point $t_0 = 0.5$ and the pulse width $\tau = 0.3333$.

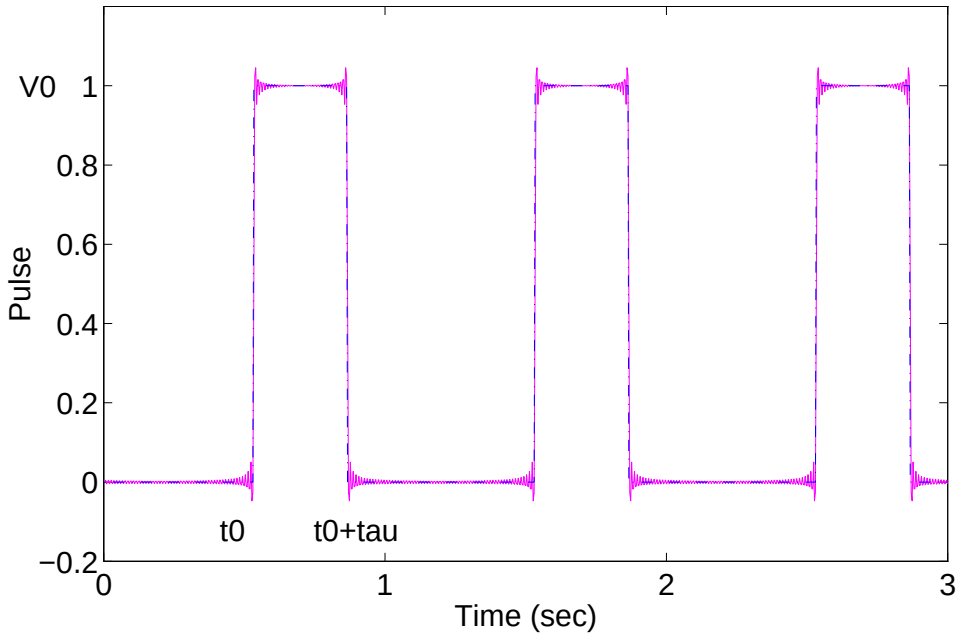


Figure 1.11: The Fourier series for a periodic sequence of rectangular pulses. The sum of the first 100 terms is shown, as in figure 1.10.

Now, a train of pulses is represented by an infinite number of sinusoidal waves. How well can it be approximated by a finite number? Look at figure 1.10. The solid curve shows the sum of the first 12 terms of the signal's Fourier series; it looks like a pulse train modulated by a sinusoid. The ideal pulses are shown as dashed lines. The approximation gets better when more terms are used, as in figure 1.11.

Let's construct the signal's *spectrum*: a plot of its Fourier coefficients. We first consider $|c_k|$ expressed in terms of the fundamental frequency f_0, remembering that $\omega_0 = 2\pi f_0$:

$$|c_k| = \frac{V_0\tau}{T}|\operatorname{sinc}(k\pi f_0\tau)| \tag{1-61}$$

The magnitude of any c_k is obtained from (1-61) by using the known values of pulse width τ and signal period $T = 1/f_0$, and selecting the desired value of $k, k = 0, 1, 2, \ldots$. Instead of evaluating (1-61) at these discrete frequencies, let us sketch the envelope of $|c_k|$ by considering the frequency kf_0 to be a continuous variable. That is, though $f = kf_0$ can really only take on the discrete values (the harmonic frequencies) $0, f_0, 2f_0, 3f_0, \ldots$, but we may think of k for the moment as a continuous variable. When f is zero, $|c_k|$ is evidently $V_0\tau/T$, and when f has increased to $1/\tau$, $|c_k|$ is zero. In fact from (1-55) when $f\pi\tau = m\pi$ $(m = 1, 2, 3, \ldots)$, $\operatorname{sinc}(f\pi\tau) = 0$, which yields zeros at frequencies $f = m/\tau$.

The resultant line spectrum and envelope is plotted in figure 1.12. The figure is the plot of the envelope of the Fourier coefficients $\{c_k\}$, or the spectrum of the periodic sequence of rectangular pulses, versus frequency $f = kf_0$. The magnitude of its peaks decays at the rate of $1/k$.

There are several observations and conclusions which we may make about the line spectrum of a periodic sequence of rectangular pulses as given in figure 1.12. With respect to the envelope of the discrete spectrum, it is evident that the width of the envelope depends upon changes in τ, the pulse width. As a matter of fact, the *shape* of the envelope does not vary with changes in T; instead, changes in T correspond to changes in the frequency interval between spectral lines. If the pulse period T is increased (f_0 is decreased), the number of spectral lines between zero frequency and $1/\tau$ increases and the amplitude of each line decreases. Figures 1.13 and 1.14 illustrate this.

A shift in the phase t_0 does not change the line spectrum, that is, $|c_k|$ is not a function of $t0$. The relative phase of the frequency components *do* change with the choice of t_0 (I just haven't plotted those).

When talking about the generalized square wave, with fixed period T and pulse width τ increasing or decreasing, the result is a family of different waveforms. When $\tau = T/2$, we have a 50% duty cycle square wave. When τ takes on different percentages of T, we get different duty cycles. Also, when τ becomes really small, the waveform become a train of impulses.

Self-Test Exercise

See A.1 #17 for the answer.

1. Prove (1-56).

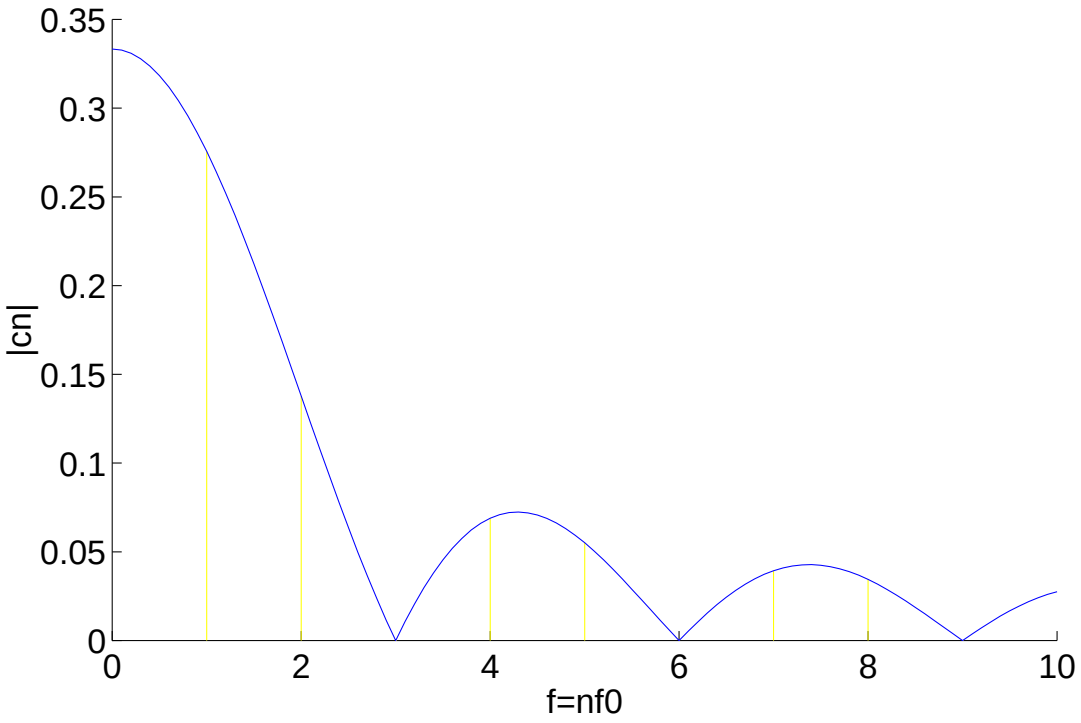


Figure 1.12: The spectrum of a periodic sequence of rectangular pulses: the magnitude of c_k vs. frequency $f = kf_0$, where $f_0 = 1/T$ (light gray vertical lines). The envelope for continuous k is plotted as the black curve. Zero points in the spectrum $|c_k|$ are at $kf_0 = n/T = m/\tau$, $m = 1, 2, 3, \ldots$.

1.8 Problems

1. Discuss how does fork mass affects tuning fork vibration frequency.

2. What musical instruments might be governed by properties analogous to that of a tuning fork?

3. When you add two sinusoids with frequencies that differ by δ, the result is beating. Imagine you're tuning a guitar. As the two strings get closer together in tone, what happens to the beat frequency? Show how this result is predicted by the sum of the complex sinusoids.

4. Implement a C++ or Java `Complex` class for representing complex numbers. Your class should include at least the following methods:

   - `add()`
   - `multiply()`

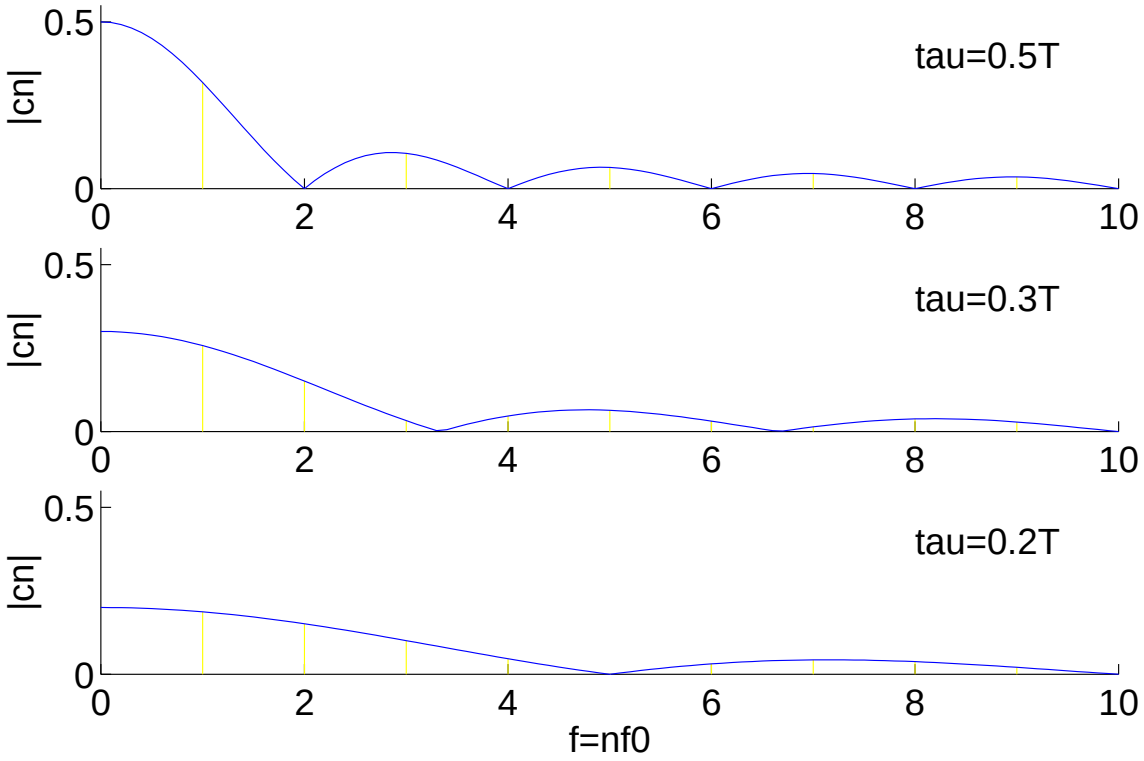


Figure 1.13: The spectrum of a periodic sequence of rectangular pulses, when T is fixed and the pulse width *tau* varies.

- `real()`
- `imag()`
- `magnitude()`
- `angle()`

Document your code so that this class could be used by someone else. Write a test program that exercises all of this class' methods. Submit hard copy of your code, your test program output, and your documentation.

5. Derive equation (1-52) from equation (1-51).

6. Determine the coefficients of the Fourier series for the signal

$$x(t) = \cos \frac{\pi}{3} t$$

.

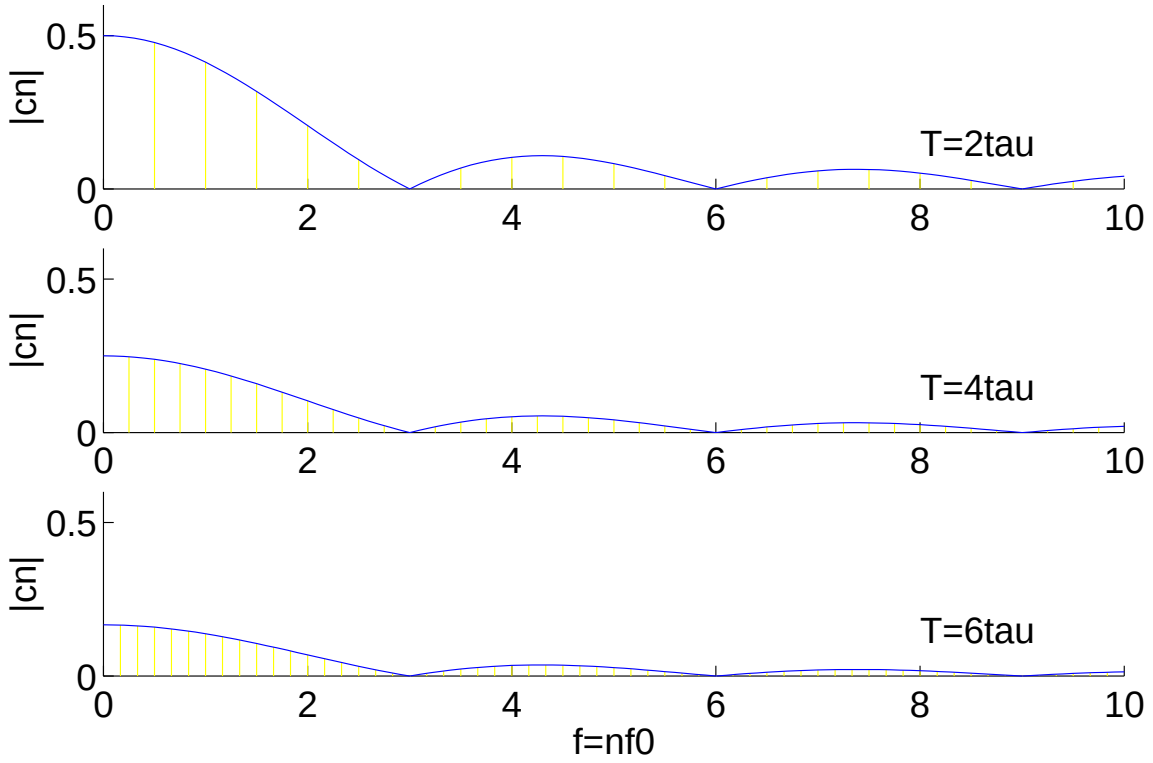


Figure 1.14: The spectrum of a periodic sequence of rectangular pulses with fixed pulse width *tau* and varying period T.

1.9 Further Reading

- James H McClellan, Ronald W. Schafer, and Mark A. Yoder, *DSP First: A Multimedia Approach*, Prentice Hall, 1998, chapters 1–3 (§3.1–3.4), appendix A.
- Martin D. Levine, *Vision in Man and Machine*, McGraw-Hill, 1985, chapter 1, sections 2.1, 2.2.
- Robert S. Tannenbaum, *Theoretical Foundations of Multimedia*, Computer Science Press, 1998, chapters 1 & 2.
- Donald Hearn & M. Pauline Baker, *Computer Graphics*, Second Edition, Prentice Hall, 1997, sections 2.1–2.4.
- A. Murat Tekalp, *Digital Video Processing*, Prentice Hall, 1995, chapters 1 & 2.

2 Signals in the Computer

In chapter 1, you were introduced to the nature of physical signals and how they can be described mathematically. In this chapter, I move onward to discuss how these "real world," analog signals end up inside computers: analog to digital conversion (A/D conversion, or ADC). I then develop the first little bit of a fundamental mathematical representation of such *discrete* signals. In the process, I visit the problems that digitization creates. At the end of this chapter, you should have a basic grasp of how ADC works and the tradeoffs involved in A/D parameters versus the characteristics of the analog signal.

2.1 From the physical to the digital

Physical signals fundamentally involve application of energy to cause some physical quantity to change. For instance, sound is carried by waves of changing air pressure; images are patterns of emitted and/or reflected light. On the other hand, computer signals are collections of binary numbers — vectors for sound, 2D arrays for images, or sequences of 2D arrays for video. The process of bridging this gap is the process of connecting the analog world to the digital computer, or *data acquisition*. There are three parts to this process:

1. transduction,
2. sampling,
3. and quantization.

Figure 2.1 shows these three steps and the intervening representations of the signal. This is an important point: the "real" signal is the physical one; what we seek to do is to produce

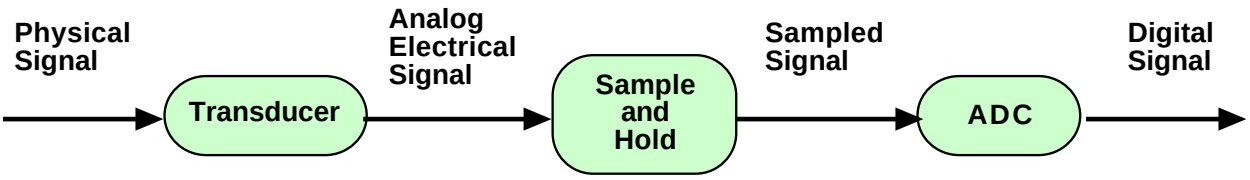


Figure 2.1: Block diagram of the data acquisition process.

a *representation* of this signal within the computer. Our goal is that this representation will carry all the information of interest that the original had. In this chapter, I will concern myself primarily with two ways that information (fidelity) is destroyed: sampling and quantization.

But, before either can be performed, the physical signal must be converted into an analog, electrical signal. This process, known as *transduction*, involves a sensor that responds to the physical signal and produces an electrical output. A microphone is such a sensor: it might include a membrane that vibrates in response to air pressure changes which in turn moves a coil of wire around a magnet to produce an electric current in the coil. Image sensors are typically composed of 2D arrays of charge-coupled devices (CCDs), in which photons affect the leakage of electrical charge. A sensor is typically connected to signal conditioning hardware (not shown), which amplifies its output to match the subsequent stages' inputs and may perform filtering operations (we'll discuss this filtering later in the chapter). The result of transduction is truly an analog signal: it is an electrical waveform whose value is proportional to the physical signal.

Figure 2.1 also presents the basic process of digitization — converting an analog electrical signal into a sequence of binary numbers. Digitization involves two processes: *sampling* and *quantization*. In the former process, the value of the analog signal is measures at regular intervals of time (the *sampling interval*). The output of a sample and hold (S/H) device will maintain a fixed level in between sampling times. This makes the quantization process easier: the analog-to-digital converter (ADC) takes an analog signal at some fixed voltage (the sampled signal) and produces a binary output that approximates the voltage — quantization.

2.2 Sampling

A S/H acts like a switch and an analog memory device. While I may talk of sampling a signal at a *point* in time, a real device requires a *period* of time to perform its function, and this includes sampling. Over some short period of time, the S/H closes its "switch" and the analog signal is presented to the memory device (which can be considered to be a capacitor, for example). The memory device's internal voltage takes a short time to reach equilibrium with the applied voltage. After this *aperture time*, the "switch" is opened, and the output, sampled signal should stay stable while the ADC does its job. However, just as a S/H has a finite aperture time rather than taking an instantaneous sample, its output also doesn't stay perfectly constant — some of the stored electrical charge leaks away, and thus the sampled signal "sags" towards zero volts. While these limitations of the physical device are not unimportant, we will focus on the basic idea of sampling at points in time and assume the S/H performs like an ideal device. Figure 2.2 shows the input/output behavior of such an idealized device.

We've already stated that we'd like the digitization process to retain all the information in the original physical signal (or, at least, that not being possible, to retain all that can be practically). Sampling alone can destroy information. In the case of figure 2.2, no

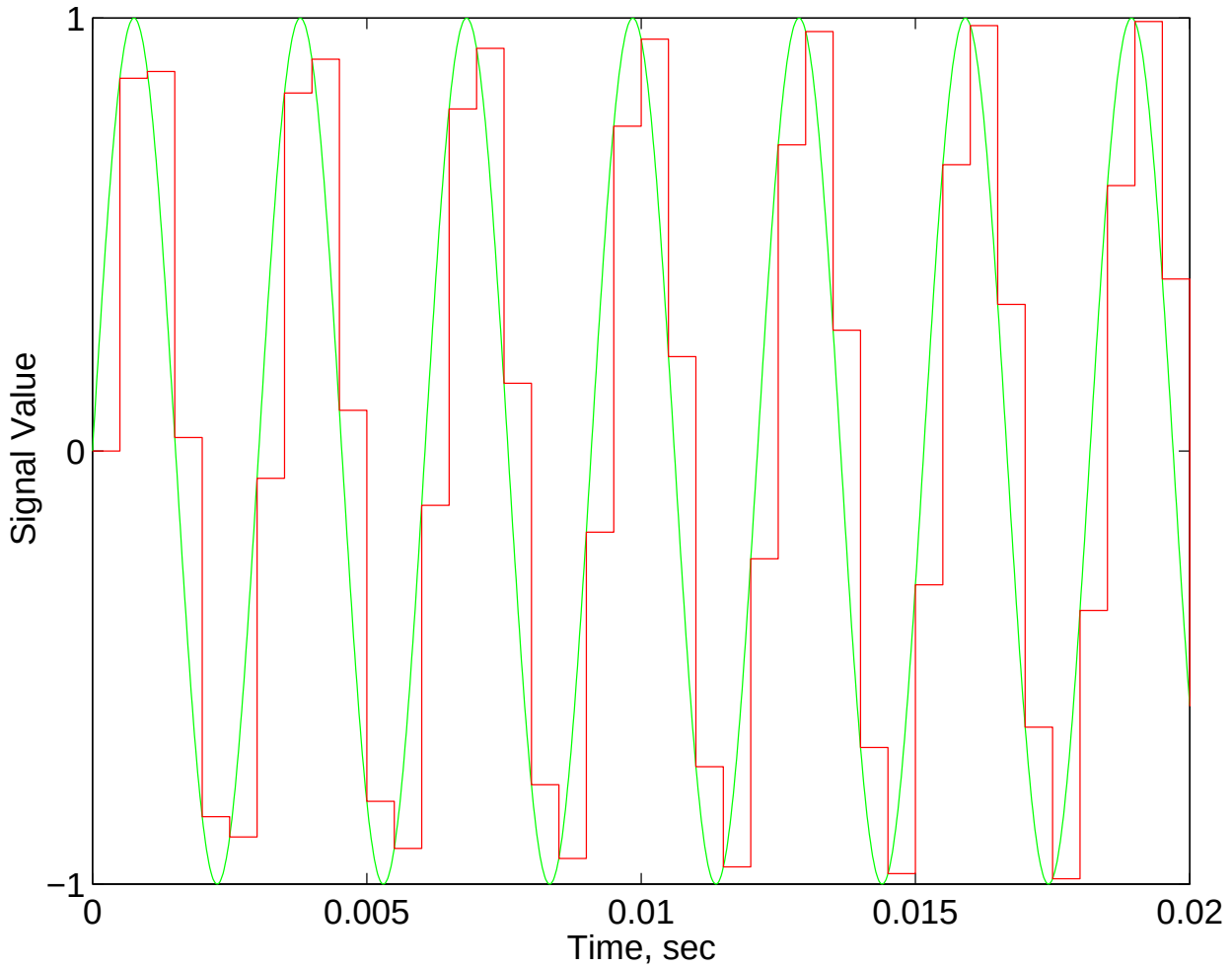


Figure 2.2: Sample and hold output. Analog input signal (sinusoidal curve) is a 330Hz sine wave; sampling rate is 2000Hz. Output of S/H is dark lines.

information is lost (the original waveform could in principle be reconstructed) because the sampling rate, at 2000Hz, is much higher than the sinusoid's frequency, 330Hz. Of course, the higher the sampling rate the more expensive the S/H and ADC, and the more data produced at the output (and produced at a higher data rate). How slowly can we sample a sinusoid? Figure 2.3 shows the same sinusoid as 2.2, this time sampled at 300Hz. In the 20ms plotted, the original signal goes through approximately six cycles. If we look at the sampled signal, it looks like a sampled version of a sinusoid that goes through only a half cycle. In other words, it appears that the relatively low sampling rate has caused the 330Hz frequency component of the original signal to appear as an *alias* at perhaps 25 or 30 Hz.

This aliasing will occur for a sinusoid whenever the sampling rate is less than twice the sinusoid's frequency. Or, alternatively, given a particular sampling rate, only sinusoids with frequencies up to one-half that rate will be accurately represented. This cutoff frequency is

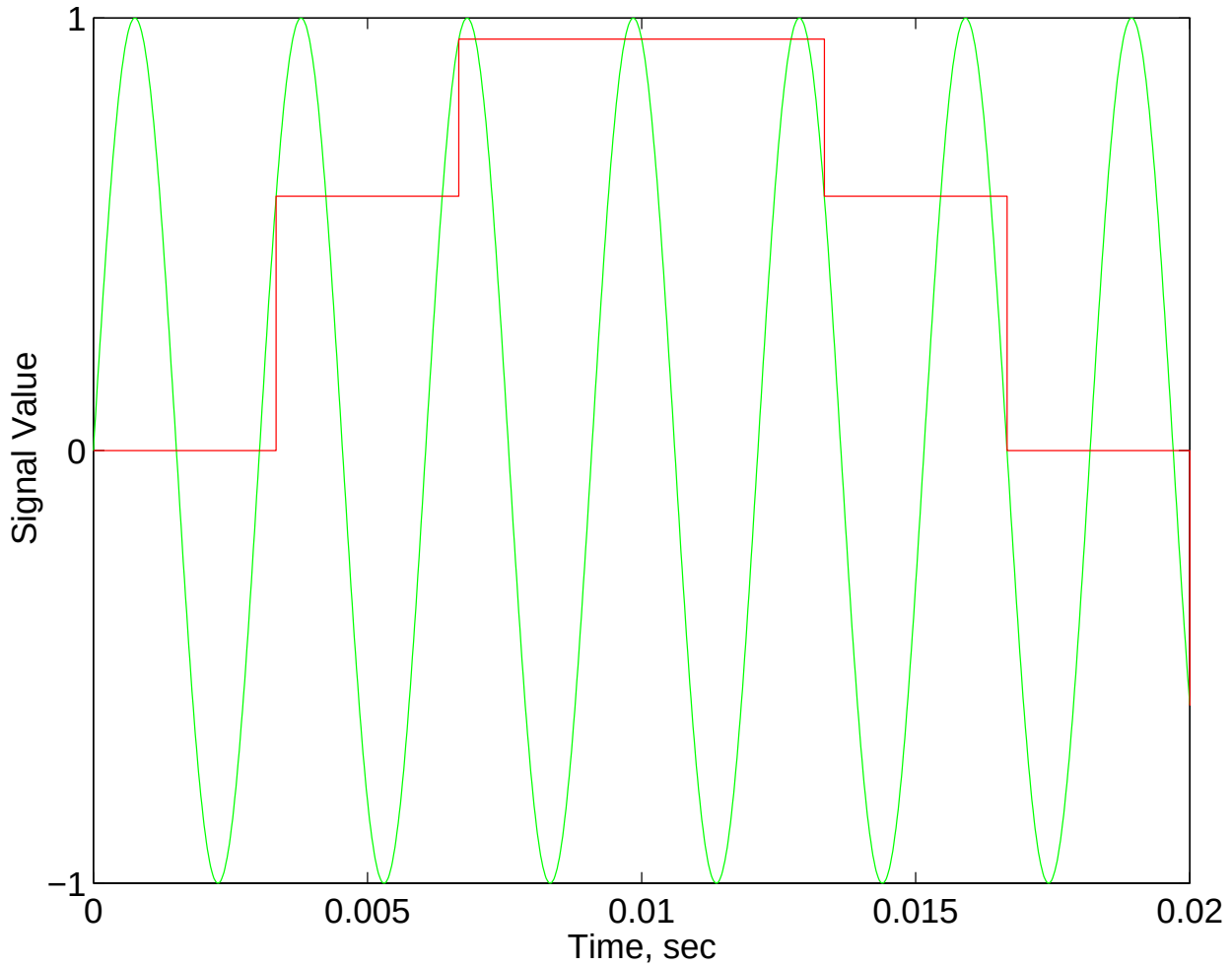


Figure 2.3: Sample and hold output. Analog input signal (green) is a 330Hz sine wave; sampling rate is 300Hz. Output of S/H is plotted in red.

called the *Nyquist frequency*. Let's suppose we sample a phasor $x(t)$ at a sampling interval of T_s. The original, analog phasor is $x(t) = e^{j\omega_0 t}$. The sampled version of the phasor is:

$$\begin{aligned} x[n] &= x(\underbrace{nT_s}_{\substack{\text{sample} \\ \text{times}}}) \\ &= e^{j\omega_0 nT_s} \end{aligned} \tag{2-1}$$

Equation (2-1) gives the value of the signal at the sample times: the phasor represented as a sequence of measurements, each taken at a time that is a multiple of the sampling interval T_s. This was done by replacing t by nT_s, the sampling times ($\{0, T_s, 2T_s, 3T_s, \ldots\}$). $x[n]$ is a *discrete* signal (a function of discrete time), corresponding to the red waveforms in figures 2.2 and 2.3.

The fundamental reason for aliasing is that the original signal is periodic. This means

that every cycle of the signal is identical, and so samples taken then will be identical, too. We can't tell if two samples were taken from two successive cycles, or if a single cycle was skipped in between them, or if a hundred cycles were skipped. Let's explore this in more detail by purposefully sampling the signal at too low of a rate. That means that, after sample $x[n-1]$ is taken, the next sample — $x[n]$ — will be taken some number of cycles later. Call the number of cycles between the two samples k. Clearly, $x[n+1]$ will then occur k cycles after $x[n]$, and so on. Since the period of the signal is $2\pi/\omega_0$, we can write the sampling interval as $T_s' = (T_s + k2\pi/\omega_0)$, where T_s' is the sampling interval written to show explicitly that it is greater than one cycle of the sinusoid. As in equation (2-1), we can write the sampled signal as:

$$\begin{aligned} x[n] &= e^{j\omega_0 n T_s'} \\ &= e^{j\omega_0 n(T_s + k2\pi/\omega_0)} && \text{(substitute expression for cycle skipping, } T_s') \\ &= e^{jnT_s(\omega_0 + k2\pi/T_s)} && \text{(factor out } T_s) \\ &= e^{j\omega_0' nT_s} && \text{(where we've called } \omega_0' = \omega_0 + k2\pi/T_s) \end{aligned} \tag{2-2}$$

By rearranging the terms as in equation (2-2), we see that sampling at too low a rate means that the discrete signal $x[n]$ can be interpreted as a phasor with frequency $\omega_0' = \omega_0 + k2\pi/T_s$, $k = \ldots, -2, -1, 0, 1, 2, \ldots$.

Now, remember that we are dealing with a real-valued signal, not a complex-valued one; we use the complex exponential as a representation that makes the math easier. For a real-valued signal, the imaginary component of the complex exponential must be canceled out. We can do this easily by adding another complex exponential at a negative frequency, since $\cos\omega_0 t = 1/2e^{j\omega_0 t} + 1/2e^{-j\omega_0 t}$. So, we should really write ω_0' as $\omega_0' = \pm\omega_0 + k2\pi/T_s$. Replacing the sampling interval T_s with the sampling rate in radians/second, $\omega_s = 2\pi/T_s$, we see that the set of ambiguous frequencies is:

$$\begin{aligned} \{\omega_0'\} &= \pm\omega_0 + \frac{k2\pi}{2\pi/\omega_s} \\ &= \pm\omega_0 + k\omega_s \end{aligned} \tag{2-3}$$

Let's consider the case where $\omega_s/2 < |\omega_0| < \omega_s$, that is, the sinusoid's frequency is just a bit greater than the Nyquist frequency (or, alternatively, you could think of it as the sampling rate being just a bit too slow: $\omega_s < 2|\omega_0|$). Call the amount that $|\omega_0|$ exceeds $\omega_s/2$ by $\omega_a = |\omega_0| - \omega_s/2$. Let's figure out what happens to the negative frequency component, $-\omega_0$. Using ω_a, we can rewrite $-\omega_0$ as

$$-\omega_0 = -\omega_a - \omega_s/2 \tag{2-4}$$

(this is simply solving the ω_a equation for $-\omega_0$).

But we know that, according to equation (2-3), the sinusoid's frequency really corresponds to a set of ambiguous frequencies, and so we can write equation (2-4) as $-\omega_0 + k\omega_s =$

$-\omega_a - \omega_s/2 + k\omega_s$ to show *all* of the ambiguous frequencies. To simplify matters, let's just look at the "first" ambiguous frequency at $k = 1$:

$$\underbrace{\omega_0'}_{\text{apparent frequency}} = \underbrace{\omega_s/2}_{\text{Nyquist frequency}} - \omega_a \tag{2-5}$$

So, the apparent frequency is one that is just a bit below the Nyquist frequency. In fact, if we substitute back in the definition of ω_a, we get that the apparent frequency is located at $\omega_s - \omega_0$. So, the first ambiguous frequency for the negative frequency component of a real sinusoid is located at a positive frequency.

What if the sampling frequency is not just a bit too slow? In that case, $\omega_0 > \omega_s$. To find the location of the $k = 1$ ambiguous frequency, we can follow the above procedure and find that we can just subtract multiples of ω_s from ω_0 until its absolute value is less than $\omega_s/2$. Now we can see that the sampled signal in figure 2.3 must have a frequency of 330-300=30Hz (subtracting the sampling frequency of 300Hz from the sinusoid's frequency of 330Hz one time).

At the end of chapter 1, I explained how *any* function can be represented as a sum of sinusoids — its Fourier series. If you look at the above discussion, you will see that, if we replace the phasor in, say, equation (2-2) by a sum of phasors at frequencies $\omega_1, \omega_2, \omega_3, \ldots$, we will reach the same conclusions, but they will now apply to *each frequency component*: each will produce aliases. In the case of the square wave of Figure 1.8, the frequencies of the sinusoids have no upper limit — the signal's *spectrum* is infinite. Certainly, then, there will be frequencies greater than $\omega_s/2$, regardless of how fast we sample.

How can we avoid this aliasing? We've been discussing the Nyquist frequency in terms of the sampling frequency; that the signal must have frequency components less than one-half the sampling frequency. Another way of looking at sampling is to require that the input signal be *band limited*: that it have a maximum frequency component, rather than an infinite spectrum. Given that the signal is band limited with maximum frequency ω_m (which we can achieve using *filters*, as described in chapters 3 and 5), then we can set our sampling rate so that it is at least $2\omega_m$.

Self-Test Exercises

See A.2 #1–3 for answers.

1. Aliasing can happen in the world around you. Identify the source of the original signal and the sampling mechanism in the following situations:

   (a) The hubcap of a car coming to a stop in a motion picture;

   (b) A TV news anchor squirming while wearing a tweed jacket;

   (c) A helicopter blade while the helicopter is starting up on a sunny day.

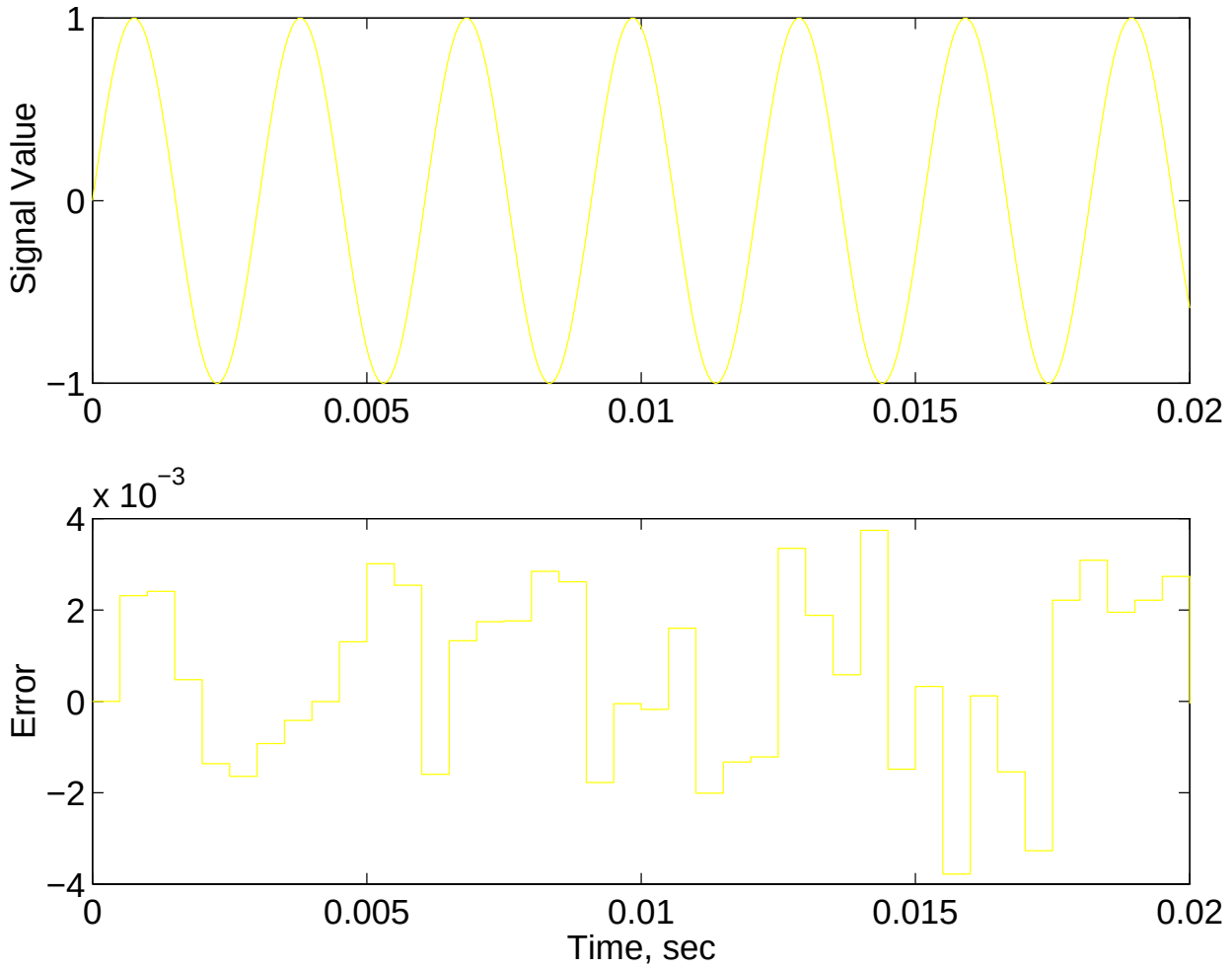


Figure 2.4: Error in output (bottom) of an ADC that uses 8 bits to represent values in the range [-1, +1] for the S/H output in figure 2.2. Original analog signal is shown in top plot.

2.3 Quantization

So we can now deal with the problems raised by discretizing time using a sample and hold device. There is another discretization that occurs, however: a discretization of the analog level output from the S/H to the finite number of binary values output by the ADC. This process is formally known as *quantization*. Since an ideal analog signal has an infinite number of possible values, while the computer representation (be it integer or floating point) has a finite number of values, there will be some error.

This error is illustrated in figure 2.4. To produce this figure, the output of the S/H from figure 2.2 was quantized to 256 levels, each corresponding to 1/256 of the distance between -1 and +1. This result was then subtracted from the ideal S/H output and plotted as the error shown in the bottom plot in the figure. How much can this error be? The error can be as much as one part in ±1/2LSB (the S/H signal is within LSB/2 of the ADC output). What

are its statistics? Assuming that Nature hasn't conspired against us, we would expect the mean of this error to be zero. If all ADC input values have equal probability, then the errors should also, which means that it is uniformly distributed between minus and plus one-half. Remembering our statistics, the standard deviation of a continuous random variable is the square root of its variance,

$$\sigma^2 = \int_{-\infty}^{+\infty} (x-\mu)^2 \mathrm{d}x \tag{2-6}$$

In this case, $\mu = 0$ and the variable's range is [-1/2, +1/2]. The result is a standard deviation of $\sigma = 1/\sqrt{12}\,\mathrm{LSB} \approx 0.29\,\mathrm{LSB}$ (how [answer in A.2 #4]?). In the case of an 8-bit ADC, this comes to $0.29/256 = 1/883$ of the ADC's full range.

Noise is ubiquitous in the physical world — and this includes analog electronics (digital electronics have noise, too, but its effect on digital signals is different). Sometimes, this noise is truly random, or *stochastic*, perhaps produced by thermal effects within the circuitry. Other times, the noise is merely unwanted signal, such as "background noise" in an audio recording. Noise is usually quantified as a ratio — *signal-to-noise ratio*, or SNR — and expressed in decibels (a logarithmic scale). To convert a ratio of two magnitudes R_M = Mag(signal)/Mag(noise) to decibels, we calculate $\mathrm{SNR_{dB}} = 20\log_{10} R_M$. (If we were talking about a ratio of powers R_P, it would be $\mathrm{SNR_{dB}} = 10\log_{10} R_P$.) So, for example, if there was no other noise present than that from quantization by an 8-bit ADC, the SNR would be $20\log_{10} 883 \approx 59$dB. For a 12-bit ADC, the SNR is 83dB; for 16-bit, the SNR is 107dB.

Let's suppose that the ADC input signal has a range of 5 volts (V), and that it includes noise (from whatever source) with a standard deviation of 1 millivolt (mV). If we use an 8-bit ADC, 5V corresponds to 255 and 1mV is then 0.051 LSB. (What is the SNR for the original signal (answer in A.2 #5) ? When we add random variables, we add their variances, and so the standard deviation of their sum is the square root of the sum of their variances, $\sigma' = \sqrt{\sigma_1^2 + \sigma_2^2}$. For an 8-bit ADC, then, we have the total noise (inherent plus quantization) of $\sqrt{0.051^2 + 0.29^2} = 0.294\,\mathrm{LSB}$. So, almost all of the noise in the digitized signal is caused by quantization (the noise on the output represents a 476% increase from the input noise)! If we go to 12 bits, the total noise is $\sqrt{0.82^2 + 0.29^2} = 0.87\,\mathrm{LSB}$ — quantization increases the noise by 6%. From a design point of view, we *select* the ADC based in part on the expected noise in the analog signal and the maximum noise we can tolerate in the digital one.

MATLAB and Sound Files

- A MATLAB .m file for demonstrating quantization using tones is available at `http://courses.washington.edu/css457/ebook/quantdemo1.m`.

- A MATLAB .m file for demonstrating quantization using a more interesting sound is available at `http://courses.washington.edu/css457/ebook/quantdemo2.m`, along with a data file at `http://courses.washington.edu/css457/ebook/amoriole2.mat`.

Self-Test Exercises

See A.2 #6 for the answer.

1. What ratio of amplitudes is represented by one bel?

2.4 Dynamic Range

So, one reason we might choose an ADC with more bits is to reduce the effects of quantization noise. There is another reason: our desire to represent both low and high amplitude signals with reasonable fidelity. The need for this *dynamic range* can result in us needing more bits. Let's use as an example the digitization of an orchestra where the ratio of the low amplitude passages to the high amplitude ones is 1/1000 (60dB). If we use an 8-bit ADC, then 255 corresponds to the highest amplitude and the lowest amplitude is 0.255 LSB. In other words, we have used just about all the bits for the loudest passages, leaving nothing for the quiet parts! For a 12-bit ADC, the lowest amplitudes are allocated about 2 bits; for a 16-bit ADC, only about 6 bits. The reason for this is that we've coded the signal *linearly*, with the step between ADC outputs being the same for low and high amplitude signals. Logically, however, it would seem unlikely that our ears would be as sensitive to slight changes in loud passages as they would be to the same changes to quiet ones.

This intuition is correct, and so consideration of the *psychophysics* of hearing (the intersection of the physics of sound and the psychology of perception) leads us to a solution for audio digitization: *companding*. We first pass the analog signal through a "squashing function" before digitization. This nonlinear function doesn't modify the quiet passages (the region of the function near zero is basically linear, with a slope of one). However, it causes changes in loud passages to result in smaller changes in the signal to be digitized. This is equivalent to allocating more bits to the quiet passages and fewer to the loud ones. Assuming we use the exact inverse function on any audio output provided by our system, there should be little noticeable effect of this companding.

Of course, this is all based on the assumption that we are digitizing a signal that will be listened to. If, on the other hand, the signal is some other kind of data not for "human consumption," then small steps at high amplitude may be just as important as those at low ones, and so companding won't be usable — it would destroy valuable information.

2.5 Periodic and Aperiodic Signals

In this chapter, I talked about a signal as being a sum of sinusoids, as I did in chapter 1. In general, a periodic signal can be expressed as a sum of *both* sines and cosines, which we can express as the sum of complex sinusoids:

$$f(t) = \sum_{k=-\infty}^{+\infty} c_k e^{jk2\pi t/T} \tag{2-7}$$

Equation (2-7) is the general Fourier series, which can be used to represent any periodic signal with period T. In this series, all of the complex sinusoids have periods that are multiples of T (frequencies that are multiples of $2\pi/T$). So, the frequency content, or *spectrum*, of a periodic signal (the c_ks) is *discrete*: it only has values at particular frequencies.

As we shall see in chapter 6, for an aperiodic signal, the Fourier series becomes the *Fourier integral*, and the signal's spectrum is continuous:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(j\omega) e^{j\omega t} \mathrm{d}\omega \tag{2-8}$$

In equation (2-8), the function $F(j\omega)$ describes the magnitude of the complex sinusoid at frequency ω. This is the continuous (because ω can take on all values, not just a discrete set) spectrum of an aperiodic signal. I'll revisit this topic later, in more detail.

2.6 Problems

1. The analytical formula for synthesis of a 50% duty cycle square wave as a sum of harmonically related sine waves (its Fourier series) is:

$$x(t) = \frac{4}{\pi} \left(\sin(\omega_0 t) + \frac{1}{3}\sin(3\omega_0 t) + \frac{1}{5}\sin(5\omega_0 t) + \cdots \right)$$

   Write a MATLAB program to compute and plot the spectrum of a square wave given this equation. Your program should prompt the user for ω_0 and which harmonic is the maximum to plot. Note that you are *not* being asked to plot the square wave as a function of time; you should plot the amplitudes of the component sinusoids as a function of the sinusoids' frequencies (like figure 1.12). Hand in hard copy of your MATLAB code and a plot of the spectrum with 10 harmonics. You may find the MATLAB function `stem()` useful.

2. Modify your above program to show how these harmonics are aliased if the square wave is sampled at ω_s (prompting the user for ω_s). Hand in hard copy of your code and an example plot for $\omega_s < \omega_0$.

3. Modify your program to play the sound of the aliased square wave (synthesize $x(t)$ from problem 2). Next, write a program to generate a band-limited square wave from a sum of sinusoids, given ω_0 and ω_s, with no frequency components above the Nyquist limit. Play that sound (the MATLAB function `soundsc()` is useful for this). Compare the sounds with and without (from problem 1, given that ω_s determines ω_{max}) aliasing. Can you find an interesting use for intentional aliasing? Hand in your code's hard copy, plots of the sample square wave and the band limited square wave (as functions of *time*), and your written comments.

2.7 Further Reading

James H McClellan, Ronald W. Schafer, and Mark A. Yoder, *DSP First: A Multimedia Approach*, Prentice Hall, 1998, chapter 4 (§4.1–4.3).

3 Filtering and Feedforward Filters

3.1 Introduction

In this chapter, I will introduce you to the most basic type of algorithm for processing digital signals: feedforward filters. We start with the concept of *filtering* and the operation of basic feedforward filters. By the end, you should understand some important terms related to filters, for example, *frequency response*, *phase response*, *transfer function* and *zeros* of a *transfer function*. You should be able to implement simple digital filters on a computer and use them to solve some simple signal processing problems.

The concept of filtering should not be new to you. For example, if you were interested in filtering and opened this book, you would see that this chapter deals with filtering and so you would pay attention to this chapter and ignore the others. Your mind performed a *bandpass* filter with the pass band being [3]. Some other kinds of filters are *low pass*, *high pass*, and *band stop*. Using the same example, a low pass filter allows you to attend to all the chapters in the book, from the beginning to the end of the pass band, say chapter 3; the high pass on the other hand will pass only the chapters beyond a certain one, say beyond chapter 4; the band stop is the one that allows you to pay attention to all the chapters *except* the stop band, say [3].

In the signal processing domain, filters exclude and/or include signal *frequencies*. For example, consider a signal $x(t)$ (where t is time) with four sinusoidal components. It has frequencies at $f_1 = 50$, $f_2 = 100$, $f_3 = 250$, and $f_4 = 350$:

$$x = \sin(2\pi t f_1) + \sin(2\pi t f_2) + \sin(2\pi t f_3) + \sin(2\pi t f_4) \tag{3-1}$$

Self-Test Exercise

See A.3 #1 for the answer.

1. Is the signal of equation 3-1 periodic? If so, what is its period?

Figure 3.1 shows a plot of about 0.1 second of this signal. Its four frequencies are shown as green peaks in figure 3.4. Let's say that we would like to have a filter to keep the $f_3 = 250$Hz sinusoid and get rid of the f_1, f_2, and f_4 sinusoids. This filter can be visualized as in figure 3.2. Here, the horizontal axis is frequency and the vertical axis is magnitude of the

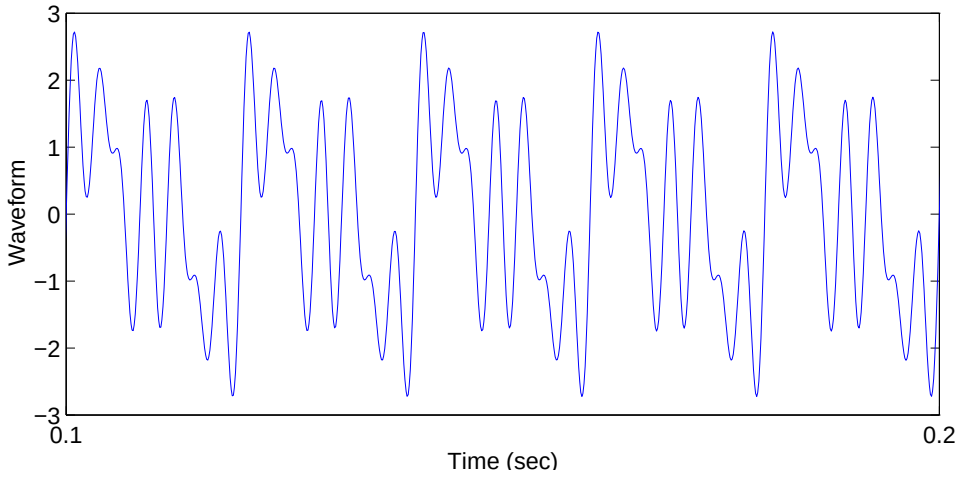


Figure 3.1: A sinusoidal signal with four frequency components: $f_1 = 50$, $f_2 = 100$, $f_3 = 250$, $f_4 = 350$Hz.

filter's response. The filter is designed to have a frequency passband between 200 and 300 Hz, which means that only the signal's frequency components within this band are allowed to pass. Figure 3.3 shows this filter's output — in other words, the filtered signal — plotted along time. Another way to look at the filtered signal is its frequency components, which are shown in figure 3.4 (magenta). This latter graph clearly shows that, after filtering, the signal is very close to a 250 Hz sinusoid, exactly as expected. You can hear these two signals as sounds: http://courses.washington.edu/css457/ebook/sine4.au is the sum of four sinusoids and http://courses.washington.edu/css457/ebook/filtered_sine4.au is the filtered version.

At this point, you should have an idea of what filters do. Next, I will explain *how* filters work.

3.2 Feedforward Filters

3.2.1 Delaying a phasor

In chapter 1, we learned the term *phasor*, a complex sinusoid expressed as $e^{j\omega t}$. We also saw that this representation makes the math simpler for adding sinusoids, at least. A phasor's magnitude is one, its frequency is ω, and it angle is ωt, where t is time. It moves around the unit circle counter-clockwise along time. If we delay it by τ sec, then the delayed time is $t - \tau$, and we can write this as:

$$e^{j\omega(t-\tau)} = e^{-j\omega\tau} e^{j\omega t} \tag{3-2}$$

This is the product of two phasors at the same frequency. This doesn't do anything except rotate the phasor by $-\omega\tau$ (i.e., it doesn't change its magnitude).

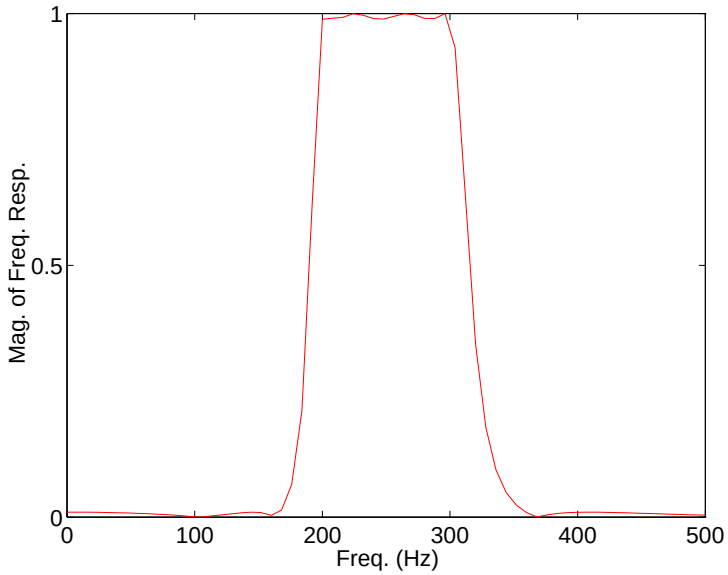


Figure 3.2: Filter's frequency response. The pass band is [200 300] Hz.

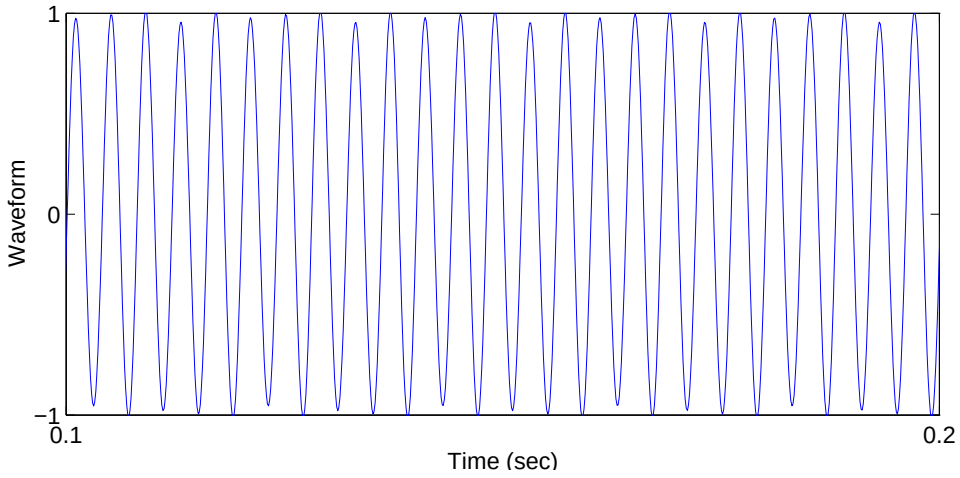


Figure 3.3: The signal (3-1) after filtering out frequencies 50, 100 and 350Hz.

3.2.2 A simple feedforward filter

Filters combine delayed versions of signals. We have already seen that signals are made up of phasors. A simple feedforward filter's *block diagram* is shown in figure 3.5. A block diagram is a type of data flow diagram: it shows flow of signal *data* within the filter (in contrast to a *flow chart*, which shows flow of *control* within a program. In figure 3.5, the input signal data $x(t)$ branches, with one copy sent to an adder ("+") and another sent to a delay block. The output of the delay block is sent to a multiplier, which multiplies it by the constant b_1 (sometimes, multiplication by a constant is just indicated by writing the constant next to a link). Its output, $y(t)$, is the summation of its input and a scaled (or *weighted*) version of its

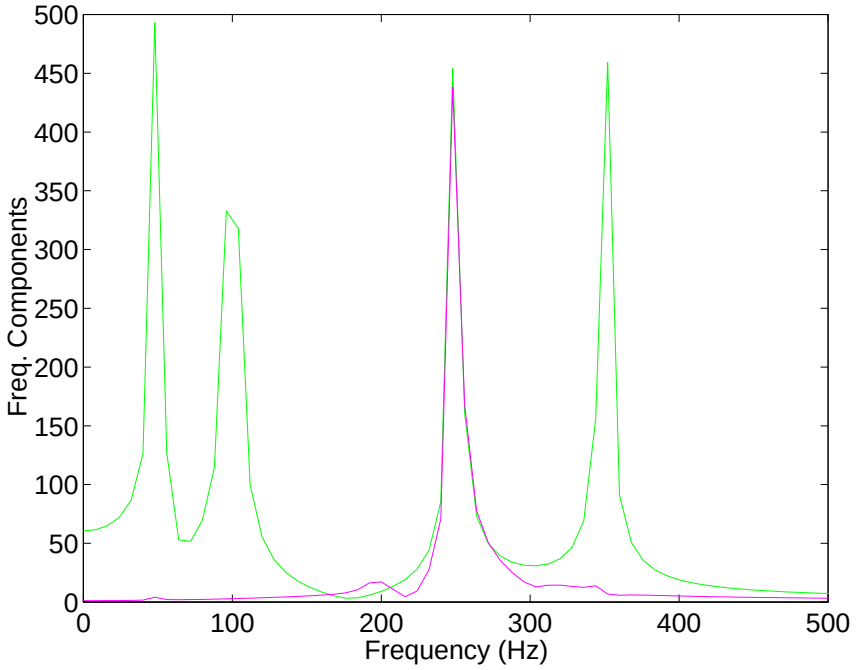


Figure 3.4: Spectrum of four-sinusoid signal (green) and its filtered version (magenta).

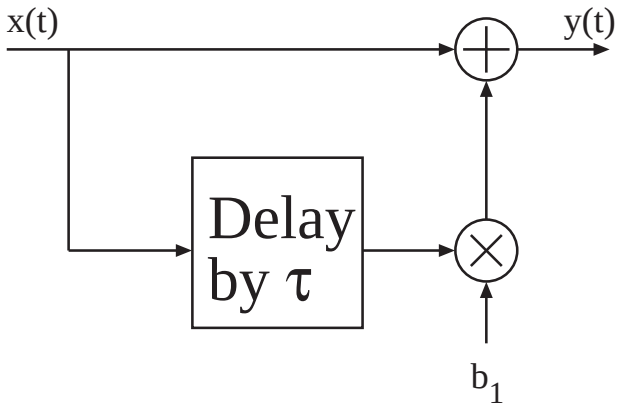


Figure 3.5: A basic feedforward filter block diagram.

input delayed by τ. Given an input signal $x(t)$, its delayed version $x(t-\tau)$, and coefficient b_1, the output $y(t)$ can be written by inspection of the block diagram as:

$$y(t) = x(t) + b_1 x(t-\tau) \tag{3-3}$$

When the input is a phasor, $x(t) = e^{j\omega t}$, equation (3-3) becomes:

$$y(t) = e^{j\omega t} + b_1 e^{j\omega(t-\tau)} \tag{3-4}$$

Notice that the input and the output have the same frequency, ω. We can factor out the

$e^{j\omega t}$ in equation (3-4) to obtain:

$$y(t) = [1 + b_1 e^{-j\omega\tau}]e^{j\omega t} \tag{3-5}$$
$$= \mathcal{H}(\omega)e^{j\omega t} \tag{3-6}$$
$$= \mathcal{H}(\omega)x(t) \tag{3-7}$$

where

$$\mathcal{H}(\omega) = \frac{y(t)}{x(t)} = 1 + b_1 e^{-j\omega\tau} \tag{3-8}$$

is called the filter's *frequency response*: how its output varies as a function of the input frequency (i.e., the ratio of output to input). Remember that τ is a constant delay; what can vary here (besides t, which is the same for both $x(t)$ and $y(t)$) is the frequency of the input, ω. So, $\mathcal{H}(\omega)$ is a complex function of frequency. Frequency response is generally written in a polar form with its magnitude response $|\mathcal{H}(\omega)|$ and phase response $\theta(\omega)$:

$$\mathcal{H}(\omega) = |\mathcal{H}(\omega)|e^{-j\theta(\omega)} \tag{3-9}$$

According to equations (3-6) and (3-9), the filter multiplies its input by its magnitude response $|\mathcal{H}(\omega)|$ and shifts its phase by the phase response $\theta(\omega)$. These two functions define the filter's behavior. For this particular example, considering only the magnitude,

$$|\mathcal{H}(\omega)| = |1 + b_1 e^{-j\omega\tau}| \tag{3-10}$$
$$= |1 + b_1^2 + 2b_1 \cos(\omega\tau)|^{\frac{1}{2}} \tag{3-11}$$

(leaving the derivation to an upcoming self-test exercise).

Since $|\cos(\omega\tau)| \leq 1$, the maximum value $|\mathcal{H}(\omega)|$ can reach is $(1 + b_1)$, which occurs when the angle $\omega\tau = n\pi, n = 0, 2, 4, ...$ (zero or even multiples of π). Why is this (answer in A.3 #2)? Similarly, the minimum of $|\mathcal{H}(\omega)|$ is $1 - b_1$, which happens when $\omega\tau = n\pi, n = 1, 3, 5...$ (odd multiples of π). For example, let's say $\tau = 167\mu$s. Remembering that ω (radians/second) is converted to Hz by $\omega = 2\pi f$, this delay corresponds to a filter with passbands centered at $2\pi f\tau = n\pi$, or $f = n/(2\tau)$ for even n, producing maxima at 0, $1/\tau$ (6 kHz), and $2/\tau$ (12 kHz) and notches at $f = n/(2\tau)$ for odd n, or $1/(2\tau)$ (3 kHz), $3/(2\tau)$ (9 kHz), and $5/(2\tau)$ (15 kHz).

You can easily see how the input signal's frequency components (remember that all periodic functions can be expressed as sums of complex sinusoids) will be altered by the filter: when they are within the filter's passbands, they will be passed through; they will be reduced in magnitude or filtered out when their frequency matches the low magnitude response. Figure 3.2 is another example of a bandpass filter with passband [200 300] Hz. The input signal's frequencies out of this range are filtered out. This is shown in figure 3.4: in the output signal, only the frequency component at $f = 250$ Hz is left.

Self-Test Exercise

See A.3 #3 for the answer.

1. Use Euler's formula and the definition of the magnitude of a complex vector to derive (3-11) from (3-10).

3.2.3 Digital Filters

Electrical engineers have spent a lot of time developing different kinds of filter transfer functions for different classes of filters. You may run across names like Butterworth or Chebyshev. The motivation behind this has been to produce filters with good properties (flatness of the passband, steepness of the rolloff from the passband, minimal phase distortion) that is still implementable in analog hardware (operational amplifiers, resistors, and capacitors). However, once we digitize a signal, we can filter it in a computer because filtering is a mathematical operation and that's what computers do. By directly implementing a filter's frequency response, we can implement digital filters that may be difficult or impossible to implement in analog hardware.

To be implemented on a computer, an analog filter must be discretized in its variables, to yield a *digital filter*. There are two important points here related to digital filters compared to analog filters:

1. Time is expressed as an integer times a *sampling period*, T_s.
2. Only frequencies below the *Nyquist frequency* can be represented.

For the first point, digitized time t or τ is in form of:

$$t = nT_s \tag{3-12}$$

or,

$$\tau = mT_s \tag{3-13}$$

where n and m are integers, and T_s is the time interval between samples or the sampling period, whose units are sec/sample. Note in equation (3-13) that we can only delay a digital signal by an integer number of sampling intervals. When the *sampling rate* is f_s (samples/second), $T_s = 1/f_s$. So the phasor becomes $e^{j\omega t} = e^{j\omega nT_s}$, with its exponent having units of radians/sec × samples × sec/samples = radians.

It's rather cumbersome to have to carry around T_s or f_s in all our equations. Additionally, if we keep these variables we will always need to note and remember the signal in question's Nyquist frequency. To simplify our notation, let's define our digital $\hat{\omega}$ to be the normalized analog ωT_s (with units of radians/sample) and use the sample counter n, rather than the analog time t. Similarly, our normalized frequency will be $\hat{f} = fT_s = f/f_s$, with units of cycles/sample. Our phasor then becomes $e^{j\omega nT_s} = e^{j\hat{\omega}n}$, and has the same form as the analog

Units:	
ω	radians/second
$\hat{\omega}$	radians/sample
$\hat{\omega}_{Nyquist} = \pi$	radians/sample
f	cycles/second (Hz)
$\hat{f}$	cycles/sample
$f_{Nyquist} = 1/2$	cycles/sample
T_s	seconds/sample
f_s	samples/second
n	samples

version, except that we are now using the sample counter instead of continuous time.

For the second point, recall that the Nyquist frequency is:

$$f_{Nyquist} = \frac{f_s}{2} \tag{3-14}$$

Only frequencies below $f_{Nyquist}$ will be accurately represented after sampling, beyond it they are aliased to frequencies below $f_{Nyquist}$. It is important to restrict the frequency range to $f_{Nyquist}$ so you can get correct results.

Again, for convenience, the digital frequency can be normalized by f_s,

$$\hat{f} = \frac{f}{f_s} \tag{3-15}$$

and

$$\hat{f}_{Nyquist} = \frac{f_{Nyquist}}{f_s} = \frac{f_s/2}{f_s} = \frac{1}{2} \tag{3-16}$$

Since the analog $f \leq f_s/2$, $\hat{f}$ is a fraction that ranges between zero and 1/2 — a fraction of the digital sampling rate. In other words, for a sinusoid, we can only have up to 1/2 of a cycle per sample. We can multiply it by f_s to convert it back to the analog units of Hz, if necessary.

If $\hat{f}_{Nyquist}$ is 1/2, the normalized $\hat{\omega}_{Nyquist}$ is

$$\hat{\omega}_{Nyquist} = 2\pi \hat{f}_{Nyquist} = 2\pi \times 1/2 = \pi \tag{3-17}$$

Going back to our simple feedforward filter, the discrete representation of equation (3-11) with a one time step delay ($\tau = T_s$) is:

$$|\mathcal{H}(\hat{\omega})| = |1 + b_1^2 + 2b_1 \cos(\hat{\omega})|^{\frac{1}{2}} \tag{3-18}$$

Since $\hat{\omega}_{Nyquist} = \pi$, $|\mathcal{H}(\hat{\omega}_{Nyquist})| = (1 - b_1)$. This response corresponds a notch at $\hat{f} = 1/2$ (one-half the sampling rate). The filter has a broad band from 0 until the notch, so it can pass all the frequencies from 0 to below 1/2 — this filter is a lowpass filter.

Self-Test Exercises

See A.3 #4–5 for answers.

1. Suppose that we sample a signal at 1000Hz. For each of the following analog frequencies f, determine ω, $\hat{f}$, and $\hat{\omega}$. Indicate if that frequency will be aliased.
   (a) $f = 100$Hz.
   (b) $f = 200$Hz.
   (c) $f = 500$Hz.
   (d) $f = 1000$Hz.

2. Suppose that we sample a signal at 44.1kHz (the sampling rate used in audio CDs). For each of the following analog frequencies f, determine ω, $\hat{f}$, and $\hat{\omega}$. Indicate if that frequency will be aliased.

   (a) $f = 100$Hz.

   (b) $f = 1000$Hz.

   (c) $f = 10000$Hz.

   (d) $f = 20000$Hz.

   (e) $f = 25000$Hz.

3.2.4 Delay as an Operator

We're still on our quest to make the mathematics of filter design and analysis as simple as possible. Generally, a feedforward filter can have many delays, not just one, and the input $x[n]$ to output $y[n]$ relation can be written as:

$$\begin{aligned} y[n] &= b_0 x[n] + b_1 x[n-1] + \cdots + b_M x[n-M] \\ &= \sum_{k=0}^{M} b_k x[n-k] \end{aligned} \tag{3-19}$$

where M is the number of delays. We shall refer to this as the filter's *defining equation*. Each term in the summation corresponds to a parallel pathway in the block diagram with a particular delay and constant multiplier.

When $x[n]$ is the phasor $e^{jn\hat{\omega}}$ this becomes

$$y[n] = \sum_{k=0}^{M} b_k e^{jn\hat{\omega} - jk\hat{\omega}} \tag{3-20}$$

$$= e^{jn\hat{\omega}} \sum_{k=0}^{M} b_k e^{-jk\hat{\omega}} \tag{3-21}$$

$$= \underbrace{x[n]}_{\text{original signal}} \quad \underbrace{\sum_{k=0}^{M} b_k e^{-jk\hat{\omega}}}_{\text{delaying terms}} \tag{3-22}$$

Now that you're comfortable with the concept of multiplication by the phasor $e^{-jk\hat{\omega}}$ being a delay, let's get rid of it. Seriously, though, it's a lot of writing; this phasor is acting as an *operator* which, when applied to another phasor, delays it. We can use another symbol for convenience's sake. In this case, we define an *operator* z as follows:

$$z = e^{j\hat{\omega}} \tag{3-23}$$

This is a symbol that represents the application of an action on an object (an operation, so z is an operator). So a k time delay $e^{-jk\hat{\omega}}$ can be written as

$$z^{-k} = e^{-jk\hat{\omega}} \tag{3-24}$$

Here z^{-k} is a k time *delay operator*, where $k = 0, 1, 2, \ldots$ denotes the $0, 1, 2, \ldots k, \ldots$ time step delay. Operators are a general concept in mathematics, which can be used in many other circumstances to simplify notation.

Example 1: Consider a "square" operator S, which squares the thing on which it operates. When it operates on a variable x, we get:

$$Sx = x^2$$

Example 2: The transpose operator T is commonly used in linear algebra. It *transposes* the matrix to which it is applied, exchanging its rows and columns. For example, when applied to the matrix $\mathbf{A}$,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

the result is

$$\mathsf{T}\mathbf{A} = \mathbf{A}^{\mathsf{T}} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

An operator can be thought of as purely notation (though this can be a subtle point): it stands for a mathematical operation. We could use a functional notation, for example delay($x[n]$) just as well. However, if the mathematical operation has certain properties, then the operator notation is much more useful, as its syntax gives us "direct access" to these properties because of our familiarity with them from other mathematical operations. In the case of delay, the z^{-k} operator notation "works" because of the properties of multiplication and division and their effects on exponents (exponents add during multiplication; exponents are negated when dividing). We are not "really" multiplying or dividing, in the sense that the mathematical multiplication and division operations are not being used, but applying the delay operator or its inverse "looks like" multiplication or division.

Now we're ready to analyze how a digital filter responds to an input. Let the vector $X = \{x[0], x[1], \ldots, x[n], \ldots\}$ be the entire digital input signal: the ordered set of all samples. We use the same notation to produce the vector Y for the ordered set of output samples. Instead of using just one sample ($x[n]$ and $y[n]$) as in equation (3-22), let's substitute X and Y and the delay operator z^{-k} to obtain an equation that describes the action of a filter on an entire digital signal (in other words, how to compute *all* of the elements of Y

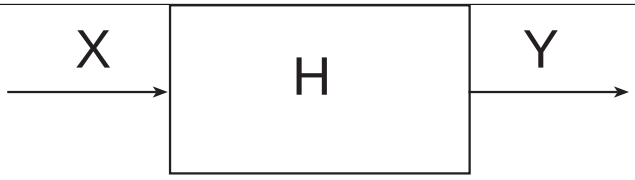


Figure 3.6: Treat transfer function as a black box.

"simultaneously" using a single parallel operation):

$$Y = \sum_{k=0}^{M} b_k z^{-k} X \tag{3-25}$$

$$= [b_0 + b_1 z^{-1} + \cdots + b_k z^{-k} + \cdots] X \tag{3-26}$$

The benefit of using the delay operator is that it makes the task of factoring out the entire signal X simple. If we call the expression in the square brackets $H(z)$,

$$H(z) = \sum_{k=0}^{M} b_k z^{-k} \tag{3-27}$$

$$= b_0 + b_1 z^{-1} + \cdots + b_k z^{-k} + \cdots \tag{3-28}$$

we get

$$Y = H(z)X \tag{3-29}$$

$H(z)$ is also an operator, which transfers the input signal X to the output signal Y, so it is called the filter's *transfer function*. The relation between the analog frequency response and digital transfer function is

$$\mathcal{H}(\hat{\omega}) = H(z)|_{z=e^{j\hat{\omega}}} \tag{3-30}$$

So, we evaluate the transfer function $H(z)$ at the frequency $\hat{\omega}$ by substituting $z = e^{j\hat{\omega}}$ to get the frequency response. For $M = 1$ (a filter with a single delay), this yields

$$Y = H(z)X = [b_0 + b_1 z^{-1}]X \tag{3-31}$$

Equation (3-31) says that the output equals the weighted sum of the input signal and the input signal delayed by one time step (one sampling interval).

We can treat the transfer function as a black box that does everything that the filter needs to do. We only need to pay attention to the input and output, as in figure 3.6. This give us a way to combine different filters, simply by composing block diagrams. Consider two simple filters $H_1(z)$ and $H_2(z)$ connected in series. $H_1(z)$ has input X and output W, and $H_2(z)$ takes $H_1(z)$'s output as its input and outputs Y. Starting from the output of this system, this can be written

$$Y = H_2(z)W = H_2(z)[H_1(z)X] = \underbrace{[H_2(z)H_1(z)]}_{\text{combined transfer function}} X \tag{3-32}$$

This suggests that the combined transfer function is $H_2(z)H_1(z)$. In fact, we can interchange the order

$$H_2(z)H_1(z) = H_1(z)H_2(z) \tag{3-33}$$

Because the transfer function is a polynomial, this gives us a way to represent digital filtering as just multiplication by a polynomial (and the product of two polynomials is just another polynomial).

Self-Test Exercises

See A.3 #6–8 for answers.

1. Write equation (3-22) for $k = 0, 1, 2, 3$, then write the transfer function for each.
2. Given the signal $x(t) = \sin t$ and the derivative operator $D = \mathrm{d}/\mathrm{d}t$, what is $Dx(t)$?
3. When

$$H_1(z) = b_0 + b_1 z^{-1}$$
$$H_2(z) = b'_0 + b'_1 z^{-1}$$

and b_0, b_1, b'_0, and b'_1 are constants, show that $H_2(z)H_1(z) = H_1(z)H_2(z)$.

3.2.5 The z-plane

I just used z as a delay operator without much comment as to why I picked the same symbol as I've used for the complex plane. The operator $z = e^{j\hat{\omega}}$ is obviously a complex variable in the z-plane. For any value of $\hat{\omega}$, it lies on a circle of radius one, at an angle of $\hat{\omega}$ relative to the positive real axis — it is the polar representation of a complex number. As $\hat{\omega}$ varies from zero to 2π (or $-\pi$ to $+\pi$, if you prefer not to consider $\hat{\omega} > \hat{\omega}_{Nyquist}$), its path is the unit circle in the z-plane. Since the Nyquist frequency $\hat{\omega}_{Nyquist} = \pi$, we are only interested the top of half of the circle running from $\hat{\omega} = 0$ to $\hat{\omega} = \pi$.

Let's examine the defining equation of a simple digital filter with one time delay:

$$y[n] = x[n] - b_1 x[n-1] \tag{3-34}$$

Just for convenience, I have used subtraction instead of summation (or, equivalently you can think that I used a negative weight on the delayed signal). Using the delay operator z and the transfer function $H(z)$, this becomes:

$$Y = [1 - b_1 z^{-1}]X = H(z)X \tag{3-35}$$

Obviously,

$$H(z) = 1 - b_1 z^{-1} = (1 - b_1 z^{-1})\frac{z}{z} = \frac{z - b_1}{z} \tag{3-36}$$

For our purposes, we shall restrict ourselves to considering z to be on the unit circle ($z = e^{j\hat{\omega}}$). The magnitude of $H(z)$ is the same as the magnitude of $\mathcal{H}(\hat{\omega})$, so the magnitude of the frequency response is

$$|\mathcal{H}(\hat{\omega})| = |H(z)|_{z=e^{j\hat{\omega}}} = \frac{|z - b_1|_{z=e^{j\hat{\omega}}}}{|z|_{z=e^{j\hat{\omega}}}} = |z - b_1|_{z=e^{j\hat{\omega}}} \tag{3-37}$$

because $|z| = 1$ when we're on the unit circle. You can see that although $z = 0$ makes the denominator of $H(z)$ zero, we're not considering that case — we've already said that we're on the unit circle: that $|z| = 1$. The value that makes the denominator of $H(z)$ zero is called a *pole*, which we will talk about in a subsequent chapter.

Equation (3-37) tells us that the magnitude of the transfer function is the distance between z and b_1 in the complex plane. Since z is a vector from the origin to the unit circle and b_1 is a constant, which can be any number here, $|\mathcal{H}(\hat{\omega})|$ is equal to the length of the vector from b_1 to the unit circle where z points. As this length becomes shorter, $|\mathcal{H}(\hat{\omega})|$ becomes smaller. We can see that is the case when z nears b_1.

The other thing that equation (3-37) tells us is that when $z = b_1$, $\mathcal{H}(\hat{\omega}) = 0$, so b_1 is a *root* of $\mathcal{H}(\hat{\omega})$ — also called a *zero* — which makes the magnitude of the frequency response reach its minimum. Obviously, when the zero (b_1) is near $\hat{\omega} = 0$, this results in a high pass filter because it doesn't pass frequencies near zero (low frequencies). Similarly, when the zero (b_1) is near $\hat{\omega} = \pi$ we obtain a low pass filter: high frequency components are filtered out.

Example 3: Consider the general two-time-step delay feedforward filter,

$$y[n] = x[n] + b_1 x[n-1] + b_2 x[n-2]$$

where b_1 and b_2 are real constants. Let's analyze its behavior. The transfer function is

$$\begin{aligned} H(z) &= 1 + b_1 z^{-1} + b_2 z^{-2} \\ &= (1 + b_1 z^{-1} + b_2 z^{-2}) \frac{z^2}{z^2} \\ &= \frac{z^2 + b_1 z + b_2}{z^2} \end{aligned}$$

The magnitude response is

$$|H(z)| = \frac{|z^2 + b_1 z + b_2|}{|z^2|} \tag{3-38}$$

It has two poles at zero; however, as we already know, $|z^2| = 1$, so they don't affect the magnitude response. So $|H(z)|$ becomes:

$$|H(z)| = |z^2 + b_1 z + b_2| \tag{3-39}$$

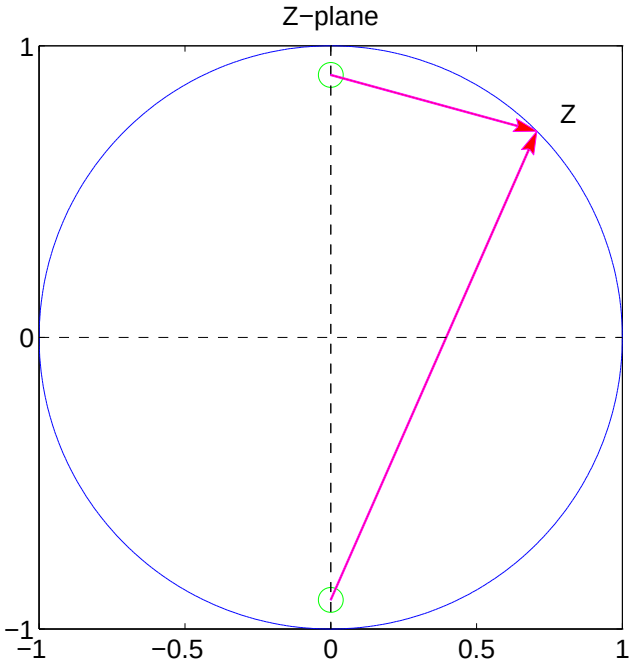


Figure 3.7: Two zero feedforward filter. $r = 0.9$, $\hat{\omega}_0 = \pm\pi/2$

The zeros of this magnitude response are merely the roots of a polynomial of order two,

$$z_{1,2} = \frac{-b_1 \pm \sqrt{b_1^2 - 4b_2}}{2} \tag{3-40}$$

For real b_k, the possible types of zeros are:

- When $b_1^2 > 4b_2$, there are two real zeros.
- When $b_1^2 = 4b_2$, there are repeated zeros at $-b_1$.
- When $b_1^2 < 4b_2$, a pair of complex conjugate zeros result.

Let's call the two solutions of equation (3-40) z_1 and z_2. Since these are the roots of the magnitude response, we can factor the polynomial in (3-39) that describes the magnitude response as:

$$|H(z)| = |(z - z_1)(z - z_2)| \tag{3-41}$$

We can write the zeros in polar form, for convenience of visualization, as

$$z_i = r_i e^{j\hat{\omega}_{0_i}}, \quad i = 1, 2 \tag{3-42}$$

where $r_i > 0$ are the radii where the zeros are located, and $\hat{\omega}_{0_i}$ their angles. Depending on the angles $\hat{\omega}_{0_i}$, the zeros can be either real or complex. For example, when $\hat{\omega}_{0_i} = 0$, the zero lies on the z-plane's real axis, and when $\hat{\omega}_{0_i} = \pi/2$, it is on the imaginary axis. Actually,

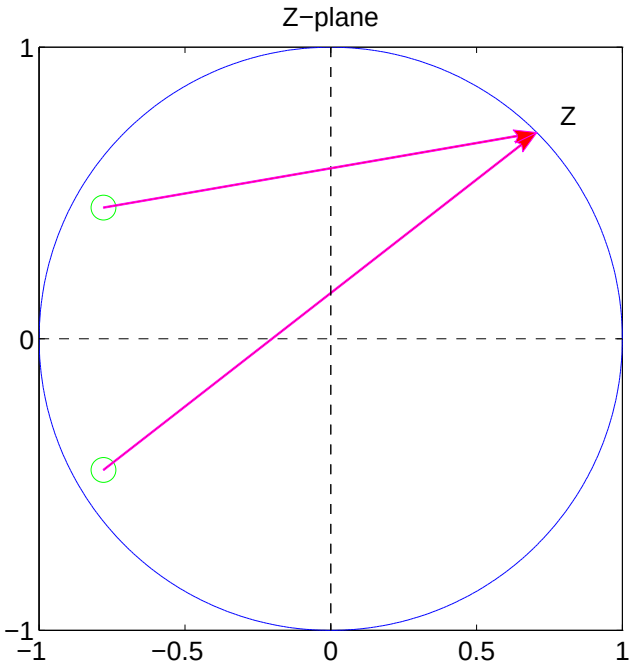


Figure 3.8: Two zero feedforward filter. $r = 0.9$, $\hat{\omega}_0 = \pm 5\pi/6$

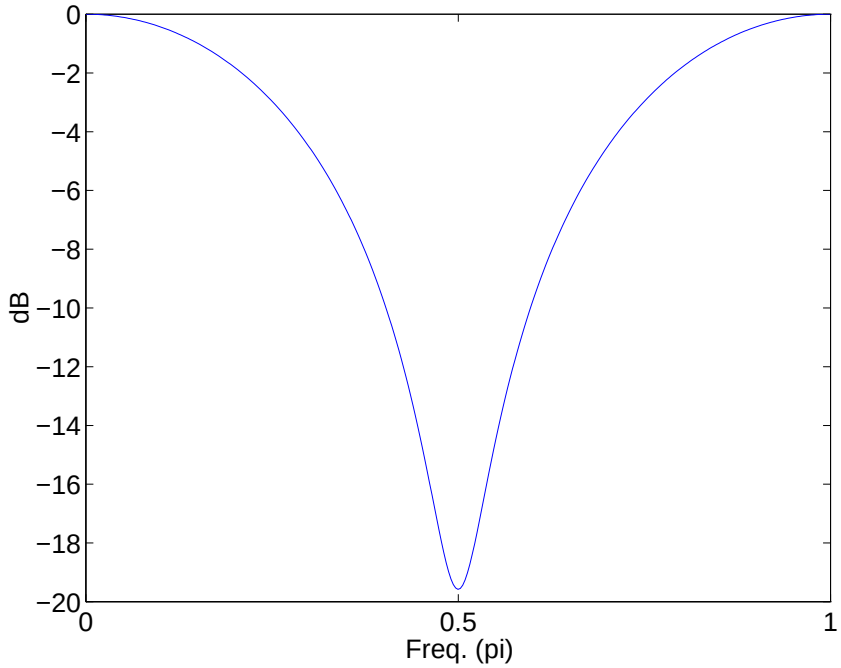


Figure 3.9: Magnitude of frequency response for the filter in figure 3.7.

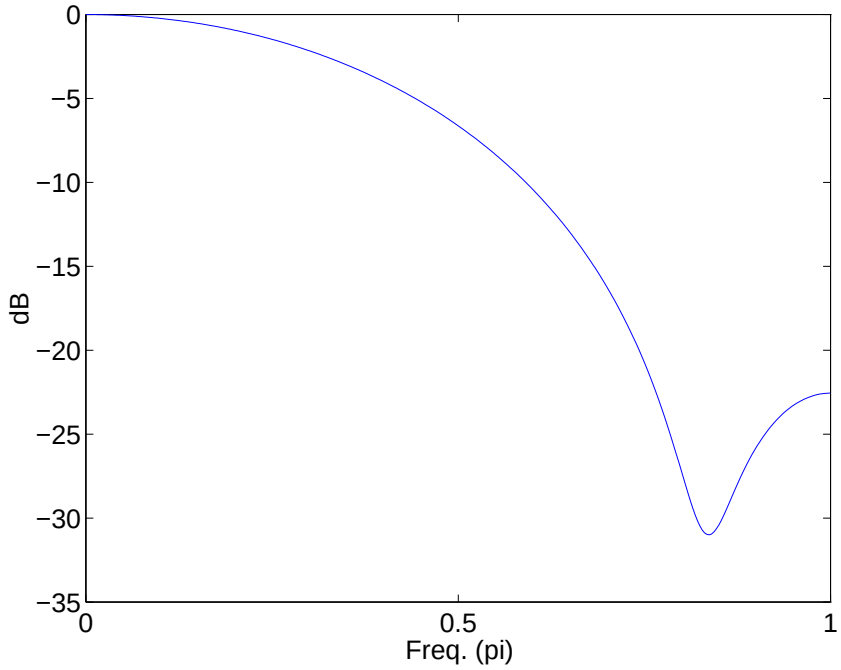


Figure 3.10: Magnitude of frequency response for the filter in figure 3.8.

when a filter's coefficients are real, if one of the zeros is a complex number, the other one will be its conjugate mate (if one of them is real, the other will be real, too). So, in the case where one zero is imaginary at $\hat{\omega}_{0_i} = \pi/2$, the other zero's angle would be $\hat{\omega}_0 = -\pi/2$. So, for this filter and real coefficients, the pair of complex zeros can be written as

$$z_{1,2} = re^{\pm j\hat{\omega}_0} \tag{3-43}$$

Figures 3.7 and 3.8 show two different sets of zero locations. The magnitude responses for those two sets of zeros are presented in figures 3.9 and 3.10.

3.2.6 Phase Response

So far we have been talking about the magnitude response of $\mathcal{H}(\hat{\omega})$. In the last topic of this chapter, let's talk about is its phase response. We already know that $\mathcal{H}(\hat{\omega})$ can be expressed in polar form, with its magnitude and angle

$$\mathcal{H}(\hat{\omega}) = |\mathcal{H}(\hat{\omega})|e^{j\theta(\hat{\omega})} \tag{3-44}$$

$|\mathcal{H}(\hat{\omega})|$ is the magnitude response and $\theta(\hat{\omega})$ is the phase response. The phase response can be computed as

$$\theta(\hat{\omega}) = \angle\mathcal{H}(\hat{\omega}) = \arctan\left(\frac{\text{Im}[\mathcal{H}(\hat{\omega})]}{\text{Re}[\mathcal{H}(\hat{\omega})]}\right) \tag{3-45}$$

When the input signal is a phasor, $x[n] = e^{jn\hat{\omega}}$, the filter's output is

$$y[n] = \underbrace{|\mathcal{H}(\hat{\omega})|e^{j\theta(\hat{\omega})}}_{\mathcal{H}(\hat{\omega})} e^{jn\hat{\omega}} = \underbrace{|\mathcal{H}(\hat{\omega})|}_{\text{change in magnitude}} \underbrace{e^{j(n\hat{\omega}+\theta(\hat{\omega}))}}_{\text{phase shift}} \tag{3-46}$$

So, what a filter does to a phasor (one frequency of the input) is to change the input's magnitude at that frequency by multiplying by $|\mathcal{H}(\hat{\omega})|$ and shift its phase (which is the same thing as delaying it) by the phase response $\theta(\hat{\omega})$.

Example 4: Let's examine the previous example (from the discussion of the z-plane):

$$y[n] = x[n] + b_1 x[n-1] + b_2 x[n-2]$$

Its transfer function is

$$H(z) = 1 + b_1 z^{-1} + b_2 z^{-2}$$

To get the filter's phase response, we need to access the frequency response's real and imaginary parts. Therefore the frequency response is

$$\begin{aligned} \mathcal{H}(\hat{\omega}) &= 1 + b_1 e^{-j\hat{\omega}} + b_2 e^{-2j\hat{\omega}} \\ &= 1 + b_1(\cos\hat{\omega} - j\sin\hat{\omega}) + b_2(\cos 2\hat{\omega} - j\sin 2\hat{\omega}) \\ &= \underbrace{(1 + b_1\cos\hat{\omega} + b_2\cos 2\hat{\omega})}_{\text{Re}[\mathcal{H}(\hat{\omega})]} \underbrace{-j(b_1\sin\hat{\omega} + b_2\sin 2\hat{\omega})}_{j\,\text{Im}[\mathcal{H}(\hat{\omega})]} \end{aligned} \tag{3-47}$$

where we have used Euler's formula to rewrite it so that we can separate its real and imaginary components. We can obtain the magnitude response from the square root of the sum of the square of these components and the phase response using equation (3-45).

When the magnitude response of a filter is plotted, the y axis scale is usually expressed in decibels (dB), a logarithmic scale. We can take advantage of this to slightly simplify computation its computation. To do this, we note that the *square* of the magnitude response is the sum of the squares of the real and imaginary components:

$$|\mathcal{H}(\hat{\omega})|^2 = [1 + b_1\cos\hat{\omega} + b_2\cos 2\hat{\omega}]^2 + [b_1\sin\hat{\omega} + b_2\sin 2\hat{\omega}]^2 \tag{3-48}$$

A quantity is converted to dB by taking twenty times the logarithm (base ten). $|\mathcal{H}(\hat{\omega})|$ can be converted to dB as

$$\begin{aligned} |\mathcal{H}(\hat{\omega})|_{dB} &= 20\log_{10}|\mathcal{H}(\hat{\omega})| = 10\log_{10}|\mathcal{H}(\hat{\omega})|^2 \\ &= 10\log_{10}\left\{[1 + b_1\cos\hat{\omega} + b_2\cos 2\hat{\omega}]^2 + [b_1\sin\hat{\omega} + b_2\sin 2\hat{\omega}]^2\right\} \end{aligned}$$

Which is a trivial matter to compute for any value of $\hat{\omega}$ (on a computer) for plotting purposes. The phase response is

$$\theta(\hat{\omega}) = \arctan\left[\frac{-(b_1\sin\hat{\omega} + b_2\sin 2\hat{\omega})}{1 + b_1\cos\hat{\omega} + b_2\cos 2\hat{\omega}}\right] \tag{3-49}$$

Consider a special case: $b_1 = 0$ and $b_2 = 1$ (a filter with a two-step delay). Substituting these values into equations (3-48) and (3-49), the squared magnitude becomes

$$\begin{aligned}|\mathcal{H}(\hat{\omega})|^2 &= (1 + \cos 2\hat{\omega})^2 + \sin^2 2\hat{\omega} \\ &= 1 + 2\cos 2\hat{\omega} + \cos^2 2\hat{\omega} + \sin^2 2\hat{\omega} \\ &= 1 + 2\cos 2\hat{\omega} + 1 \\ &= 2(1 + \cos 2\hat{\omega}) \\ &= 2(1 + 2\cos^2 \hat{\omega} - 1) \\ &= 4\cos^2 \hat{\omega}\end{aligned}$$

Where the next to last step made use of the double-angle identity, $\cos 2\theta = 2\cos^2 \theta - 1$. The square root of this is

$$|\mathcal{H}(\hat{\omega})| = 2|\cos \hat{\omega}|$$

With the substitution, the phase response becomes

$$\theta(\hat{\omega}) = \arctan\left(\frac{-\sin 2\hat{\omega}}{1 + \cos 2\hat{\omega}}\right) \tag{3-50}$$

If we remember the double-angle formulae,

$$\begin{aligned}\sin 2\hat{\omega} &= \frac{2\tan \hat{\omega}}{1 + \tan^2 \hat{\omega}} \\ \cos 2\hat{\omega} &= \frac{1 - \tan^2 \hat{\omega}}{1 + \tan^2 \hat{\omega}}\end{aligned}$$

and substitute them into equation (3-50), we get

$$\begin{aligned}\theta(\hat{\omega}) &= \arctan\left(\frac{-2\tan \hat{\omega}}{2}\right) \\ &= \arctan(-\tan \hat{\omega}) \\ &= \begin{cases} -\hat{\omega} & 0 \le \hat{\omega} < \pi/2 \\ \pi - \hat{\omega} & \pi/2 < \hat{\omega} \le \pi \end{cases}\end{aligned}$$

In other words *both* the magnitude and phase responses are functions of $\hat{\omega}$ — they change the input in a frequency-dependent way. According to equation (3-40), the two zeros of the transfer function are at:

$$z_{1,2} = \frac{\pm\sqrt{-4}}{2} = \pm\frac{2j}{2} = \pm j$$

which are a pair of complex zeros on the imaginary axis. Using polar coordinates they are

$$z_{1,2} = e^{\pm j\pi/2} \tag{3-51}$$

with $r = 1$. The final result for the transfer function, written in polar form, is:

$$\mathcal{H}(\hat{\omega}) = |\mathcal{H}(\hat{\omega})|e^{j\theta(\hat{\omega})} = 2\cos \hat{\omega} e^{-j\hat{\omega}} \tag{3-52}$$

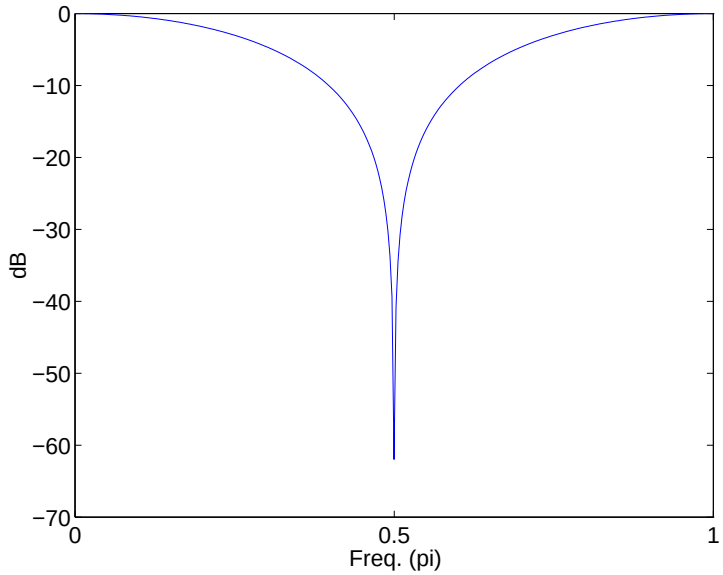


Figure 3.11: Magnitude response of two time delay feedforward filter, with $b_1 = 0, b_2 = 1$ in (3-53). Its zeros are at $e^{\pm j\pi/2}$.

We can illustrate the effect of this filter in a simple manner by examining the response when the input is a phasor (a single frequency, $x[n] = e^{jn\hat{\omega}}$):

$$y[n] = \mathcal{H}(\hat{\omega})x[n] = 2\cos\hat{\omega}e^{j(n-1)\hat{\omega}} \tag{3-53}$$

We can see that the effect of the filter on the input signal is to delay it by one sampling interval (from the $(n-1)$ term in the exponential) and to multiply it by $2\cos\hat{\omega}$. Notice that the delay is independent of its frequency. When all the frequency components of a signal are delayed by an equal amount, say the filter has *no phase distortion*.

The magnitude response and phase response for this special case are shown in figure 3.11 and figure 3.12.

Self-Test Exercises

See A.3 #9–10 for answers.

1. Prove $|z^2| = 1$ in equation (3-38).

2. Starting with the factored magnitude response in equation (3-41), derive expressions for b_1 and b_2 in terms of z_1 and z_2.

3.2.7 Implementing Digital Filters

Implementing feedforward digital filters is really quite straightforward. Let's first look at how we would implement the two time delay feedforward filter $y[n] = x[n] + b_1x[n-1] + b_2x[n-2]$,

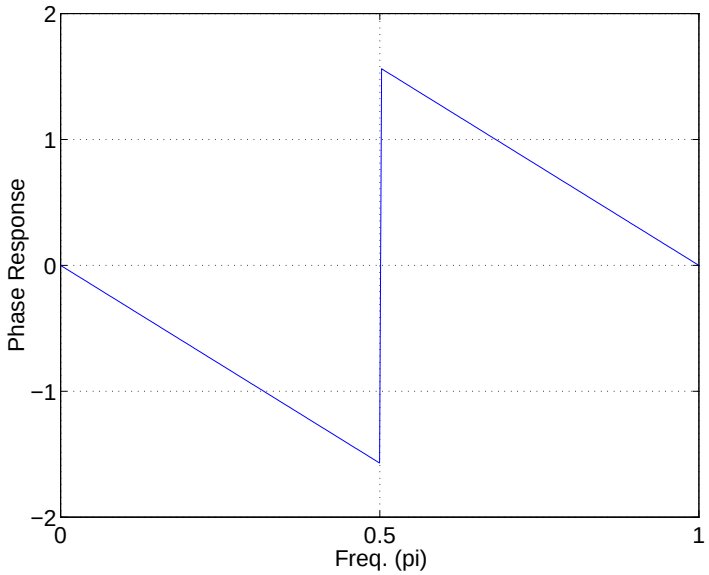


Figure 3.12: Phase response of the same frequency response of figure 3.11.

when $b_1 = 0, b_2 = 1$, in MATLAB. I'll present segments of a MATLAB script to do the computation and plot some results with my comments on it interspersed below:

```
% exp_ff_filter.m
% feedforward filter : y_t = x_t + b1x_t + b2x_t
% when a1=0, a2=1;

clf;
clear

% --- Generate an input signal ---
Fs=100;                 % Sampling frequency Fs, samples/second
N=100;                  % make N samples
Ts = 1/Fs;              % time interval, seconds/sample
t = (1:N)*Ts;           % discrete time axis (sample times, in seconds)
f1 = 5;                 % input signal's frequency
f2 = 25;
b1 = 0;                 % filter coefficients
b2 = 1;
x=sin(2*pi*t*f1)+sin(2*pi*t*f2);      % input signal
```

At this point, I've generated a discrete signal `x` by first producing a vector `t` containing all the time points at which the sampling should occur (100 samples at 100Hz = 1 second). The signal has two frequency components: one at 5Hz and one at 25Hz (both well below the Nyquist frequency).

```
% --- Filtering ---
```

```
y = zeros(size(x));
y(3:N) = x(3:N) + b1*x(2:N-1) + b2*x(1:N-2);
y = real(y);
```

In this particular example, I have the twin luxuries of taking advantage of MATLAB's built-in vector operations and being able to hold the entire input signal in a vector. I initialized `y` to zero to allocate memory at one time (MATLAB will automatically reallocate memory to expand or contract vectors and matrices, but that can be very time consuming). Because there is a two time step delay, I can't compute a value for `y(1)` or `y(2)`, so those stay zero. Since I'm only going to plot the magnitude response, I only keep the real part of `y`.

```
% --- plot magnitude of the input and output signals' spectra ---
spx=fft(x,512);          % original signal's fft
spy=fft(y,512);
fstep=(Fs/2)/256;        % frequency step
f=(0:255)*fstep;         % frequency axis
plot(f,abs(spx(1:256)'), 'g', f,abs(spy(1:256)'), 'm');
                         %plot spx and spy in the same plot
xlabel('Freq. (Hz)', 'Fontsize', 16);
ylabel('Freq. Components', 'Fontsize', 16);
set(gca, 'Fontsize', 16);
input('Press key to continue')
```

I want to see what happened to the signal's frequency components, and so I use the built-in MATLAB function `fft()`. I'll talk about the Fourier transform and FFT in section 6.1. See the MATLAB manuals (especially the graphics guide) to see what the `plot()` and `set()` are doing. Figure 3.13 is the plot this produces.

```
% --- Plot orignal and filtered signal ---
plot(t, x, 'b', t, y, 'r' );
xlabel('Time (sec)', 'FontSize', 16);
ylabel('Time waveform', 'FontSize', 16);
set(gca, 'FontSize', 16);
```

And finally, I plot the input and output. They're shown in figure 3.14.

How would one implement this in a language like C, C++, Java, etc? Let's not worry about the plotting issue: that would certainly have to be dealt with, but it would involve either getting a graphics library, writing one's own, or using an external plotting package (ideally, one targeted at scientific and engineering applications, rather than one written for business). All the elementary vector and matrix operations provided by MATLAB can be implemented as loops. I will discuss the `fft()` function in section 6.2.5. The only remaining issue is that of keeping the entire input and output signal in memory.

In general, it is not possible to keep the entire input and output signal in memory. In fact, many (if not most) digital signal processing applications involve real-time processing,

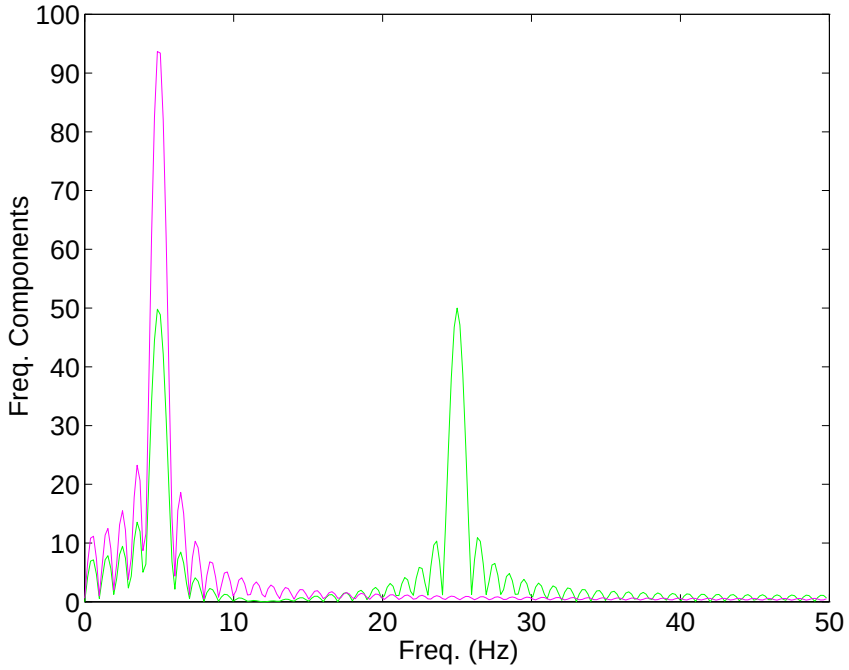


Figure 3.13: Figure 3.14's spectrum.

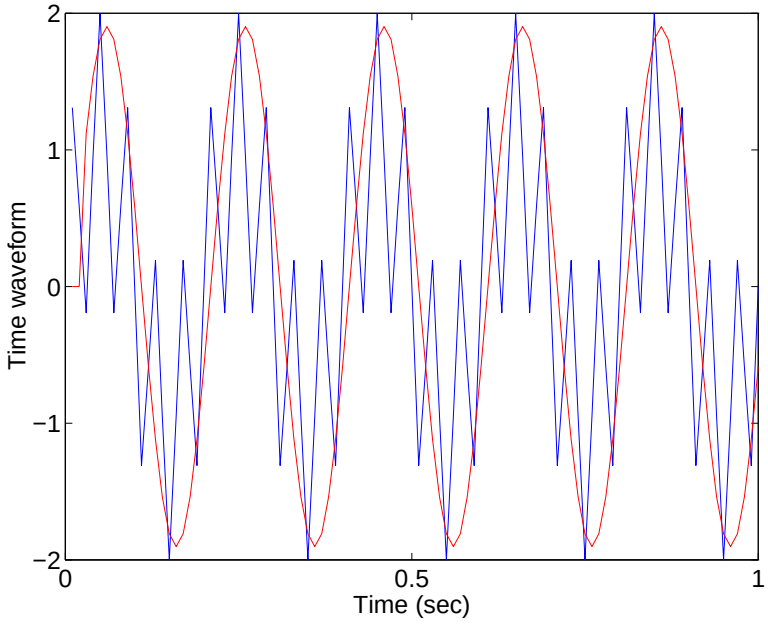


Figure 3.14: Two frequency component sine wave and its filtered version, using the feedforward filter $y[n] = x[n] + x[n-2]$, $f_1 = 5Hz$ and $f_2 = 25Hz$. Zeros are at $z_0 = 0.99e^{\pm j\pi/2}$.

so the computer system really needs to be viewed as just a stage in a processing pipeline, with the input flowing in and the output flowing out (just like the block diagrams used to illustrate filters). We can therefore only devote a relatively small amount of memory to hold the part of the input and output needed for the current computation. If n delayed samples of the input are needed to compute each output value, we will need storage for $n+1$ input samples (the current one plus the delayed samples). For a feedforward filter, there is no need to save output values; they can be sent out as soon as they are computed.

What abstract data type (ADT) should hold the delayed inputs (answer in A.3 #11)? As each input sample comes in, it displaces the oldest input sample from our buffer of $n+1$ values. This is clearly a FIFO ADT — a *queue*. As you should remember, a queue is very efficiently and simply implemented in an array, especially when the queue size is fixed. Furthermore, you should remember how to treat the array as though it were circular using indexing modulo the queue size.

3.3 Problems

1. In Example 4, we used a large number of trigonometric identities to go from the filter's frequency response in (3-47) to the results in (3-52) and (3-53) for the special case of $b_1 = 0$ and $b_2 = 1$. There is an easier way. First, substitute the values of $b_{1,2}$ into (3-47), and then use the fact that we can always factor a real number into a product of complex conjugate numbers, i.e., $1 = e^{j\hat{\omega}}e^{-j\hat{\omega}}$. A little bit of algebra and Euler's formula should allow you to derive the results (3-52) and (3-53) of Example 4 much easier.

2. For the filter whose magnitude response is described in (3-41), plot the location of a pair of complex conjugate zeros at $r = 0.8$ and $\hat{\omega}_0 = \pm\pi/4$. Using MATLAB, compute and plot the filter's magnitude response. Submit your code with your figures.

3. In the self-test exercise on page 51, two filters with transfer functions $H_1(z) = b_0 + b_1 z^{-1}$ and $H_2(z) = b'_0 + b'_1 z^{-1}$ were connected in series, and it was shown that they could be connected in either order to produce the same composite effect (the same overall transfer function). Redo this exercise using the defining equations for the two filters, i.e., $y_1[n] = F_1(x[n])$ for the filter with transfer function $H_1(z)$ and $y_2[n] = F_2(x[n])$ for the filter with transfer function $H_2(z)$. In other words, show that $F_2(F_1(x[n])) = F_1(F_2(x[n]))$.

4. Use MATLAB to plot the results of filtering the following signals with the filter: $y[n] = x[n] + x[n-1]$. Submit your code with your results.

   (a) $x[n] = n$

   (b) $x[n] = \sin(n\pi/100)$

   (c) $x[n] = \begin{cases} +1 & n \bmod 5 \text{ even} \\ -1 & n \bmod 5 \text{ odd} \end{cases}$

5. Check the following factorization: $z^2 - z + 1 = (z - e^{j\pi/3})(z - e^{-j\pi/3})$

3.4 Further Reading

- James H McClellan, Ronald W. Schafer, and Mark A. Yoder, *DSP First: A Multimedia Approach*, Prentice Hall, 1998, ch. 5 (§5.1–5.3), 6 (§6.1, 6.4–6.7).

4 The Z-Transform and Convolution

Two signal processing tools — the *z transform* and *convolution* — are introduced in this chapter. These operations play important roles in the analysis of discrete-time signals (which is what we do in the computer). We shall see that they are related — the convolution of two time-domain signals (which is what we do when we filter a signal) is equivalent to multiplication of their corresponding z-transforms. This is one example of how these representations can greatly simplify computation.

After studying this chapter, you should be able to understand what the z-transform and convolution are, and how to implement them. You should understand the differences between them and the other transforms: Fourier series, Fourier transform, and discrete Fourier transform. You will enrich your knowledge of filter transfer functions with its time domain representation: its *impulse response*.

4.1 Domains

Up to this point, we have covered the Fourier series representation of a signal as a weighted sum of sinusoids. In effect, the Fourier series *transforms* a finite and periodic, continuous signal in the time *domain* into an infinite, discrete spectrum in the frequency *domain*. When we use the term *domain*, we merely mean a particular way of looking at a signal. In this case, we have two different ways of thinking about our signals: as functions of time or as functions of frequency. These are equivalent, in the sense that we can convert the signal's representation back and forth between the two domains without loss of information (neglecting matters such as roundoff error).

In future chapters, we will cover two other transforms:

Fourier transform transforms an infinite, continuous signal in the time domain into an infinite, continuous spectrum in the frequency domain.

Discrete Fourier transform transforms a finite, discrete signal in the time domain into a finite, discrete spectrum in the frequency domain.

Here, however, we will learn about the *z-transform*, which converts an infinite, discrete signal in the time domain into a finite, continuous spectrum in the frequency domain. The z-transform fills the last combination among "finite vs. infinite; continuous vs. discrete".

Table 4.1: Summary of frequency transforms.

Transform	Time Domain	Frequency Domain
Fourier Series	Finite, Continuous	Infinite, Discrete
Fourier Transform	Infinite, Continuous	Infinite, Continuous
Discrete Fourier Transform	Finite, Discrete	Finite, Discrete
Z-Transform	Infinite, Discrete	Finite, Continuous

Table 4.1 summarizes all four transforms. As you can see, continuous versus discrete in the time domain transforms to infinite versus finite in the frequency domain, while finite versus infinite in the time domain transforms to discrete versus continuous in the frequency domain.

4.2 The z-transform

The *z-transform* of a discrete time signal $x[n]$, $n = 0, \pm 1, \pm 2, \ldots, \pm\infty$ is defined as the power series

$$X(z) \equiv \sum_{k=-\infty}^{\infty} x[k] z^{-k} \tag{4-1}$$

where z is a continuous complex variable. It transforms the time domain, infinite, discrete sequence into its complex plane representation $X(z)$. Since the z-transform is an infinite power series, it exists only for those values of z for which this series converges. The *region of convergence* (ROC) of $X(z)$ is the set of all values of z for which $X(z)$ has a finite value. We consider the z-transform to be a transform between the time domain and the frequency domain because we can substitute $z = e^{j\hat{\omega}}$ into equation (4-1) to get the signal's (finite and continuous) frequency content $\mathcal{X}(\hat{\omega})$ — just as we previously made the same substitution to derive a feedforward filter's frequency response from its transfer function.

The relationship between $x[n]$ and $X(z)$ can be indicated by the *transform pair*

$$\underbrace{x[n]}_{\substack{\text{function of} \\ \text{sample \#}}} \overset{\mathbf{Z}}{\longleftrightarrow} \underbrace{X(z)}_{\substack{\text{function of} \\ \text{complex } z}} \tag{4-2}$$

Self-Test Exercises

See A.4 #1–2 for answers.

1. Determine the z-transform for the sequence $x[n] = \{1, 2, 5, 7, 0, 1\}$, $n = 0, 1, 2, 3, 4, 5$

2. Determine the z-transform of the sequence $x[n] = \{1, 2, 5, 7, 0, 1\}$, $n = -2, -1, 0, 1, 2, 3$

4.2.1 Example: z-transform of an impulse

The *unit impulse* or *unit sample* signal is the δ function,

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} \tag{4-3}$$

It has value of zero for every sample except $n = 0$, for which it has a value of one. Substituting this signal into (4-1) to get its z-transform, we have

$$\begin{aligned} \Delta(z) &= \sum_{k=-\infty}^{\infty} \delta[k] z^{-k} \\ &= 1z^{-0} = 1 \end{aligned} \tag{4-4}$$

that is

$$\delta[n] \overset{\mathbf{Z}}{\longleftrightarrow} 1 \tag{4-5}$$

Since the frequency content of the signal is the magnitude of its z-transform on the unit circle in the z-plane,

$$|\mathcal{D}(\hat{\omega})| = |\Delta(e^{j\hat{\omega}})| = 1 \tag{4-6}$$

This tells us that the frequency content is the same for all frequencies: an impulse has a *flat spectrum*.

What about time shifted impulses,

$$\delta[n - n_0] = \begin{cases} 1 & n = n_0 \\ 0 & n \neq n_0 \end{cases}, \quad n_0 > 0 \tag{4-7}$$

$$\delta[n + n_0] = \begin{cases} 1 & n = -n_0 \\ 0 & n \neq -n_0 \end{cases}, \quad n_0 > 0 \tag{4-8}$$

In these cases the nonzero value is not at sample zero, but at samples n_0 or $-n_0$. We can compute the z-transform as before,

$$\begin{aligned} \Delta(z) &= \sum_{k=-\infty}^{\infty} \delta[k - n_0] z^{-k} \\ &= 1z^{-n_0} = z^{-n_0} = \frac{1}{z^{n_0}} \end{aligned} \tag{4-9}$$

for $\delta[n-n_0]$. The z-transform for this shifted unit impulse has one value, z^{-n_0}, for any $z \neq 0$.

$$\delta[n - n_0] \overset{\mathbf{Z}}{\longleftrightarrow} \frac{1}{z^{n_0}}, \quad n_0 > 0 \tag{4-10}$$

Its frequency content is also one, just like $\delta[n]$ (remember that we compute the spectrum for values of z on the unit circle),

$$|\mathcal{D}(\hat{\omega})| = |\Delta(e^{j\hat{\omega}})| = |e^{-jn_0\hat{\omega}}| = 1 \tag{4-11}$$

This is not surprising at all, because it is, after all, just a time-shifted version of $\delta[n]$. I leave $\delta[n + n_0]$ as a self-test exercise.

Self-Test Exercises

See A.4 #3–4 for answers.

1. Sketch equation (4-6).
2. Compute the z-transform and frequency content for the signal $\delta[n + n_0]$.

4.2.2 Example: z-transform of exponential signal

An exponential signal is defined as

$$x[n] = \begin{cases} \alpha^n & n \geq 0 \\ 0 & n < 0 \end{cases} \tag{4-12}$$

where α can be any real or complex value less than one. The signal consists of an infinite number of samples. The z-transform of this signal is

$$\begin{aligned} X(z) &= \sum_{k=-\infty}^{\infty} x[k] z^{-k} \\ &= \sum_{k=0}^{\infty} \alpha^k z^{-k} \\ &= \sum_{k=0}^{\infty} (\alpha z^{-1})^k \end{aligned} \tag{4-13}$$

This is an infinite *geometric series*: a sum in which each successive term is the previous term times some (unchanging) expression (i.e., the ratio of successive terms is constant). The ratio of two successive terms in a geometric series like equation (4-13) is called its *common ratio*, which in this case is αz^{-1}. We can show this by rewriting equation (4-13) as $X(z) = 1 + \alpha z^{-1} + \alpha^2 z^{-2} + \cdots$. Rewriting the $i + 1^{\text{st}}$ element of this series as a recurrence relation (in terms of the i^{th}), we get $X(z)_{i+1} = X(z)_i \alpha z^{-1}$, which shows the common ratio.

If we have a geometric series in which successive terms b_i and b_{i+1} have the common ratio r (i.e., $b_{i+1}/b_i = r$), then any term in the series can be expressed in terms of the first term as

$$b_i = b_0 r^i$$

So, a geometric series can be expressed as a sum of these, $b_0 + b_0 r + b_0 r^2 + \cdots$. We can factor out the zeroth term, leaving us with the task of simplifying $1 + r + r^2 + \cdots$. Multiplying this sum by $(1 - r)/(1 - r)$, we obtain

$$\begin{aligned} (1 + r + r^2 + \cdots)\frac{1 - r}{1 - r} &= \frac{1 + r + \cdots - r - r^2 - \cdots}{1 - r} \\ &= \frac{1 - r^N}{1 - r} \end{aligned} \tag{4-14}$$

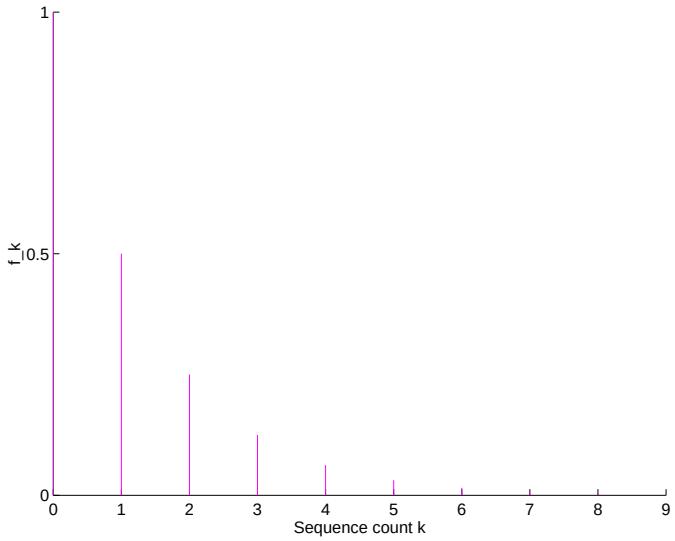


Figure 4.1: The exponential signal $x[n] = (1/2)^n, n = 0, 1, 2, \ldots$.

In this case, $|r| < 1$ so $r^N \to 0$ as $N \to \infty$:

$$1 + r + r^2 + r^3 + \ldots = \frac{1}{1-r}, \quad \text{if } |r| < 1 \tag{4-15}$$

Consequently, for $|r| = |\alpha z^{-1}| < 1$ or $|z| > |\alpha|$, $X(z)$ converges to

$$X(z) = \frac{1}{1 - \alpha z^{-1}}, \quad |z| > |\alpha| \tag{4-16}$$

In the z-plane, $|z| > |\alpha|$ refers to any z that is outside of the radius $|\alpha|$ circle. We see that in this case, the z-transform provides a compact alternative representation of the signal $x[n]$.

Let's check out some special cases:

When α is a real number, say $\alpha = 1/2$: The discrete signal in this case is

$$x[n] = \begin{cases} \left(\frac{1}{2}\right)^n & n \geq 0 \\ 0 & n < 0 \end{cases} \tag{4-17}$$

or

$$x[n] = \left\{1, \frac{1}{2}, \left(\frac{1}{2}\right)^2, \left(\frac{1}{2}\right)^3, \ldots, \right\} \tag{4-18}$$

Figure 4.1 shows the graph of the signal $x[n]$.

Replacing α with $1/2$ in (4-16), its z-transform is expressed as

$$X(z) = \frac{1}{1 - \frac{1}{2}z^{-1}}, \quad |z| > \frac{1}{2} \tag{4-19}$$

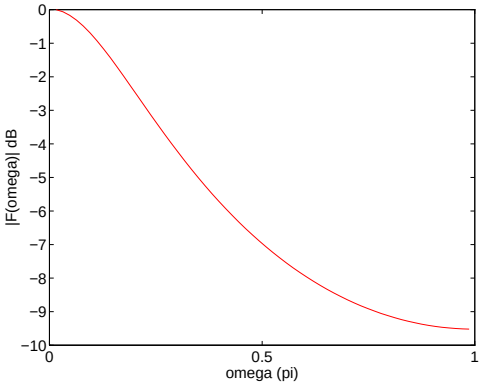


Figure 4.2: Frequency content of the signal shown in figure 4.1.

As you should be familiar with now, the frequency content of this signal is the magnitude of its z-transform on the unit circle $z = e^{j\hat{\omega}}$ in the z-plane, which is

$$|\mathcal{X}(\hat{\omega})| = |X(e^{j\hat{\omega}})| = \left| \frac{1}{1 - \frac{1}{2}e^{-j\hat{\omega}}} \right| \tag{4-20}$$

Figure 4.2 is the plot of $|\mathcal{X}(\hat{\omega})|$ versus frequency ω. The amplitude decreases along increasing frequency. Its peak is at zero frequency, which is called *DC* (which literally means "direct current," implying the constant — actually mean — component of the signal).

When $\alpha = 1$: For $\alpha = 1$, the sequence becomes

$$x[n] = u[n] = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases} \tag{4-21}$$

or

$$u[n] = \{1, 1, 1, \ldots\}, \quad k \geq 0 \tag{4-22}$$

This is a discrete time, infinite duration *unit step* signal. Notice the difference between the unit step signal and the unit impulse signal. The latter only has one nonzero value at one particular time, the former has value one for all time after some particular time. By analogy with $\delta[n]$, a unit step occurring at sample k is called $u[n-k]$. Substituting $\alpha = 1$ into (4-16) we have

$$U(z) = \frac{1}{1 - z^{-1}}, \quad |z| > 1 \tag{4-23}$$

We can see that the pole (a zero in the denominator; you'll learn more about this in Chapter 5) is at $z = 1$, where the z-transform has an infinite value.

Let's evaluate the frequency content of the unit step signal. If we evaluate $\mathcal{U}(\hat{\omega})$ on the

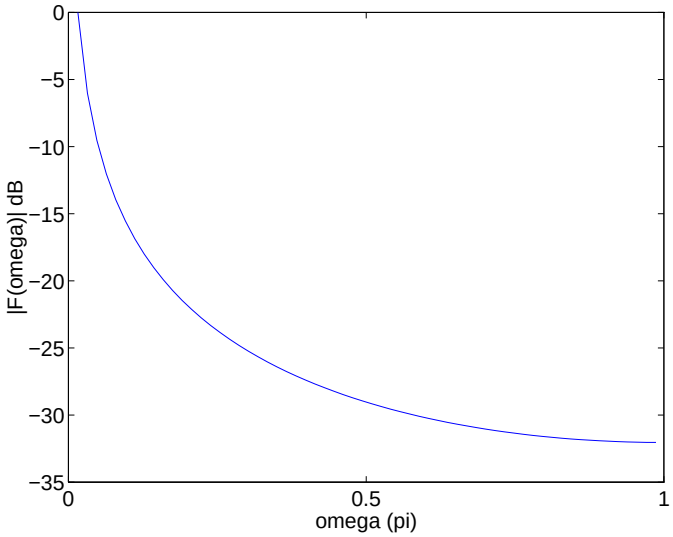


Figure 4.3: Frequency content of the unit step signal.

unit circle (except at $z = 1$), we obtain

$$\begin{aligned}\mathcal{U}(\hat{\omega}) = U(e^{j\hat{\omega}}) &= \frac{1}{1 - e^{-j\hat{\omega}}} \frac{e^{j\hat{\omega}/2}}{e^{j\hat{\omega}/2}} \\ &= \frac{e^{j\hat{\omega}/2}}{e^{j\hat{\omega}/2} - e^{-j\hat{\omega}/2}} \\ &= \frac{e^{j\hat{\omega}/2}}{2j \sin \hat{\omega}/2} \\ &= \frac{e^{j(\hat{\omega}/2 - \pi/2)}}{2 \sin \hat{\omega}/2}, \quad \hat{\omega} \neq 2\pi k, k = 0, 1, \ldots \end{aligned} \tag{4-24}$$

because $-j = e^{-j\pi/2}$ (see the self-test exercises). Hence, the presence of a pole (a zero in the denominator) at $z = 1$ (that is, at $\hat{\omega} = 0$) creates a problem only when we want to compute $|\mathcal{U}(\hat{\omega})|$ at $\hat{\omega} = 0$, because $|\mathcal{U}(\hat{\omega})| \to \infty$ as $\hat{\omega} \to 0$. For any other value of $\hat{\omega}$, $|\mathcal{U}(\hat{\omega})|$ is finite.

Figure 4.3 shows a plot of $|\mathcal{U}(\hat{\omega})|$ vs. $\hat{\omega}$. Since the signal is a unit step, and so has a constant value from zero onwards, we might expect the signal to have zero frequency components at all frequencies except at $\hat{\omega} = 0$, but that is not the case. The reason is that the signal is not a constant for all $-\infty < n < \infty$. Instead, it is turned on at $n = 0$. This *abrupt jump* creates all the frequency components existing in the range $0 < \hat{\omega} \leq \pi$. Generally, *all* signals which start at a finite time will have nonzero frequency components everywhere in the frequency axis from zero up to the Nyquist frequency. All such signals can be considered to be the product of some infinite signal with a unit step; we will see the effect of this on their spectrum when we explore convolution in section 4.3.

Optional: When α is a complex number, $\alpha = Re^{j\theta}$: When $\alpha = Re^{j\theta}$, equations (4-12) and (4-16) become

$$x[n] = \begin{cases} R^k e^{jn\theta} & n \geq 0 \\ 0 & n < 0 \end{cases} \tag{4-25}$$

$$X(z) = \frac{1}{1 - Re^{j\theta} z^{-1}}, \quad |z| > |R| \tag{4-26}$$

Obviously $z = Re^{j\theta}$ is the pole (a zero in the denominator; you'll learn more about this in Chapter 5). Equation (4-26) can be broken into real and imaginary parts using Euler's formula:

$$\begin{aligned} X(z) &= \frac{1}{1 - Re^{j\theta} z^{-1}} \\ &= \frac{1}{1 - R(\cos\theta + j\sin\theta) z^{-1}} \\ &= \frac{1}{1 - R\cos\theta z^{-1} - jR\sin\theta z^{-1}} \\ &= \frac{1 - R\cos\theta z^{-1} + jR\sin\theta z^{-1}}{[1 - R\cos\theta z^{-1}]^2 - [jR\sin\theta z^{-1}]^2} \\ &= \frac{1 - R\cos\theta z^{-1} + jR\sin\theta z^{-1}}{1 - 2R\cos\theta z^{-1} + R^2 z^{-2}} \\ &= \frac{1 - R\cos\theta z^{-1}}{1 - 2R\cos\theta z^{-1} + R^2 z^{-2}} + j\frac{R\sin\theta z^{-1}}{1 - 2R\cos\theta z^{-1} + R^2 z^{-2}} \end{aligned} \tag{4-27}$$

We break the signal into two parts, too:

$$\mathrm{Re}\{x[n]\} = R^n \cos(k\theta), \quad n \geq 0 \tag{4-28}$$

$$\mathrm{Im}\{x[n]\} = R^n \sin(k\theta), \quad n \geq 0 \tag{4-29}$$

From real part we get

$$R^n \cos(n\theta) \overset{\mathbf{Z}}{\longleftrightarrow} \frac{1 - R\cos\theta z^{-1}}{1 - 2R\cos\theta z^{-1} + R^2 z^{-2}} \tag{4-30}$$

and from imaginary parts we have

$$R^n \sin(n\theta) \overset{\mathbf{Z}}{\longleftrightarrow} \frac{R\sin\theta z^{-1}}{1 - 2R\cos\theta z^{-1} + R^2 z^{-2}} \tag{4-31}$$

When $R < 1$, $R^n \cos(n\theta)$ and $R^n \sin(n\theta)$ are damped cosine and sine waves.

Self-Test Exercises

See A.4 #5–7 for answers.

1. What is the derivative of $u[n-k]$ (the unit step at time step k)?
2. Show that $e^{j\hat{\omega}/2} - e^{-j\hat{\omega}/2} = 2j\sin\hat{\omega}/2$.
3. Prove that $e^{-j\pi/2} = -j$.

4.3 Convolution

Convolution is an important operation for implementing digital filters. Let's first define what convolution is. For two infinite, discrete signals $x[n]$ and $h[n]$, the *convolution* $y[n]$ of them at sample n is defined as

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] \tag{4-32}$$

The notation for convolution is "$*$", so this can be written as

$$Y = X * H \tag{4-33}$$

where Y, X, and H are the signals with samples at n being $y[n]$, $x[n]$, and $h[n]$.

Let's consider the case when both x and h both start at zero. This is equivalent to saying that both have values of zero before that sample. So, $h[k] = 0$ when $k < 0$ and $x[n-k] = 0$ when $n - k < 0$. Equation (4-32) becomes

$$y[n] = \sum_{k=0}^{n} h[k]x[n-k] \tag{4-34}$$

Actually, this is a more realistic situation than k's summation from $-\infty$ to ∞.

We expand the summation in (4-34) as

$$\begin{aligned} y[n] = h[0]x[n] + h[1]x[n-1] + h[2]x[n-2] + \ldots + h[k]x[n-k] + \ldots \\ + h[n-2]x[2] + h[n-1]x[1] + h[n]x[0] \end{aligned} \tag{4-35}$$

Using this formulation, you may show that the convolution $X*H$ has the following properties:

1. Commutative: $X * H = H * X$
2. Distributive : $X * (H_1 + H_2) = X * H_1 + X * H_2$
3. Associative : $(X * H) * G = X * (H * G)$

Obviously, $x * \vec{0} = \vec{0} * x = 0$ (where $\vec{0}$ is a vector of all zeros). Then how about $\vec{1} * \vec{1}$ (where $\vec{1}$ is a vector of all ones, in this case $\vec{1} = u[n]$; see the self-test exercise)?

4.3.1 Example of Convolution

Determine the convolution $e^n * e^n$, $n = 0, 1, 2, \ldots,$. Using (4-34),

$$\begin{aligned} e^n * e^n &= \sum_{k=0}^{n} e^k e^{n-k} \\ &= \sum_{k=0}^{n} e^n = e^n \sum_{k=0}^{n} 1 = ne^n \end{aligned} \tag{4-36}$$

Self-Test Exercises

See A.4 #8–9 for answers.

1. Determine if $u[n] * H \neq H$ is true, where $h[n] = n$, $n = 0, 1, 2, \ldots$ (a *ramp*).
2. Compute $u[n] * u[n]$.

4.3.2 Implementing Convolution

From observation of equation (4-35), we know that for a fixed sample n, $y[n]$ can be computed by the term-by-term multiplication of the sequence

$$\{h[0], h[1], h[2], \ldots, h[k], \ldots, h[n-2], h[n-1], h[n]\} \tag{4-37}$$

and the time-reversed sequence

$$\{x[n], x[n-1], x[n-2], \ldots, x[n-k], \ldots, x[2], x[1], x[0]\} \tag{4-38}$$

We just multiply the corresponding terms (for example, $h[k]x[n-k]$), then add these products. This produces the convolution for one sample n. Remember, however, that the output $y[n]$ is also a sequence, $n = 0, 1, 2, \ldots$. We need to repeat this process for all $\{n\}$ to get the full sequence, as shown in algorithm 4.1.

Algorithm 4.1 Discrete convolution.

Require: $h[n]$ is a finite, discrete signal, $n = 0, 1, 2, \ldots$
Require: $x[n]$ is a finite, discrete signal, $n = 0, 1, 2, \ldots$
Ensure: $y[n]$ is the convolution $X * H$, $n = 0, 1, 2, \ldots$
  for $n = 0, 1, 2, \ldots$ **do**
    Reverse $x[k]$ to produce $x'[k] = x[n-k]$, $k = 0, 1, 2, \ldots, n$
    $s[k] = h[k]x'[k]$
    $y[n] = \sum_{k=0}^{n} s[k]$
  end for

A more-or-less direct implementation of this algorithm in C is:

```
/* Convolution of two vectors

   Input: vectors x and h of lengths nx and nh (nh < nx)
   Output: vector y of length nx + nh - 1 (storage already allocated)
 */
void convolve(int x[], unsigned nx, int h[], unsigned nh, int y[])
{
   for (unsigned n=0; n<nx+nh-1; n++) {
      y[n] = 0;
```

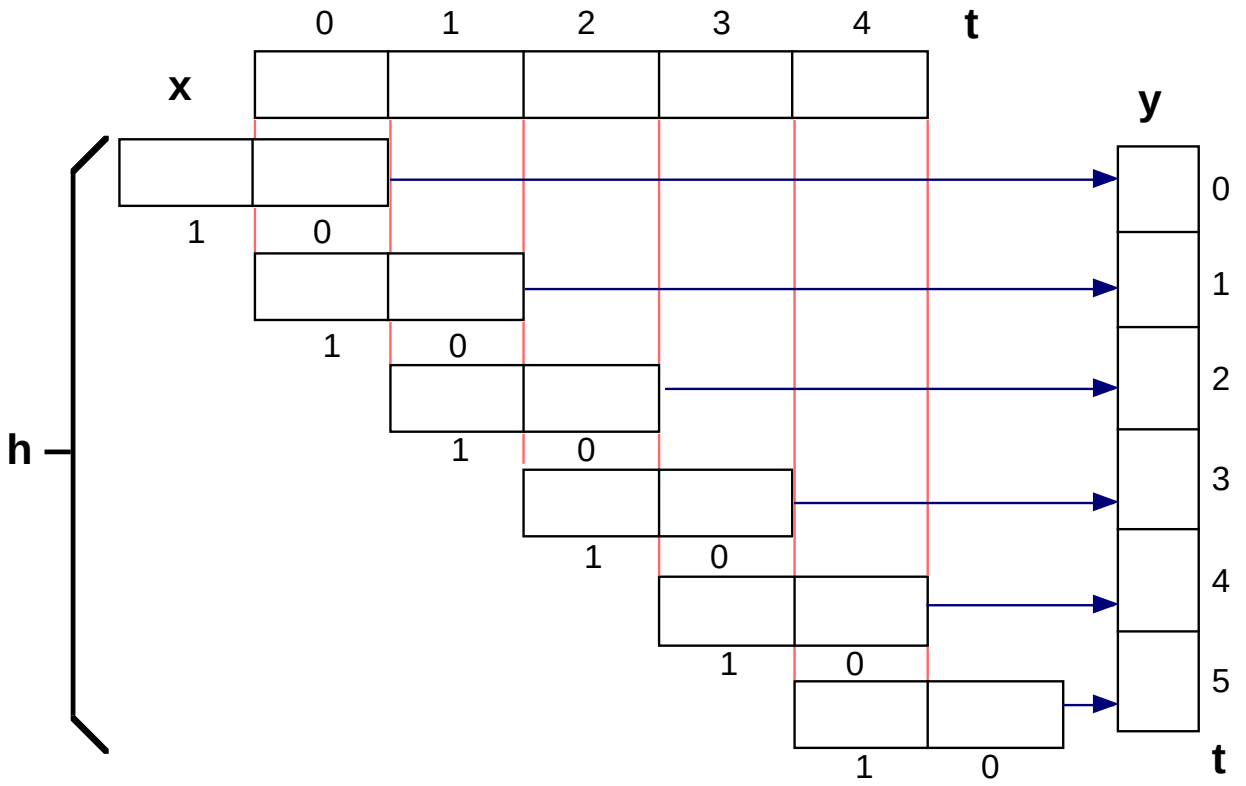


Figure 4.4: Example of convolution function execution for $n_x = 5$ and $n_h = 2$.

```
      for (unsigned k=max(0,n-nh+1); k<=min(nx-1,n); k++)
         y[n] += x[k] * h[n-k];
   }
}
```

You'll notice I did something nasty here: I reversed the identity of x and h! Of course, this is perfectly OK, given the commutative property of convolution. I did this because I am assuming that h (the shorter vector) is a property of the filter (we will see about this later) and, in my opinion, it is easier to think about reversing it than reversing the signal x. If for no other reason, this makes sense because h has a shorter length.

The implementation is for finite-length signals, rather than the infinite ones we've been discussing. It assumes that h is shorter than x ($n_h < n_x$). This h input is often called the convolution *kernel* because, as we shall see, h represents the action of our signal processing system while x is the actual input signal. Notice that, for small $n < n_h - 1$, not all of h is used. This is equivalent to multiplying the unused elements of h against the zero values of $x[k]$, $k < 0$. Similarly, for large $n > nx - 1$, part of h is also unused — multiplied against the zero values of $x[k]$, $k > n_x - 1$. Figure 4.4 illustrates function execution for a signal X of length 5 and a kernel H of length 2. This is one way to deal with the *boundary conditions* associated with the convolution: what to do at the ends of the signal X. In general, there are three ways of dealing with these boundary conditions:

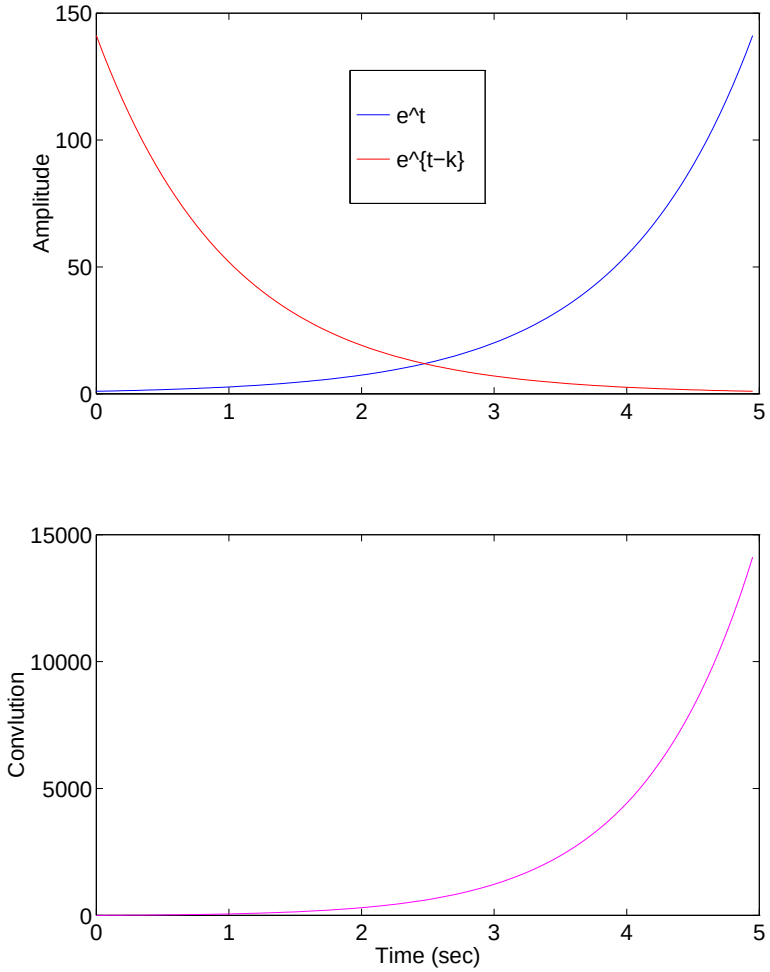


Figure 4.5: Convolution of $e^n * e^n$. Top blue line is e^n, top red line is time reversed version of e^n, the bottom plot is the result of convolution.

1. "Pad" X with zeros past its ends. In effect, this is what was done in the code above.
2. "Reflect" X by copying element k to index $-k$. This is sometimes done in image processing operations.
3. "Truncate" the convolution at the ends of X. This means that n would cover the range $n_h - 1 \leq n \leq n_x$ and Y would be shorter than X.

MATLAB has the built-in convolution function `conv`, which takes two vectors as inputs and outputs the convolution result with length equal to one less than the sum of the two input vector lengths. You can use "`help conv`" to get more information (how does `conv` deal with boundary conditions [answer in A.4 #10]?)

Let's look at the convolution of $X * H = e^n * e^n$ again. In this case, X and H are the same function. In figure 4.5 (top), X is shown as a blue curve and the time-reversed H is shown as a red curve. The convolution result is in the bottom graph.

Table 4.2: Some properties of the z-transform.

Property	Time Domain, $Z^{-1}\{\cdot\}$	z-Domain, $Z\{\cdot\}$
Linearity	$a_1x[n] + a_2y[n]$	$a_1X(z) + a_2Y(z)$
Time shift	$x[n-k]$	$z^{-k}X(z)$
Scaling in the z-domain	$a^n x[n]$	$X(a^{-1}z)$
Time reversal	$x[-n]$	$X(z^{-1})$
Differentiation in the z-domain	$nx[n]$	$-z\frac{\mathrm{d}X(z)}{\mathrm{d}z}$
Convolution	$x[n] * y[n]$	$X(z)Y(z)$

Self-Test Exercises

See A.4 #11 for the answer.

1. Use MATLAB to compute the convolution $e^{-n} * e^{-n}$ and plot the result.

4.4 Properties of the Z-Transform

The z-transform is a very powerful signal processing tool because it has some very important properties. I list some of these properties in table 4.2, where the time-domain signals $x[k]$ and $y[k]$ have z-transforms of $X(z)$ and $Y(z)$.

Knowing these properties can be very convenient. For example, the z-transform of a signal shifted (delayed) by k samples, $x[n-k]$, is $z^{-k}X(z)$; this is our familiar z (delay) operator. You can also see that the convolution property of the z-transform means that convolution in the time domain is multiplication in the z-domain. So, if we have the z-transform of two signals, it is much easier to perform convolution. Later on, you will find out that this is very important in filtering. Let's prove this property:

From (4-34) a convolution of $x[n]$ and $h[n]$ is defined as:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] \tag{4-39}$$

The z-transform of $y[n]$ is

$$\begin{aligned} Y(z) &= \sum_{n=-\infty}^{\infty} y[n]z^{-n} \\ &= \sum_{n=-\infty}^{\infty} \left(\sum_{k=-\infty}^{\infty} h[k]x[n-k] \right) z^{-n} \end{aligned} \tag{4-40}$$

Interchanging the order of the summations (which is equivalent to factoring out the $h[k]$ and distributing the z^{-n} over the inner summation),

$$Y(z) = \sum_{k=-\infty}^{\infty} h[k] \left(\sum_{n=-\infty}^{\infty} x[n-k] z^{-n} \right) \tag{4-41}$$

The inner summation is merely the z-transform of $x[n]$ shifted by k samples. Applying the time shift property of the z-transform, we obtain

$$\begin{aligned} Y(z) &= \sum_{k=-\infty}^{\infty} h[k] X(z) z^{-k} \\ &= X(z) \sum_{k=-\infty}^{\infty} h[k] z^{-k} = X(z) H(z) \end{aligned} \tag{4-42}$$

Which is the product of the two z-transforms. I'll present a couple examples of using these properties.

4.4.1 Example: Time Shifting

Remember that I discussed that the z-transform of the unit impulse $\delta[n]$

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} \tag{4-43}$$

is 1 and that the z-transform of the shifted unit impulse $\delta[n-k]$ is z^{-k}? Using the time-shifting property of the z-transform, this is easy to determine:

$$\delta[n] \overset{\mathbf{Z}}{\longleftrightarrow} 1 \tag{4-44}$$

then

$$\delta[n-k] \overset{\mathbf{Z}}{\longleftrightarrow} 1 \times z^{-k} = z^{-k} \tag{4-45}$$

4.4.2 Example: Convolution

Given the signals

$$x[n] = \{1, -3, 2, 1\}, \quad n = 0, 1, 2, 3 \tag{4-46}$$

and

$$h[n] = \begin{cases} 1 & n = 0, 1 \\ 0 & n = 2, 3 \end{cases} \tag{4-47}$$

use the z-transform to compute their convolution $Y = X * H$.

According to (4-1),

$$X(z) = 1 - 3z^{-1} + 2z^{-2} + z^{-3} \tag{4-48}$$

$$H(z) = 1 + z^{-1} \tag{4-49}$$

Then using the convolution property of the z-transform, we have

$$\begin{aligned} Y(z) = X(z)H(z) &= (1 - 3z^{-1} + 2z^{-2} + z^{-3})(1 + z^{-1}) \\ &= 1 - 2z^{-1} - z^{-2} + 3z^{-3} + z^{-4} \end{aligned} \tag{4-50}$$

Now we can easily get the inverse z-transform $y[n] = x[n] * h[n]$ from the result $Y(z)$. Again from the z-transform definition (4-1),

$$y[n] = x[n] * h[n] = \{1, -2, -1, 3, 1\} \tag{4-51}$$

We can also compute the convolution directly, according to the convolution definition (4-34). There are only 4 nonzero terms in $x[n]$ and 2 in $h[n]$. For a fixed sample n, the convolution is given by

$$\begin{aligned} y[n] &= \sum_{k=0}^{n} x[k]h[n-k] \\ &= x[0]h[n] + x[1]h[n-1] + x[2]h[n-2] + x[3]h[n-3] \end{aligned} \tag{4-52}$$

Changing n gives the sequence of the convolution output as a function of time. In the following computation, I use $h[n-k] = 0$, if $n - k < 0$ and $x[k] = 0$ and $h[k] = 0$ if $k > 3$:

$$\begin{aligned} y[0] &= x[0]h[0] = 1 \\ y[1] &= x[0]h[1] + x[1]h[0] = 1 - 3 = -2 \\ y[2] &= x[0]h[2] + x[1]h[1] + x[2]h[0] = 0 - 3 + 2 = -1 \\ y[3] &= x[0]h[3] + x[1]h[2] + x[2]h[1] + x[3]h[0] = 0 + 0 + 2 + 1 = 3 \\ y[4] &= x[1]h[3] + x[2]h[2] + x[3]h[1] = 0 + 0 + 1 = 1 \\ y[n] &= 0, \quad n > 4 \end{aligned}$$

Therefore, this also gives the result

$$y[n] = x[n] * h[n] = \{1, -2, -1, 3, 1\} \tag{4-53}$$

Self-Test Exercises

See A.4 #12 for the answer.

1. Prove the scaling property of the z-transform; that is, if

$$x[n] \overset{\mathbf{Z}}{\longleftrightarrow} X(z)$$

then

$$a^n x[n] \overset{\mathbf{Z}}{\longleftrightarrow} X(a^{-1}z)$$

4.5 Impulse Response and the Transfer Function

Recall the a filter's input/output relationship is summarized by the transfer function discussed in chapter 3 (and which you'll see again in chapter 5). Let's denote the input signal as $x[n]$ and output as $y[n]$ in the time domain. You learned that the the transfer function in the z domain is $H(z)$. What is its time domain representation? We will answer this shortly; first I would like to give a name to the time domain transfer function, $h[n]$: the filter's *impulse response*. The filter's input/output relationship can be written using $h[n]$ as:

$$y[n] = x[n] * h[n] \tag{4-54}$$

This says that the output of filter results from the convolution between input signal $x[n]$ and filter's impulse response $h[n]$. From earlier in this chapter, you now know that convolution of two signals in the time domain is equivalent to multiplying their z-transforms in the z domain. So, we obtain the input/output relationship via the transfer function in the z domain as

$$Y(z) = X(z)H(z) \tag{4-55}$$

where $X(z)$ and $Y(z)$ are the z-transforms of $x[n]$ and $y[n]$ and $H(z)$ is the z-transform of $h[n]$. Actually, this $H(z)$ is just the transfer function we talked about in chapter 3. In other words, a filter's transfer function is the z-transform of its impulse response!

As a simple example of how to use the z-transform to determine a filter's transfer function from its defining equation, consider the feedforward filter:

$$y[n] = b_0 x[n] + b_1 x[n-k] \tag{4-56}$$

Applying the z-transform to both sides,

$$Y(z) = b_0 X(z) + b_1 z^{-k} X(z) \tag{4-57}$$

because of the time shift property of the z-transform. This can be rearranged to be

$$Y(z) = (b_0 + b_1 z^{-k}) X(z) \tag{4-58}$$

So the transfer function is

$$H(z) = b_0 + b_1 z^{-k} \tag{4-59}$$

and therefore, we have

$$Y(z) = H(z)X(z) \tag{4-60}$$

Applying the inverse z-transform, we can get the time domain representation of $H(z)$, or the impulse response $h[n]$. In a similar manner, we can get the output $y[n]$ from $Y(z)$. In fact, using (4-55), we found an easy way to compute a filter's response to a signal if we have already know the signal's z-transform and the filter's transfer function.

Let's see why $h[n]$ is called the impulse response. Remember the unit impulse, which is a signal that has the value one at $n = 0$ and zero otherwise, and which can be expressed as

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} \tag{4-61}$$

The impulse response of a filter is its output when a unit impulse is applied. Now we compute these two functions' convolution,

$$y[n] = \sum_{k=0}^{n} \delta[k]h[n-k] = h[n] \tag{4-62}$$

because all of the terms except $k = 0$ drop out (since $\delta[k] = 0$ for all $k \neq 0$). This tell us that the filter response $y[n]$ to a unit impulse is $h[n]$. This is why $h[n]$ is called the impulse response.

Applying the z-transform to both side of the above equation, we get

$$Y(z) = H(z) \tag{4-63}$$

Therefore, the *transfer function* can be viewed as the z-transform of the filter's *impulse response*, or its impulse response in the z domain.

If we restrict z to lie on the unit circle, $z = e^{j\hat{\omega}}$, from (4-55) we obtain

$$\mathcal{Y}(\hat{\omega}) = \mathcal{H}(\hat{\omega})\mathcal{X}(\hat{\omega}) \tag{4-64}$$

where $\mathcal{X}(\hat{\omega})$ is the signal's frequency content, $\mathcal{Y}(\hat{\omega})$ is the frequency content of the filter output and $\mathcal{H}(\hat{\omega})$ is the filter's frequency response.

4.6 Problems

1. Compute the z-transform of

$$x[n] = \begin{cases} (-1)^n & n \geq 0 \\ 0 & n < 0 \end{cases} \tag{4-65}$$

   and determine its frequency content.

2. Determine the z-transform of the signal

$$x[n] = \begin{cases} \cos \hat{\omega}_0 n & n \geq 0 \\ 0 & n < 0 \end{cases} \tag{4-66}$$

3. Find the z-transform of the signal

$$x[n] = \begin{cases} 1/n! & n \geq 0 \\ 0 & n < 0 \end{cases}$$

   Recall that $0! = 1$.

4. The impulse response of a system is $h[n] = \{1, 2, 1, -1\}$, $n = -1, 0, 1, 2$. Determine the response of the system to the input signal $x[n] = \{1, 2, 3, 1\}$, $n = 0, 1, 2, 3$.

4.7 Further Reading

- James H McClellan, Ronald W. Schafer, and Mark A. Yoder, *DSP First: A Multimedia Approach*, Prentice Hall, 1998, chapter 7 (§7.1–7.6.3).

5 Feedback Filters

5.1 Introduction

In this chapter, I will continue to introduce filters, this chapter focusing on *feedback filters*, in which previous outputs are combined with new inputs to produce new outputs. I will talk about their structure, function, and the differences between feedforward and feedback filters. We will see that sometimes it is better to combine these two kinds of filters. I will also talk a bit about digital filter design. After this chapter, you should understand important concepts like the *poles* (as compared to *zeros*) of a transfer function, *impulse response*, and *bandwidth*. You should know about features of feedback filters and special types of such filters, such as *resons*. You should be able to design simple digital filters, implement them on a computer and use them to solve some simple signal processing problems.

5.1.1 Poles

Figure 5.1 presents the block diagram of a filter with one feedforward and one feedback term; the feedback filter's signal flowgraph is shown on the right. Compared to the feedforward one on the left in the figure, we notice that instead of combining the input signal with a delayed version, here the output signal is delayed and "fed back" to be combined with the input. The feedback processing alone is expressed by

$$y[n] = x[n] + a_1 y[n-1] \tag{5-1}$$

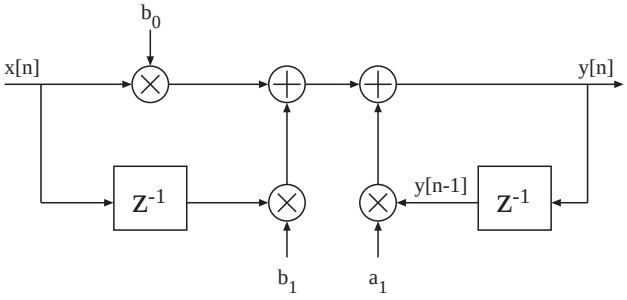


Figure 5.1: Block diagram of a simple filter with both feedforward and feedback terms.

(ignoring the factor of b_0) which is the equation for a simple feedback filter with one delayed component. $a_1 y[n-1]$ is the feedback term. You can see that the output at $n-1$ is used to compute the output at time n. The combined filter's full defining equation is

$$y[n] = b_0 x[n] + b_1 x[n-1] + a_1 y[n-1] \tag{5-2}$$

but for the moment we will concentrate on just the feedback.

From equation (5-1), the z-transforms of the input and output signals, and the delay operator z^{-1}, we can get the filter's transfer function as follows:

$$\begin{aligned} Y(z) &= X(z) + a_1 z^{-1} Y(z) \\ Y(z)[1 - a_1 z^{-1}] &= X(z) \\ Y(z) &= \frac{1}{1 - a_1 z^{-1}} X(z) \end{aligned} \tag{5-3}$$

Finally we have the transfer function:

$$\begin{aligned} H(z) &= \frac{Y(z)}{X(z)} = \frac{1}{1 - a_1 z^{-1}} \\ &= \frac{z}{z - a_1} \end{aligned} \tag{5-4}$$

The magnitude response is:

$$|\mathcal{H}(\hat{\omega})| = |H(z)| = \left| \frac{z}{z - a_1} \right| = \left| \frac{1}{z - a_1} \right| \tag{5-5}$$

($|z| = 1$ because we are only concerned with the magnitude when z is on the unit circle, and so its magnitude is always one).

The values of z that make the denominator of the transfer function zero (the roots of the denominator polynomial, where the transfer function becomes infinite) are called its *poles*. In (5-5), there is one pole at $z = a_1$. In general, a_1 is not on the unit circle, and so the phasor z approaches it but is never equal to it. Just as we did for zeros, we can draw a line from the pole to the unit circle to indicate the distance between z and the pole. However, now this distance is in the denominator. Therefore, instead of $H(z)$ having a notch or dip when z nears a zero, it has a *peak* when z nears a pole.

Example 1 For the one pole filter (5-1), $a_1 = re^{j\hat{\omega}_0}$. Figure 5.2 shows this pole when $r = 0.9$ and $\hat{\omega}_0 = \pi/2$. Figure 5.3 shows its magnitude response. As expected, $|\mathcal{H}(\hat{\omega})|$ has a "hill" near $\hat{\omega}_0 = \pi/2$. As r moves closer to one, the hill becomes steeper.

5.1.2 Example: Computing Transfer Function and Impulse Response

Compute the transfer function and impulse response of the system described by the feedback filter,

$$y[n] = 2x[n] + \frac{1}{2} y[n-1] \tag{5-6}$$

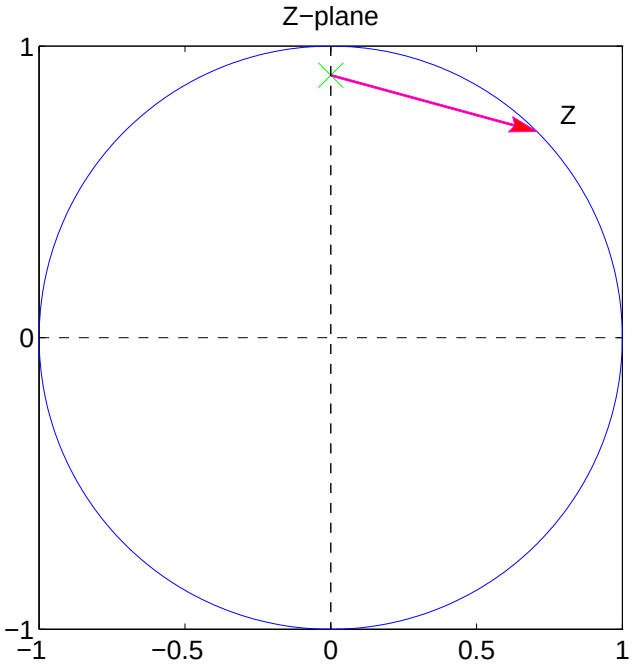


Figure 5.2: One pole in the z-plane, located at $r = 0.9$ and $\hat{\omega}_0 = \pi/2$.

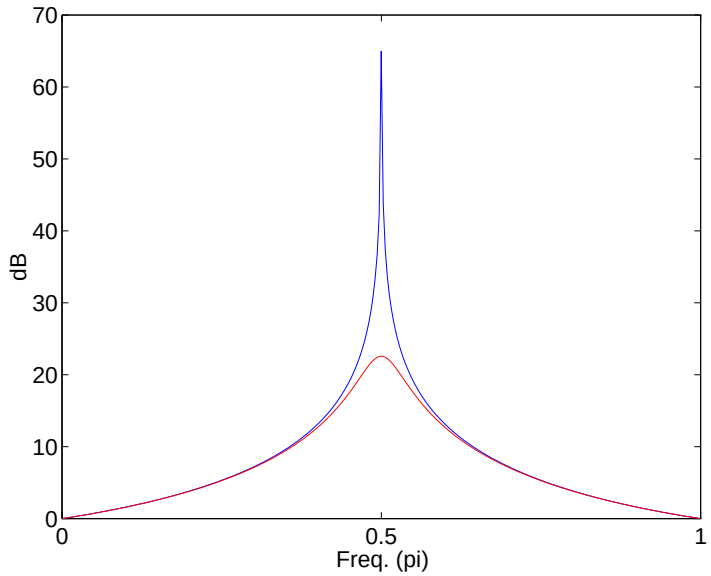


Figure 5.3: Magnitude response of two filters with poles located at $\hat{\omega}_0 = \pi/2$ and $r = 0.9$ (bottom) or $r = 1$ (top).

By computing the z-transform of the both side of above equation and using the fact that

$$y[n-1] \overset{\mathbf{Z}}{\longleftrightarrow} z^{-1}Y(z) \tag{5-7}$$

we obtain,

$$Y(z) = 2X(z) + \frac{1}{2}z^{-1}Y(z). \tag{5-8}$$

Hence the transfer function is

$$H(z) = \frac{Y(z)}{X(z)} = \frac{2}{1-\frac{1}{2}z^{-1}} = \frac{2z}{z-\frac{1}{2}} \tag{5-9}$$

$H(z)$ has a pole at $z = 1/2$ and zero at $z = 0$. Now we compute its impulse response. Previously we had an example of an exponential signal (4-12) and its z-transform (4-16), so we have

$$a^n \overset{\mathbf{Z}}{\longleftrightarrow} \frac{1}{1-az^{-1}}, \quad n \geq 0, |z| > a \tag{5-10}$$

This is of the same form as (5-9), with $a = 1/2$. Using this result, we obtain the inverse transform

$$h[n] = 2\left(\frac{1}{2}\right)^n, \quad n \geq 0 \tag{5-11}$$

where the left and right sides of equation (5-11) are the inverse z-transforms of the left and right sides of (5-9). This is the corresponding impulse response.

5.1.3 Stability

Let's examine equation (5-1) again. Note that $y[n]$ doesn't only depend on $x[n]$. This means that, even if the input signal is zero after some initial value, this feedback term can continue to have a nonzero value and so the filter can still produce some output. This is a major difference from feedforward filters, where the output depends only on the input. As an extreme example, consider the case where the input signal is a *unit impulse*,

$$x[n] = \delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n > 0 \end{cases} \tag{5-12}$$

which means it only has a value of one at the beginning, and then turns off. The output of any filter for a unit impulse is called, logically enough, its *impulse response*. For the filter

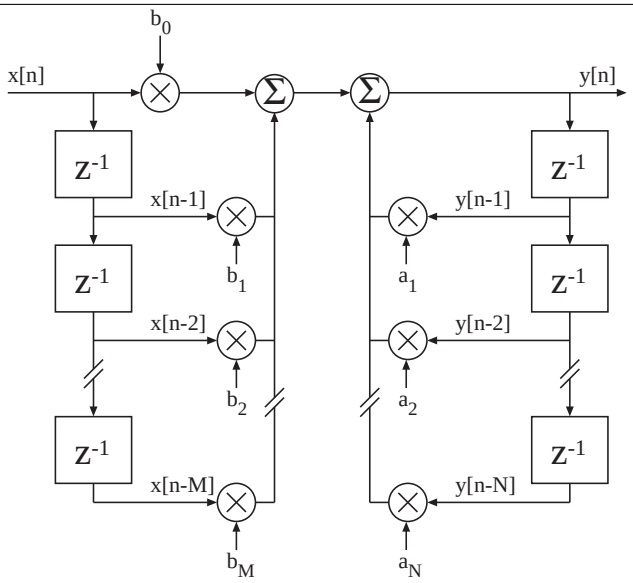


Figure 5.4: Block diagram of a filter with M feedforward and N feedback terms.

in (5-1) with coefficient $a_1 = 0.5$, the impulse response is:

$$\begin{aligned} y[0] &= x[0] = 1 \\ y[1] &= x[1] + 0.5y[0] = 0 + 0.5 \times 1 = 0.5 \\ y[2] &= x[2] + 0.5y[1] = 0 + 0.5 \times 0.5 = 0.25 = 0.5^2 \\ y[3] &= x[3] + 0.5y[2] = 0 + 0.5 \times 0.25 = 0.125 = 0.5^3 \\ &\vdots \\ y[n] &= 0.5^n \\ &\vdots \end{aligned} \tag{5-13}$$

You can see that the output $y[n]$ continues after $n = 0$, but the value becomes smaller each time, and will decay towards zero. On the other hand, if $a_1 > 1$, the output becomes bigger and bigger, and goes to infinity. This second behavior is called *unstable*. From equation (5-5), we know that a_1 is a pole of the filter. In fact, a feedback filter with multiple poles p_i is stable if and only if

$$|p_i| < 1, \text{ for all } i \tag{5-14}$$

In other words, *all* the poles must be inside the unit circle. Here is the basic idea for a four-step (well, three-step-and-a-note) proof of this:

1. **A feedback filter with N feedback terms can be decomposed into N one-pole feedback filters.**

The equation for a feedback filter with N feedback terms (the right hand side of the general feedforward/feedback filter block diagram in figure 5.4) can be written as

$$y[n] = x[n] - \sum_{k=1}^{N} a_k y[n-k] \tag{5-15}$$

Its transfer function is

$$\begin{aligned} H(z) &= \frac{1}{1 + \sum_{k=1}^{N} a_k z^{-k}} \\ &= \frac{z^N}{z^N + \sum_{k=1}^{N} a_k z^{N-k}} \end{aligned} \tag{5-16}$$

Let $p_i, i = 1, 2, ..., N$ be its poles (the roots of the denominator polynomial). Let's not worry about how we can get them (we know how if we can factor the denominator polynomial, so let's assume we have already done that). So, (5-16) can be rewritten in the form

$$H(z) = \frac{z^N}{(z-p_1)(z-p_2)\ldots(z-p_N)} \tag{5-17}$$

$$= \frac{z^N}{\prod_{k=1}^{N}(z-p_k)} \tag{5-18}$$

Recalling our pre-calculus, a fraction with a product of terms as in (5-18) can be rewritten using a partial fraction expansion as

$$H(z) = \frac{A_1}{(1-p_1 z^{-1})} + \frac{A_2}{(1-p_2 z^{-2})} + \ldots + \frac{A_N}{(1-p_N z^{-N})} \tag{5-19}$$

$$= \sum_{k=1}^{N} \frac{A_k}{(1-p_k z^{-k})} \tag{5-20}$$

Each of the terms

$$\frac{A_k}{(1-p_k z^{-k})} \tag{5-21}$$

is a one-pole feedback filter. So, the complete N-pole filter's output is the sum of N one-pole filters' outputs. This is called a parallel one-pole filter.

2. **A one-pole feedback filter is stable if and only if its impulse response is stable.**

Consider the unit impulse $\delta[n]$,

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n > 0 \end{cases} \tag{5-22}$$

Any discrete input signal can be thought of as a weighted sum of delayed unit impulses,

$$\begin{aligned} x[n] &= \{x[0], x[1], x[2], \ldots\} \\ &= x[0]\delta[n] + x[1]\delta[n-1] + x[2]\delta[n-2] + \cdots \end{aligned} \tag{5-23}$$

We can see that the filter's response to any signal will be stable if and only if its response to each of these impulses is stable.

3. **An impulse response is stable if and only if its pole's $|p| < 1$.**

   Equation (5-13) tell us that the i^{th} one-pole feedback filter's impulse response is the signal with samples

   $$1, p_i, p_i^2, p_i^3, \ldots \tag{5-24}$$

   Obviously, the impulse response is stable only when the pole's $|p_i| < 1$

4. **An impulse response is neither stable nor unstable if $|p| = 1$.**

5.1.4 Resonance and Bandwidth

Resonance is the increase in a filter's magnitude response in the region near a pole. An example is shown in figure 5.3. The pole is at frequency $\hat{\omega} = \pi/2$, which is the peak of the magnitude response. The magnitude is small when away from the pole.

One other thing I have mentioned before is that the peak becomes steep when r nears one. The steepness is measured by the filter's *bandwidth*. Bandwidth, B, is defined as the width of the filter's response (i.e., the range of frequencies) at half its maximum power output. Bandwidth is an important measure of a filter's performance; it indicates which frequencies are passed and which are filtered out.

The filter's power is the square of its amplitude, so the bandwidth can also be measured at $1/\sqrt{2}$ of its peak amplitude value. These half power levels are denoted by $|\mathcal{H}(\hat{\omega})|_B^2$ (the $1/\sqrt{2}$ amplitude points written as $|\mathcal{H}(\hat{\omega})|_B$). If the peak value of power is $|\mathcal{H}(\hat{\omega})|_p^2$ and the peak amplitude is $|\mathcal{H}(\hat{\omega})|_p$, the bandwidth points are located at

$$\underbrace{|\mathcal{H}(\hat{\omega})|_B}_{\substack{\text{cutoff} \\ \text{amplitude}}} = \frac{1}{2}\underbrace{|\mathcal{H}(\hat{\omega})|_p^2}_{\substack{\text{peak} \\ \text{power}}} = \frac{1}{\sqrt{2}}\underbrace{|\mathcal{H}(\hat{\omega})|_p}_{\substack{\text{peak} \\ \text{amplitude}}} \tag{5-25}$$

Since $20\log_{10}(1/\sqrt{2}) = -3\text{dB}$

$$\frac{|\mathcal{H}(\hat{\omega})|_B}{|\mathcal{H}(\hat{\omega})|_p} = -3\text{dB} \tag{5-26}$$

and so the *cutoff amplitude* for a filter are those for which its output is reduced 3dB from its peak. The values of $\hat{\omega}$ at which the filter's output is reduced by this much define the edges of its *passband* and are called its *cutoff frequencies*; they are sometimes called its "minus three deebee points". We will call these frequencies $\hat{\omega}_B$ in the simple case where the passband

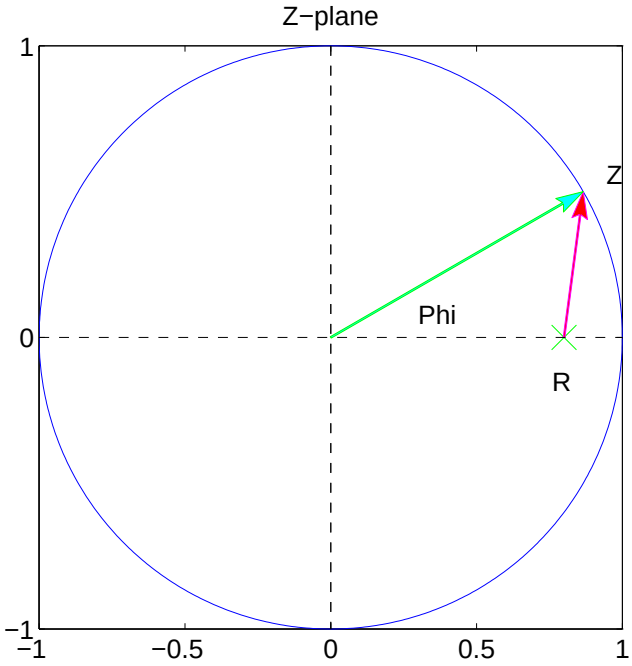


Figure 5.5: One pole $a_1 = re^{j\hat{\omega}_0}$ in z-plane. $r = R$, $\hat{\omega}_0 = 0$ and $z = e^{j\hat{\omega}}$

is symmetrical and the two cutoff frequencies are the same amount above and below the frequency at peak filter output, $\hat{\omega}_p$.

Let's take a look at the case in which a pole is on the real axis and no other poles are nearby; the pole is $a_1 = re^{j\hat{\omega}_0}$, $r = R$ and $\hat{\omega}_0 = 0$ (so $a_1 = R$). A point on the unit circle is $z = e^{j\hat{\omega}}$. This situation is illustrated in figure 5.5. We know that the one pole transfer function is

$$H(z) = \frac{1}{1 - a_1 z^{-1}} \tag{5-27}$$

with magnitude

$$|H(z)| = \left| \frac{1}{1 - a_1 z^{-1}} \right| \tag{5-28}$$

Consider the inverse square of the magnitude, substituting in our expressions for a_1 and z,

$$\begin{aligned}
\frac{1}{|\mathcal{H}(\hat{\omega})|^2} &= |1 - Re^{j\hat{\omega}}|^2 \\
&= |1 - R\cos\hat{\omega} - jR\sin\hat{\omega}|^2 && \text{(using Euler's formula)} \\
&= (1 - R\cos\hat{\omega})^2 + R^2\sin^2\hat{\omega} && \left(|H|^2 = \mathrm{Re}[\mathcal{H}]^2 + \mathrm{Im}[\mathcal{H}]^2\right) \\
&= 1 - 2R\cos\hat{\omega} + R^2\cos^2\hat{\omega} + R^2\sin^2\hat{\omega} \\
&= 1 - 2R\cos\hat{\omega} + R^2 && \left(\cos^2\theta + \sin^2\theta = 1\right)
\end{aligned} \tag{5-29}$$

The peak of $|\mathcal{H}(\hat{\omega})|^2$ should be the value of angle $\hat{\omega}_p = \hat{\omega}$ that makes $1/|\mathcal{H}(\hat{\omega})|^2 = 1 - 2R\cos\hat{\omega} + R^2$ minimum, that is when $\hat{\omega} = 0$. The power at this peak is determined by substituting $\hat{\omega} = 0$ into equation (5-29) and taking its reciprocal:

$$|\mathcal{H}(\hat{\omega})|_p^2 = \frac{1}{(1-R)^2} \tag{5-30}$$

According to equation (5-25),

$$|\mathcal{H}(\hat{\omega})|_B = \frac{1}{2}|\mathcal{H}(\hat{\omega})|_p^2 = \frac{1}{2(1-R)^2} \tag{5-31}$$

The corresponding $\hat{\omega}_B$ can be obtained by substituting $|\mathcal{H}(\hat{\omega})|_B$ for $|\mathcal{H}(\hat{\omega})|$ in equation (5-29) and solving for frequency:

$$2(1-R)^2 = 1 - 2R\cos\hat{\omega}_B + R^2 \tag{5-32}$$

$$\cos\hat{\omega}_B = 2 - \frac{1}{2}\left(R + \frac{1}{R}\right) \tag{5-33}$$

The filter's bandwidth is the span $[-\hat{\omega}_B, \hat{\omega}_B]$ — a distance of $2\hat{\omega}_B$. When R is close to one, we can express R as a small amount ϵ less than one: $R = 1 - \epsilon$. We can then take advantage of the expansions:

$$\frac{1}{R} = \frac{1}{1-\epsilon} = 1 + \epsilon + \epsilon^2 + \epsilon^3 + \cdots \tag{5-34}$$

and

$$\cos\epsilon = 1 - \frac{\epsilon^2}{2!} + \frac{\epsilon^4}{4!} + \cdots \tag{5-35}$$

So, from (5-33),

$$\begin{aligned}\cos\hat{\omega}_B &= 2 - \frac{1}{2}[\underbrace{(1-\epsilon)}_{R} + \underbrace{(1+\epsilon+\epsilon^2+\epsilon^3+\cdots)}_{\frac{1}{R}}] \\ &= 1 - \frac{\epsilon^2}{2} - O(\epsilon^3) \\ &\approx \cos\epsilon\end{aligned} \tag{5-36}$$

(where $O(\epsilon^3)$ is shorthand for terms in the expansion of order ϵ^3 or higher). Therefore, $\hat{\omega}_B \approx \epsilon$. So, when R is close to the unit circle,

$$B = 2\hat{\omega}_B \approx 2\epsilon = 2(1-R) \tag{5-37}$$

or

$$R \approx 1 - B/2 \tag{5-38}$$

If we want a low-pass filter with a particular bandwidth, equation (5-38) gives us a way to determine its pole location.

Self-Test Exercises

See A.5 #1–2 for answers.

1. Derive equation (5-33) from (5-32).

2. In the situation where the sampling rate is 44,100Hz and the desired bandwidth is 20Hz, R in (5-38) is 0.998575. Solve for R the situation where the desired bandwidth is 200Hz. Is it true that when R is far away from one, B grows large?

5.2 Mixing Feedback and Feedforward Filters

We have now seen feedforward filters with zeros in the transfer function and feedback filters with poles. Zeros suppress frequency components and poles enhance them. Quite often, we want to combine poles and zeros to improve the filter's features, such as the flatness of the passband and the abruptness with which its response transitions between the passband and the stop band. Generally, the transfer function of a filter with block diagram in figure 5.4 can written as

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} \tag{5-39}$$

and the filter is

$$y[n] = \underbrace{b_0 x[n] + b_1 x[n-1] + \cdots + b_M x[n-M]}_{\text{feedforward terms}} - \underbrace{a_1 y[n-1] - \cdots - a_n y[n-N]}_{\text{feedback terms}} \tag{5-40}$$

Depending on the coefficients $\{b_k\}$ and $\{a_\ell\}$, the filter will show different features. Some of these features have proven so useful that these forms of (5-39) have acquired special names, such as elliptic, Butterworth, etc (usually, based on the form of the numerator and/or denominator polynomial).

5.3 Implementation

Just as in chapter 3, feedback filters have their delays implemented with queues which are "circular arrays". However, there are two additional complications that we must deal with: complex poles and accuracy of numerical computation.

5.3.1 Avoiding Complex Numbers

In principle, there is no reason that we couldn't perform all our computations using complex numbers and just output the real part of the result as the filtered signal. However, we often want to perform filtering in real time, and so would like to avoid any unnecessary computation. A cute trick allows us to eliminate the need for complex numbers.

Remember that all complex poles will appear in conjugate pairs. Since each pole is a root of a polynomial, that means that the denominator of the filter's transfer function will have even degree (excepting the real-valued poles, of course), and when it is factored the conjugate pairs (z_i, z_i^*) will appear as $(z-z_i)(z-z_i^*)$. If we multiply this out, we get $z^2 - 2\,\mathrm{Re}[z_i]z + |z_i|^2$. In other words, the imaginary parts of poles cancel each other out. Since we're already using b_k for the feedforward coefficients and a_ℓ for the feedback ones in the filter equation, let's set $c_i = -2\,\mathrm{Re}[z_i]$ and $d_i = |z_i|^2$. If the transfer function's denominator is of order N (N even), then we can multiply out the polynomials for each complex conjugate pair and rewrite the denominator as the product of $N/2$ second-order polynomials

$$(z^2 + c_0 z + d_0)(z^2 + c_1 z + d_1) \cdots (z^2 + c_{N/2-1} z + d_{N/2-1}) \tag{5-41}$$

with no need for complex arithmetic. We've done this before symbolically, using Euler's formula to eliminate the imaginary part of a complex conjugate pair. The point here is that, since complex poles always appear in conjugate pairs, we can do this with even quite complicated filters to eliminate the need for performing complex arithmetic or processing complex numbers — a significant optimization.

5.3.2 Limitations of Numerical Accuracy

When we talk of the mathematics of filters, we assume that numbers have infinite precision. Unfortunately, this is not the case for computer implementation. These days, computers (including digital signal processors) typically use either 32 or 64 bits to represent numbers (be they fixed or floating point). That may seem like a great deal of precision, but it is unfortunately not uncommon for intermediate results in numerical computation to require many more bits to retain needed precision. There are a number of ways this can happen, but one is where a computation is performed iteratively, so that a long chain of operations is applied to inputs before they become outputs. At each step of this chain, the result has limited precision. In other words, the number we get is not exactly correct — in general, it can't be, since we don't have infinite precision.

This loss of precision can mount rapidly, eventually destroying the result. We can think of this limited precision as an *error*. If the input is from a 16-bit A/D, for example, and we assume it does its conversion absolutely accurately to the smallest bit (which, as you have seen in chapter 2 and will revisit in chapter 7, is probably not realistic), then that's around $\log_{10} 2^{16} \approx 5$ decimal digits from the A/D, to be generous. Each arithmetic operation we perform, regardless of the number of bits we use, can have the undesirable effect of decreasing the number of digits of precision we have. Sums of approximate values have errors which are the sums of their addends' errors. Multiplication tends to magnify errors. Let's see this with a simple example.

Example 4 Consider the following two recurrence relations for computing the series $\{x[n]\} = \{1/3^n\}$ $(n = 0, 1, 2, \ldots)$:

$$x'[n] = \frac{1}{3}x'[n-1] \tag{5-42}$$

$$x''[n] = \frac{10}{3}x''[n-1] - x''[n-2] \tag{5-43}$$

Both of these equations are mathematically "correct". *They also look like the expressions we use for our filters.* Yet, they yield very different results because of loss of significance. Let's use an initial value of $x'[0] = 0.99996$ for (5-42) and initial values of $x''[0] = 1$ and $x''[1] = 0.33332$ for (5-43). This is an initial error of 0.00004 for $x'[0]$ and $0.000013\bar{3}$ for $x''[1]$. I'll use MATLAB to compute the first ten terms in each sequence with double-precision accuracy for each calculation. The MATLAB code is

```
% significance.m
% 2/22/02 MDS
% Examine how error grows in an iterative computation
stdout = 1;

% This isn't efficient, but it is straightforward

% Values of n for computation
n = [0:10];

% Series 1: the real McCoy
x = 1./3.^n;

% Series 2: x'[n] = 1/3 x'[n-1]
xp = 0.99996;

for n = [1:10],
  xp(n+1) = 1/3 * xp(n);  % Remember, MATLAB indices start at 1, not 0
end;

% Series 3: x''[n] = 10/3 x''[n-1] - x''[n-2]
xpp = [1 0.33332];

for n = [2:10],
  xpp(n+1) = 10/3 * xpp(n) - xpp(n-1);
end;

% Print out the results
fprintf(stdout, ' n          x[n]            x''[n]          x''''[n]\n');
for n = [0:10],
  fprintf(stdout, '%2.1d\t%12.10f\t%12.10f\t%12.10f\n', ...
```

```
          n, x(n+1), xp(n+1), xpp(n+1));
end;
```

The output this script produces is:

```
 n          x[n]           x'[n]          x''[n]
 0     1.0000000000   0.9999600000   1.0000000000
 1     0.3333333333   0.3333200000   0.3333200000
 2     0.1111111111   0.1111066667   0.1110666667
 3     0.0370370370   0.0370355556   0.0369022222
 4     0.0123456790   0.0123451852   0.0119407407
 5     0.0041152263   0.0041150617   0.0029002469
 6     0.0013717421   0.0013716872  -0.0022732510
 7     0.0004572474   0.0004572291  -0.0104777503
 8     0.0001524158   0.0001524097  -0.0326525834
 9     0.0000508053   0.0000508032  -0.0983641945
10     0.0000169351   0.0000169344  -0.2952280648
```

The first column is accurate to the full precision of IEEE floating point (that is, the computation is that accurate; the output was limited to ten decimal places). The second column's error is *stable*: it decreases in an exponential manner, and the value for $x'[10]$ is within 7×10^{-10} of the actual value. The third column's error is *unstable*: it increases exponentially.

Couple this with the need for precise pole placement, and I think you can see that we need to be careful about our implementations. We saw in this chapter's section on combining feedforward and feedback filters the general transfer function for a filter (5-39) and its direct implementation (5-40), which involved M delayed values of x and N delayed values of y.

However, it is better to implement such a digital filter as a cascade of second-order ones, by factoring the numerator and denominator (which should be "easy," since we know the locations of the poles and zeros) and multiplying out terms with complex conjugate pairs as described in the previous section. The resultant fraction will have numerator and denominator which are each a product of second-order polynomials. Each ratio of second-order polynomials is a subfilter, which can be implemented by a simple update equation. If we take the output of one subfilter and present it as the input of the next, the output of the last subfilter will be equivalent to the output of the entire original filter (we saw this cascade of filters in chapter 3). Note that, though each filter has a very short (two step) queue, the series combination produces what amounts to a longer overall queue. In practical applications, this shouldn't be much of a computational penalty.

5.4 Problems

5.5 Further Reading

- James H McClellan, Ronald W. Schafer, and Mark A. Yoder, *DSP First: A Multimedia Approach*, Prentice Hall, 1998, chapter 8 (§8.1–8.6, 8.8).

6 Spectral Analysis

We begin our study of signal frequency analysis with the representation of continuous-time periodic and aperiodic signals by means of the *Fourier Transform*. This is followed by a treatment of discrete-time signals, that is the *Discrete Fourier Transform* and an efficient algorithm for computing it: the *Fast Fourier Transform* (FFT). After this chapter, you should understand what they are and how the FFT works. You should also understand related terms, for example, *fundamental frequency*, *harmonics*, *spectrum*, *time domain*, and *frequency domain*. You should be able to use them to do some frequency analysis of a signal. We will also go over some problems that you need to keep in mind when using them: *power leakage* caused by sudden changes in the signal, the *tradeoff between time and frequency resolution*, and the function of *windows*. You should be able to use this knowledge to guide what you do to obtain accurate results in estimating spectral information.

6.1 The Fourier Transform

We developed the Fourier series to represent a *periodic* signal as a linear combination of harmonically related complex exponentials:

$$x(t) = \sum_{k=-\infty}^{+\infty} c_k e^{j2\pi k f_0 t} \tag{6-1}$$

Here, the c_k are the Fourier coefficients and the sequence $\ldots, c_{-1}, c_0, c_1, \ldots$ can be thought of as the frequency domain representation (the amplitudes of the complex sinusoids) of the time domain signal $x(t)$. For continuous time *aperiodic* signals $x(t)$, the Fourier Transform is defined. It can be derived from the Fourier series allowing the signal period T to go to infinity. Here I'll just give its definition:

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi f t} df \tag{6-2}$$

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt \tag{6-3}$$

Equation (6-3) is the *Fourier transform* of $x(t)$: a function of the continuous frequency variable f. Equation (6-2) is called the *inverse Fourier transform*, which yields $x(t)$ when

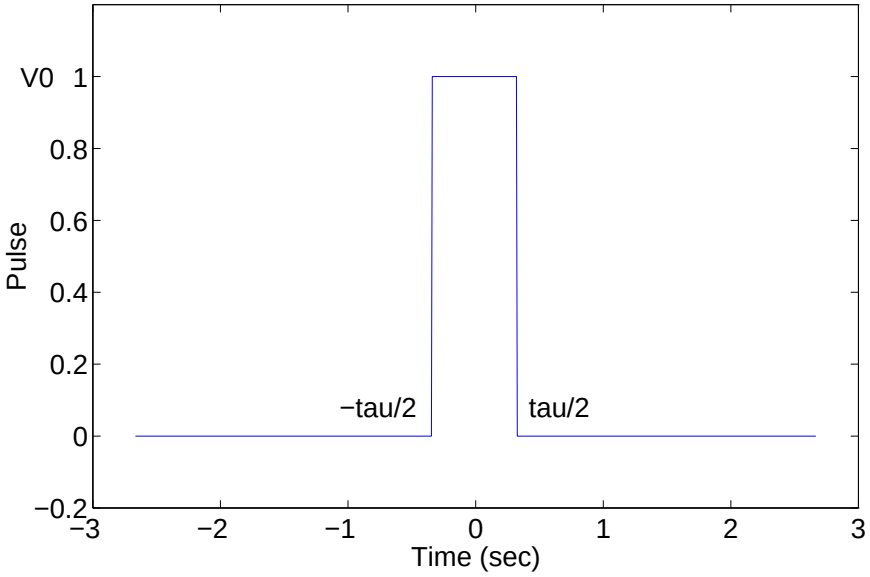


Figure 6.1: A rectangular pulse with width τ.

$X(f)$ is known. The Fourier transform converts an infinite-duration (aperiodic), continuous signal in the time domain into an infinite, continuous spectrum in the frequency domain. It is apparent that the essential difference between the Fourier series and the Fourier transform is that the spectrum in the latter is continuous (the Fourier series yields a discrete line spectrum; it has non-zero values only at specific frequencies) and hence the synthesis of an aperiodic signal from its spectrum is accomplished by means of integration instead of summation. The Fourier transform pair in (6-3) and (6-2) can also be expressed in terms of the radian frequency variable $\omega = 2\pi f$. Since $f = \omega/(2\pi)$, the pair becomes

$$x(t) = \int_{-\infty}^{\infty} X(\omega) e^{j\omega t} d\omega \tag{6-4}$$

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \tag{6-5}$$

6.1.1 Example: Fourier transform of a rectangular pulse

Let's look at the rectangular pulse I wrote about before, but here the pulse is aperiodic — there is just a single pulse instead of a train of them. The signal is defined as

$$x(t) = \begin{cases} 1 & |t| \leq \tau/2 \\ 0 & |t| > \tau/2 \end{cases} \tag{6-6}$$

and illustrated in Figure 6.1.

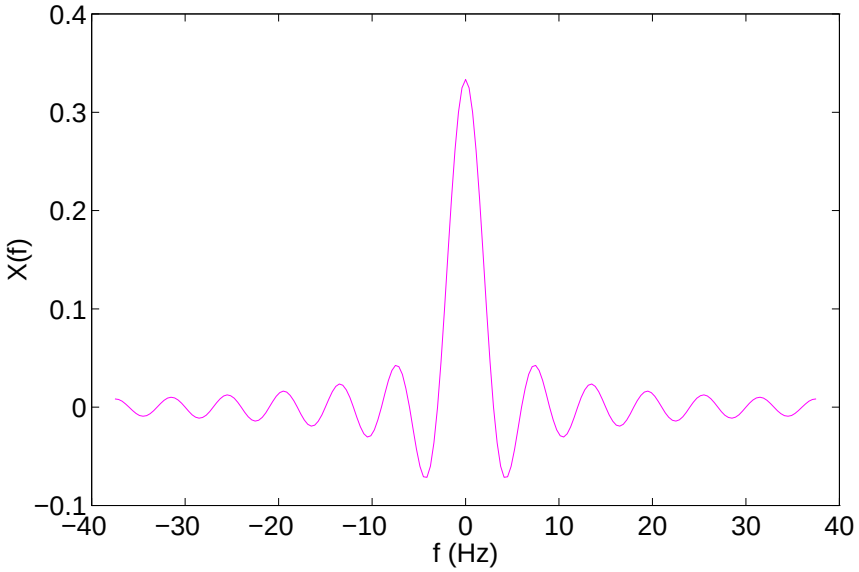


Figure 6.2: Fourier transform of the pulse in figure 6.1.

By applying (6-3), we find that

$$\begin{aligned} X(f) &= \int_{-\tau/2}^{+\tau/2} e^{-j2\pi ft} dt \\ &= \frac{e^{-j2\pi ft}}{-j2\pi f}\Bigg|_{-\tau/2}^{+\tau/2} \\ &= \frac{1}{-j2\pi f}[e^{-j\pi f\tau} - e^{j\pi f\tau}] \\ &= \tau\,\mathrm{sinc}(\pi f\tau) \end{aligned} \tag{6-7}$$

The final step in deriving (6-7) made use of the observation that the difference of a complex number and its conjugate is just twice its imaginary part, $(a - jb) - (a + jb) = -2jb$. In polar notation, then, $e^{-j\pi f\tau} - e^{j\pi f\tau} = -2j \sin \pi f\tau$, using Euler's formula. We observe that $X(f)$ is real and has the shape of the sinc function we mentioned previously. The important difference here is that it is a *continuous* spectrum instead of line spectrum. It is shown in figure 6.2.

As with the periodic rectangular pulses, from (6-7) we note that the zero crossings of $X(f)$ occur at multiples of $1/\tau$. Furthermore, the width of the main lobe, which contains most of the signal energy, is equal to $1/\tau$. As the pulse duration τ decreases or increases, the main lobe becomes broader or narrower, and more energy is moved to the higher or lower frequencies, respectively, as is illustrated in figure 6.3. Thus, as the single pulse is expanded (compressed) in time, its transform is compressed (expanded) in frequency.

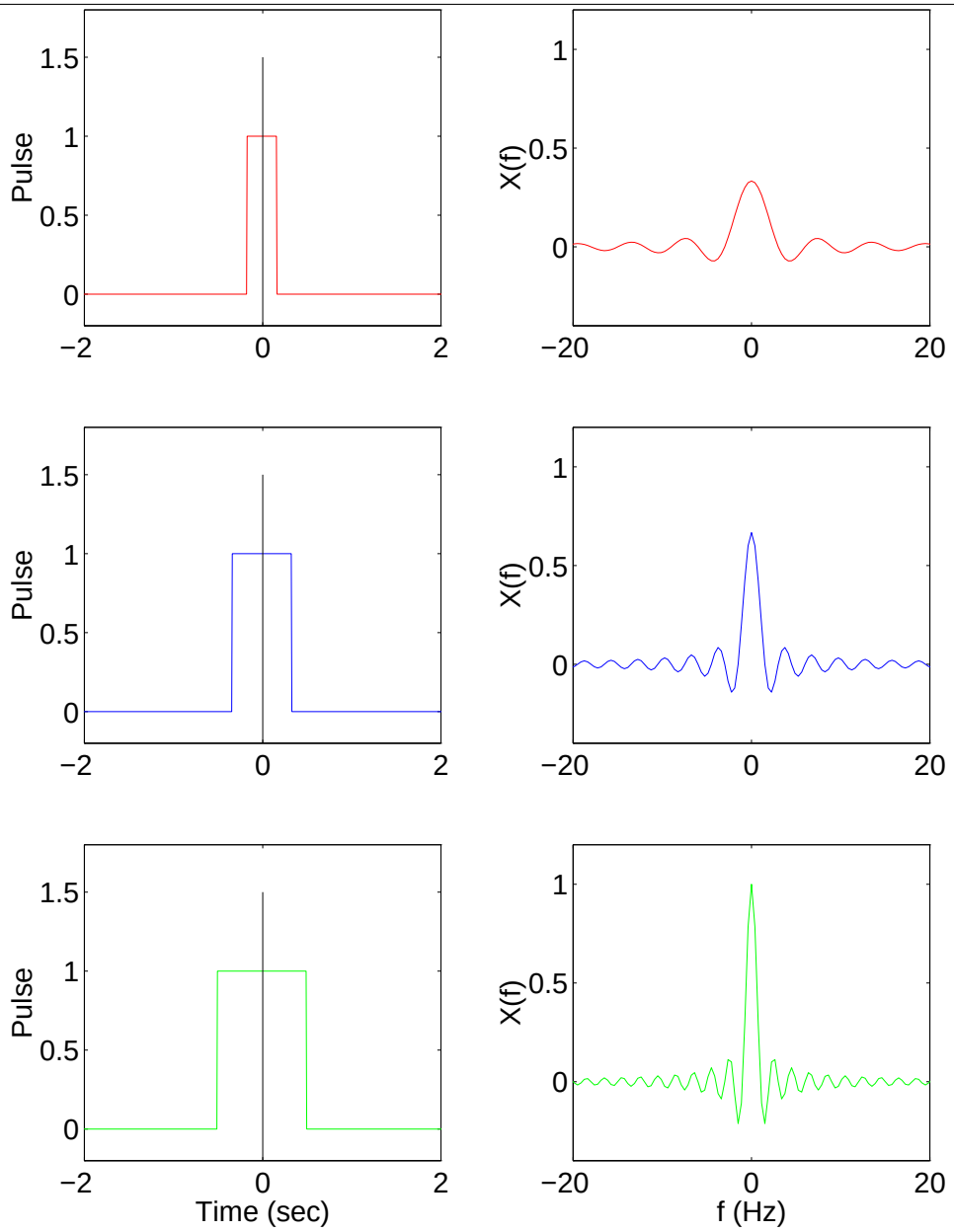


Figure 6.3: Fourier transform of a rectangular pulse for various width values τ.

6.2 The Discrete Fourier Transform

So far, I have talked about the Fourier series (which is for a periodic, continuous signal) and its discrete spectrum and the Fourier transform (which is for an aperiodic, continuous signal) and its continuous spectrum. Now, I will talk about periodic, *discrete* signals and their spectra, which are discrete.

Let's consider a finite digital signal $\{x[n]\}$ $(n = 0, 1, 2, \ldots, N-1)$. The discrete Fourier

transform (DFT) is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-jnk2\pi/N}, \quad k = 0, 1, 2, \ldots, N-1 \tag{6-8}$$

where X is the spectrum of x, and its synthesis equation or inverse DFT ($IDFT$) is

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{jnk2\pi/N}, \quad n = 0, 1, 2, \ldots, N-1 \tag{6-9}$$

Obviously, the DFT converts a finite (periodic), discrete signal in the time domain into a finite, discrete spectrum in the frequency domain.

We can derive the DFT informally from the Fourier series of a periodic signal. Let's compare equation (6-8) and that for the coefficients of the Fourier series, equation (1-35), which I repeat here:

$$c_k = \langle f(t), e^{jk\omega_0 t} \rangle = \frac{1}{T} \int_0^T f(t) e^{-jk\omega_0 t} dt \tag{6-10}$$

First, notice that the DFT uses a summation within $[0, N-1]$ and the series uses an integral within one period T. This difference arises because here the signal is discrete and finite (and so we write $x[n]$ instead of $f(t)$). The factor of $1/T$ in (6-10) is equivalent to that of $1/N$ in (6-9), where I have arbitrarily included that factor in the IDFT instead of the DFT.

Now let's compare the basis (the complex exponential). The frequency here is also discretized. Remember that a sampled signal has a Nyquist frequency of $\omega_s/2$, or π radians/sample, where ω_s is the sampling rate, equivalent to 2π radians/sample. Let's discretize this signal's frequencies using N equally-spaced points. The step $\Delta\hat{\omega}$ between neighboring frequencies (in radians/sample) is

$$\Delta\hat{\omega} = \frac{2\pi}{N} \tag{6-11}$$

This is what equation (6-8) indicates within the sum: that whatever nk is, it will be a multiple of $2\pi/N$. The k^{th} frequency in radians can be written as

$$k\Delta\hat{\omega} = k\frac{2\pi}{N}, \quad k = 0, 1, 2, \ldots, N-1 \tag{6-12}$$

Substituting $k\Delta\hat{\omega}$ in (6-12) for $k\omega_0$ in the Fourier series basis should convince you that the DFT is the discrete version of the Fourier series. The coefficient $X[k]$ is the magnitude of the spectrum of x at frequency $k2\pi/N$.

Table 6.1 illustrates how the step between frequencies changes as the signal duration and sampling rate (and thus number of samples) change. Increasing the signal duration, and therefore the number of samples, without changing the sampling rate (which keeps the Nyquist cutoff constant) decreases the step between frequencies (or, if you prefer to think of this another way, increases the frequency resolution). On the other hand, decreasing the sampling rate decreases the number of samples (assuming the signal duration stays constant) and the Nyquist cutoff, so the step between frequencies stays constant.

T (s)	f_s (Hz)	N	$\Delta\hat{\omega}$ (rad/sample)	$\Delta\omega$ (rad/s)	Δf (Hz)
1	10	10	$\pi/5$	2π	1
10	10	100	$\pi/50$	$\pi/5$	1/10
100	10	1000	$\pi/500$	$\pi/50$	1/100
100	5	500	$\pi/250$	$\pi/50$	1/100
100	1	100	$\pi/50$	$\pi/50$	1/100

Table 6.1: Some examples of how $\Delta\hat{\omega}$, $\Delta\omega = \Delta\hat{\omega} f_s$, and $\Delta f = \Delta\omega/2\pi = f_s/N$ change for signals of different duration and sampling rate.

6.2.1 Derivation of the IDFT [Optional]

Next, let's see how we can get the synthesis equation IDFT from the DFT. Multiplying both sides of (6-8) by $e^{jn'k2\pi/N}$ and then summing over all k,

$$\sum_{k=0}^{N-1} X[k]e^{jn'k2\pi/N} = \sum_{k=0}^{N-1}\sum_{n=0}^{N-1} x[n]e^{-jnk2\pi/N}e^{jn'k2\pi/N} \tag{6-13}$$

Switching the order of summation of n and k on the right side, plus some algebraic rearrangement yields

$$\sum_{k=0}^{N-1} X[k]e^{jn'k2\pi/N} = \sum_{n=0}^{N-1} x[n]\sum_{k=0}^{N-1} e^{jk(n'-n)2\pi/N} \tag{6-14}$$

It can be shown that

$$\sum_{k=0}^{N-1} e^{jk(n'-n)2\pi/N} = \begin{cases} N & n = n' \\ 0 & n \neq n' \end{cases} \tag{6-15}$$

$$= N\delta[n-n'] \tag{6-16}$$

where $\delta[n-n']$ is defined as

$$\delta[n-n'] = \begin{cases} 1 & n = n' \\ 0 & n \neq n' \end{cases} \tag{6-17}$$

Substituting the above result into (6-14), we get the IDFT (but for that pesky factor of $1/N$):

$$\sum_{k=0}^{N-1} X[k]e^{jn'k2\pi/N} = \sum_{n=0}^{N-1} x[n]N\delta[n-n'] = Nx[n'] \tag{6-18}$$

To simplify the notation, I will use a vector notation for the discrete signal $\mathbf{x} = \{x[n]\}$ and its spectrum $\mathbf{X} = \{X[k]\}$. The DFT and IDFT operations become $N \times N$ matrices $\mathbf{F}$ and $\mathbf{F^{-1}}$, respectively. An element in row k and column n of $\mathbf{F}$ is

$$[\mathbf{F}]_{kn} = e^{-jnk2\pi/N}, \quad k, n = 0, 1, 2, \ldots, N-1 \tag{6-19}$$

Then the DFT and IDFT can be written as

$$\mathbf{X} = \mathbf{F}\mathbf{x} \tag{6-20}$$

$$\mathbf{x} = \mathbf{F}^{-1}\mathbf{X} \tag{6-21}$$

or

$$\mathbf{x} \xrightarrow{\mathbf{F}} \mathbf{X} \tag{6-22}$$

$$\mathbf{X} \xrightarrow{\mathbf{F}^{-1}} \mathbf{x} \tag{6-23}$$

6.2.2 Finite vs. Infinite Signals

In this book, I have used (and will continue to use) the terms "periodic" and "finite" interchangeably (and, similarly, "aperiodic" and "infinite"). When we compute the Fourier series or the DFT of a finite signal of duration T, we do so only within that interval of time. In effect, what we are doing for finite signals is "pretending" that the signal is actually periodic with period T and that we are only looking at one of its periods.

On the other hand, if a signal is actually of infinite duration, we would use the Fourier transform. However, this is not a process that we can perform on a computer, where infinite duration signals cannot exist (the duration of any signal is limited by the available storage). We have to chop the signal off at a length that the computer can handle, ignoring the error it may produce (or testing to make sure that the error is tolerable for our application). This brings us back to the DFT for processing the resulting finite signal as an *approximation* of the infinite signal's Fourier transform. Note that, even outside the computer, our time is finite (even the lifespan of the universe is finite) and so no physically realizable signal is actually infinite.

6.2.3 Properties of the DFT

Let's consider some of the properties of the DFT of physically realizable signals. These signals are real, so $x[n] = x[n]^*$. What is the spectrum of such a signal like? We can think of the signal's frequency content or spectrum, $X[k]$ ($k = 0, 1, \ldots, N-1$), as a sequence of N complex numbers. This is known as a N *point FFT*. If we compute $X[N-k]$ using (6-8),

we get

$$\begin{aligned} X[N-k] &= \sum_{n=0}^{N-1} x[n] e^{-jn(N-k)2\pi/N} \\ &= \sum_{n=0}^{N-1} x[n] e^{-jn2\pi} e^{jnk2\pi/N} \\ &= \sum_{n=0}^{N-1} x[n] e^{jnk2\pi/N} && \text{(since } e^{-jn2\pi} = 1\text{)} \\ &= \left[\sum_{n=0}^{N-1} x[n]^* e^{-jnk2\pi/N} \right]^* && \text{(since } ab = ((ab)^*)^* = (a^*b^*)^*\text{)} \end{aligned}$$

Because $x[n] = x[n]^*$,

$$\begin{aligned} X[N-k] &= \left[\sum_{n=0}^{N-1} x[n] e^{-jnk2\pi/N} \right]^* \\ &= X[k]^* \end{aligned} \tag{6-24}$$

So for example, when $N = 8$, $k = 0, 1, 2, 3, 4, 5, 6, 7$, and

$$X[7] = X[1]^* \quad X[6] = X[2]^* \quad X[5] = X[3]^* \tag{6-25}$$

(When $k = 0$, $X[8] = X[0]^*$ but $X[8]$ is out of spectrum, which ranges from 0 to 7 only. This is the "DC", or zero-frequency, component of the signal.)

This can be represented diagrammatically as

$$\underset{\substack{\uparrow \\ \text{DC}}}{0} \quad \overbrace{1 \quad \overbrace{2 \quad 3 \quad 4 \quad 5}^{} \quad 6 \quad 7}^{} \tag{6-26}$$

Obviously, it is symmetric about $k = 4$. When N is an even number, the spectrum has $N/2$ at the center; when N is an odd number, the center is between $\lfloor N/2 \rfloor$ and $\lceil N/2 \rceil$. So, for $N = 7$,

$$\underset{\substack{\uparrow \\ \text{DC}}}{0} \quad \overbrace{1 \quad \overbrace{2 \quad 3 \quad 4}^{} \quad 5 \quad 6}^{} \tag{6-27}$$

What this means is that the spectrum over frequencies $\lceil N/2 \rceil$ to $N-1$ has the same magnitude as over 1 to $\lfloor N/2 \rfloor$, but with negative phase angle (which means negative frequencies on the unit circle). In fact, $N/2$ corresponds to the Nyquist frequency in radians/sample. We can see this from (6-12), when $k = N/2$

$$\hat{\omega}_{N/2} = \frac{N}{2} \frac{2\pi}{N} = \pi \tag{6-28}$$

Converting this to Hz, we get

$$\hat{f}_{N/2} = \hat{\omega}_{N/2}\frac{f_s}{2\pi} = \pi\frac{f_s}{2\pi} = \frac{f_s}{2} \tag{6-29}$$

where f_s is the sampling rate. The above equation shows that $\hat{f}_{N/2}$ is the Nyquist frequency in Hz. The conclusion is that the magnitude of the transform of a real valued signal is an *even* function of frequency and so we only need to plot its spectrum in range $[0, N/2]$, corresponding to the range of frequencies from 0 to the Nyquist frequency. This also means that the DFT of a real signal has (slightly more than) half the information of the DFT of a complex-valued signal with the same number of samples (which makes sense, since each sample has an imaginary component of zero).

Self-Test Exercises

See A.6 #1 for the answer.

1. Show which frequencies will be equal for a spectrum with:
   (a) 16 points.
   (b) 15 points.

Example: Spectrum of an exponential

Consider the exponential signal

$$x[n] = a^n, \quad 0 < a < 1, \quad n = 0, 1, 2, \ldots, N-1 \tag{6-30}$$

The signal is plotted in the top of figure 6.4

The DFT of the sequence $x[n]$ is

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n]e^{-jnk2\pi/N} \\ &= \sum_{n=0}^{N-1} a^n e^{-jnk2\pi/N} \end{aligned} \tag{6-31}$$

The common ratio for this geometric series (the ratio of two successive terms) is $ae^{-jk2\pi/N}$. Accordingly, the summation in (6-31) becomes

$$\begin{aligned} X[k] &= \frac{1 - a^N e^{-jkN2\pi/N}}{1 - ae^{-jk2\pi/N}} \\ &= \frac{1 - a^N}{1 - ae^{-jk2\pi/N}} \end{aligned} \tag{6-32}$$

The magnitude of $X[k]$ for $N = 50$ and $a = 0.8$ is plotted in the bottom of figure 6.4. Since this is a real signal, $|X[k]|$ is an even function of frequency, and so we can just plot the first half of the range of 0 to $N/2$. That is shown in figure 6.5

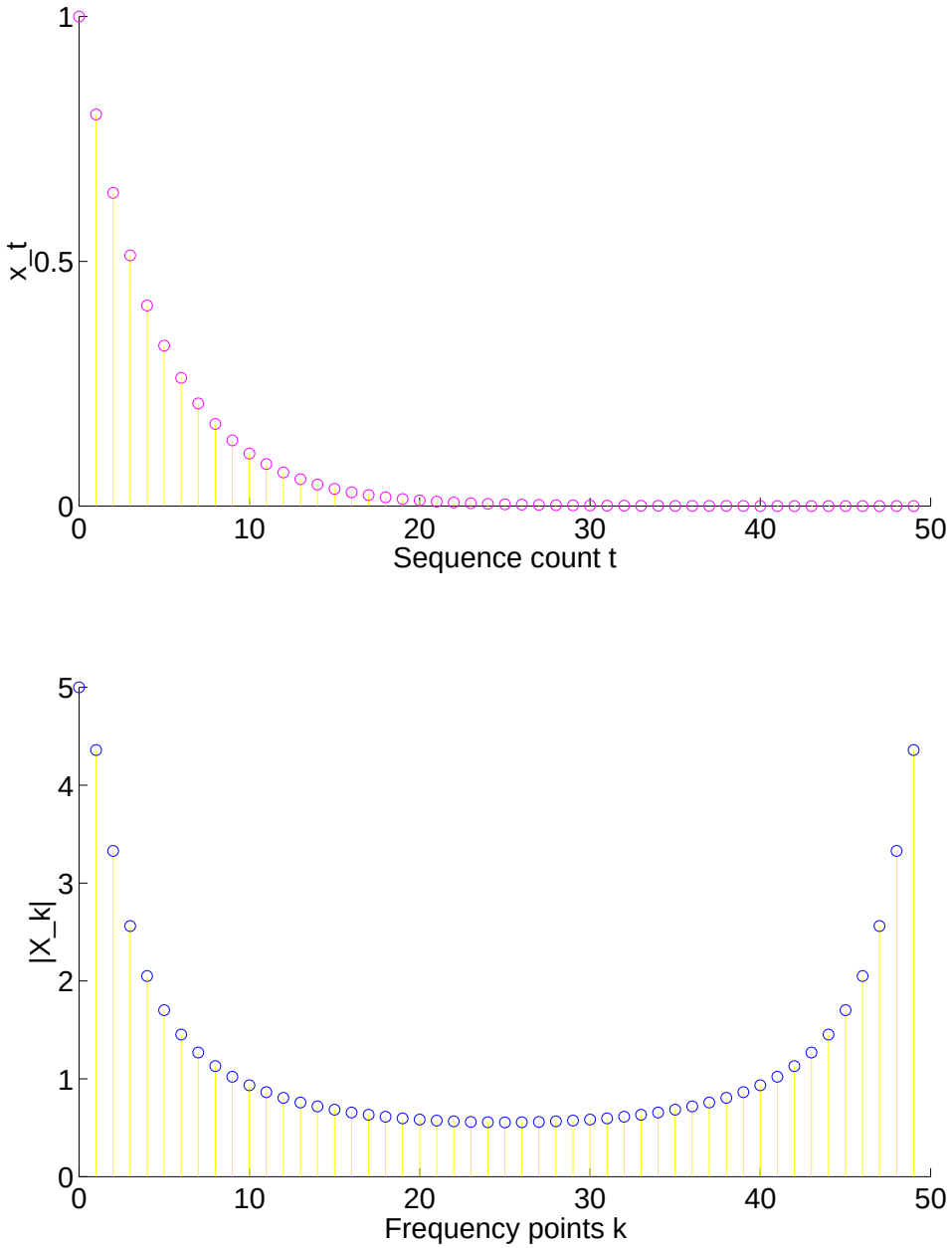


Figure 6.4: Plot of the sequence $x[n] = a^n$, $0 < a < 1$ (top) and the magnitude of its DFT (bottom), where $a = 0.8$ and $N = 50$.

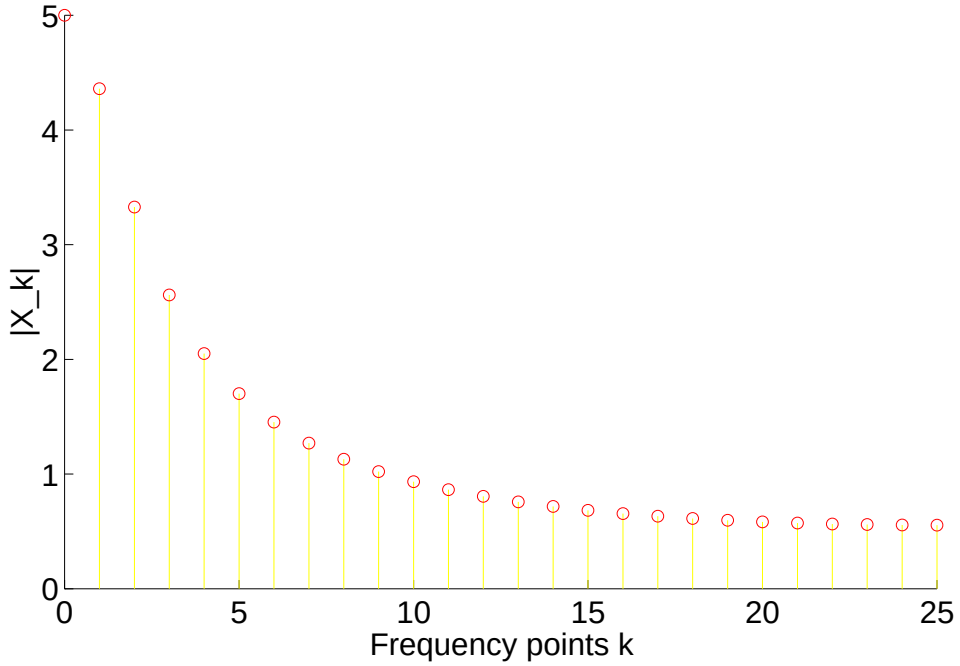


Figure 6.5: The magnitude of DFT for the signal 6.4 top is plotted for the first half of its range.

6.2.4 Computing the DFT Directly

Later on, I will introduce a method for computing the DFT efficiently. In view of the importance of the DFT in various digital signal processing applications, such as linear filtering, correlation analysis, and spectrum analysis, its efficient computation is a topic that has received considerable attention by many mathematicians, engineers, and applied scientists. But, let's first consider the direct, brute-force approach.

To compute the DFT, we need to compute the sequence $\{X[k]\}$ of N complex-valued numbers given another sequence of data $\{x[n]\}$ of length N, according to the formula (6-8):

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-jnk2\pi/N}, \quad k = 0, 1, 2, \ldots, N-1$$

In general, the data sequence $x[n]$ is also assumed to be complex valued. We observe that for each value of k, direct computation of $X[k]$ involves N complex multiplications ($4N$ real multiplications and $2N$ real additions) and $N-1$ complex additions ($2N-2$ real additions). Consequently, to compute all N values of the DFT requires $4N^2$ complex multiplications and $4N^2 - 2N$ complex additions. So, the direct approach is an $O(N^2)$ algorithm, and you should recognize that this is not terribly efficient to say the least.

6.2.5 The Fast Fourier Transform Algorithm

The development of computationally efficient algorithms for the DFT is made possible if we adopt a divide-and-conquer approach, called Fast Fourier Transform (FFT). This approach is based on the decomposition of an N point DFT into successively smaller DFTs, in a manner similar to how mergesort or quicksort divide large sorting problems into smaller sorting problems.

A DFT of length N can be rewritten as the sum of two DFTs, each of length $N/2$. One of the two is formed from the even-numbered points of the original N, while the other uses the odd-numbered points. This division can be arrived at straightforwardly:

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n]e^{-jnk2\pi/N} \\ &= \sum_{n=0}^{N/2-1} x[2n]e^{-jk(2n)2\pi/N} + \sum_{n=0}^{N/2-1} x[2n+1]e^{-jk(2n+1)2\pi/N} \\ &= \sum_{n=0}^{N/2-1} x[2n]e^{-jkn2\pi/(N/2)} + e^{-jk2\pi/N} \sum_{n=0}^{N/2-1} x[2n+1]e^{-jkn2\pi/(N/2)} \\ &= X[k]^{even} + e^{-jk2\pi/N} X[k]^{odd}, \quad k = 0, 1, 2, \ldots, N-1 \end{aligned} \tag{6-33}$$

where

$$X[k]^{even} = \sum_{n=0}^{N/2-1} x[2n]e^{-jkn2\pi/(N/2)} \tag{6-34}$$

$$X[k]^{odd} = \sum_{n=0}^{N/2-1} x[2n+1]e^{-jkn2\pi/(N/2)} \tag{6-35}$$

where $k = 0, 1, 2, \ldots, N/2 - 1$.

$X[k]^{even}$ denotes the k^{th} component of the DFT of length $N/2$ formed from the even components of the original $x[n]$, while $X[k]^{\text{odd}}$ is the corresponding transform of length $N/2$ formed from the odd components. The transforms $X[k]^{even}$ and $X[k]^{odd}$ are periodic in k

with length $N/2$. So each is repeated through two cycles to obtain $X[k]$:

$$\begin{aligned} X[k+N/2]^{even} &= \sum_{n=0}^{N/2-1} x[2n]e^{-j(k+N/2)n2\pi/(N/2)} \\ &= \sum_{n=0}^{N/2-1} x[2n]e^{-jkn2\pi/(N/2)} = X[k]^{even} \end{aligned} \tag{6-36}$$

$$\begin{aligned} X[k+N/2]^{odd} &= \sum_{n=0}^{N/2-1} x[2n+1]e^{-j(k+N/2)n2\pi/(N/2)} \\ &= \sum_{n=0}^{N/2-1} x[2n+1]e^{-jkn2\pi/(N/2)} = X[k]^{odd} \end{aligned} \tag{6-37}$$

because $e^{-jkn2\pi} = 1$.

The good thing is that this process can be continued recursively, just as in the divide-and-conquer sorting algorithms. Having reduced the problem of computing $X[k]$ to that of computing $X[k]^{even}$ and $X[k]^{odd}$, we can do the same reduction of $X[k]^{even}$ to the problem of computing the transform of its $N/4$ even-numbered elements and $N/4$ odd-numbered elements. In other words, we can define $X[k]^{even-even}$ and $X[k]^{even-odd}$ to be DFT of the points which are respectively even-even and even-odd on the successive subdivisions of the data. The same process can be applied to recursively divide $X[k]^{odd}$.

It is easy to imagine continuing this process until the DFT size is one, if N is a power of two. Though there are special algorithms if N is not a power of two, we can assume that the data is just padded with zeros up to the next power of two. Having done that, we will continue developing the FFT algorithm for N a power of two as a general-purpose DFT algorithm. With this condition on N, we can continue applying the dividing method until we have subdivided the data all they way down to transforms of length 1. The DFT of $x[n]$ for some n, and $N = 1$ is $x[n]$! Therefore,

$$X[k]^{even-even-odd-\cdots} = x[n], \quad \text{for some } n \tag{6-38}$$

The results of adjacent pairs of these one-point DFTs can be merged using (6-33): $X[k]^{even} + e^{-jk2\pi/N}X[k]^{odd}$. In this case $k = \{0, 1\}$, $N = 2$, and the one-point DFTs are the even and odd signal values themselves, so this reduces to,

$$\begin{aligned} X_2[0] &= x[even] + e^{-j(0)2\pi/2}x[odd] \\ &= x[even] + x[odd] \end{aligned} \tag{6-39}$$

$$\begin{aligned} X_2[1] &= x[even] + e^{-j(1)2\pi/2}x[odd] \\ &= x[even] - x[odd] \end{aligned} \tag{6-40}$$

where I have used the subscript "2" to indicate that this is a 2-point DFT.

Algorithm 6.1 Recursive FFT.

Require: x is a discrete function of time of length $N = 2^m$
Ensure: X is the DFT of x (also of length N)
  if $N = 1$ **then**
    Return x
  else
    Divide x into x^{even} and x^{odd} {the even and odd numbered points}
    $X^{even} = \text{FFT}(x^{even})$
    $X^{odd} = \text{FFT}(x^{odd})$
    for $k = 0, 1, 2, \ldots, N-1$ **do**
      $X_N[k] = X_{N/2}[k]^{even} + e^{-jk2\pi/N} X_{N/2}[k]^{odd}$
    end for
    Return X
  end if

Table 6.2: Example of the effect of bit reversal on an eight-element input vector.

Input		Bit-Reversed Result	
Decimal	Binary	Binary	Decimal
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

This is the basic FFT. High-level pseudocode for a recursive implementation is given in algorithm 6.1, where I have used subscripts to distinguish between the values of the $N/2$-point FFT and the N-point FFT. The base case for the recursion is that the length of the signal is one. As I'm sure you're familiar, recursive algorithms are short, simple, and elegant, but not always as efficient as non-recursive ones. There is a non-recursive approach to the even and odd dividing, and it is really neat, so I'll go over it.

We start with the problem of how to sort even and odd numbers and how to know which n goes to which pattern of $X[k]^{even-\cdots}$ and $X[k]^{odd-\cdots}$. The successive subdivisions of the data into even and odd parts are tests of successive low-order (least significant) bits of the binary representation of n. This is true because the least significant bit of an binary number is what determines if it is even or odd, and the act of dividing a binary number by two is equivalent to shifting it to the right one bit, which makes the next bit the new least significant bit. We take the original vector of data $x[n]$ and rearrange it into bit-reversed order (see table 6.2

for an example for $N = 8$), so that the individual numbers are not in order of n, but of the number obtained by bit-reversing n. The function of bit-reversing is to take care of all those even and odd divisions to the final list for N one-point transforms!

How do we recombine these non-recursively to produce the result? Starting with the one-point transforms, we combine adjacent pairs to get two-point transforms according to (6-33), then combine adjacent pairs of pairs of 4-point transforms, and so on, until we combine $N/2$ point transforms into the final transform of N total data points.

Algorithm 6.2 Iterative FFT.

Require: x is a discrete function of time of length N
Ensure: X is the DFT of x (also of length N)
  if N is not a power of two **then**
    Pad x with zeros until its length is the next power of two
  end if
  Shuffle the $x[n]$ in bit-reversed order
  $Size = 2$ {$Size = 1$ DFTs already done}
  while $Size < N$ **do**
    Compute $N/Size$ DFTs from the existing ones as $X_{Size/2}[k]^{even} + e^{-jk2\pi/Size} X_{Size/2}[k]^{odd}$

    $Size = Size \times 2$
  end while

In summary, the bit-reverse-based nonrecursive FFT algorithm is given in algorithm 6.2. This is illustrated in figure 6.6.

As far as the computational complexity of this algorithm is concerned, each iteration of the loop is $O(N)$ (we're computing $N/Size$ DFTs, each with $Size$ elements), and the loop executes $\log_2 N$ times, so the whole algorithm is $O(N \log_2 N)$. Directly computing the DFT (i.e., not using the FFT algorithm) takes $O(N^2)$.

Example: 8-Point FFT

Let's compute the FFT of the signal, $x[n] = \{0, 1, 0, 1, 0, 1, 0, 1\}$, $n = \{0, 1, 2, 3, 4, 5, 6, 7\}$. The first thing we'll do is write the signal out as a 1D array, with indices in binary:

0	1	0	1	0	1	0	1
000	001	010	011	100	101	110	111

Next, we sort the elements according to their bit-reversed indices:

0	0	0	0	1	1	1	1
000	100	010	110	001	101	011	111

Each of these elements is already a 1-point FFT; we now begin merging them into 2-point ones ($X_2[0]$ and $X_2[1]$), as in equations (6-39) and (6-40):

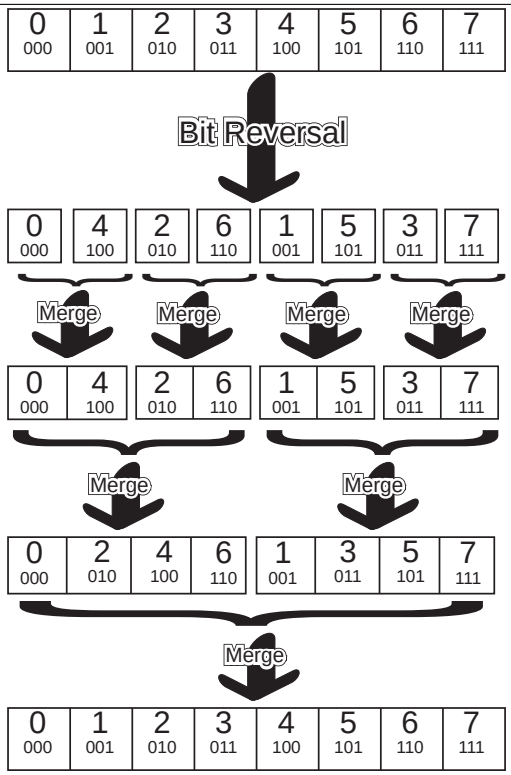


Figure 6.6: Schematic outline of an iterative 8-point FFT

0	0	0	0	2	0	2	0
0	1	0	1	0	1	0	1

Now we have four 2-point FFTs; we now merge adjacent pairs of points. So, we will be computing $X_4[k]$ for $k = \{0, 1, 2, 3\}$. To compute elements 2 and 3, we need to remember equations (6-36) and (6-37): we get $X_2[2] = X_2[0]$ and $X_2[3] = X_2[1]$. The resulting equations are:

$$\begin{aligned}
X_4[0] &= X_2[0]^{even} + e^{-j(0)2\pi/4} X_2[0]^{odd} \\
X_4[1] &= X_2[1]^{even} + e^{-j(1)2\pi/4} X_2[1]^{odd} \\
X_4[2] &= X_2[0]^{even} + e^{-j(2)2\pi/4} X_2[0]^{odd} \\
X_4[3] &= X_2[1]^{even} + e^{-j(3)2\pi/4} X_2[1]^{odd}
\end{aligned}$$

We can do the arithmetic in the exponents,

$$\begin{aligned}
X_4[0] &= X_2[0]^{even} + e^{0} X_2[0]^{odd} \\
X_4[1] &= X_2[1]^{even} + e^{-j\pi/2} X_2[1]^{odd} \\
X_4[2] &= X_2[0]^{even} + e^{-j\pi} X_2[0]^{odd} \\
X_4[3] &= X_2[1]^{even} + e^{-j3\pi/2} X_2[1]^{odd}
\end{aligned}$$

and then simplify,

$$X_4[0] = X_2[0]^{even} + X_2[0]^{odd}$$
$$X_4[1] = X_2[1]^{even} - jX_2[1]^{odd}$$
$$X_4[2] = X_2[0]^{even} - X_2[0]^{odd}$$
$$X_4[3] = X_2[1]^{even} + jX_2[1]^{odd}$$

yielding:

0	0	0	0	4	0	0	0
0	1	2	3	0	1	2	3

We repeat this process one more time to get the 8-point final result,

$$X_8[0] = X_4[0]^{even} + e^{-j(0)2\pi/8} X_4[0]^{odd}$$
$$X_8[1] = X_4[1]^{even} + e^{-j(1)2\pi/8} X_4[1]^{odd}$$
$$X_8[2] = X_4[2]^{even} + e^{-j(2)2\pi/8} X_4[2]^{odd}$$
$$X_8[3] = X_4[3]^{even} + e^{-j(3)2\pi/8} X_4[3]^{odd}$$
$$X_8[4] = X_4[0]^{even} + e^{-j(4)2\pi/8} X_4[0]^{odd}$$
$$X_8[5] = X_4[1]^{even} + e^{-j(5)2\pi/8} X_4[1]^{odd}$$
$$X_8[6] = X_4[2]^{even} + e^{-j(6)2\pi/8} X_4[2]^{odd}$$
$$X_8[7] = X_4[3]^{even} + e^{-j(7)2\pi/8} X_4[3]^{odd}$$

producing:

$$X_8[0] = X_4[0]^{even} + X_4[0]^{odd}$$
$$X_8[1] = X_4[1]^{even} + e^{-j\pi/4} X_4[1]^{odd}$$
$$X_8[2] = X_4[2]^{even} - jX_4[2]^{odd}$$
$$X_8[3] = X_4[3]^{even} + e^{-j3\pi/4} X_4[3]^{odd}$$
$$X_8[4] = X_4[0]^{even} - X_4[0]^{odd}$$
$$X_8[5] = X_4[1]^{even} + e^{-j5\pi/4} X_4[1]^{odd}$$
$$X_8[6] = X_4[2]^{even} + jX_4[2]^{odd}$$
$$X_8[7] = X_4[3]^{even} + e^{-j7\pi/4} X_4[3]^{odd}$$

The final result is:

4	0	0	0	-4	0	0	0
0	1	2	3	4	5	6	7

To interpret this, since we're dealing with a real-valued signal, we refer back to (6-26). $X_8[0]$ is the DC value, proportional to the mean value of the signal (in fact, it would be equal to the mean if we divide by N). We need only examine elements 1 through 4, since the spectrum is periodic. Only element 4 is nonzero. This makes sense, since the original signal repeated every other sample, which is right at the Nyquist frequency (π), corresponding to element 4.

Self-Test Exercises

See A.6 #2–4 for answers.

1. Prove that the DFT of $x[n]$ for any $n = m$ and $N = 1$ is $x[m]$.
2. Perform step-by-step division for the example given in table 6.2 to prove the final result is equal to the bit-reversed input.
3. Perform the 4-point FFT of the signal $x[n] = \{1, 2, 3, 4\}$, $n = \{0, 1, 2, 3\}$ by hand.

6.3 The inverse DFT

The inverse DFT (*IDFT*) can be computed using method similar to that used to compute the DFT, that is the FFT. Let's take look at (6-9) again:

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{jnk2\pi/N}, \quad n = 0, 1, 2, \ldots, N-1$$

Take the complex conjugate of both sides of this equation and multiply by N,

$$Nx[n]^* = \sum_{k=0}^{N-1} X[k]^* e^{-jnk2\pi/N} = \text{DFT}(X[k]^*) \tag{6-41}$$

Compare this to (6-8); the right side is just the forward DFT of $X[k]^*$. So, $x[n]$ can be computed from $X[k]$ as

$$x[n] = \frac{1}{N}[\text{DFT}(X[k]^*)]^* \tag{6-42}$$

Given the DFT of a signal $\{X[k]\}$ ($k = 0, 1, 2, \ldots, N-1$), the algorithm for computing its IDFT $\{x[n]\}$ using the FFT is:

1. take $\{X[k]\}$'s conjugate: $\{X[k]^*\}$
2. take $\{X[k]^*\}$'s forward transform using the FFT: FFT($\{X[k]^*\}$)
3. take the conjugate of the result and divide by N to get $\{x[n]\}$

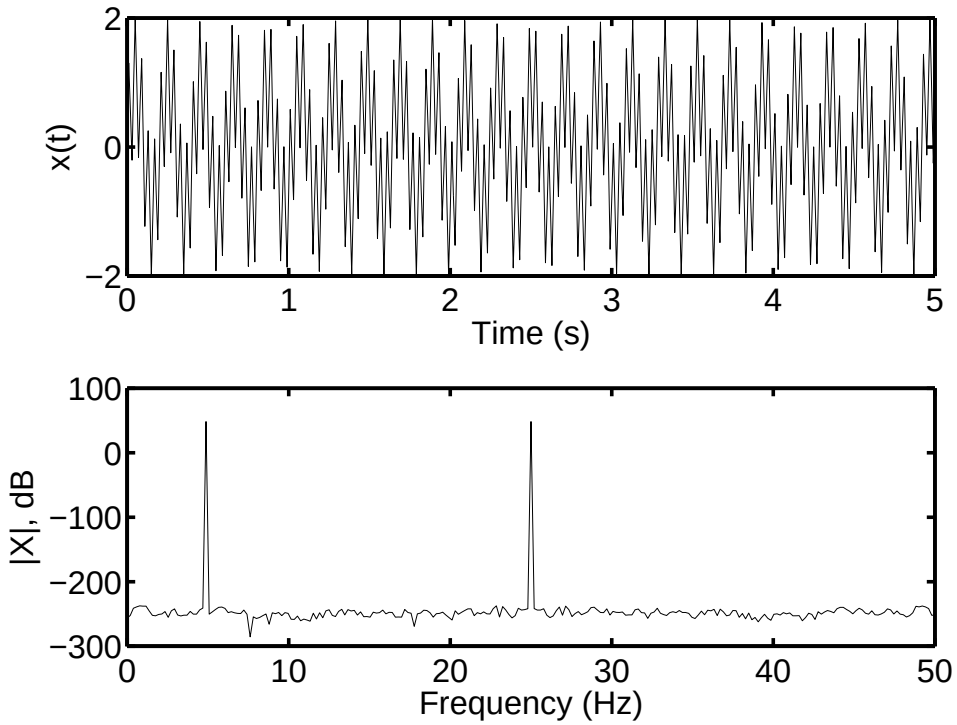


Figure 6.7: Sum of two sinusoids x with $N = 512$ samples (top) and its FFT $|X|$ (bottom). The sampling frequency is $f_s = 100$Hz, $f_1 = 25\Delta f = 4.8828$Hz and $f_2 = 128\Delta f = 25.0$Hz, where frequency step $\Delta f = f_s/N = 0.1953$Hz/sample.

6.3.1 Example: Sum of Two Sinusoids

Let's investigate the FFT using the example of two sinusoids,

$$x(t) = \sin 2\pi f_1 t + \sin 2\pi f_2 t \tag{6-43}$$

I'll set the sampling rate to $f_s = 100$Hz, so the sampling interval is $T_s = 1/f_s = 0.01$s. A finite-length (N point) segment of the digital version of above signal becomes,

$$x[n] = \sin(2\pi f_1 n T_s) + \sin(2\pi f_2 n T_s), \quad n = 0, 1, \ldots, N-1 \tag{6-44}$$

The corresponding frequency step in Hz is $\Delta f = f_s/N$, so $\hat{f} = k\Delta f$, $k = 0, 1, 2, \ldots, N/2$. First, let's use $N = 512$ and set the frequency components of the signal to be $f_1 = 25\Delta f = 4.8828$Hz and $f_2 = 128\Delta f = 25.0$Hz. The signal and its FFT result are shown on the top and bottom of figure 6.7.

As we expected, the two components appear in locations 4.8828Hz and 25.0Hz. Their magnitudes are around 300dB. Theoretically, all other value should be zero except these two. The nonzero values away from these two frequencies represent unavoidable computational errors introduced.

Now, let's choose another a pair of frequency values, $f_1 = 5$Hz and $f_2 = 25$Hz and check the result in figure 6.8 (top).

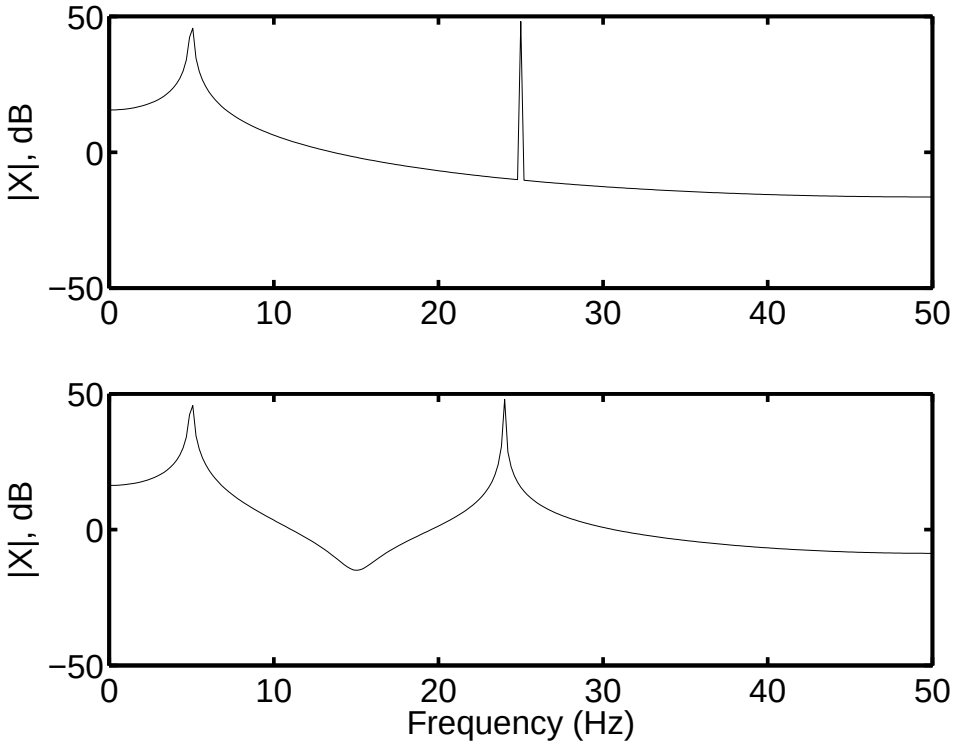


Figure 6.8: Similar input signals as in figure 6.7, but here the two frequencies are $f_1 = 5$Hz and $f_2 = 25$Hz for the top plot and $f_1 = 5$Hz and $f_2 = 24$Hz for the bottom.

How about $f_1 = 5$Hz and $f_2 = 24$Hz?, See figure 6.8 (bottom). In both cases in this figure, the peaks are below 40dB, down from 300dB. Why? This is caused by the discrete nature of the frequency representation and the fact that one or two of the signal's frequency components are not lined up with the set of discrete frequencies $\{\hat{f}\}$ — they fall in between two DFT points. For the pair $f_1 = 5$Hz and $f_2 = 25$Hz, $f_1 = 5$Hz falls between the points $k = 25$ and 26 and $f_2 = 25$Hz is lined up with $k = 128$. So, the first peak is broad and its energy shows up in significant amounts at all of the DFT frequencies. The second one is narrow and sharp, but because of the first peak's energy distribution, its amplitude is low too (actually, the amplitudes of all the "in between" points have been raised). For the second pair of $f_1 = 5$Hz and $f_2 = 24$Hz, neither are lined up so both of them are broad and low.

6.4 Power Leakage [Optional]

Power leakage is the phenomenon where power at a particular frequency (called the *center frequency* or *central lobe*) "leaks" into neighboring frequencies, which results in the center frequency peak being reduced, the peak width becoming broader, and nearby frequencies' (or side lobes') amplitudes increasing. This happens when the signal abruptly changes, for

instance suddenly turning on or off. Let's see the case of a truncated signal.

When we compute a signal's spectrum, values of the signal for all time are required. However, in practice, we observe signals for only finite durations. Therefore, the spectrum of a signal can only be approximated from a finite data record. Let's say we have an analog signal, we sample it at a rate f_s, and we limit the duration of the signal to the time interval $T_0 = NT_s$, where N is the number of samples and $T_s = 1/f_s$ is the sample interval. We denote the original, infinite-duration discrete signal as $\{x[n]\}$ and duration limited signal as $\{y[n]\}$. This is equivalent to multiplying $\{x[n]\}$ by a rectangular window $w[n]$ of length N. That is,

$$y[n] = x[n]w[n] \tag{6-45}$$

where

$$w[n] = \begin{cases} 1 & 0 \leq n \leq N-1 \\ 0 & otherwise \end{cases} \tag{6-46}$$

The rectangular window $w[n]$ sharply chopped the original signal to get the finite signal $y[n]$.

I'll use a sinusoid as an example for the signal $x[n]$,

$$x[n] = \cos \hat{\omega}_0 n \tag{6-47}$$

Using Euler's formula, the signal also can be written as

$$x[n] = \frac{1}{2}\left(e^{j\hat{\omega}_0 n} + e^{-j\hat{\omega}_0 n}\right) \tag{6-48}$$

The windowed version of this signal is

$$y[n] = \frac{1}{2}\left(e^{j\hat{\omega}_0 n} + e^{-j\hat{\omega}_0 n}\right) w[n] \tag{6-49}$$

Setting $\hat{\omega}_k = k2\pi/N$ (the N radian/sample frequency points in the FFT) in the DFT formula and applying it,

$$\begin{aligned} Y[k] \equiv Y(\hat{\omega}_k) &= \sum_{n=0}^{N-1} w[n]\frac{1}{2}(e^{j\hat{\omega}_0 n} + e^{-j\hat{\omega}_0 n})e^{-j\hat{\omega}_k n} \\ &= \frac{1}{2}\sum_{n=0}^{N-1} w[n](e^{-j(\hat{\omega}_k - \hat{\omega}_0)n} + e^{-j(\hat{\omega}_k + \hat{\omega}_0)n}) \\ &= \frac{1}{2}(W(\hat{\omega}_k - \hat{\omega}_0) + W(\hat{\omega}_k + \hat{\omega}_0)) \end{aligned} \tag{6-50}$$

where $W(\hat{\omega})$ is the DFT of the window $w[n]$. Actually, $w[n]$ can be viewed as the long pulse

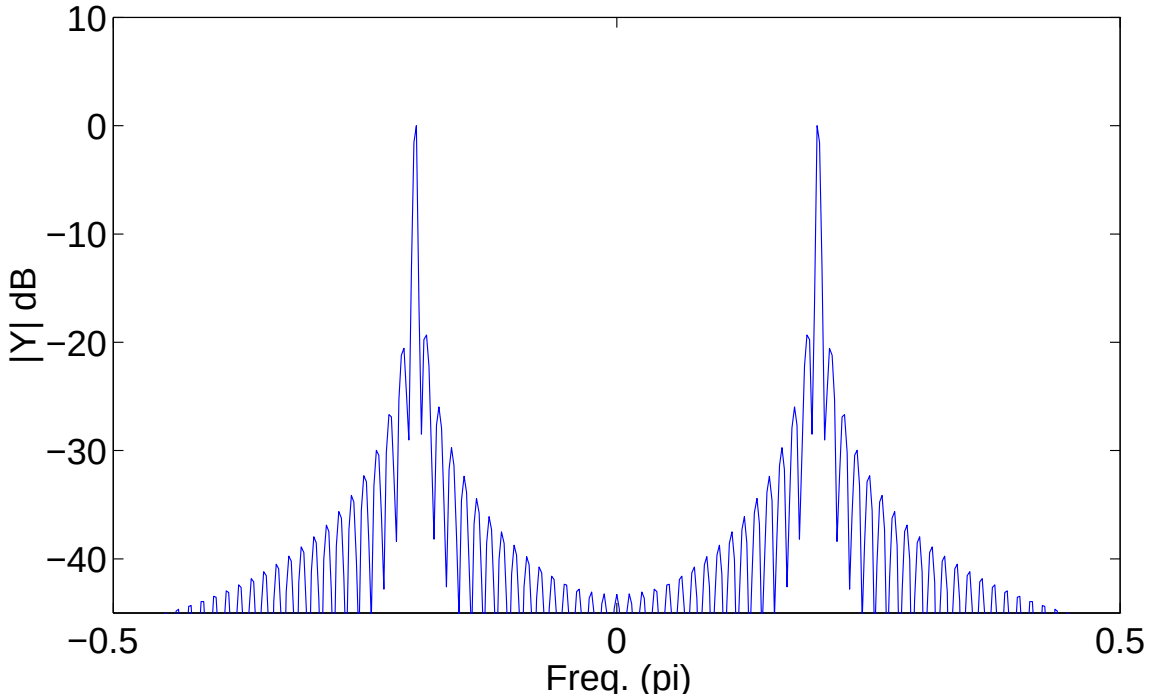


Figure 6.9: Magnitude spectrum of windowed signal $x[n] = \cos \hat{\omega}_0 n$, $\hat{\omega}_0 = 0.2\pi$ and the rectangular window length is $N = 128$. This Illustrates the occurrence of power leakage.

we talked about in chapter 1. Its DFT is

$$\begin{aligned} W[k] \equiv W(\hat{\omega}_k) &= \sum_{n=0}^{N-1} w[n] e^{-j\hat{\omega}_k n} \\ &= \sum_{n=0}^{N-1} e^{-j\hat{\omega}_k n} \qquad (6\text{-}51) \\ &= \frac{\sin(\hat{\omega}_k N/2)}{\sin \hat{\omega}_k/2} e^{-j\hat{\omega}_k (N-1)/2} \qquad (6\text{-}52) \end{aligned}$$

Figure 6.9 is a plot of $Y(\hat{\omega}_k)$ with a window length of $N = 128$, and $\hat{\omega}_0 = 0.2\pi$. The power of the original signal sequence $x[n]$ — concentrated at a single frequency $\hat{\omega}_0 = 0.2\pi$ — has been spread by the rectangular window into the entire frequency range. That is called *power leakage*. The rectangular window's effect is to cut out a piece of the original "long" digital signal; this action is equivalent to turning the signal suddenly on at $n = 0$ and off at $n = N$. This causes the power leakage.

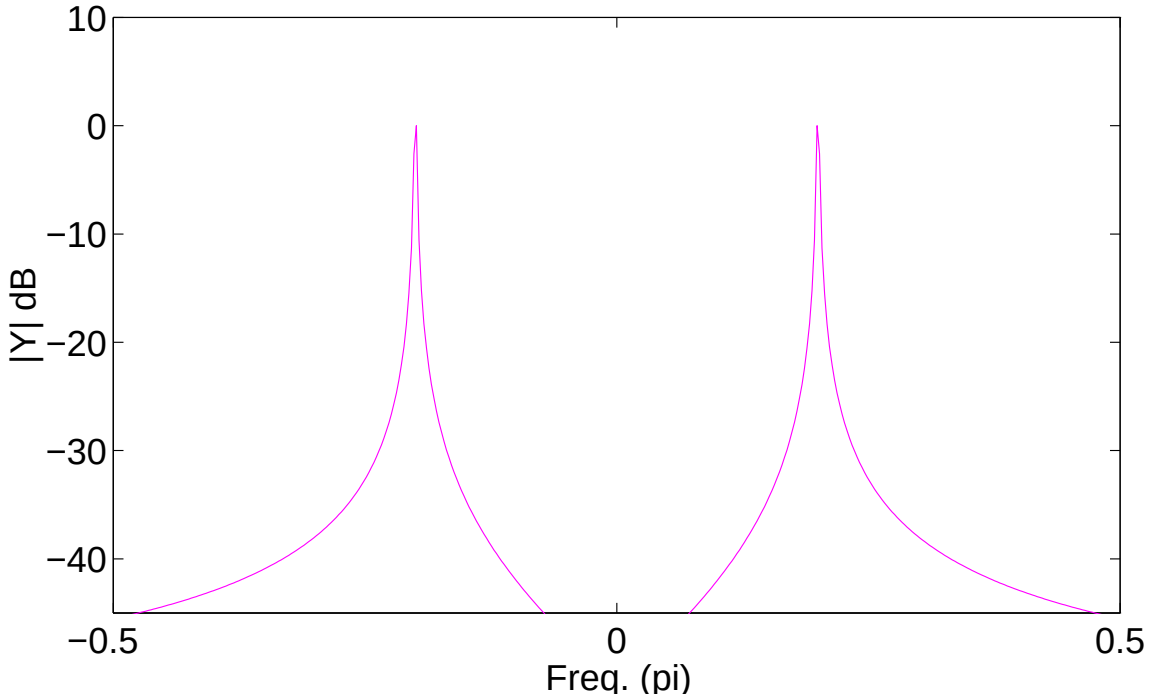


Figure 6.10: Magnitude spectrum of a one-sided signal $x[n] = \cos \hat{\omega}_0 n$ with $\hat{\omega}_0 = 0.2\pi$.

Let's see what happens if we let the window grow infinity long (or $N \to \infty$), that is

$$\begin{aligned} W[n] \equiv W(\hat{\omega}_k) &= \sum_{n=0}^{\infty} w[n] e^{-j\hat{\omega}_k n} \\ &= \sum_{n=0}^{\infty} e^{-j\hat{\omega}_k n} \end{aligned} \tag{6-53}$$

This is an infinite geometric series. Its common ratio is $e^{-j\hat{\omega}_k}$, so

$$\begin{aligned} W[k] &= \frac{1}{1 - e^{-j\hat{\omega}_k}} \\ &= \frac{1}{2je^{-j\hat{\omega}_k/2} \sin(\hat{\omega}_k/2)} \\ &= \frac{-je^{j\hat{\omega}_k/2}}{2\sin(\hat{\omega}_k/2)} \end{aligned} \tag{6-54}$$

Substituting this result back into (6-50), we obtain the spectrum of a one-sided signal x_t. This is plotted in figure 6.10.

Comparing (6-52) and (6-54), the magnitude of the former has N zeros equally spaced on the frequency axis, except at the peak frequency $\hat{\omega}_0$, where there is a 0/0 situation, which becomes 1. All these zeros contribute to the spectrum's oscillation. The magnitude of the latter one does not have a zero.

Self-Test Exercise

See A.6 #5 for the answer.

1. Fill in the steps leading from (6-51) to (6-52).

6.5 Tradeoff Between Time and Frequency Resolution [Optional]

Windowing not only distorts the spectral estimate due to leakage effects, it also reduces spectral resolution. There exists a tradeoff between time and frequency resolution. Wider windows produce finer frequency resolutions, which means you have the ability to distinguish between nearby spectral peaks. However, it yields worse time resolution, meaning that, if you deal with a time-varying signal (a signal with spectrum varying along time), the longer the time window is, the more information is combined within it and the more it misrepresents *when* the spectral components occurred.

To illustrate the concept of frequency resolution, let's use a digital signal consisting of three frequency components,

$$x[n] = \cos\hat{\omega}_0 n + \cos\hat{\omega}_1 n + \cos\hat{\omega}_2 n \tag{6-55}$$

When $x[n]$ is truncated to N samples in the range $0 \le n \le N-1$, the windowed spectrum is

$$Y(\hat{\omega}_k) = \frac{1}{2}[W(\hat{\omega}-\hat{\omega}_0)+W(\hat{\omega}-\hat{\omega}_1)+W(\hat{\omega}-\hat{\omega}_2)+W(\hat{\omega}+\hat{\omega}_0)+W(\hat{\omega}+\hat{\omega}_1)+W(\hat{\omega}+\hat{\omega}_2)] \tag{6-56}$$

Examining (6-52) again, the first zero crossing on either side of the center frequency is the $\hat{\omega}_k$ that satisfies

$$\frac{\hat{\omega}_k N}{2} = \pi \tag{6-57}$$

which is the first frequency at which $\sin(\hat{\omega}_k N/2) = 0$, or $\hat{\omega}_k = 2\pi/N$. The width of the center lobe is twice this, because there is no zero at $\hat{\omega}_k = 0$. The center lobes of the two window functions $W(\hat{\omega} - \hat{\omega}_i)$ and $W(\hat{\omega} - \hat{\omega}_j)$ — corresponding to two frequency components $\hat{\omega}_i$ and $\hat{\omega}_j$ $(i \ne j)$ — will overlap if $|\hat{\omega}_i - \hat{\omega}_j| < 4\pi/N$. As a result, the two spectral lines (peaks) of x_t will be indistinguishable. Only if $|\hat{\omega}_i - \hat{\omega}_j| \ge 4\pi/N$ can we see two separate lobes in the spectrum $Y(\hat{\omega}_k)$. Therefore, our ability to resolve spectral lines of different frequencies is limited by the window main lobe width, that is $4\pi/N$. As window length N increases, the spacing between two resolvable frequencies decreases as $1/N$, and so we get better frequency resolution. Figures 6.11 and 6.12 illustrate this conclusion. In these two figures, the signal has the three components $\hat{\omega}_0 = 0.1\pi$, $\hat{\omega}_1 = 0.12\pi$ and $\hat{\omega}_2 = 0.3\pi$. The length of the window for 6.11 is $N = 16$, where the two close components $\hat{\omega}_0$ and $\hat{\omega}_1$ are indistinguishable. The window length for 6.12 is $N = 128$, and now we can see two peaks around the 0.1π area because the main lobes have become narrow. At the same time, the side lobes becomes lower,

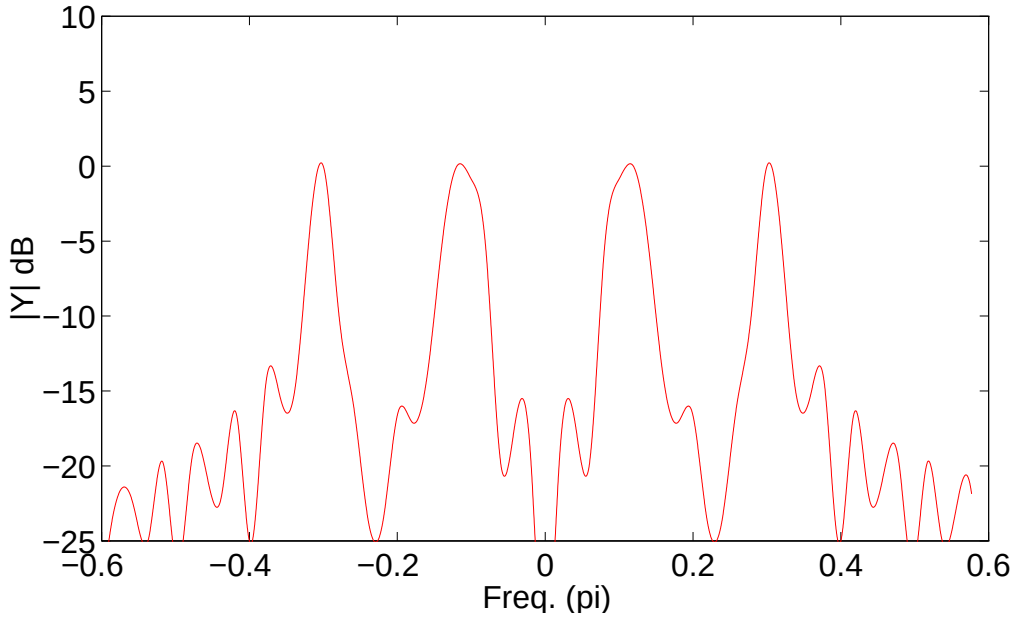


Figure 6.11: Magnitude spectrum of the signal (6-55) made up of three sinusoids. $\hat{\omega}_0 = 0.1\pi$, $\hat{\omega}_1 = 0.12\pi$, $\hat{\omega}_2 = 0.3\pi$, and the rectangular window length was $N = 16$. Because of the small window size, $\hat{\omega}_0$ and $\hat{\omega}_1$ are indistinguishable.

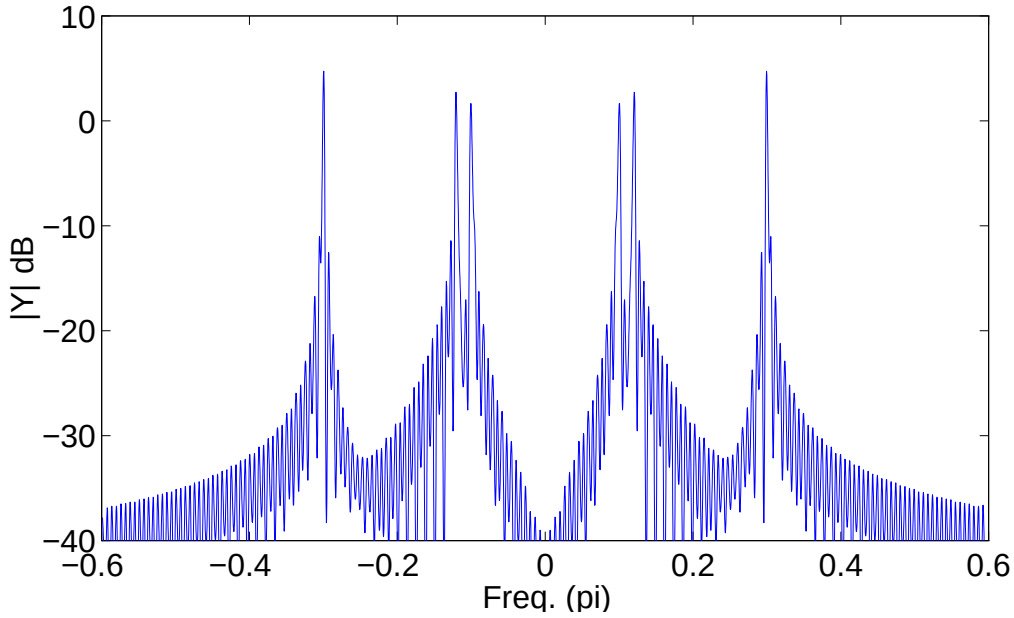


Figure 6.12: The same figure as 6.11, but the rectangular window length $N = 128$ allows separation of the $\hat{\omega}_0$ and $\hat{\omega}_1$ frequency components.

which means that less central frequency power leaks into neighboring frequencies. This is why the center frequency peak also becomes higher.

You might then ask if it is always good to extend the window length to get better frequency resolution. The answer is no. Depending on the signal, a longer segment might provide worse spectral information. This is the case when the signal is time-varying, that is the signal's spectrum changes along time (in every instant, the signal has different spectrum or frequency components). When we estimate the spectrum of a windowed signal using the DFT, we treat the segment as though it were a periodic signal (even though it is not), which repeats with period equal to the segment length. You can see that, for a signal that changes with time, this approach mixes together time domain information from the entire segment. The actual spectrum we get from a windowed signal is the *average* spectrum in that window. This means that, for a time-varying signal, a long window will misrepresent what the actual frequencies are (or, rather it will smear the frequencies together). The frequency information is all there, but we don't know *when* they occurred — we only know that it was sometime during the window. This suggests that the best approach would be to decrease the window size to get better time resolution and to get a more accurate time estimate. Unfortunately, as we just learned, short windows result in bad frequency resolution. This is the so-called time/frequency tradeoff. You must do your best to choose a window size that meets both your time and frequency resolution criteria.

A good way to understand this tradeoff is using a *spectrogram*, or short-time Fourier transform. This is a method to deal with time-varying signals to get a compromise result for both the time and frequency domains. A spectrogram is a sequence of FFT results produced by a sliding window. Sliding window processing starts at some time with a fixed window length, computes an FFT, slides the window by a fixed increment, recomputes the FFT, and repeats this sliding and FFT computation for the duration of interest. The result is usually plotted as a surface in a plot with the X axis being time, the Y axis frequency, and surface height being power. The resulting spectrogram is a two-dimensional function with independent variables time and frequency. The magnitude of the spectrogram gives the spectrum within windows at particular times.

An example of a bird call and its spectrogram is shown in figures 6.13 and 6.14. For each, the top plot is a section of the bird call waveform and the bottom is a spectrogram with two variables, time and frequency. Spectrogram color shows the magnitude of the spectrogram with red high and blue low, as shown by the color bar on the right.

In figure 6.13, the window length is 512 samples with an overlap of 511. Let's check the details of the spectral components with large magnitudes. Starting at the beginning, there is a component at around 1.5kHz which ends a little before 200 milliseconds. A second component is at around 3.3kHz starting at approximately 50ms continuing until 300ms, but becoming broader at around 100ms. Just before this component's end, it has a broad band frequency range of 1.7kHz to 3.5kHz (time ranging from 150 to 250ms). A similar pattern can be seen in the span of 400–700ms.

Compare this to figure 6.14, where the window length is 128 samples. From the point of view of time, the patterns look narrower, which means they have better temporal resolution

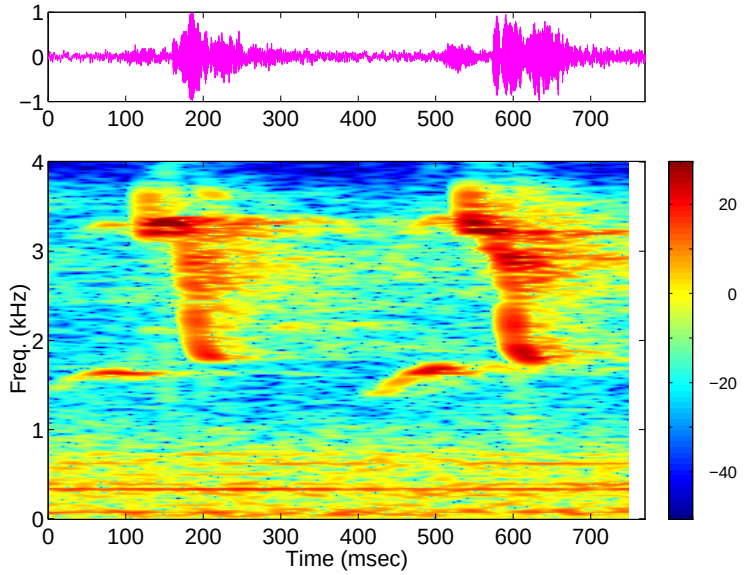


Figure 6.13: A bird call waveform (top) and its spectrogram (bottom). The window size is 512 samples and the overlap is 511 samples (increment is 1). Color indicates magnitude, with color key on the right of the figure.

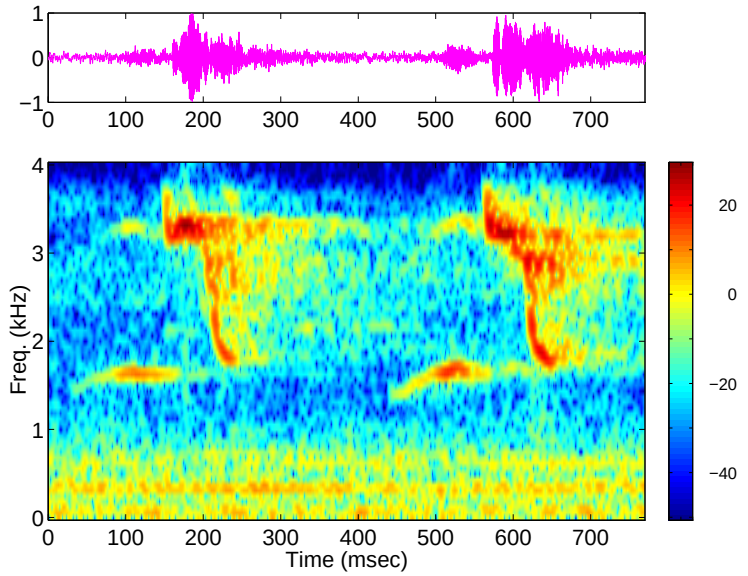


Figure 6.14: The same data as figure 6.13, but the window size is 128 samples and overlap is 127 (increment is 1).

(because of the shorter window). We can now see, for example, that there is a complex structure in the temporal evolution of the 1.7–3.5kHz frequency band. However, the patterns are broader and "blockier" along the frequency axis — the frequency resolution got worse (notice the 1.5kHz component in the area of 50–150ms) because of the short window.

Figures 6.15 and 6.16 are another pair of examples. They show similar effects from the tradeoff between time and frequency resolution.

6.6 Windowing [Optional]

In the discussion of power leakage, I mentioned that turning a signal on and off suddenly, abruptly truncating it (which is what a rectangular window does) will cause power leakage from the central lobe to side lobes. To reduce leakage when we select a segment of a signal, instead of using an abrupt truncation — like a rectangular window produces — we could select a data window function $w[n]$ that has lower side lobes in the frequency domain. Examples of some popular window functions include Hamming, Hann, Bartlett, Blackman, Gaussian, etc. Each of these has its own characteristics. I will give some examples of what these windows look like and what their effects are.

Hamming Window

The text book has some detail about the Hann window in example 9.5; here I introduce the Hamming window, which is very similar. The coefficients of an L-point Hamming window are computed from the equation,

$$\text{Hamming}[n] = 0.54 - 0.46\cos(2\pi n/(L-1)) \quad n = 0, 1, \ldots, L-1 \tag{6-58}$$

It consists of a cycle of a cosine, droping to 0.08 at the end-points and with a peak value of one. A plot of a Hamming window is shown in figure 6.17

Bartlett Window

The L-point Bartlett window is defined as:

- For L odd

$$\text{Bartlett}[n] = \begin{cases} \frac{2n}{L-1} & 0 \le n \le \frac{L-1}{2} \\ 2 - \frac{2n}{L-1} & \frac{L-1}{2} \le n \le L-1 \end{cases} \tag{6-59}$$

- For L even

$$\text{Bartlett}[n] = \begin{cases} \frac{2n}{L-1} & 0 \le n \le \frac{L-1}{2} \\ \frac{2(L-n-1)}{L-1} & \frac{L-1}{2} \le n \le L-1 \end{cases} \tag{6-60}$$

This is a triangular window with maximum height of one and zeros at samples 0 and $L-1$. It is plotted in figure 6.18.

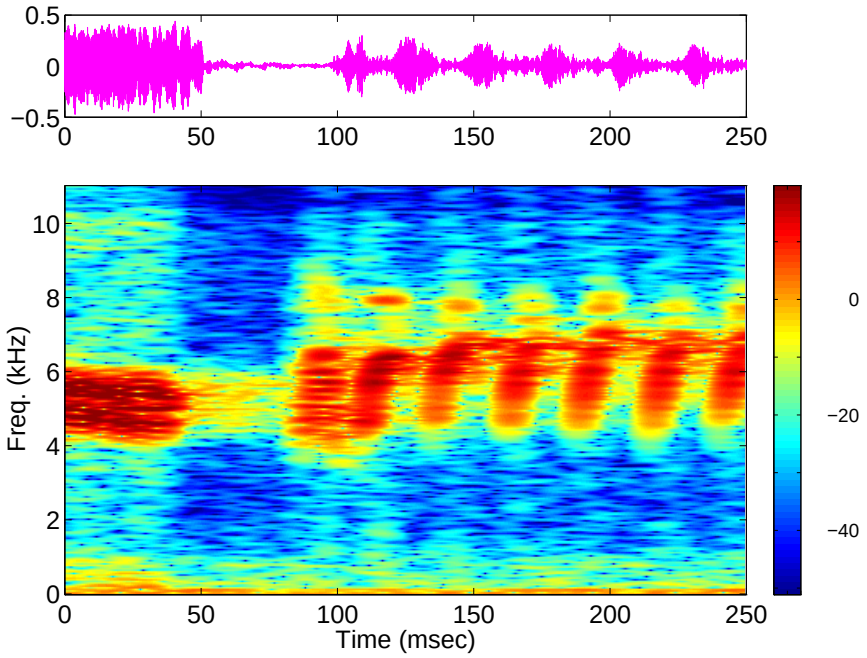


Figure 6.15: Another bird call waveform (top) and its spectrogram (bottom), with window size of 512 samples and overlap of 511 samples.

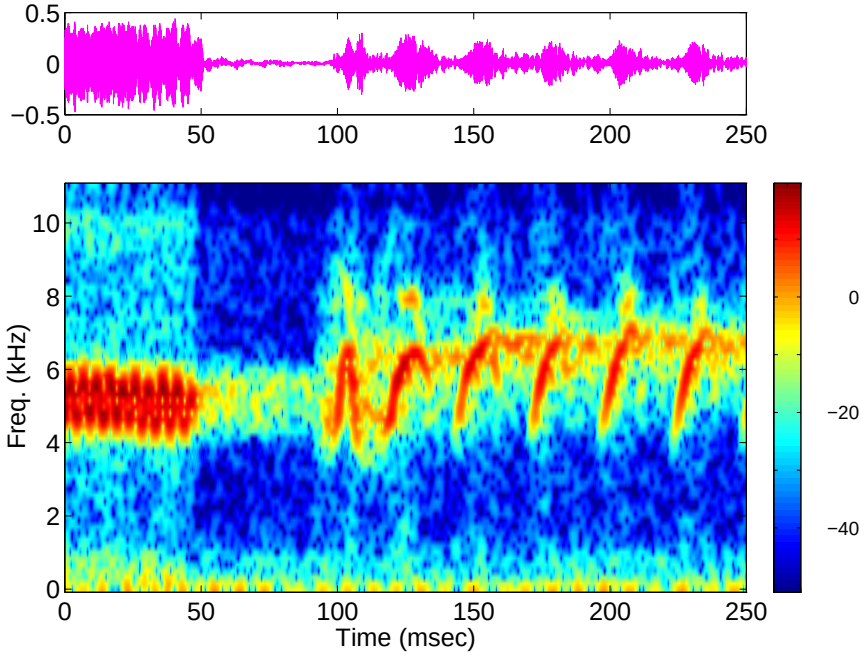


Figure 6.16: Same data as figure 6.15, but with window size of 128 samples and overlap of 127.

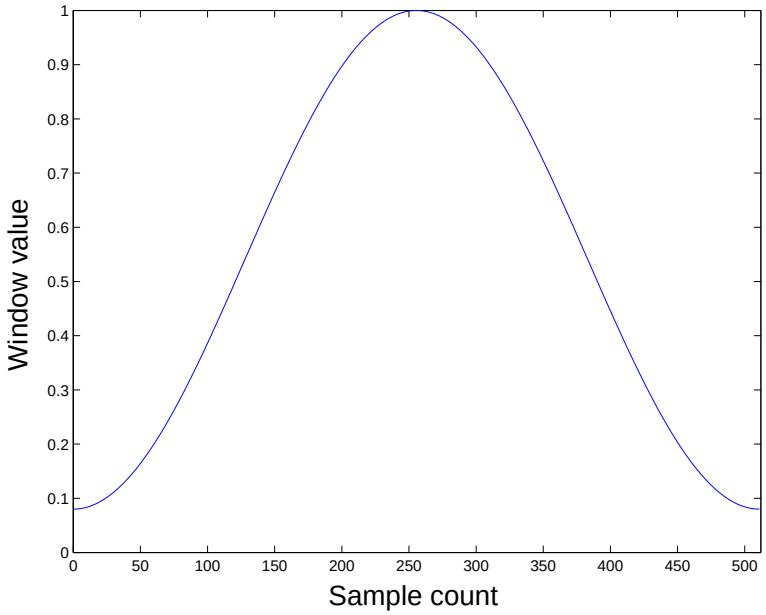


Figure 6.17: A 512-point Hann window in the time domain.

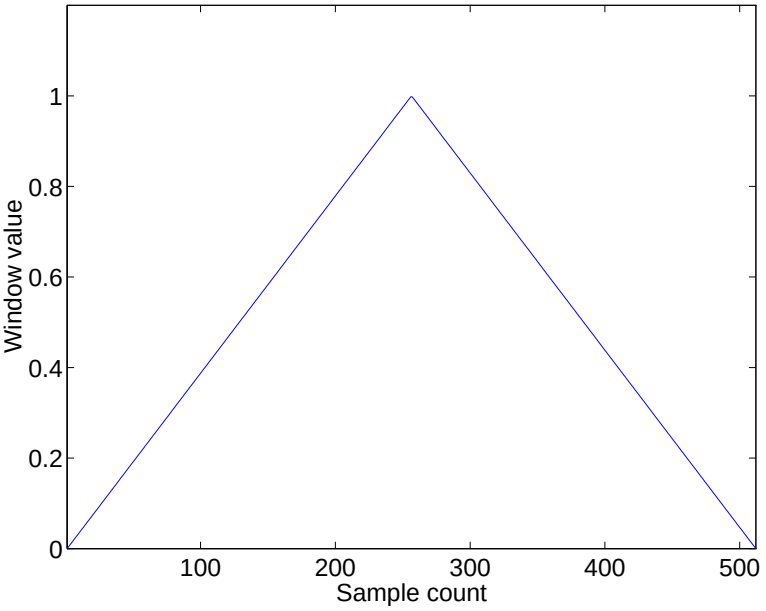


Figure 6.18: The 512-point Bartlett window in the time domain.

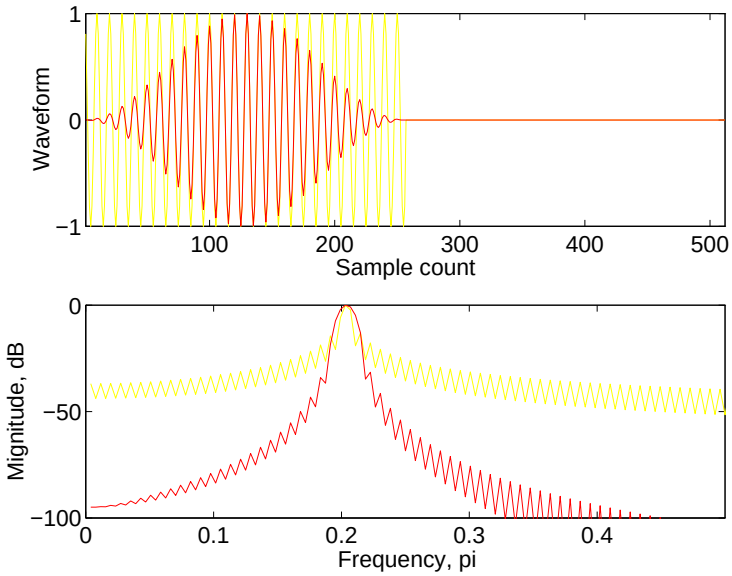


Figure 6.19: Hann windowed cosine waveform ($\hat{\omega}_0 = 0.2\pi$) and its spectrum, compared to rectangular window. The top is the cosine waveform windowed by rectangular (yellow) and Hann (red) windows, the bottom is their corresponding spectra.

Using Window Functions

How do we use window functions? For a general signal sequence $x[n]$, the time domain relationship between the windowed sequence $y[n]$ and original sequence is

$$y[n] = x[n]w[n] \tag{6-61}$$

where $w[n]$ is a window function in time domain. Previously, I discussed the special case of a rectangular window; now I will consider $w[n]$ as a general function.

Example: Windowed Sinusoid

Consider again a sinusoid signal $x[n]$,

$$x[n] = \cos(\hat{\omega}_0 n); \tag{6-62}$$

Instead of using a rectangular window, let's first use a Hann window and see the results. The windowed signal $y[n]$ is

$$y[n] = x[n]\,\mathrm{Hann}[n] = \cos(\hat{\omega}_0 n)\,\mathrm{Hann}[n] \tag{6-63}$$

where the Hann window is as described in (6-58). $y[n]$ is shown in figure 6.19 (top, red curve); its spectrum is the red curve in the bottom plot. Since $x[n]$ only has one component $\hat{\omega}_0 = 0.2\pi$, ideally there should be only one δ function peak at frequency 0.2π. However, as you now know, that is not likely to be the case because of power leakage. The magnitude

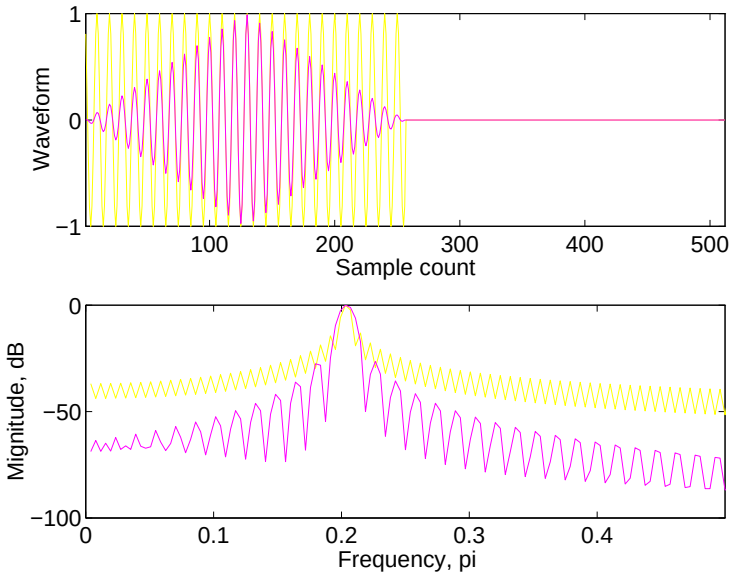


Figure 6.20: Bartlett windowed cosine waveform and its spectrum, compared to rectangular window. The top is the cosine waveform windowed by rectangular (yellow) and Bartlett (magenta), the bottom is their corresponding spectra.

value is gradually damped down away from the center lobe around $\hat{\omega}_0$. The leakage is smaller with the Hann window than a rectangular one (yellow curves). On the other hand, the rectangular windowed signal has a narrower peak than the Hann windowed one. So, we can see that the Hann window decreases power leakage by sacrificing peak resolution.

Let's see the result of using a Bartlett window. For the same cosine waveform, the results are shown in figure 6.20. We can reach similar conclusions about the Bartlett window. But, there is a difference between the two windows, as you can see in figure 6.21, which is a comparison between the Hann and Bartlett windows for the cosine wave. Though it appears that the Hann window is superior, there are a number of issues (beyond the scope of this course) that I haven't discussed; the choice of window function involves a number of tradeoffs and depends on the signals being processed and the goal of the processing.

Example: Windowed Bird Call

Here, I will once again examine the bird call spectrogram I introduced earlier. When I computed that spectrogram, I cheated: I used a Hann window, instead of a rectangular one. Now, instead of using a Hann window in figure 6.14, if I use rectangular one, I get figure 6.22. The increased power leakage into the side lobes of the two main frequency components is readily apparent. This leakage messes up the spectral appearance and affects how accurately we can estimate the shape of the frequency components. That illustrates how important it is to choose the right window.

Figure 6.23 presents a similar result for the bird call originally Hann windowed in figure 6.16.

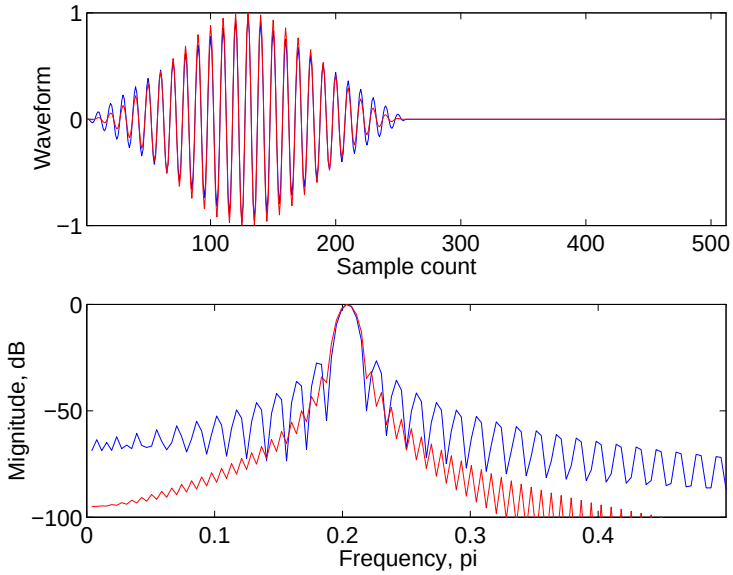


Figure 6.21: Comparison between Hann (red, lower curve) and Bartlett (blue, upper curve) windows for a cosine wave.

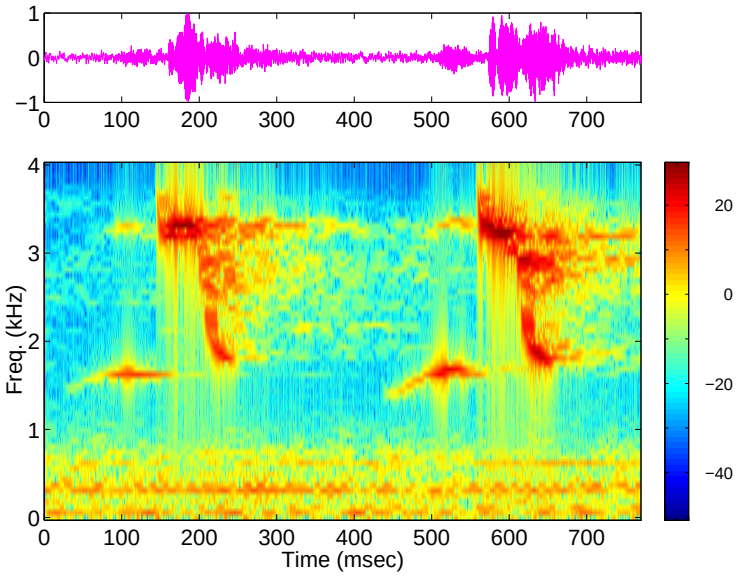


Figure 6.22: Similar spectrogram as figure 6.14, but here a rectangular window is used.

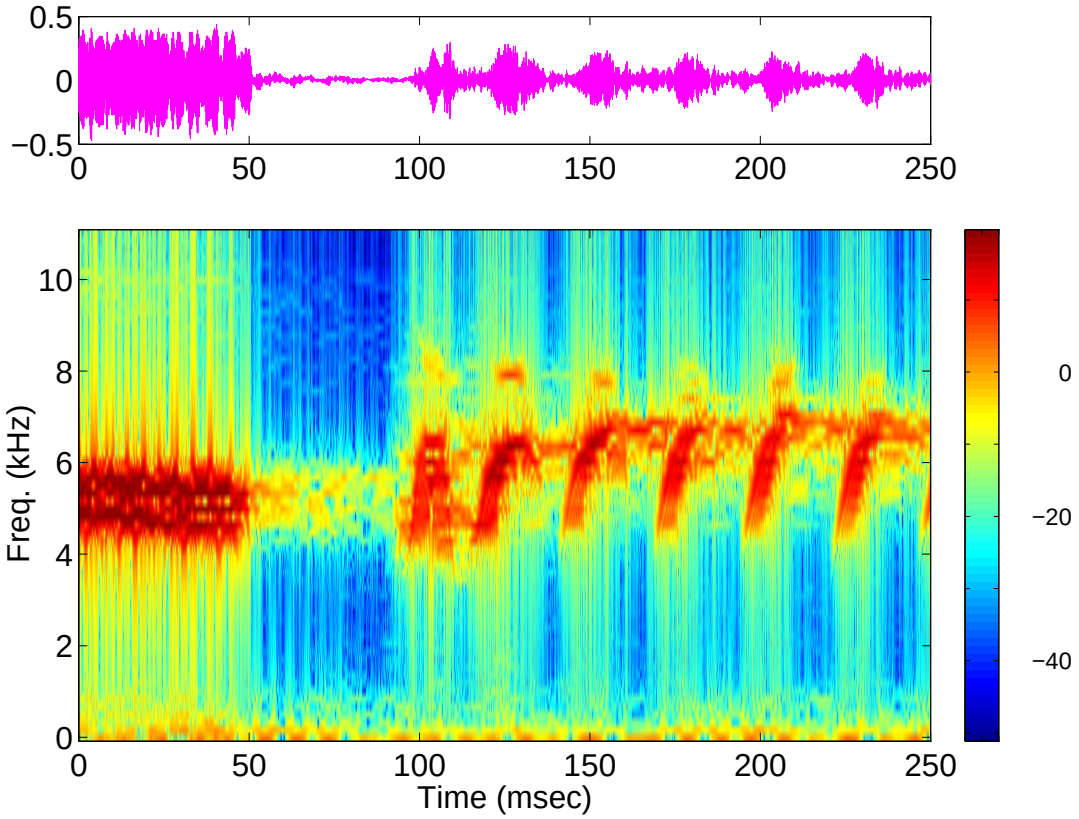


Figure 6.23: Similar spectrogram as figure 6.16, but here a rectangular window is used.

Self-Test Exercise

See A.6 #6 for the answer.

1. Plot the Hann and Hamming windows in the time domain and compare their shapes.

6.7 Problems

1. If $x(t) = 100 \sin 20\pi t$ for $-0.05 < t < 0.05$ and $x(t) = 0$ for $t < -0.05$ and $t > 0.05$, evaluate its Fourier transform at $\omega =$ (a) 0, (b) 10π, (c) -10π, (d) 20π.

2. Trace by hand the 8-point FFT algorithm for the following signals, showing the intermediate steps. Plot their magnitude spectra.

   (a) $x[n] = \{-1, 1, -1, 1, -1, 1, -1, 1\}$

   (b) $x[n] = \{0, 1, 1, 0, 1, 1, 0, 1\}$

   (c) $x[n] = \{0, 1, 2, 0, 1, 2, 0, 1\}$

   (d) $x[n] = \{0, 1, 2, 3, 0, 1, 2, 3\}$

3. Use your favorite programming language to implement the iterative FFT defined in algorithm 6.2.

4. Use Matlab's built-in FFT subroutine to compute the following DFTs and plot the magnitudes $|X_k|$ of those DFTs using Matlab.

   (a) The 64-point DFT of the sequence

   $$x[n] = \begin{cases} 1 & n = 0, 1, \ldots, 15 \\ 0 & \text{otherwise} \end{cases}$$

   (b) The 64-point DFT of the sequence

   $$x[n] = \begin{cases} 1 & n = 0, 1, \ldots, 7 \\ 0 & \text{otherwise} \end{cases}$$

   (c) The 128-point DFT of the sequence in 4b.

   (d) The 64-point DFT of the sequence

   $$x[n] = \begin{cases} 10e^{n\pi/8} & n = 0, 1, \ldots, 64 \\ 0 & \text{otherwise} \end{cases}$$

   Answer the following questions:

   (e) What is the frequency interval between successive samples for the plots in (a–d)?

   (f) What is the value of the spectrum at zero frequency (DC value) in the plots in (a–d)?

5. It is common practice to *normalize* windows by assuring that the sum of their values is equal to one, i.e.,

   $$\sum_{n=0}^{L-1} w[n] = 1$$

   . Show that the Hamming window is normalized by dividing its values by $0.54L - 0.46$.

6. Given the bird call data at http://courses.washington.edu/css457/ebook/amoriole2-1.txt (4000 samples), and its sampling frequency $f_s = 8$kHz, use MATLAB to compute its spectrogram, plot the waveform and its spectrogram (use window sizes of 128 and 512 with Hann [hanning] windows and an increment of 1). Use only the base MATLAB toolbox (*not* the Signal Processing Toolbox). Submit the resulting figures and code.

6.8 Further Reading

- James H McClellan, Ronald W. Schafer, and Mark A. Yoder, *DSP First: A Multimedia Approach*, Prentice Hall, 1998, chapter 9.

7 Compression

This chapter investigates how we might reduce the data processing, transmission, and storage requirements for a multimedia system by reducing the number of bits needed for signal representation. I introduce the concept of *information* to quantify the *content* of multimedia data, and show how this is distinct from the *representation* of multimedia data. I show how the choice of representation affects the number of bits required to represent multimedia information — that multimedia data can be *compressed* without *loss*. I also show how additional compression can be achieve by sacrificing information content: *lossy* compression.

By the end of this chapter, you should understand the main kinds of compression algorithms, their chief features, and their tradeoffs. You will know when it is appropriate to sacrifice information for data size and when it is not. You will also have an idea of how knowledge of human perceptual capabilities is essential for designing lossy compression schemes. Finally, you will be able to implement some basic compression algorithms yourself.

7.1 Signals and Information

Up to this point, we've been implicitly assuming that a signal is sampled, quantized, and then processed. However, there is more to multimedia computing than just running a stream of samples through a filter. Among other functions, multimedia systems also need to store and transmit data (either among components within a single computer, or across networks to other machines). A critical issue for system performance is the *volume* of data that must be stored or moved around. For example, suppose that we are digitizing audio at CD quality. If a rate of 44,100 samples/second at 16bits/sample, what is the digital data rate in bits/second(answer in A.7 #1)? If we are digitizing high-quality video — 1k x 1k pixels/frame, 30 frames/sec, 24 bits/pixel — what is the bit rate (answer in A.7 #2)? Clearly, multimedia systems require not only high processing power (for real-time operation), but also high I/O bandwidth, large memory and storage capacity and speed, and fast networks. However, you should be familiar by now with the idea that a bit of thought can often lead to significant savings in algorithm run time or memory requirements. In this chapter, I will focus on the latter: how we can *encode* digital multimedia information to reduce its volume.

The fundamental ideas related to signal coding were developed by Shannon at Bell Labs in the late 1940s. He was concerned with transmitting a signal (either via radio or along a wire) so that it could be reconstructed reliably at the receiver despite any noise corruption

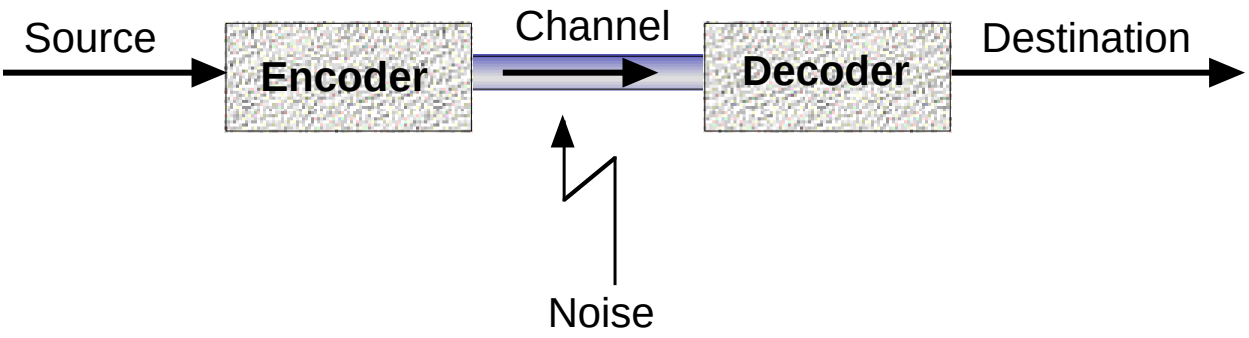


Figure 7.1: Shannon's model for coding information for transmission.

that might occur along the way. The basic model for this problem is presented in figure 7.1. Information (which we can think of as a bit stream) at the source is passed through some sort of encoder (which transforms the source bitstream into another one) and then transmitted along a *channel*. While in transmission, the bitstream may become corrupted by noise, which we can think of as randomly flipping bits with some probability. The goal of the decoder is to convert the received bitstream back into a replica of the original signal.

While the above problem is phrased in terms of coping with noise in a channel, it is intimately tied to the number of bits transmitted. The conceptual key to this is the separation of the *information content* in a signal from its *representation*. One goal of encoding is

to choose a representation that allows the underlying information to be preserved in the presence of noise (*channel coding*). The other goal — which is relevant to this chapter — is that the representation use the fewest possible bits to do the job (*source coding*). To accomplish this, we need first to quantify the *information content* of a signal: its *entropy*. The basis for a mathematical description of information is a common-sense one: information is something you don't already know. If I tell you something you don't already know, I've given you information; if I tell you something you already know, then you've received none.

In multimedia terms, we are talking about the information content in a digital signal. You might say that, unless you already know the signal being sent, the entire signal is new information to you. However, this is not true. For example, if the signal is a sampled sine wave, after you've received a few samples, you should be able to *predict* the next ones. If each sample is 16 bits, and you can predict the next sample's value to an accuracy of 14 bits (in other words, the 16-bit number that you predict is off by, on average, 2 bits), then the transmitted signal really only contains 2 bits of information per sample. One way to achieve compression, then, would be to choose an alternative representation for the signal in which each sample only took 2 bits — the number of bits sent would equal the signal's information content.

Web Links:

Introduction to data compression
http://www.faqs.org/faqs/compression-faq/part2/section-1.html

Entropy in Information & Coding Theory
http://www.math.psu.edu/gunesch/Entropy/infcode.html

Primer on Information Theory
ftp://ftp.ncifcrf.gov/pub/delila/primer.ps

LZW Data Compression http://www.dogma.net/markn/articles/lzw/lzw.htm

Practical Huffman coding http://www.compressconsult.com/huffman/

Interactive Data Compression Tutor
http://www.eee.bham.ac.uk/WoolleySI/All7/body0.htm

The Data Compression Library
http://dogma.net/DataCompression/

So, one scheme for compression is to remove *redundancies* in the data: that part of the data which conveys no additional information, given previous data. This is termed *lossless compression*, because no information is lost. Another approach is one which considers the use to which the data will be put, and selectively eliminates unneeded or unimportant information — *lossy compression*. These two coding schemes and their subcategories are presented in figure 7.2. I will discuss many of these in the rest of this chapter. In each case, I will be concerned with the following algorithm characteristics:

- the degree of information loss,
- the encoding complexity,
- and the decoding complexity.

Why do I separate out encoding and decoding complexity? Isn't decoding just the opposite of encoding? While that may be true conceptually, there is no guarantee that, for any particular coding scheme, encoding and decoding algorithms will have the same run time. Schemes for which they do are called *symmetric*; those for which they are not are *asymmetric*. For some applications, symmetric coding is necessary, while for others one end (typically

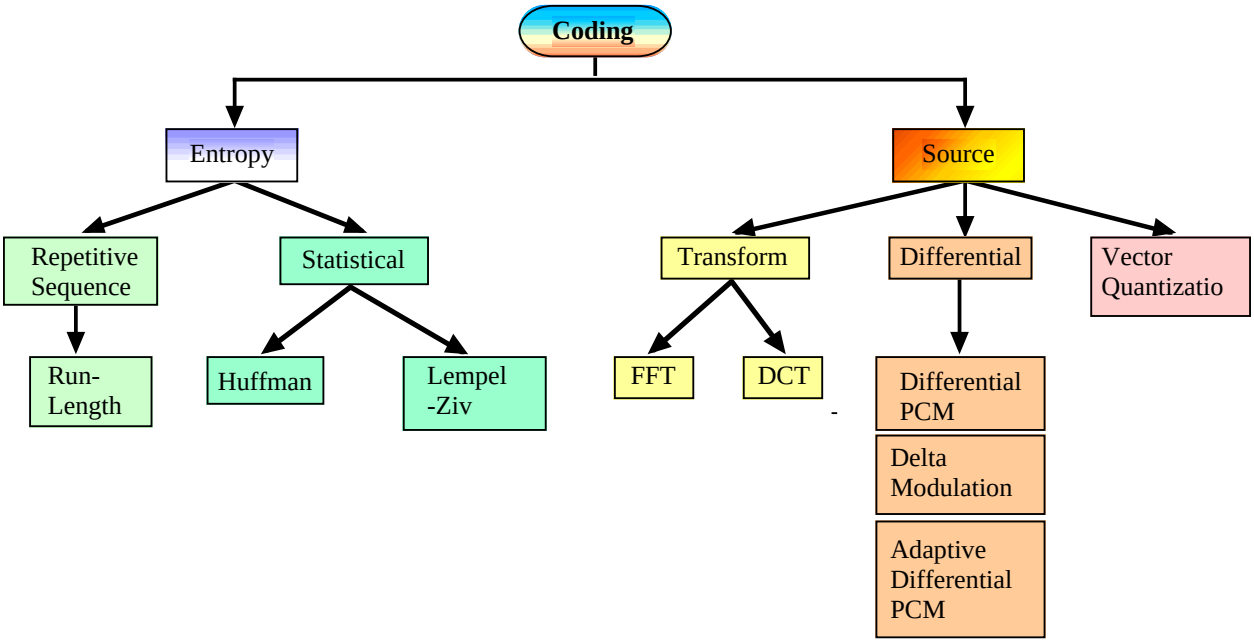


Figure 7.2: A taxonomy of coding/compression schemes.

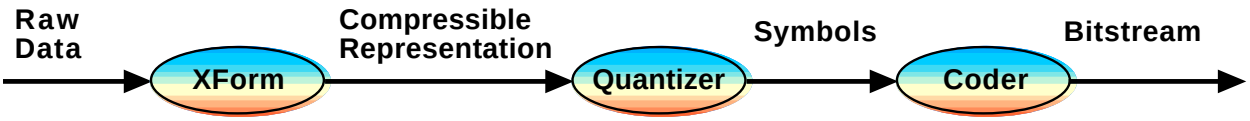


Figure 7.3: Generalized scheme for encoding.

the encoder) can be allowed to take significantly more time (presumably to produce greater compression or better signal "quality" given some level of compression). In symmetric applications, the hardware and the available processing time is usually also symmetric, while for asymmetric applications one end may have much faster hardware and/or more time.

So, we arrive at the general scheme for encoding in figure 7.3. The data to be encoded can first be transformed (the "xform" block) to make it more amenable to compression (to produce a more compressible representation). The goal of such a transform is essentially to *expose* the signal's underlying redundancy. For example, if the signal were a sine wave, its Fourier transform representation would be more easily compressible: it would have a single value at one frequency, as opposed to the original time domain function's sequence of samples. This representation may also make it easier to separate out components based on their "importance" (allowing more information to be eliminated from less important components). I'll discuss this last point in the sections on lossy compression.

Once the data has been transformed, it is then converted into a sequence of symbols

for transmission. This *quantization* step limits the number of different symbols to be used. So, though the original signal might have 8 bits per sample, it may not be necessary to retain all 256 symbols. In general, we may map each input symbol to an output symbol, or we may take N input symbols and produce one output symbol (this might be the case when "runs" of increasing inputs are common — we might substitute a single symbol or a shorter sequence for a stereotypical increasing sequence). Companding is one example of this kind of quantization: more symbols are allocated to quiet sections of an audio stream, where small amplitude differences are noticable, and fewer to loud sections, where much larger differences are needed for changes in volume to be noticable. This stream of symbols are then coded to produce a bit stream, which might represent each symbol with a varying number of bits, for example. So, first a quick self-test, and then we'll look at each of these blocks and compression schemes in more detail.

Self-Test Exercises

See A.7 #3–6 for answers.

1. If a signal is sent in which all samples have the same value, what is the information content in bits (ignoring the first sample)?
2. What kind of signal would have maximum information content?
3. Can you give an example of an application which would demand symmetric coding?
4. Can you give an example of an application which could allow asymmetric coding?

7.2 Entropy (Lossless) Compression

Lossless, or entropy, compression ignores the *semantics* (meaning) of the data. It is based instead purely on the statistics of the symbols in the data. These statistics can be the frequencies of different symbols (how often each occurs) or the existence of certain *sequences* of symbols. In the former case, we have statistical compression; in the latter, a category for which repetitive sequence compression is the simplest case.

7.2.1 Repetitive Sequence Compression

When two people are having a telephone conversation, it is common for there to be pauses when nobody is speaking. In still images, it is not unusual for large areas to have the same (usually, background) color. In video, areas that correspond to moving objects change from one frame to another while other, larger areas don't change. All of these situations have the same feature in their raw stream of samples: long sequences (1D, 2D, or 3D) which are identical. Many bits are used to send a relatively small amount of information.

The idea of *run-length encoding* (RLE) is simple: replace long sequences (*runs*) of identical samples with a special code that indicates the value to be repeated and the number of

times to repeat it. For 8 bits/sample data, we mught reserve one symbol (say, zero) as a "flag" to indicate the start of a run length code. A run would then be replaced with a zero, a byte containing the symbol to repeat (1–255), and one or more bytes as the repetition count (how many bytes to use we would have to decide ahead of time, based on what we know about typical run lengths for the particular kind of signals being processed). So, for example, a run of 112 'A's in a text file could be encoded as: *flag*, 'A', 112.

We can also extend RLE to work for cases where sequences of symbols, rather than just one, are repeated.

7.2.2 Statistical Compression

Statistical compression schemes work by assigning *variable-length codes* to symbols based on their frequency of occurrence. By assigning shorter codes to more frequently occurring symbols, the average number of bits per symbol can be reduced.

Huffman Coding

When we sample data, we almost always do so with a fixed number of bits per sample — a fixed number of bits per symbol. So, we can consider 8-bit sampling as quantization of a signal into 256 levels, or equivalently as representing it as a sequence of symbols, where there are 256 symbols available.

This is usually *not* the most space-efficient coding scheme, for the simple reason that some symbols are more common than others. If instead we use a variable-length representation, and let more common symbols be encoded in fewer bits, then we can save a considerable amount of memory. While it may be the case that, for a particular type of signal, the statistics of symbol usage are fairly stable across multiple data sources, let's not make this assumption. Thus, we would expect to use the sampled signal itself as the source of statistical information for code construction.

A *Huffman code* is a variable-length symbol representation scheme which is optimal in the case where all symbol probabilities are integral powers of 1/2. Since the number of bits per symbol is variable, in general the boundary between codes will *not* fall on byte boundaries. So, there is no "built in" demarcation between symbols. We could add a special "marker," but this would waste space. Rather than waste space, a set of codes with a *prefix property* is generated: each symbol is encoded into a sequence of bits so that no code for a symbol is the prefix of the code for any other. This property allows us to decode a bit string by repeatedly deleting prefixes of the string that are codes for symbols. This prefix property can be assured using binary trees.

Two example codes with the prefix property are given in Table 7.1. Decoding code 1 is easy, as we can just read three bits at a time (for example, decode "001010011" [answer in A.7 #7]). For code 2, we must read a bit at a time so that, for instance, "1101001" would be read as "11"='2', "01"='3', and "001"='4'. (What would the symbol sequence be for "01000001000" [answer in A.7 #8]?) Clearly, the average number of bits per symbol is less for code 2 (2.2 versus 3, for a saving of 27%).

Table 7.1: Two binary codes.

Symbol	Probability	Code 1	Code 2
1	0.12	000	000
2	0.35	001	11
3	0.20	010	01
4	0.08	011	001
5	0.25	100	10

So, assuming we have a set of symbols and their probabilities, how do we find a code with the prefix property such that the average length of a code for a character is a minimum? The answer is the *Huffman algorithm*. The basic idea is that we select the two symbols with the the lowest probabilities (in Table 7.1, '1' and '4'), and replace them with a "made up" symbol (let's call it s_1) with probability equal to the sum of the original two (in this example, 0.20). The optimal prefix code for this set is the code for s_1 (to be determined later) with a zero appended for '1' and a one appended for '4'. This process is repeated, until all symbols (real or "made up") have been merged into one "super-symbol" with probability 1.0.

If you think about this merging of pairs of characters, what we are doing is constructing a binary tree from the bottom up. To find the code for a symbol, we follow the path from the root to the leaf that corresponds to it. Along the way, we output a zero every time we follow a left child link, and a one for each right link (or we could use ones for right children and zeros for left, as long as we are consistent). If only the leaves of the tree are labeled with symbols, then we are guaranteed that the code will have the prefix property (since we only encounter one leaf on the path from the root to a symbol). An example code tree (for code 2 in table 7.1) is in Figure 7.4.

To compress a signal, then, we build the Huffman tree (there are more efficient algorithms which don't actually build the tree) and then produce a *look up table* (like table 7.1) that allows us to generate a code for each symbol (or decode the symbol at decompression time). We need to send this table with the compressed signal (or store it in the compressed file).

As I said at the beginning of this section, Huffman coding is only optimal if the symbol probabilities are integral multiples of 1/2. For the more general case, *arithmetic coding* can be used.

Lempel-Ziv Compression

In the 1970s, Lempel and Ziv developed two (patented) families of compression algorithms based on a *dictionary* approach. In a nutshell, in one family the algorithm builds a data structure (dictionary) with entries being sequences of symbols found in the input data. As the input is scanned, it tries to find the longest sequence of symbols that already exists in the dictionary. If this is successful, the entry number for that dictionary entry is transmitted. If unsuccessful, a sequence is added to the dictionary and also transmitted. This approach

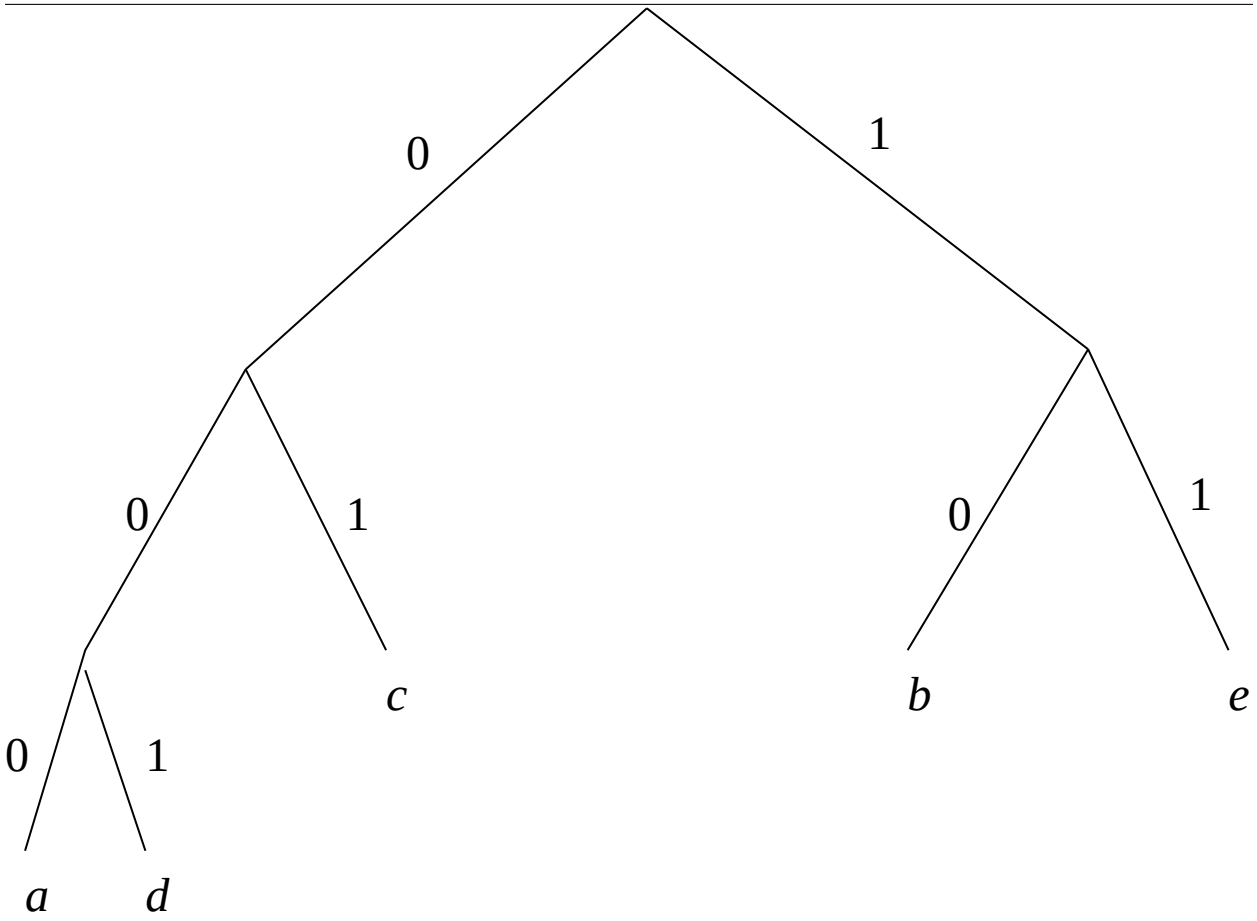


Figure 7.4: Binary tree with prefix property code.

starts with sequences of pairs of symbols and, as the encoding process continues, adds longer and longer sequences to the dictionary. As a result, long duration signals can be significantly compressed, as long stretches are found to be repeats of previously-seen data.

7.3 Source (Lossy) Compression

In certain situtations, it may be appropriate to sacrifice some of the information in the original signal to obtain increased compression. This may be the case, for instance, when a human observer cannot perceive the additional information (and therefore won't notice its lack). This inability to perceive the difference may be innate to human perception or it may be a product of the delivery technology (audio system, video monitor, etc.) or a more subtle interaction of sampling, the original signal, and human perception (as is the case for differential compression).

7.3.1 Differential Compression

Recall that when we sample a signal, the discrete representation is limited to frequencies below the Nyquist cutoff. It is not uncommon, however, for a signal to be significantly *oversampled*: the Nyquist cutoff is much higher than the signal's bandwidth. Even if this is not always the case, there may be long stretches of signal for which it is. For example, an orchestral recording may have stretches when no high-pitched instruments are playing.

When a signal lack high-frequency components, this is equivalent to saying that it changes slowly along time (high frequencies have high derivatives, low frequencies have small derivatives). If a signal changes slowly then, in its sampled version, successive samples are very similar. Let's go back to the idea of information content being the part of a message which you don't know. If we use each sample as a *prediction* of the next, then the *difference* between them is the information contained in the second. Ideally, then we should just transmit this difference.

This is the idea behind *differential pulse code modulation* (DPCM): we use the i^{th} sample of a signal x_i as the prediction for the next, x_{i+1}, and just transmit the difference, $\Delta x_{i+1} = x_{i+1} - x_i$. Of course, we start our encoding by sending a complete sample, x_0, and then continue with just the differences.

In what way is this lossy? It quite possibly isn't, depending on the number of bits in the original samples, the number of bits in the differences we send, the possible difference values in the signal, and how we treat them. For instance, if our original samples are 8 bits and we allow 4 bits for differences, we can accommodate differences of up to ± 7 between samples (using a two's complement representation for the differences). If all actual differences are less than of equal to ± 7, then no loss results. What if actual differences are greater? We have three basic options:

1. Output the full sample, rather than just a difference.

2. Assume this is an infrequent anomaly, outputting the maximum difference possible and retaining the actual difference internally. When subsequent differences are less than the maximum, modify them so that the output differences allows the coded signal to "catch up" to the value of the input.

3. Use the limited number of bits to cover larger differences by assuming they are multiplied by a constant factor, in effect "re-quantizing" them. If the factor was a constant value of '2', then 4 bits would cover ± 14.

The first case is very straightforward, and clearly results in no losses. For the second approach, the reconstructed signal is not the same as the original — information has been lost. However, the information is a rare, sudden change in the signal, and if the reconstructed signal caught up with the original fairly quickly, it is likely to be unnoticable. The third approach is also lossy, as it is incapable of representing differences that fall in between the quantized levels.

At the extreme, we can allocate only one bit per difference: this is called *delta modulation* (DM). In this case, we need to interpret a '0' as a -1 difference and a '1' as a +1 difference (we

could multiply these by a constant factor), so a constant input produces a "0101010101..." sequence, rather than a "0000000000..." one. Assuming the sampling rate is high enough, differences more than ± 1 will be rare, and loss will be minimal.

A more general approach to DPCM would be to use something other than just the value of one sample to predict the next. Thus, the differences to be sent would be $\Delta_{i+1} = \mathcal{F}(x_{i-j}, \ldots, x_i)$. This is the approach that *adaptive differential pulse code modulation* (ADPCM) uses. It *adapts* to the signal, using past experience to select the quantization levels that will be used to encode differences. This means that loud sections and quiet sections can have different steps between quantization levels. The international videoconferencing standards ITU G.726 use ADPCM to encode audio.

7.3.2 Transform Compression

Going back to figure 7.3, one thing that an encoding scheme can do is to transform the original signal into a domain that allows for better compression. To allow for better compression, a representation needs to isolate redundancy. For one-dimensional signals like audio, redundancy is apparent in the *sequence of samples*. All of the previous compression schemes are based on this sequential redundancy. For a signal sampled along time like sound, *temporal redundancy* is apparent. For a signal that is spatially sampled, like an image, there is a (2D) *spatial redundancy* (pixels near each other tend to have similar values). The question is: is the above noted temporal or spatial domain the domain in which the signal has its greatest redundancy, or is there some other domain in which more redundancy would be apparent? As you might guess, there are situations in which this is the case (for the aproach to be practical, we just need to make sure that an *inverse transform*, which brings us back to the original signal domain, exists).

One such domain is the frequency domain. Especially in images, a spectral representation tends to have great apparent redundancy. In such a representation, an image is considered to be composed of the sum of sinusoids (just as for sound) that are functions of space (instead of time, as for sound). The "only" conceptual jump here then is the one from 1D signals to 2D signals, which we'll defer to chapter 8. For the time being, let's think of images as being one-dimensional, like sound, so we can talk about 1D Fourier transforms.

I previously said that pixels tend to be similar to those nearby. This is another way of saying that the change in intensity as a function of space is low — that low-frequency processes are involved. Repetition coding assumes that there is no change, while DPCM either places a limit on change or quantizes the changes. Rather than doing these, let's take the Fourier transform of our signal. We have now decomposed it according to frequency. If mostly low-frequency processes have produced our signal, then the coefficients for low frequencies will have higher values than those for high frequencies: the low frequency components carry most of the information.

We can now take the obvious approach: match the number of bits in a representation to the amount of information contained. In this case, rather than use the same number of bits for each frequency coefficient, we assign more bits to the low frequencies and fewer to the high. This approach, which in effect codes different frequency bands separately, is also

called *sub-band coding*.

You probably remember that the Fourier transform (and FFT) have both magnitude and phase. We can simplify matters if we use a transform that uses only real arithmetic. This is one of the motivations behind using the discrete cosine transform (DCT) instead. The DCT coefficients can be expressed as:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cos 2\pi nk/N \tag{7-1}$$

We throw away absolute phase information and assume that the signal has even symmetry, but since we don't care what happens beyond the bounds of the signal, this is fine. Loss of relative phase information is another matter, but after all, this *is* a lossy compression technique. In chapter 8, I will place these losses in the context of human perception.

7.4 Problems

For each of the following MATLAB programming assignments, please email me your .mat file and your "best" result (the result that best shows the operation of your program). For your result, send both the original and "reconstituted" compressed version.

1. Write a program that performs run-length coding on images (you may treat the image as a one-dimensional vector for processing purposes, and in fact MATLAB will allow you to treat 2D matrices in this fashion). What percent compression do you get for black and white images (like those that might be produced for sending a fax)? What percent for color images generated from drawing programs? What percent for images from a digital camera? What percent for vectors filled with random numbers?

2. Write a program that performs simple, lossy DPCM coding and decoding on sampled audio. Test it using audio samples. For samples of people speaking, how many bits do you need for the differences for the result to be still intelligible? For it to be of quality comparable to the original?

3. For the DCPM examples, can you gain any additional compression by applying run-length coding to the output of the DPCM coder? Why or why not?

4. One way to apply transform coding of a signal is to divide it into *non*-overlapping *windows* and transform and code each window separately. Using the same audio samples as before, apply DCT coding (MATLAB has `DCT` and `IDCT` functions, part of the Signal Processing Toolkit. If this toolkit isn't available, use the magnitude of the `FFT`). (Hint: keep the window size short; maybe 8 or 16 samples.) Do not try to quantize the DC (zero frequency) component, but *do* experiment with quantizing the high frequency ones. What is the effect with no quantization? See how much compression you can get by heavily quantizing and/or eliminating high frequency components.

7.5 Further Reading

- J. Crowcroft, M. Handley, & I. Wakeman, *Internetworking Multimedia*, Morgan Kaufmann, 1999, chapter 4 (§ 4.1–4.5).
- A. Murat Tekalp, *Digital Video Processing*, Prentice Hall, 1995, chapters 18, 19, 21.
- K.R. Rao & J.J. Hwang, *Techniques & Standards for Image, Video & Audio Coding*, Prentice Hall, 1996, chapters 4, 5.

8 Audio & Video Compression and Coding

This chapter continues our coverage of compression begun in chapter 7 with descriptions of current standard compression schemes for audio and video information. By the end of this chapter, you will have a basic understanding of how the most common compression and coding algorithms use the fundamentals of lossless and lossy compression. You will also understand how the nature of the multimedia application influences compression algorithm design.

Each coding algorithm or standard that I will discuss in this chapter shows clearly the tradeoffs inherent in lossy versus lossless compression, the need for symmetric coding and decoding versus the ability to asymmetrically place more processing power in the encoder, storage versus transmission, and human perception versus machine processing. I will begin with audio coding schemes, move on to those for still images, and conclude with video.

More specifically, each coding scheme makes implicit or explicit decisions about each of the following issues; I ask you to consider what those decisions are when you read about each standard.

8.0.1 Issues in Coding Method Selection

- What are the application constraints? Are there signal quality requirements? Limits on system complexity? Upper bounds on end-to-end transmission delay?
- Will the codec (CODer/DECoder) be implemented in software, hardware, or a hybrid combination of both?
- Should the encoding be reversible, or can it be lossy?
- Besides overall constraints on algorithm efficiency, are there constraints on efficiency *consistency*? In other words, can the amount of computation required vary (based on the time-varying nature of the signal), or must it be independent of the signal's content?
- Does the algorithm need to be tolerant of transmission errors? Does it need to be able to correct them? How should it deal with data lost in transmission?
- If a lossy algorithm is desired, what kind of information can be lost? How do we decide what information to throw away?

- Does the data representation need to accommodate future scalability? For example, are we building a codec for images up to some maximum size, or will we want it to work with larger images in the future?
- How many times will the media be encoded? Decoded?
- Will we need to synchronize the encoded signal with other media, or are we encoding it "in isolation"?
- Will our system need to be compatible with other methods? Will we need to "transcode" our representation to other formats, or transcode other formats to our representation, efficiently?

8.1 Audio Coding Standards

8.1.1 Speech Coding for Telephony

Pulse code modulation (PCM), as discussed in chapter 7, is the foundation for most of the major audio coding standards. The ITU (International Telecommunications Union) has defined the following audio coding standards (among others) for "low quality" (i.e., telephone quality) audio:

G.711 This is audio pulse-code modulation (PCM) in support of video conferencing, with a bandwidth of 64K bits/second. It recommends a sampling rate of 8000 samples/second. The audio data is logarithmically encoded (which has the effect of companding) to 8 bits, quantized to 212 levels. The encoding itself can be either A-law (this is mostly used in Europe) or μ-law (which is mostly used in the US and by most computer hardware and software).

G.721 This is an ADPCM-based standard for 32K bit/second audio.

G.726 This replaces G.721, allowing conversion between 64Kbps and 40, 32, 24, or 16kbps.

G.727 This standard extends G.726 for embedded applications, including transmission over packet-switching networks (packetized voice protocol, or PVP, which is G.764).

G.722 and G.725 These standards are targeted at higher-quality speech transmission, with a signal bandwidth of 50Hz to 7kHz. They are targeted at a transmission rate of 64kbps.

There are other approaches that aim to produce high-quality speech with low bandwidth requirements (below 16kbps). These include LPC (linear predictive coding) and CELP (code excited linear predictor). In both cases, a model of the speaker's vocal tract is generated and the parameters of this model are transmitted to the receiver. After that, only enough data is sent to allow for the receiver to *synthesize* speech using the model. So, instead of hearing

a processed version of the speaker's voice, the listener hears a synthesized approximation to their voice. To improve speech quality, CELP sends error information (the difference between the actual signal and the model's output) in addition to the model information.

8.1.2 High-Quality Audio Coding

There are a number of standards used to encode audio at quality levels usable for music, television, movies, etc. — up to audiophile levels (except perhaps for those folks who insist that tube amplifiers and vinyl are necessary to capture the warmth of the original music).

MPEG

While MPEG (Moving Picture Experts Group) is a standard for video encoding, it obviously also must include audio information and is often used in isolation for just encoding audio. MPEG is actually a family of standards, with successive members providing increased quality at higher levels of compression (at the price of increasing computational complexity). For almost all versions, the input signal is assumed to be 20kHz. (What is the minimum sampling rate for such a signal [answer in A.8 #1]?) The desire is to have quality comparable to compact disc audio, and to support multiple channels of such audio. For CD quality, at a sampling rate of 44.1kHz, 16 bits/channel and two stereo channels, the uncompressed audio stream is 1.4Mbps.

MPEG-I This standard allows for encoding of two channels (stereo) at sampling rates of 32 (FM broadcasting), 44.1 (CD), or 48 (DAT) kHz. It achieves high-quality lossy compression by incorporating a simple psychoacoustical model of human auditory perception. The algorithm uses this model to determine what information can be lost without significantly affecting the listener's perception of sound quality.

More specifically, the algorithm is based on the phenomenon called *masking*. It is perhaps not surprising that our auditory systems are not equally sensitive to all sound frequencies. We are most sensitive to sounds in the range of 2–4kHz, and increasingly less sensitive to much higher and lower frequencies. What might be surprising is that our sensitivity at one frequency can be influenced by the presence of sounds at other frequencies.

The general idea of masking is that signals can interfere with each other within the processing stages that are a part of sensory systems. For example, a signal at one frequency (a *masker*) presented around the same time as another one at a second frequency might result in the second being undetected by the sensory system. This can occur even though the sensory system is perfectly capable of perceiving the second signal in isolation (or, in combination with signals other that the masker). Masking can also occur in time (hence, my previous statement, "around the same time"), with our sensitivity to sound at some frequency recovering from its masked level over the course of around 100ms.

In other words, how important a particular frequency is for our perception of sound is a function of both the frequency itself *and* the history of signal intensities at other frequencies.

MPEG-I takes advantage of masking for compression by dynamically altering a threshold for each of a number of frequency bands, based on the signal strength in neighboring bands. It does this by passing the input signal through a *filter bank* composed of 32 bandpass filters, thereby breaking the signal into 32 bands. It then computes the amount of masking for each band based on the signal in the other bands. The information in a band is only encoded if it is above the masking threshold. If it is encoded, the number of bits to be used is computed so that the quantization noise introduced is below the masking threshold (remember the discussion of quantization noise in chapter 2?). The resulting sub-band codes are then formatted into a bitstream, perhaps with video and synchronization information.

MPEG-I actually includes three audio "layers," each successive one being an enhancement of the previous. Layer 1 uses bands of equal width and only frequency masking (no temporal masking). Since it uses only frequency masking, the codec only needs to keep one "frame" of audio information (12 samples) in memory: masking occurs only between bands at the current time. Layer 1 typically achieves compression ratios of 4:1, or 384kbps high-quality stereo.

Layer 2 uses three frames of audio information in memory: previous, current, and next. This allows it to compute temporal masking, in addition to frequency masking. This allows layer 2 to reach compression ratios of 8:1, for a 192kbps audio stream.

Layer 3 uses frequency bands which are not of equal width, to better match human auditory perception. It also seeks to eliminate redundancy between the two stereo channels (because much of the information in one channel is also present in the other). It does this by separately coding the sum (M, for "middle") of the left (L) and right (R) channels and their difference (S, for "side"). At the decoder, the two channels are reconstructed as $L = (M + S)/\sqrt{2}$ and $R = (M - S)/\sqrt{2}$. When you listen to an MP3 audio file, you are actually listening to MPEG-I, layer 3 encoded audio. Finally, layer 3 incorporates Huffman coding for additional data stream compression. Layer 3 can produce high-quality audio at a compression rate of 12:1, which corresponds to a 112kbps data stream.

Web Links:

Compression FAQ http://www.faqs.org/faqs/compression-faq/

JPEG image compression FAQ http://www.faqs.org/faqs/jpeg-faq/

Planet JPEG http://www.geocities.com/tapsemi/

MPEG section of compression FAQ http://www.faqs.org/faqs/compression-faq/part2/section-2.html

MPEG FAQ http://www.faqs.org/faqs/mpeg-faq/

MPEG.org http://www.mpeg.org/MPEG/index.html

MPEG for MATLAB http://www.cl.cam.ac.uk/~fapp2/software/mpeg/

comp.compression newsgroup news:comp.compression

comp.multimedia newsgroup news:comp.multimedia

International Telecommunications Union (ITU) http://www.itu.int/

MPEG-II This extends MPEG-I audio to five channels plus one additional, low-frequency enhancement (LFE) channel. This should be familiar to you as the basic configuration for surround-sound (the five channels are center front, front left and right, and rear left and right; the low-frequency channel is for a subwoofer). These channels could also be used to

encode multilingual stereo. The additional channels are encoded by being mixed in such a way that an MPEG-I decoder can still decode the primary left and right stereo channels from an MPEG-II stream. So, an MPEG-II audio bit stream is an MPEG-I bit stream with the additional data formatted into data blocks reserved in MPEG-I for ancillary data. Correspondingly, an MPEG-II encoder consists of an MPEG-I encoder and an MPEG-II extension encoder.

There are other implementations within the MPEG-II standard which are not backward compatible with MPEG-I. These include AAC (advanced audio encoding).

MPEG-III Because of the progress of MPEG-II in support of high-definition television, development of an MPEG-III standard was terminated.

MPEG-IV MPEG-IV is targeted at a much broader range of applications than the preceding standards. This includes not only compression and coding of audio and video, but support for structuring content for WWW and hypermedia applications, intellectual property right management, computer network quality of service signaling, interactivity by the user, and low bit rate applications. It encodes data as a composition of multimedia objects, which can include audio, video, and 3D graphical objects. This builds on the earlier work of VRML (virtual reality modeling language). Little of this has anything directly to do with high-quality audio, but it seemed to make sense to discuss this standard right after the other ones.

8.2 Still Image Coding Standards

There are a number of formats for single images. These include TIFF (tag image file format) and GIF (graphics interchange format), which are lossless formats that use Lempel-Ziv compression. Because it also serves as the basis for spatial redundancy reduction in video, I'll confine myself to discussing JPEG (Joint Photographic Experts Group).

8.2.1 JPEG

Actually, I was being a bit misleading in dismissing formats such as TIFF and GIF. Those are *file formats*, while JPEG is an *encoding scheme*. In fact, JPEG-encoded images can be stored in their "own" format (JFIF, for JPEG file interchange format) or stored within a TIFF format file (which, counterintuitively, provides greater flexibility and more advanced features).

JPEG is targeted at compression of continuous-tone color and grayscale images (as opposed to line drawings). It includes a parameterized encoder which can support four lossy or lossless modes. These modes include:

Sequential The image is encoded in a single top-to-bottom, left-to-right scan.

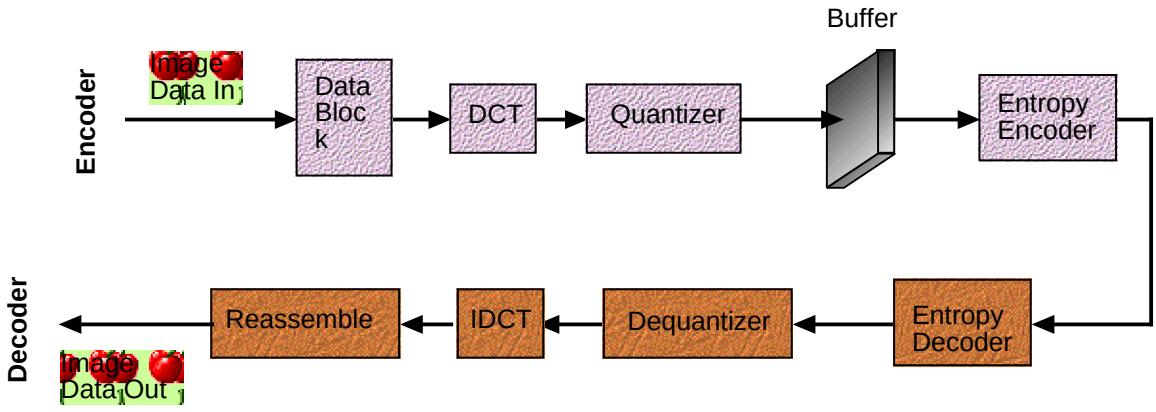


Figure 8.1: Block diagram of JPEG encoder and decoder. The DCT, quantizer, and buffer are not used for lossless mode. The buffer is only used for progressive mode.

Progressive The image is scanned multiple times, with successive scans providing information for successively better approximations to the original image.

Lossless Only entropy encoding is used.

Hierarchical Multiple versions of the image are encoded, at successively finer resolution. This allows a receiver to select the appropriate resolution; the encoder needn't know the limitations of the decoder or the constraints of its application.

Figure 8.1 presents simplified block diagrams for a JPEG encoder and decoder. There are four basic steps in the encoding process:

1. Prepare the image data by breaking it into 8-pixel by 8-pixel *blocks*. If the image is in color, each color component (i.e., red, green, blue) is separately broken into blocks (in other words, treated as though it were a separate image).

2. Decompose each block into its frequency components using the discrete cosine transform (DCT). This is a *two-dimensional* DCT, with the dimensions being the two spatial dimensions (horizontal and vertical). Let's say we use the variable x as the horizontal pixel index and y as the vertical. The location of a pixel in a block is then (x, y), where $0 \leq x \leq 7$ and $0 \leq y \leq 7$. If we use p_{xy} to refer to the pixel *value* at (x, y), then the 2D DCT of a block is

$$y_{kl} = \frac{c(k)c(l)}{4} \sum_{x=0}^{7} \sum_{y=0}^{7} p_{xy} \cos\left[\frac{(2x+1)k\pi}{16}\right] \cos\left[\frac{(2y+1)l\pi}{16}\right] \tag{8-1}$$

   where $c(k)$ and $c(l)$ are equal to $1/\sqrt{2}$ when k or l is equal to zero, and 1 otherwise.

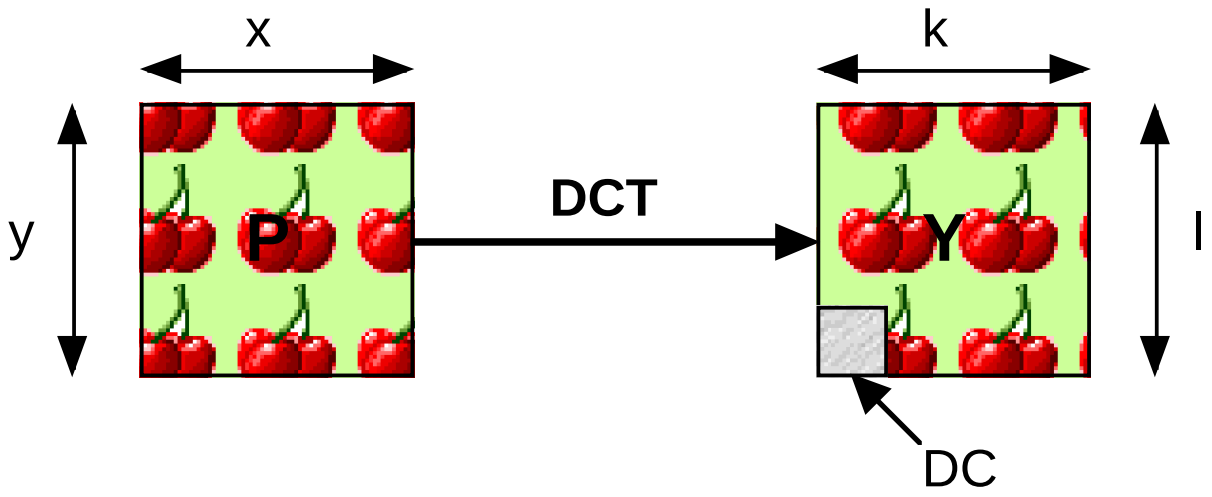


Figure 8.2: Illustration of transform of 8x8 pixel block into 8x8 spectrum by DCT. The spectrum's DC component is at y_{00}, corresponding to the average intensity within the block.

This produces an 8x8 spectrum for the block, where frequency is in cycles per (horizontal and vertical) pixel. The value at y_{00} is the DC value, and corresponds to the average pixel value for the block. Increasing k and l correspond to increasing spatial frequency (more abrupt changes in image intensity). The correspondence between the original block and its spectrum is illustrated in figure 8.2.

Figure 8.3 demonstrates the 2D spectra of two simple images. In this case, the MATLAB function `fft2()` was used, as the basic MATLAB distribution has no 2D DCT built in. The resulting complex output was converted to reals using the `abs()` function (the MATLAB code for all this is located at `http://courses.washington.edu/css457/ebook/dctdemo.m`. The top left image is a pixel block in which the pixel values vary in intensity as the sine of the x coordinate only. We would expect then that it would have nonzero spectral components for $k > 0$, because there is horizontal intensity variation. Since there is no vertical intensity variation, the spectral components in the vertical direction should all be zero for $l > 0$. This is exactly what we see in the spectrum plotted in the top right.

The bottom left block has sinusoidal intensity variation along 45-degree diagonals. In this case, the rate of intensity variation is the same in both the x and y directions, so it seems logical that the nonzero frequency components would occur at the same horizontal and vertical frequencies — for $k = l$. This is shown in the plot of the spectrum on the bottom right.

Figure 8.4 presents an example of the spectrum of a natural image. In this case, the original block (left) is someone's eye, at fairly low resolution. On the right in the figure, the spectrum of the block is presented. To show the non-DC components more clearly, the log of the spectrum is plotted (in MATLAB, `ykl = log10(abs(fft2(pxy)));`).

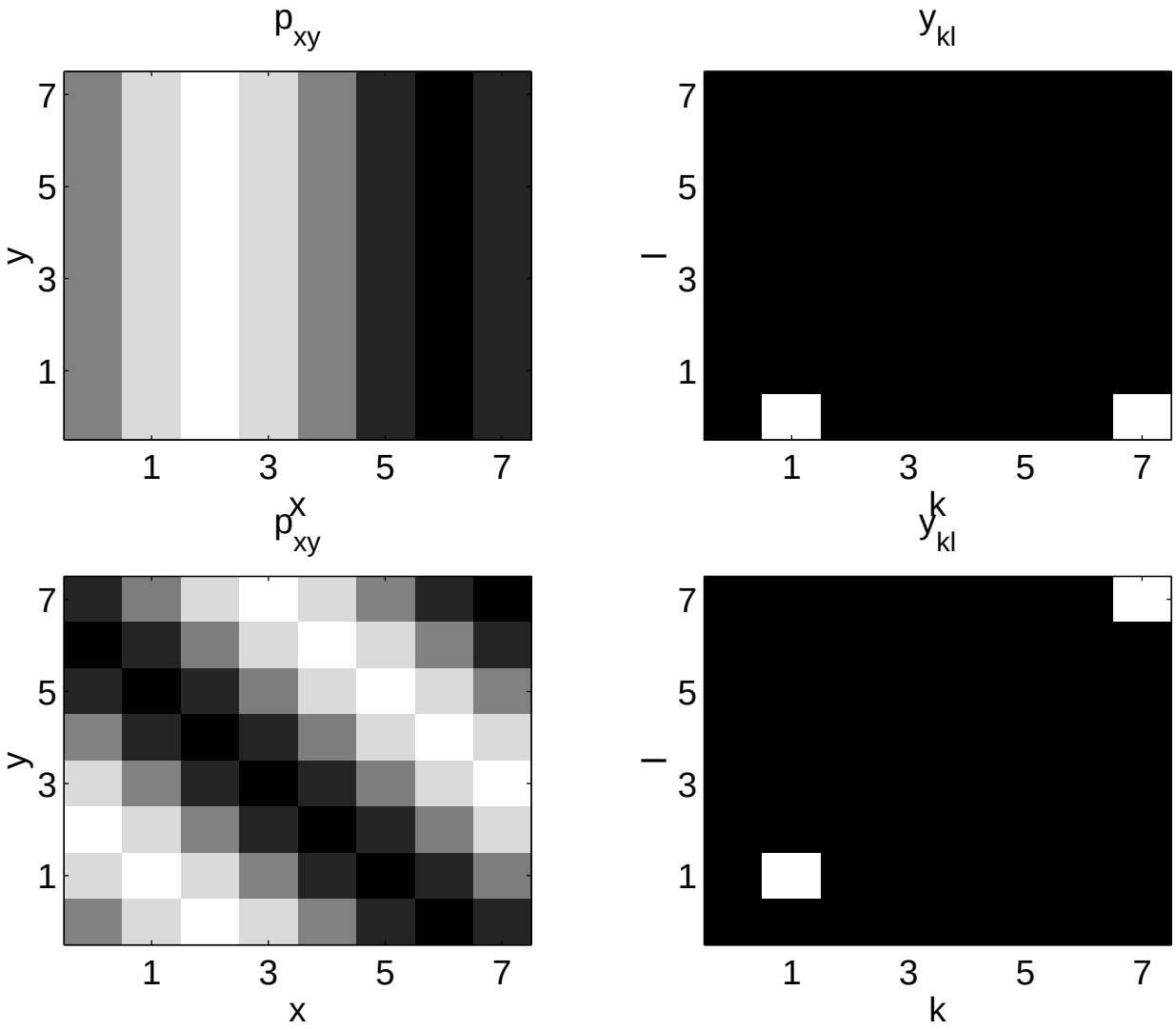


Figure 8.3: Example 2D spectra. The magnitude spectra for the 8x8 pixel blocks on the left are presented on the right. A block with horizontal sinusoidal intensity variation at a frequency of one cycle per 8 pixels and vertical frequency of zero (top left) has nonzero frequency components only for $l = 0$ (top right). A block with diagonal sinusoidal intensity variation at one cycle per 8 pixels (bottom left) has nonzero frequency components only at some $k = l$.

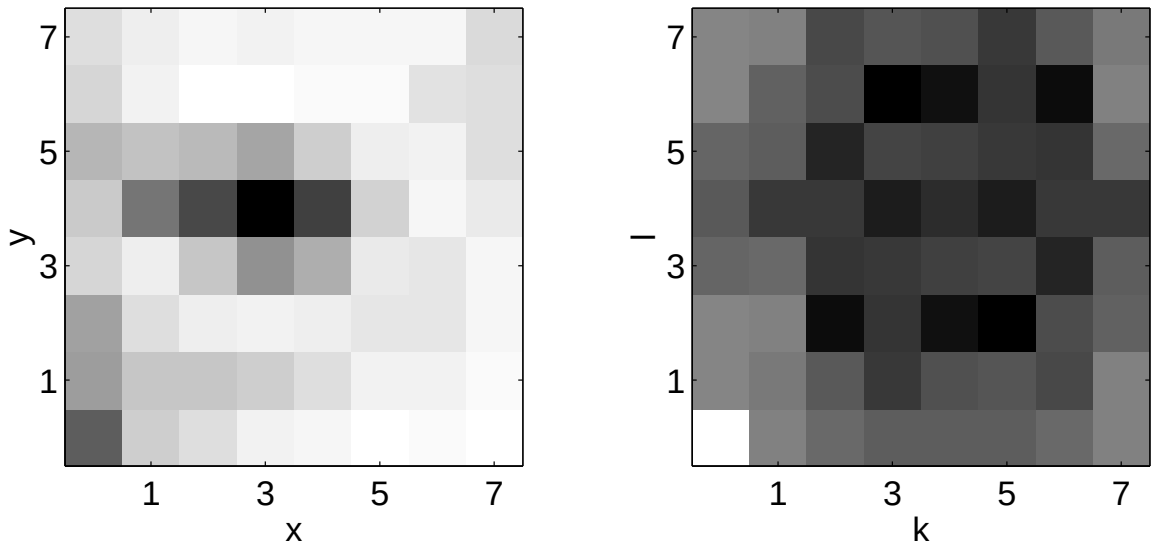


Figure 8.4: Example spectrum (right) of an 8x8 block taken from a natural image.

3. Reduce the number of bits used to quantize each frequency component. An 8x8 quantization matrix, $\mathbf{Q}$, is used to "threshold" each element of y_{kl}, with the result being $z_{kl} = \text{round}(y_{kl}/q_{kl})$. Larger values for q_{kl} mean that larger values for the corresponding spectral component y_{kl} will be ignored (treated as zero) and the effect is that fewer bits will be used to quantize y_{kl}. The values for $\mathbf{Q}$ are determined by the amount of compression desired.

4. Perform entropy encoding. The local average brightness (the DC component of the blocks) of images tends to vary slowly across the image; in other words, there is a great deal of spatial redundancy in the DC components. JPEG encodes the DC components of each block separately (all of the DC components are encoded together). As far as the other components are concerned, frequencies close to each other in a block tend to have the similar value. This is especially true for higher compression levels, where many of the high-frequency components will have been zeroed out. So, the 2D DCT block is converted to a 1D data stream by being scanned in a "zig-zag" pattern, as shown in figure 8.5. The puts the components in increasing order of frequency.

   At this point, we have two data streams: the DC components and the non-DC (AC) components. The DC stream is encoded using a predictive (difference) scheme. The

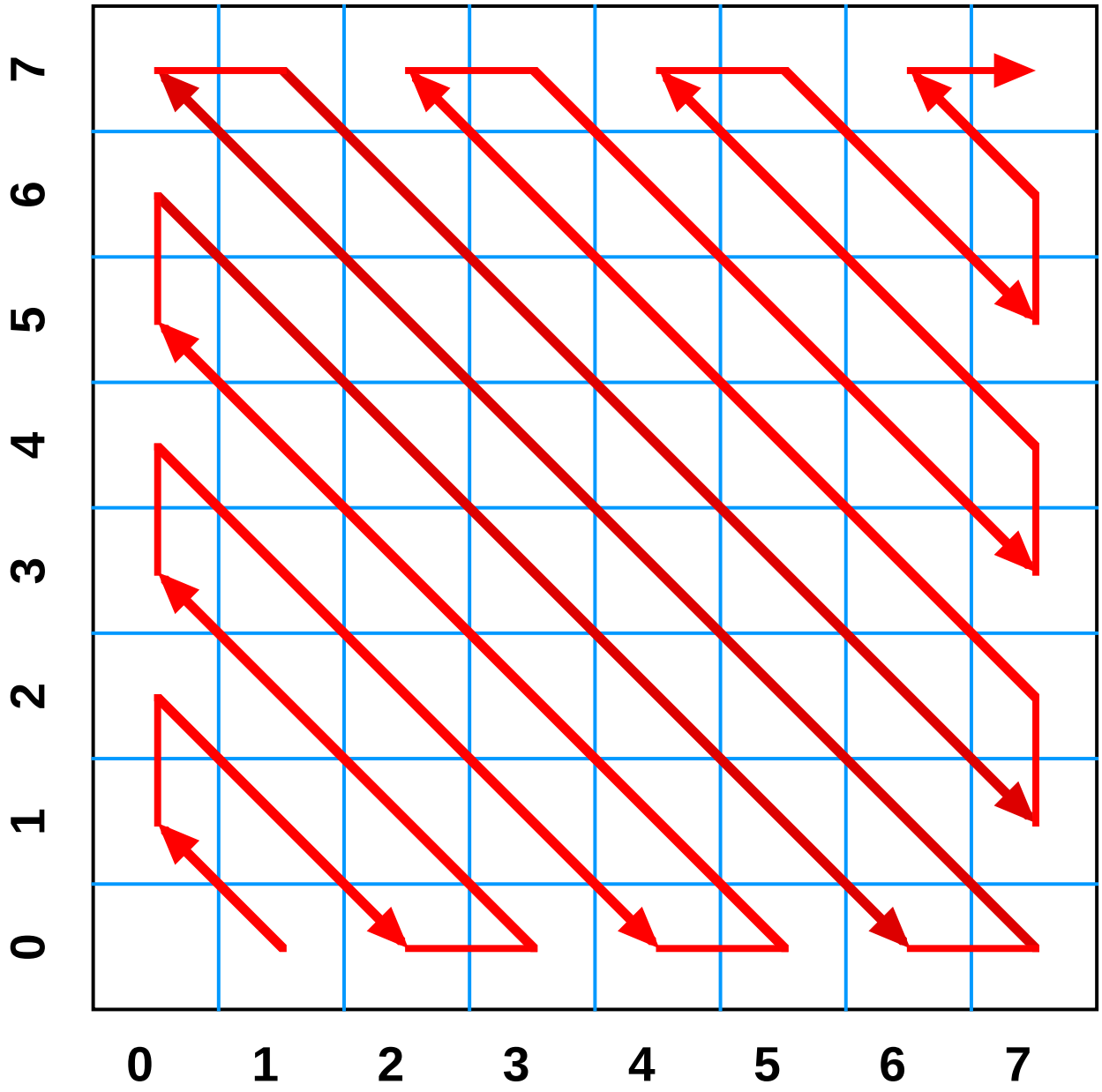


Figure 8.5: Entropy coding of JPEG DCT blocks. Non-DC frequency components are scanned in a zig-zag pattern.

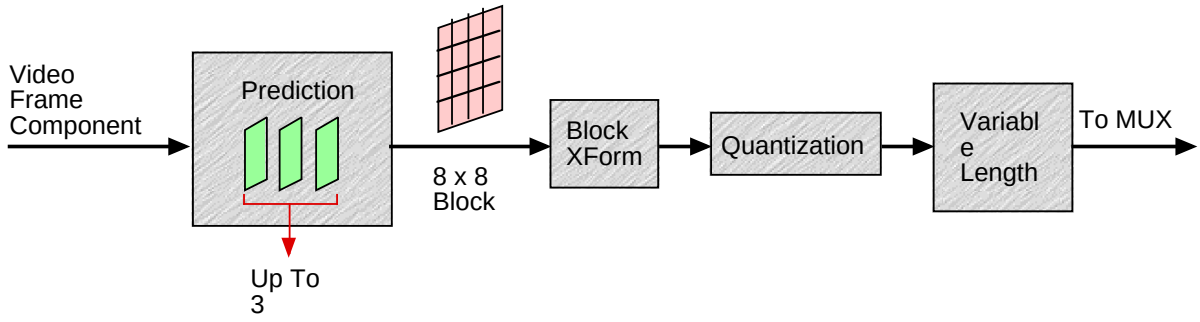


Figure 8.6: Simplified block diagram of an MPEG video source encoder. Input frames pass through a motion compensation process, 8x8 pixel blocks are converted to a spectral representation by DCT, the components are quantized according to the desired level of compression, and then the result is Huffman coded.

AC stream is run-length coded to shrink the runs of zero values. Then, both streams are Huffman or arithmetic encoded.

8.3 Video Coding Standards

Everything you've learned so far in this chapter can now be put together, because video coding involves combining audio and multiple still images. The audio and image information is combined for transmission and/or storage by *multiplexing* (MUX): interleaving segments of each. On the image side of things, there's still a great deal of redundancy in the sequence of images, because the change from one frame of video to another is usually quite small. So, moving image compression involves both spatial *and temporal* redundancy reduction. There are a number of ITU standards for videoconferencing, including H.261 and H.263. However, I will concentrate on the MPEG standards, as they follow most directly from JPEG.

8.3.1 MPEG Coding

As MPEG-II is an extension of MPEG-I (to multiple bit rates and resolutions), the following is applicable to both. As previously mentioned, MPEG is a standard for video transmission and storage. It has a higher computational complexity (on the coder side) and bandwidth requirement (2–8Mbps) than the videoconferencing standards. (Question: under what conditions is it acceptable to have greater coder complexity [answer in A.8 #2]?) On the other hand, decoding has a low enough complexity that it can be done in software.

A block diagram of the MPEG coding process is presented in figure 8.6. Except for the "prediction" block and the destination being a multiplexor, this is essentially the same as JPEG coding. Though I probably implied in the discussion of JPEG that the input could have RGB color planes, in reality MPEG color input has the three components (Y, Cb, Cr), with the first being *luminance* (brightness) and the second and third *chrominance* (color). Because the luminance channel is more important to our perception of visual detail, MPEG supports two formats in which the chrominance channels have their resolutions reduced, to either half that of luminance in both the horizontal and vertical dimensions, or half of Y horizontally only. It is also possible to keep all three matrices the same size. Each channel is then processed identically, then multiplexed in the output stream.

While each input frame is structurally identical, there are three different types of output frames:

I Frames "I," or *intra*, frames are encoded like JPEG images; in other words, coding only takes advantage of intra-frame information (information within the frame). Because I frames can be decoded in isolation, they can serve as references for random access. A video stream in which all frames are I frames is sometimes called *MJPEG*. I frames don't use the "prediction" block in figure 8.6.

P Frames "P," or *predictive*, frames use the preceding I or P frame to reduce temporal redundancy. If we ignore cuts between scenes (where the entire image changes), changes from frame to frame involve motion, either of the camera or of objects (or both). If we already know what something looked like, then a simple (x, y) vector can tell where it moved to in the current frame, greatly reducing the amount of data to send. A motion-compensation algorithm is used to determine these motion vectors. To do this, the frame is broken into 16x16 *macroblocks*. For each macroblock in the P frame, an exhaustive search is performed in the preceding I or P frame for the 16x16 pixel region which best matches it. That area in that preceding frame is used as a prediction for the macroblock in the current frame, and prediction errors and a motion vector ((x, y) offset found in the search) are computed for each 8x8 block. This corresponds to the "prediction" part of the block diagram, with two frames of memory used. These errors (and motion vectors) are then sent to the block transformation for the rest of the encoding process.

B Frames "B," or *bidirectional*, frames use both past and future I and P frames for motion-compensated prediction. Two motion vectors are computed, prediction errors are computed by interpolating between the pixel values in the past and future frames, and 8x8 blocks are passed to the DCT transformation process with pairs of motion vectors.

An MPEG coder breaks the stream of input frames into a sequence of GOPs, or *group of pictures*. It reorders the encoded I, P, and B frames so that each GOP starts with an I frame and each B frame in the GOP comes *after* the two I or P frames on which it is based. This is the best order for decoding; the decoder then converts the frames back into display order.

The MPEG standard is not just a video or audio compression standard. It encompasses a family of standards that include the entire multimedia system, including multiplexing, timing and synchronization, and a layered definition of the transmitted bitstream.

8.4 Problems

1. Discuss the decisions made with respect to the multimedia issues described at the beginning of this chapter for MPEG-I layer 3 audio and JPEG encoding.

2. Locate an interesting image to perform a simple test of the effects of frequency-dependent quantization. The MATLAB image processing or signal processing toolboxes are needed to have access to DCT functions, so we'll use the `fft2()` and `ifft2()` functions instead (if you prefer C++ or Java, then fine, but you'll have to get hold of decent DCT implementations). The only complication will be the need to deal with complex numbers, mostly using the `abs()` function. Load the image and compute its 2D FFT using `fft2()` (If your image loads as true color, with 3 color planes — which you'll know because its dimensions will be $N \times M \times 3$ — then you'll need to convert it to greyscale by adding the three components together and dividing by 3, before computing the FFT. This can be done by first converting it from `uint8` to `double` with `double`, then doing something like `a = (a(:,:,1)+a(:,:,2)+a(:,:,3)/3);`). The resulting matrix has complex values, which we will need to preserve. Use `ifft2()` to convert the FFT back and plot the result versus the original greyscale image (use `imagesc()`) to check that everything is working fine.

   Let's quantize the image's spectral content. First, find the number of zero elements in the FFT, using something like `length(find(a==0))`, where `a` is the FFT. Next, zero out all components with magnitudes below some threshold. You'll want to set the threshold somewhere between the min and max magnitudes of `a`, which you can get as `mn=min(min(abs(a)));` and `mx=max(max(abs(a)));`. Let's make four tests, with thresholds 5%, 10%, 20%, and 50% of the way between the min and max, i.e., `th=0.05*(mx-mn)+mn` (you may get better results with thresholds related to `log(abs(a))`, rather than just `abs(a)`). Zero out all FFT values below the threshold using something like:

   ```
   b = a;
   b(find(abs(a)<th)) = 0;
   ```

   (substituting `log(abs(a))` if that's how you're thresholding). You can count the number of elements thresholded by finding the number of zero elements in `b` and subtracting the number that were originally zero. This is an estimate of the amount the image could be compressed with an entropy coder.

   Convert the thresholded FFT back to an image using something like `c = abs(ifft2(b))`. For each threshold value, plot the original image and the processed image. You might

also want to print the difference between the two. Compute the mean squared error (MSE) between the original and reconstructed image (mean squared error for matrices can be computed as `mean(mean((a-c).^2))`). What can you say about the effects on the image and MSE? Write a script to automate the thresholding and reconstruction, so you can easily compute MSE for a number of thresholds. Plot MSE vs. the number of matrix values that got thresholded? Please submit the plots and your code as hard copy.

3. Go through the same procedure as above, but this time, instead of comparing the magnitude `abs(a)` to a constant, compare it to a threshold proportional to the distance from the zero frequency (if x and y are subscripts to a, then the distance is the square root of $x^2 + y^2$). How much more can you compress the image using this method for the same level of MSE?

8.5 Further Reading

- J. Crowcroft, M. Handley, & I. Wakeman, *Internetworking Multimedia*, Morgan Kaufmann, 1999, chapter 4 (§ 4.6–4.8, 4.10–4.13).
- A. Murat Tekalp, *Digital Video Processing*, Prentice Hall, 1995, chapters 20, 23, 25.
- K.R. Rao & J.J. Hwang, *Techniques & Standards for Image, Video & Audio Coding*, Prentice Hall, 1996, chapters 8–12.

9 Review and Conclusions

This chapter brings our journey to a close. In an in-person course, we would spend some time in lecture reviewing the material covered; this of course is redundant here, as you have access to all that material verbatim on-line. You also have access to the instructor for any questions you might have. Instead, what I will do is present a generic multimedia system that includes all the course material, then describe an example media system and relate its design to what we've learned in this course. The system in question is the compact disc player, which should be familiar to everyone and which is simple enough conceptually that we can actually describe it in this limited space (at least, in simplified form).

9.1 A Generic Digital Multimedia System

Figure 9.1 presents a simplified generic multimedia system which highlights the concepts covered in this course. All multimedia begins with physical signals — light, sound, etc. (Actually, that last statement isn't 100% true, as there *is* such a thing as computer-generated multimedia: computer music, computer graphics, etc.) These signals must be converted into analog electrical signals so they can be captured by computer (or, for that matter, so they could be recorded on analog media). Digitization involves converting the continuous-time analog signal to a discrete-time signal via sampling. The sampled signal is then quantized to a fixed number of bits of resolution per sample. The result is a *digital signal*. This raw signal is typically encoded for compression purposes and/or to add information to the data stream (for example, error correction codes).

At this point, the encoded signal can be treated like any other digital information manipulated by computer. It can be stored in files, transmitted over networks, processed to improve or alter it, and/or presented to human beings on a desktop computer (or, these days, a consumer device like an HDTV set).

9.2 Compact Discs

One example digital multimedia system is the compact disc, or more precisely, since I'm referring to music CDs, compact disc digital audio (CD-DA). The standards associated with CDs (the *Red Book*, defined by Philips and Sony in 1980 so that discs would be

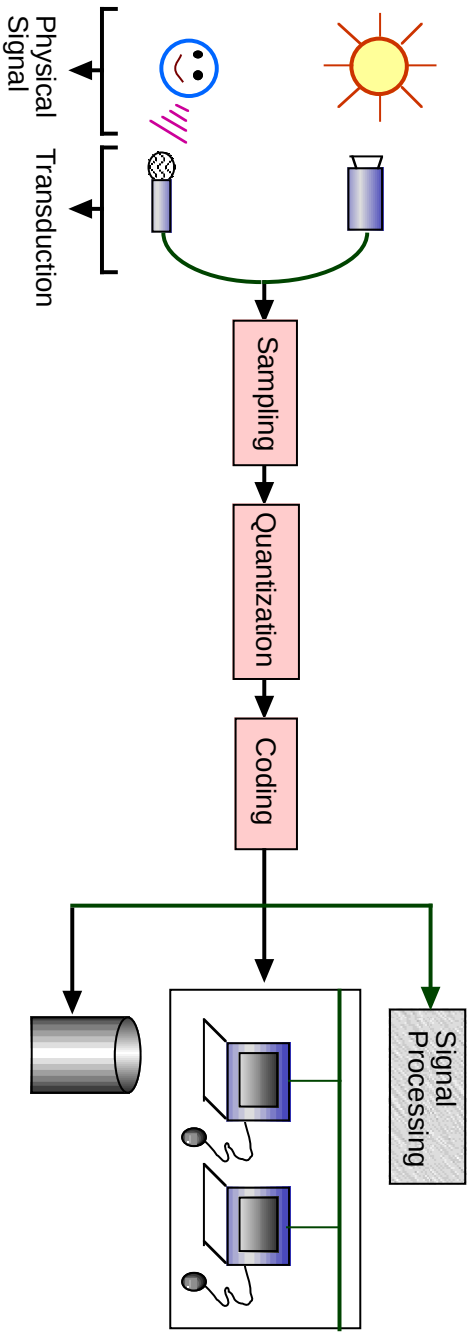


Figure 9.1: Block diagram of a generic multimedia system.

interchangeable among different manufacturers' hardware, and IEC Publication 60908) cover the disc itself and the optomechanical drives that spin it and read from it.

Information is recorded onto a CD in the pattern of pits and bumps (or *lands*) of a metal layer sandwiched between two covering plastic layers. These pits and lands are arranged in a spiral pattern, like a vinyl LP. There are two differences between the CD layout and the LP: data is recorded from the innermost part of the surface outward and the disc's rotation speed changes as the read laser moves. The change in rotation speed is necessary because a CD is a constant linear velocity (CLV) device: rotation speed is set so that the data moves past the read laser at the same rate everywhere on the disc. Near the center, the disc rotates at 500 rpm; this slows to 200 rpm at the outer edge. The rotation rate is automatically regulated by the drive mechanism to maintain a constant data rate of 4.3218 Mbps.

Data is read from the disc by a laser/detector pair. The laser illuminates a spot on the underside of the disc and the metal layer reflects this back to the detector. About 90% of the laser light is reflected by a land; pits, on the other hand, reflect only about 25% of incident light. This difference is easily detectable, and thus the *encoded* data can be read.

9.2.1 Data Encoding

Data is encoded on a CD so as to both minimize the effect of and correct for errors. This is especially important for a device that has an exposed surface and is intended to by used by ordinary consumers. Even without these considerations, error correction would be important in a device where a speck of dust could wipe out 50 bits on each of ten spirals. The CD standard uses a number of techniques to encode data so that commonly-expected errors can be detected and corrected.

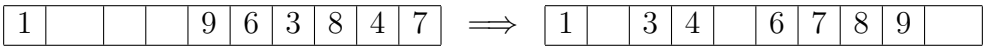
The first thing we note about error detection and correction is that it will require extra information to be sent. In effect, redundancy is introduced into the data stream in a manner such that errors are unlikely to destroy both the "original" and the "copy". (In reality, of course, the scheme is more sophisticated than just recording copies of the data.) From this, you should conclude that the data stream is not maximally compressed; in fact, CD audio is recorded uncompressed as 16 bits/sample, linearly encoded (i.e., no companding). Error detection and correction schemes involve adding bits (an *error correction code*) to each byte (or larger group) of data.

Error "Clumps"

If you consider the error generated by a scratch, dust, etc., it seems that this will obliterate a large number of consecutive data bits: a "clump" of data bits. This would seem guaranteed to eliminate not only the original data, but also the associated error correction codes.

To reduce the probability of errors in long, contiguous stretches of data, a simple scheme is to not record long, contiguous stretches of data. After all, if you don't record them, then you can't get those kinds of errors, right? This is accomplished by *interleaving*: data is shuffled before being recorded, so that errors in contiguous sections on the disc will correspond to isolated errors in the data. To demonstrate this effect, let's say that I record the numbers

one through ten in shuffled order: 1, 10, 5, 2, 9, 6, 3, 8, 4, 7. Suppose a clump of errors occurs in the second, third, and fourth numbers, rendering them unreadable. The result is:



Errors in three consecutive words on disc are widely separated in the de-interleaved data stream. An appropriate interleaving stream, shuffling data over a wide area, can cause clumps of errors to become widely distributed, and thus more likely to be correctable using the surrounding intact information. In the CD standard, the interleaving and error correction scheme is called *Cross Interleave Reed-Solomon Code*, or CIRC.

"Faking It"

Sometimes, an error will occur which is too massive for the error correction scheme. To prevent unpleasant listener experiences, CD players use varying two schemes to mask such errors: interpolation and muting.

In *interpolation*, the "good" waveform before and after the error is used to fill in the bad section with an approximation of what it might have been. A simple approach would be to just repeat the last good data value to fill in the gap; a better method would be to linearly interpolate between the two data values on either side of the gap. Interpolation schemes also can be used which "blend" the interpolated values into the gap ends more smoothly than a straight line does.

At some point, the gap becomes too big for interpolation. The workaround employed is *muting*: the volume of the music is smoothly reduced before the gap and increased back up afterwards. This avoids unpleasant effects like lound pops and clicks. Additionally, by fading out and back in smoothly, the problem is less noticable (sometimes even unnoticable).

Error Performance

CDs are not error-free devices. In fact, the typical number of errors in the raw data from a CD is one in 100,000 to 1,000,000 bits. I've already mentioned that the data rate from the CD is over 4 million bits per second, so we should expect many errors per second. CIRC error correction can repair most of these errors. Depending on the particular player's implementation (and this is never listed in the specs for an audio CD player), CIRC can deal with error clumps of up to 4000 bad bits. After CIRC, the error rate can be as low as one in 10–100 billion bits (if the player uses a good implementation; there is no requirement that all of the error correction capability in CIRC be used by the player to correct errors). In practice, a CD in reasonably good shape might have one uncorrected error, which would have to be dealt with by interpolation or muting.

The Data Encoding Process

CD-DA data is considered to be in two stereo channels sampled at 44.1kHz at 16 bits/sample. The samples from each channel are arranged in alternating order (left 16 bits, right 16 bits,

Table 9.1: Part of the eight-to-fourteen modulation lookup table.

Data Symbol (8 bits)	CD Word (14 bits)
00000000	01001000100000
00000001	10000100000000
00000010	10010000100000
00000011	10001000100000
00000100	01000100000000
00000101	00000100010000
00000110	00010000100000
00000111	00100100000000

etc.) to yield 32-bit sampling periods. Six of these sampling periods will be encoded as one *frame* of data.

The next step towards assembling the frame is computing error correction coding using CIRC. The data is treated as a sequence of 8-bit symbols for this process (so each sample corresponds to two symbols). Four bytes of CIRC parity are added after the first 12 bytes of data and four are added after the second 12 bytes. So, the original 24 bytes of data has now become 32 bytes.

Each frame then has a *subcode* byte prepended to it. The subcodes in each frame contain information about the number of tracks on the disc, their start and end times, etc. Each bit of a subcode byte has a separate meaning, and a player collects these bits from 98 consecutive frames to produce eight 98-bit words with this information. This might not seem like much information, but on a full CD it would correspond to 32MB!

At this point, the data is ready to be converted to the form which will be recorded on the disc. Because of fabrication imprecision and other manufacturing considerations, CDs do *not* use pits to encode zeros and lands to encode ones (or vice versa). Instead, ones are encoded as a pit-land or land-pit transitions, while zeros produce no transitions. So, the rate of transitions (the length of pits or lands) depends on how often ones are encountered in the data stream (or, equivalently, the length of runs of zeros in the data). It is desirable to control this so that all pits fall within some range of minimum to maximum length. To accomplish this, each 8-bit symbol is converted to a 14-bit pattern using *eight-to-fourteen modulation* (EFM). This is done via a look-up table, a portion of which is presented in table 9.1. The bit patterns in each 14-bit word are selected to generate a particular rate of occurrence of ones (and, as a result, set the typical land and pit lengths). In particular, only those words with more than two but less than ten zeros in a row are chosen. Additionally, since only 256 of the possible 16K 14-bit words are used, those words are less similar than the original symbols, which yields some additional error correction capability. For example, with 256 8-bit symbols, if we flip a bit in one symbol, we get another one. With only 256 out of 16K 14-bit words used, flipping one bit is unlikely to produce another valid word (about a 1.5% chance).

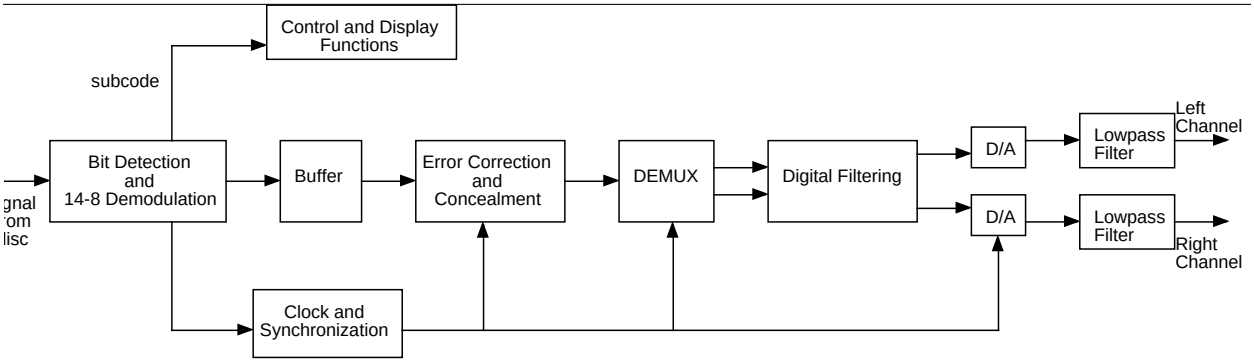


Figure 9.2: Simplified block diagram of a CD player.

There is still a need to control the transition between these 14-bit words and to fix the ratio of high to low bits. So, between each pair of words, three *merging bits* are inserted. Two of these are used to ensure that, even if the first 14-bit word ends with a one and the next starts with a one, there won't be two ones in a row. The third bit is chosen to be either a zero or a one to keep the overall ratio at 8:17.

Frames of data are now indicated by adding a 24-bit synchronization pattern before each: 100000000001000000000010 plus three merging bits. This is a set of three ones separated by tens zeros between each pair, and won't appear anywhere else in the data. Besides marking the start of a frame, this is used by the player as a clock to regulate rotation speed. The resultant frame contains 588 bits: 24 synchronization pattern bits, 336 data bits (in 24 14-bit words), 112 error correction bits (in 8 14-bit words), 14 subcode bits, and 102 merging bits (in 34 groups of three bits each).

The data is ready for recording at this point. Pit edges encode ones; while the extent of pits or lands correspond to zeros. The data stream has been encoded so that all pits and lands are between 3 and 11 bits long (in other words, there are between 3 and 11 zeros in a row anywhere in the data).

The result of all this is that only about 32% of the data on a disk is the actual audio information; the rest is overhead of EFM, merging bits, CIRC, synchronization, and subcodes.

9.2.2 CD System Signal Processing

A CD player converts the information encoded on the disc into analog audio. Figure 9.2 is a simplified block diagram of a CD player. Processing begins with detection of the pit/land transitions from the disc (which includes a control system that maintains the appropriate spindle rotation speed, moves the laser along the track of the pits, and keeps the laser focused on the metalized layer within the disc). This raw data then has the merging bits removed, and is converted from 14-bit words to 8-bit symbols. The subcode symbols are sent to a parallel pathway to support the player's user interface.

The data symbols are stored in a queue that leads to further processing. The first step

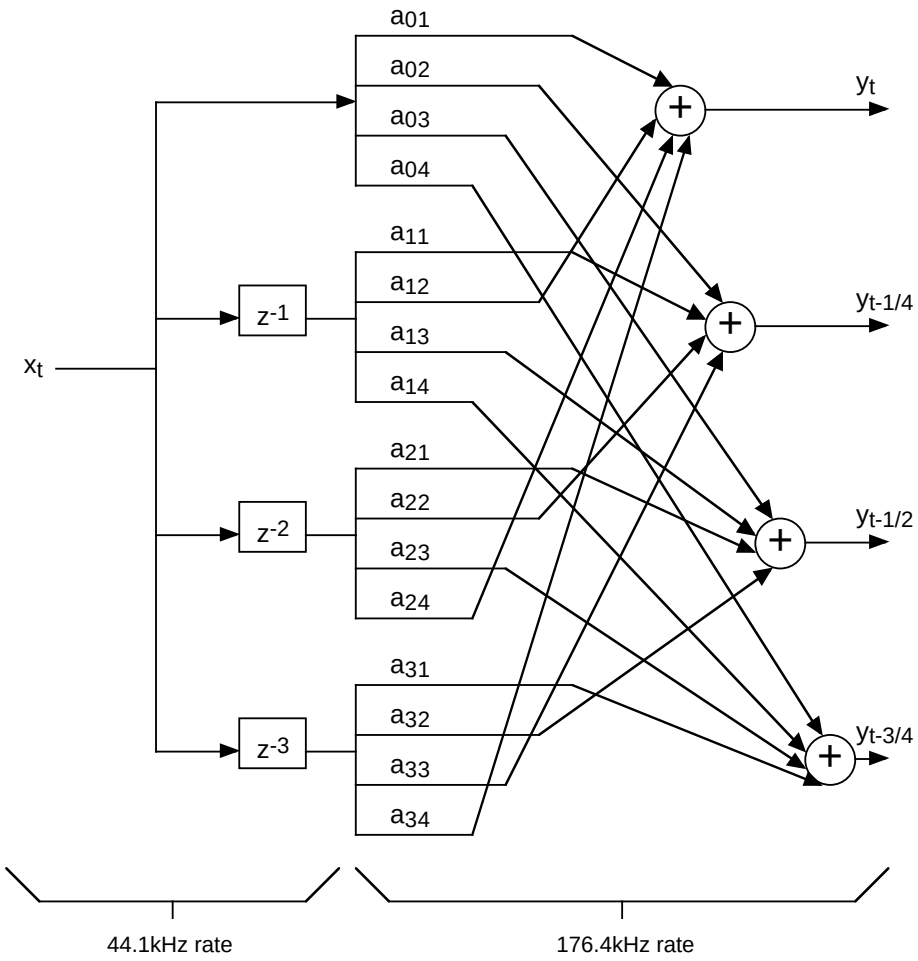


Figure 9.3: Simplified oversampling filter.

of this is error detection, correction (if possible), and concealment (if uncorrectable). The two channels (left and right) are then separated (*demultiplexed*; "DEMUX" in the figure).

Next, digital filtering involves an *oversampling* transformation, in which additional samples are interpolated between the original ones, and a low-pass filter. Oversampling results in one, three, seven, or more interpolated values being inserted between each pair of input samples. The result is a data stream which "simulates" one sampled at twice, four times, eight times, etc. the original rate of 44.1kHz. I say "simulates" here because any aliasing has already occurred when the music was originally digitized (before it was recorded to disc). The purpose of oversampling here is to produce a digital signal with no information beyond 22.05kHz (the original Nyquist limit imposed by sampling) but with a Nyquist limit of 44.1, 88.2, 176.4kHz or more. This allows the use of analog lowpass filters on the output with frequency responses which drop off relatively gently beyond 22.05kHz that still do not pass any undesirable high-frequency artifacts. Question: what's wrong with low-pass filters with abrupt cutoffs (sometimes called *brick wall* filters) (answer in A.9 #1)?

Figure 9.3 presents a simplified four-times oversampling filter. In this simplified version, an input sample plus three delayed versions are summed together to produce four output samples. The coefficients are chosen so that a low pass filter with cutoff at the input's Nyquist limit (22.05kHz) is implemented for each output. The input samples enter at 44.1kHz (and thus the delays are multiples of the sample period, $T_s = 1/44,100 = 22.7\mu$s). During each input sample period, each of the xt values is multiplied by four different coefficients and those products summed to produce four y_t, at a rate of 176.4kHz. This is then converted to analog by digital to analog converters (D/A or DAC), and then low pass filtered before being sent to the preamplifier, power amplifier, speakers, and your ears.

9.3 Conclusion

While it may have at times seemed a long and arduous journey, I hope that, looking back, you have a sense of satisfaction in the scope of understanding you've gained in this important subject. Digital multimedia is a big subject, and no single course can cover everything. However, please consider the CD overview that you've just gone through, the background required to understand it, and how incomprehensible even the basics would likely have been to you before you took this course. I'd like you to use this experience to make you confident that you could work in a team building multimedia devices, be they digital audio, video, or telecommunications (wired or wireless). I also hope that this has served to whet your appetite for more, and that you'll look for the implications for multimedia when learning about databases, or hardware, or networking, or almost any area of computing.

9.4 Further Reading

- Pohlman, Ken, *The Compact Disc Handbook*, A-R Editions, 1992.

A Answers to Self-Test Exercises

A.1 Chapter 1: Signals in the Physical World

1. Let's assume that we encode audio information as 8 bits per sample at 8000 samples per second (this would be considered very low quality). How many bits per second is that?

   Answer: 64,000 bits/second.

2. Let's say we encode each frame in a video stream as 1000x1000 pixels, 24 bits/pixel, and 30 frames/second. How many bits per second is that?

   Answer: 720 million bits/second.

3. We need to find a function of x whose second derivative is proportional to itself. Can you think of one or two?

   Answer: $\sin \omega t$ and $\cos \omega t$ both work.

4. Solve for ω in terms of k and m.

   Answer: $\omega = \sqrt{k/m}$.

5. Fork stiffness (k) clearly affects vibration frequency. As the fork get stiffer (k increases), does the vibration frequency go up or down?

   Answer: as k increases, ω (which is vibration frequency) increases.

6. If the two forks were close together, we would heard the sum of their tones. What would that sum be?

   Answer: $a_1 \cos(\omega t + \phi_1) + a_2 \cos(\omega t + \phi_2)$, where a_1 and a_2 are the vibrational amplitudes and ϕ_1 and ϕ_2 are the two phases.

7. What does $\mathbf{u} + \mathbf{v}$ look like if $\mathbf{u}$ and $\mathbf{v}$ are rotating?

   Answer: If the ωs of $\mathbf{u}$ and $\mathbf{v}$ are equal, then $\mathbf{u} + \mathbf{v}$ rotates with them, the magnitudes staying constant. Therefore, $\mathbf{u} + \mathbf{v}$ is a sinusoid of the same frequency, ω.

8. What is the sum of the two complex numbers $x + jy$ and $v + jw$?

   Answer: You add the reals and imaginaries separately, yielding $(x + v) + j(y + w)$.

9. What is the product of the two complex numbers $x + jy$ and $v + jw$?

   Answer: It works just like multiplying polynomials, $(x + jy)(v + jw) = xv + jyv + jwx + j^2yw = (xv - yw) + j(xw + yv)$ (remember that $j^2 = -1$).

10. Convert the complex number $z = x + jy$ to polar form, $R\angle\theta$.

    Answer: $R = |z| = \sqrt{x^2 + y^2}$; $\theta = \arctan(y/x)$.

11. Multiply the two polar-form complex numbers $R_1\angle\theta_1$ and $R_2\angle\theta_2$.

    Answer: $R_1R_2\angle(\theta_1 + \theta_2)$.

12. Multiply the two complex sinusoids z_1 and z_2.

    Answer: Just like multiplying the polar representation of two vectors, $z_1z_2 = R_1R_2e^{j(\theta_1+\theta_2)}$.

13. The complex conjugate is indicated by z^*. If $z = x + jy$, $z^* = x - jy$. What is the complex conjugate of the complex sinusoid, $z = Re^{j\theta}$?

    Answer: Like the polar representation, it has the same magnitude but a negative angle, $z^* = Re^{-j\theta}$.

14. Answer the following for $z = x + jy$:

    (a) What is $z + z^*$?
    Answer: $2\,\mathrm{Re}(z)$.

    (b) What is $z - z^*$?
    Answer: $2j\,\mathrm{Im}(z)$.

    (c) What is zz^*?
    Answer: $|z|^2$.

15. Prove the relationship in (1-41).

    Answer:

$$c_{-k}^* = \left[\frac{1}{T}\int_0^T f(t)e^{jk\omega_0 t}dt\right]^* = \frac{1}{T}\int_0^T f^*(t)e^{-jk\omega_0 t}dt = \frac{1}{T}\int_0^T f(t)e^{-jk\omega_0 t}dt = c_k$$

    because $f(t) = f(t)^*$ ($f(t)$ is real).

16. From equation (1-43), derive (1-44).

    Answer:

$$\begin{aligned}\mathrm{Re}[c_ke^{jk\omega_0 t}] &= \mathrm{Re}[c_k\cos k\omega_0 t + jc_k\sin k\omega_0 t] \\ &= \mathrm{Re}(c_k)\cos k\omega_0 t + \mathrm{Re}(jc_k)\sin k\omega_0 t] \\ &= \mathrm{Re}(c_k)\cos k\omega_0 t - \mathrm{Im}(c_k)\sin k\omega_0 t\end{aligned}$$

17. Prove (1-56).

    Answer: Since the denominator of $\sin\alpha/\alpha$ is zero at $\alpha = 0$, we instead evaluate this as

$$\text{sinc}(0) = \lim_{\alpha \to 0} \sin\alpha/\alpha$$

    and use L'Hôpital's rule. The limit becomes

$$\begin{aligned}\text{sinc}(0) &= \left.\frac{\frac{\mathrm{d}}{\mathrm{d}\alpha}\sin\alpha}{\frac{\mathrm{d}\alpha}{\mathrm{d}\alpha}}\right|_{\alpha=0} \\ &= \left.\frac{\cos\alpha}{1}\right|_{\alpha=0} \\ &= 1\end{aligned}$$

A.2 Chapter 2: Signals in the Computer

1. The hubcap of a car coming to a stop in a motion picture.

   Answer: Signal: the spokes; sampling: the discrete images in the movie.

2. A TV news anchor squirming while wearing a tweed jacket.

   Answer: Signal: the tweed texture; sampling: the discrete frames of the video.

3. A helicopter blade while the helicopter is starting up on a sunny day.

   Answer: Signal: the blade motion; sampling: the strobing effect as each blade blocks, then reveal, the sun.

4. In this case, $\mu = 0$ and the variable's range is [-1/2, +1/2] LSB. The result is a standard deviation (equivalent to the RMS error computed in the textbook) of $\sigma = 1/\sqrt{12}\,\text{LSB} \approx 0.29\,\text{LSB}$ (how?).

   Answer: Given the value for μ and the range of the (now definite) integral, we have:

$$\begin{aligned}\sigma^2 &= \int_{-1/2}^{+1/2} x^2 \mathrm{d}x \\ &= \left.\frac{x^3}{3}\right|_{-1/2}^{+1/2} \\ &= \frac{1}{24} + \frac{1}{24} \\ &= \frac{1}{12}\end{aligned}$$

   And so $\sigma = 1/\sqrt{12}$.

5. If we use an 8-bit ADC, 5V corresponds to 255 and 1mV is then 0.051 LSB. (What is the SNR for the original signal?

   Answer: It is $20 \log 5/0.001 = 74$dB, which is quite good.

6. What ratio of amplitudes is represented by one bel?

   Answer: A bel is ten dB; and 10dB is an amplitude ratio of $\sqrt{10} \approx 3.2$.

A.3 Chapter 3: Filtering and Feedforward Filters

1. Is the signal of equation 3-1 periodic? If so, what is its period?

   Answer: Since this signal is a sum of sinusoids with frequencies that have a common divisor, then, yes, the signal is periodic. The greatest common divisor is 50Hz, and so the smallest period for the signal is $T = 0.02$ seconds.

2. Since $|\cos(\omega\tau)| \leq 1$, the maximum value $|H(\omega)|$ can reach is $(1 + a_1)$, which occurs when the angle $\omega\tau = n\pi, n = 0, 2, 4, ...$ (zero or even multiples of π). Why is this?

   Answser: These are the values of $\omega\tau$ for which $\cos \omega\tau = +1$.

3. Use Euler's formula and the definition of the magnitude of a complex vector to derive (3-11) from (3-10).

   Answer: Subsituting $e^{-j\omega\tau} = \cos \omega\tau - j \sin \omega\tau$, we obtain $|H(\omega)| = |1 + b_1(\cos \omega\tau + j \sin \omega\tau)| = |1 + b_1 \cos \omega\tau + jb_1 \sin \omega\tau|$. The magnitude of a complex number is the square root of the absolute value of the sum of the squares of its real and imaginary parts, so $|H(\omega)| = |(1 + b_1 \cos \omega\tau)^2 + b_1^2 \sin^2 \omega\tau|^{\frac{1}{2}}$. Computing the squared values, $|H(\omega)| = |1 + 2b_1 \cos \omega\tau + b_1^2 \cos^2 \omega\tau + b_1^2 \sin^2 \omega\tau|^{\frac{1}{2}}$. Factor out the b_1^2 from the last two terms and remember that $\cos^2 \theta | \sin^2 \theta = 1$, and you're home free.

4. Suppose that we sample a signal at 1000Hz. For each of the following analog frequencies f, determine ω, $\hat{f}$, and $\hat{\omega}$. Indicate if that frequency will be aliased.

   (a) $f = 100$Hz: $\omega = 2\pi f = 200\pi$ radians/sec. $\hat{f} = f/f_s = 100/1000 = 0.1$ cycles/sample. $\hat{\omega} = 2\pi\hat{f} = 0.2\pi$ radians/sample. Not aliased.

   (b) $f = 200$Hz: $\omega = 2\pi f = 400\pi$ radians/sec. $\hat{f} = f/f_s = 200/1000 = 0.2$ cycles/sample. $\hat{\omega} = 2\pi\hat{f} = 0.4\pi$ radians/sample. Not aliased.

   (c) $f = 500$Hz: $\omega = 2\pi f = 1000\pi$ radians/sec. $\hat{f} = f/f_s = 500/1000 = 0.5$ cycles/sample. $\hat{\omega} = 2\pi\hat{f} = \pi$ radians/sample. Not aliased.

   (d) $f = 1000$Hz: $\omega = 2\pi f = 2000\pi$ radians/sec. $\hat{f} = f/f_s = 1000/1000 = 1$ cycle/sample. $\hat{\omega} = 2\pi\hat{f} = 2\pi$ radians/sample. This is aliased, because it is greater than the Nyquist limit, $f_s/2 = 500$Hz ($\hat{\omega}_{Nyquist} = \pi$).

5. Suppose that we sample a signal at 44.1kHz (the sampling rate used in audio CDs). For each of the following analog frequencies f, determine ω, $\hat{f}$, and $\hat{\omega}$. Indicate if that frequency will be aliased.

   (a) $f = 100$Hz: $\omega = 2\pi f = 200\pi$ radians/sec. $\hat{f} = f/f_s = 100/44100 \approx 0.0023$ cycles/sample. $\hat{\omega} = 2\pi\hat{f} \approx 0.014$ radians/sample. Not aliased.

   (b) $f = 1000$Hz: $\omega = 2\pi f = 2000\pi$ radians/sec. $\hat{f} = f/f_s = 1000/44100 \approx 0.023$ cycles/sample. $\hat{\omega} = 2\pi\hat{f} \approx 0.14$ radians/sample. Not aliased.

   (c) $f = 10000$Hz: $\omega = 2\pi f = 20000\pi$ radians/sample. $\hat{f} = f/f_s = 10000/44100 \approx 0.23$ cycles/sample. $\hat{\omega} = 2\pi\hat{f} \approx 1.4$ radians/sample. Not aliased.

   (d) $f = 20000$Hz: $\omega = 2\pi f = 40000\pi$ radians/sec. $\hat{f} = f/f_s = 20000/44100 \approx 0.45$ cycles/sample. $\hat{\omega} = 2\pi\hat{f} \approx 2.8$ radians/sample. Not aliased.

   (e) $f = 25000$Hz: $\omega = 2\pi f = 50000\pi$ radians/sec. $\hat{f} = f/f_s = 25000/44100 \approx 0.57$ cycles/sample. $\hat{\omega} = 2\pi\hat{f} \approx 3.6$ radians/sample. This is aliased, because it is greater than the Nyquist limit, $f_s/2 = 22.05$kHz ($\hat{\omega}_{Nyquist} = \pi$).

6. Write equation (3-22) for $k = 0, 1, 2, 3$, then write the transfer function for each.

   Answer: For $k = 0$, $y[n] = b_0 x[n]$ and $H(z) = b_0$; $k = 1$, $y[n] = x[n](b_0 + b_1 e^{-j\hat{\omega}})$ and $H(z) = b_0 + b_1 z^{-1}$; $k = 2$, $y[n] = x[n](b_0 + b_1 e^{-j\hat{\omega}} + b_2 e^{-2j\hat{\omega}})$ and $H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2}$; $k = 3$, $y[n] = x[n](b_0 + b_1 e^{-j\hat{\omega}} + b_2 e^{-2j\hat{\omega}} + b_3 e^{-3j\hat{\omega}})$ and $H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}$.

7. Given the signal $x(t) = \sin t$ and the derivative operator $D = \mathrm{d}/\mathrm{d}t$, what is $Dx(t)$?

   Answer: $Dx(t) = \cos t$.

8. When

$$H_1(z) = b_0 + b_1 z^{-1}$$
$$H_2(z) = b'_0 + b'_1 z^{-1}$$

   with b_0, b_1, b'_0, and b'_1 are constants, show that $H_2(z)H_1(z) = H_1(z)H_2(z)$.

   Answer: Multiplying $H_1(z)H_2(z)$, we obtain $(b_0 + b_1 z^{-1})(b'_0 + b'_1 z^{-1}) = b_0 b'_0 + (b_0 b'_1 + b_1 b'_0)z^{-1} + b_1 b'_1 z^{-2}$. Multiplying $H_2(z)H_1(z)$, we get $(b'_0 + b'_1 z^{-1})(b_0 + b_1 z^{-1}) = b'_0 b_0 + (b'_0 b_1 + b'_1 b_0)z^{-1} + b'_1 b_1 z^{-2}$. Because multiplication is commutative, these two expressions are equal. In other words, series combination of filters is commutative because multiplication of polynomials is commutative.

9. Prove $|z^2| = 1$ in equation (3-38).

   Answer: $|z^2| = |e^{2j\omega}|$. From Euler's formula, this is $|\cos 2\omega + j \sin 2\omega|$. Since the magnitude of a complex number in rectangular form is the square root of the sum of the squares of its real and imaginary components, and $\cos^2 2\omega + \sin^2 2\omega = 1$, $|z^2| = 1$.

10. Starting with the factored magnitude response in equation (3-41), derive expressions for b_1 and b_2 in terms of z_1 and z_2.

    Answer: The factored magnitude response is $|H(z)| = |(z - z_1)(z - z_2)|$. Multiplying the two terms out yields $|H(z)| = |z^2 - zz_1 - zz_2 + z_1z_2| = |z^2 - (z_1 + z_2)z + z_1z_2|$. Because $|H(z)| = |z^2 + b_1z + b_2|$, $b_1 = -(z_1 + z_2)$ and $b_2 = z_1z_2$.

11. What abstract data type (ADT) should hold the delayed inputs?

    Answer: a queue.

A.4 Chapter 4: The Z-Transform and Convolution

1. Determine the z-transform for the sequence $x[n] = \{1, 2, 5, 7, 0, 1\}$, $n = 0, 1, 2, 3, 4, 5$

   Answer:

$$X(z) = 1 + 2z^{-1} + 5z^{-2} + 7z^{-3} + z^{-5} \tag{A-1}$$

2. Determine the z-transform of the sequence $x[n] = \{1, 2, 5, 7, 0, 1\}$, $n = -2, -1, 0, 1, 2, 3$

   Answer:

$$X(z) = 1z^2 + 2z + 5 + 7z^{-1} + z^{-3} \tag{A-2}$$

3. Sketch equation (4-6).

   Answer: In a magnitude ($|\mathcal{X}(\omega)|$) vs. frequency plot, it is a horizontal line of value one.

4. Compute the z-transform and frequency content for the signal $\delta[n + n_0]$.

   Answer:

$$\begin{aligned}\Delta(z) &= \sum_{k=-\infty}^{\infty} \delta[k + n_0]z^{-k} \\ &= 1z^{n_0} = z^{n_0}\end{aligned}$$

   The convergence region is entire z plane, except $z = \infty$. The frequency content is $|\mathcal{D}(\hat{\omega})| = |\Delta(e^{j\hat{\omega}})| = |e^{jn_0\hat{\omega}}| = 1$.

5. What is the derivative of $u[n - k]$ (the unit step at time step k)?

   Answer: $\delta[n - k]$.

6. Show that $e^{j\hat{\omega}/2} - e^{-j\hat{\omega}/2} = 2j \sin \omega/2$.

   Answer: Euler's formula states that $e^{j\theta} = \cos\theta + j \sin\theta$. For negative angles, $e^{-j\theta} = \cos(-\theta) + j \sin(-\theta) = \cos\theta - j \sin\theta$ (because cosine is an even function and sine is an odd function). The difference of these two is $e^{j\theta} - e^{-j\theta} = 2j \sin\theta$; substitute $\theta = \hat{\omega}/2$ to finish up.

7. Prove that $e^{-j\pi/2} = -j$.

   Answer: Using Euler's formula, $e^{-j\pi/2} = \cos \pi/2 - j \sin \pi/2 = 0 - j \times 1 = -j$.

8. Determine if $u[n] * H \neq H$ is true, where $h[n] = n$, $n = 0, 1, 2, \ldots$ (a *ramp*).

   Answer: Yes, it is true:

$$\begin{aligned} u[n] * H &= \sum_{k=0}^{n} 1(n-k) \\ &= \sum_{k=0}^{n} n - \sum_{k=0}^{n} k \\ &= n^2 - \frac{n(n+1)}{2} \\ &= \frac{n(n-1)}{2} \\ &\neq n \end{aligned}$$

9. Compute $u[n] * u[n]$.

   Answer:

$$u[n] * u[n] = \sum_{k=0}^{n} 1 = n \tag{A-3}$$

   So $u[n] * u[n] \neq u[n]$.

10. How does the MATLAB function `conv` deal with boundary conditions?

    Answer: It zero pads.

11. Use MATLAB to compute the convolution $e^{-n} * e^{-n}$ and plot the result.

    Hint: use `conv` and `plot`.

12. Prove the scaling property of the z-transform; that is, if

$$x[n] \overset{\mathbf{Z}}{\longleftrightarrow} X(z)$$

    then

$$a^n x[n] \overset{\mathbf{Z}}{\longleftrightarrow} X(a^{-1}z)$$

    Answer: From (4-1),

$$\begin{aligned} Z\{a^n x[n]\} &= \sum_{n=-\infty}^{\infty} a^n x[n] z^{-n} \qquad\qquad = \sum_{n=-\infty}^{\infty} x[n](a^{-1}z)^{-n} \\ &= X(a^{-1}z) \end{aligned}$$

A.5 Chapter 5: Feedback Filters

1. Derive equation (5-33) from (5-32).

   Answer: Start from (5-32):

$$2(1-2R+R^2) = 1-2R\cos\hat{\omega}_B + R^2$$
$$1-4R+r^2 = -2R\cos\hat{\omega}_B$$

   so

$$\cos\hat{\omega}_B = -\frac{1}{2R}(1-4R+R^2)$$
$$= 2-\frac{1}{2}(R+\frac{1}{R})$$

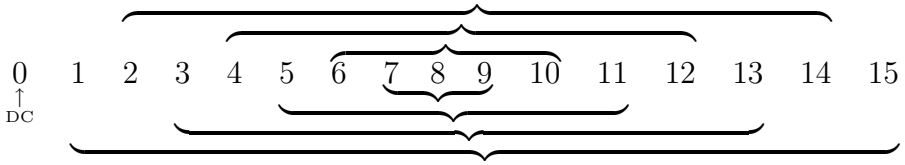
2. In the situation where the sampling rate is 44,100Hz and the desired bandwidth is 20Hz, R in (5-38) is 0.998575. Solve for R the situation where the desired bandwidth is 200Hz. Is it true that when R is far away from one, B grows large?

   Answer: $R = 1-\pi(200/44100) = 0.9858$. Yes.

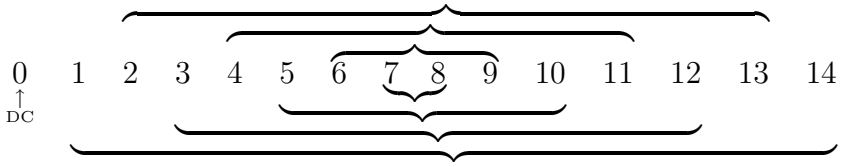
A.6 Chapter 6: Spectral Analysis

1. Show which frequencies will be equal for a spectrum with:

   (a) 16 points.
   Answer:



   (b) 15 points.
   Answer:



2. Prove that the DFT of $x[n]$ for any $n = m$ and $N = 1$ is $x[m]$.

   Answer:

$$X[k] = \sum_{n=m}^{m} x[n]e^{-jnk2\pi/1} = x[m]e^{jmk2\pi} = x[m]$$

   because k, m are integers and so $e^{jmk2\pi} = 1$.

3. Perform step-by-step division for the example given in table 6.2 to prove the final result is equal to the bit-reversed input.

   Answer: The answer given in decimal is in the following table:

Input	N/2	N/4	N/8
0	0	0	0
1	2	4	4
2	4	2	2
3	6	6	6
4	1	1	1
5	3	5	5
6	5	3	3
7	7	7	7

4. Perform the 4-point FFT of the signal $x[n] = \{1, 2, 3, 4\}$ by hand.

   Answer: We first do the bit reversal to get:

Input		Bit-Reversed Result		
Decimal	Binary	Binary	Decimal	Value
0	00	00	0	1
1	01	10	2	3
2	10	01	1	2
3	11	11	3	4

   Each row of this table corresponds to a 1-point FFT. Starting with $N = 2$, we combine neighbors to form the two-point FFTs according to equations (6-39) and (6-40):

	k	$X[k]$
even	0	4
	1	-2
odd	0	6
	1	-2

   We now have a 2-point "even" FFT and a 2-point "odd" FFT. We can combine corre-

sponding elements of this using $k = \{0, 1, 2, 3\}$ and $N = 4$; the four equations are:

$$\begin{aligned}
X[0] &= X[0]^{even} + e^{-j(0)2\pi/4} X[0]^{odd} \\
&= X[0]^{even} + X[0]^{odd} \\
X[1] &= X[1]^{even} + e^{-j(1)2\pi/4} X[1]^{odd} \\
&= X[1]^{even} + e^{-j\pi/2} X[1]^{odd} \\
&= X[1]^{even} - jX[1]^{odd} \\
X[2] &= X[0]^{even} + e^{-j(2)2\pi/4} X[0]^{odd} \\
&= X[0]^{even} + e^{-j\pi} X[0]^{odd} \\
&= X[0]^{even} - X[0]^{odd} \\
X[3] &= X[1]^{even} + e^{-j(3)2\pi/4} X[1]^{odd} \\
&= X[1]^{even} + e^{-j3\pi/2} X[1]^{odd} \\
&= X[1]^{even} + jX[1]^{odd}
\end{aligned}$$

And so the FFT values (and the *power spectrum*, $|X[k]|^2$) are:

k	$X[k]$	$\|X[k]\|^2$
0	10	100
1	-2 + j2	8
2	-2	4
3	-2 - j2	8

5. Fill in the steps leading from (6-51) to (6-52).

   Answer: Equation (6-51) is a geometric series with common ratio $e^{-j\hat{\omega}_k}$, so

   $$W(\hat{\omega}_k) = \frac{1 - e^{-j\hat{\omega}_k N}}{1 - e^{-j\hat{\omega}_k}}$$

   Using Euler's formula, the numerator becomes

   $$1 - e^{-j\hat{\omega}_k N} = e^{-j\hat{\omega}_k N/2}(e^{j\hat{\omega}_k N/2} - e^{-j\hat{\omega}_k N/2}) = 2je^{-j\hat{\omega}_k N/2} \sin(\hat{\omega}_k N/2)$$

   Similarly, the denominator is $2je^{-j\hat{\omega}_k/2} \sin(\hat{\omega}_k/2)$, so

   $$W(\hat{\omega}_k) = \frac{\sin(\hat{\omega}_k N/2)}{\sin \hat{\omega}_k/2} e^{-j\hat{\omega}_k(N-1)/2}$$

   which is (6-52).

6. Plot the Hann and Hamming windows in the time domain and compare their shapes.

   Hint: Use the MATLAB built-in commands `hamming()` and `hanning()`.

A.7 Chapter 7: Compression

1. If a rate of 44,100 samples/second at 16bits/sample, what is the digital data rate in bits/second?

   Answer: 705.6kb/s.

2. If we are digitizing high-quality video — 1k x 1k pixels/frame, 30 frames/sec, 24 bits/pixel — what is the bit rate?

   Answer: 755Mb/s.

3. If a signal is sent in which all samples have the same value, what is the information content in bits (ignoring the first sample)?

   Answer: Since you can predict all subsequent signals with 100% accuracy, no additional information is sent after the first sample (this assumes infinite signal length; otherwise, there is additional information — the number of samples).

4. What kind of signal would have maximum information content?

   Answer: It would have to be a signal in which the next sample could never be predicted at better than chance, regardless of the number of previous samples used as "clues" to the next sample's value. So, for example, if there were 8 bits/sample, the chance of predicting the next signal would have to be 1/256. Such an unpredictable signal is called *stochastic*, or *random*. In multimedia terms, *noise*.

5. Can you give an example of an application which would demand symmetric coding?

   Answer: Video conferencing. Both ends typically have the same hardware, both ends must perform both encoding and decoding, and both operations must be done in real time.

6. Can you give an example of an application which could allow asymmetric coding?

   Answer: Video broadcasting. If the broadcast is not live, then large computers can be allowed long times to optimize a recording. Even in a live broadcast, the studio can invest more money in encoding equipment than the viewer in decoding equipment (TVs or computers).

7. Two example codes with the prefix property are given in Table 7.1. Decoding code 1 is easy, as we can just read three bits at a time (for example, decode "001010011").

   Answer: "2, 3, 4".

8. What would the symbol sequence be for "01000001000"?

   Answer: "3141".

A.8 Chapter 8: Audio & Video Compression and Coding

1. For almost all versions, the input signal is assumed to be 20kHz. What is the minimum sampling rate for such a signal?

   Answer: 40kHz.

2. Question: under what conditions is it acceptable to have greater coder complexity?

   Answer: When coding is done once and not in real time, or is done by someone with a lot of money, like a TV station.

A.9 Chapter 9: Review and Conclusions

1. Question: what's wrong with low-pass filters with abrupt cutoffs (sometimes called *brick wall* filters)?

   Answer: Phase distortion; filters with steep cutoffs introduce large, frequency dependent delays, or phase shifts.

Index