Note that if carries are ignored, subtraction of two single-digit binary numbers yields the same bit as addition. Computers use high and low voltage values to express a bit, and an array of such voltages express numbers akin to positional notation. Logic circuits perform arithmetic operations.

Exercise 5.2.3

Add twenty-five and seven in base 2. Note the carries that might occur. Why is the result "nice"? The variables of logic indicate truth or falsehood. $\mathbf{A} \cap \mathbf{B}$, the AND of **A** and **B**, represents a statement that both **A** and **B** must be true for the statement to be true. You use this kind of statement to tell search engines that you want to restrict hits to cases where both of the events **A** and **B** occur. $\mathbf{A} \cup \mathbf{B}$, the OR of A and B, yields a value of truth if either is true. Note that if we represent truth by a "1" and falsehood by a "0," **binary multiplication corresponds to AND and addition (ignoring carries) to XOR**. XOR, the exclusive or operator, equals the union of $\mathbf{A} \cup \mathbf{B}$ and $\mathbf{A} \cap \mathbf{B}$. The Irish mathematician George Boole discovered this equivalence in the mid-nineteenth century. It laid the foundation for what we now call Boolean algebra, which expresses as equations logical statements. More importantly, any computer using base-2 representations and arithmetic can also easily evaluate logical statements. This fact makes an integer-based computational device much more powerful than might be apparent.

5.3 The Sampling Theorem

5.3.1 Analog-to-Digital Conversion

Because of the way computers are organized, signal must be represented by a fnite number of bytes. This restriction means that **both** the time axis and the amplitude axis must be **quantized**: They must each be a multiple of the integers. l2 Quite surprisingly, the Sampling Theorem allows us to quantize the time axis without error for some signals. The signals that can be sampled without introducing error are interesting, and as described in the next section, we can make a signal "samplable" by filtering. In contrast, no one has found a way of performing the amplitude quantization step without introducing an unrecoverable error. Thus, a signal's value can no longer be any real number. Signals processed by digital computers must be **discrete-valued**: their values must be proportional to the integers. Consequently, **analog-to-digital conversion introduces error**.

5.4 The Sampling Theorem

Digital transmission of information and digital signal processing all require signals to first be "acquired" by a computer. One of the most amazing and useful results in electrical engineering is that signals can be converted from a function of time into a sequence of numbers **without error**: We can convert the numbers back into the signal with (theoretically) **no** error. Harold Nyquist, a Bell Laboratories engineer, first derived this result, known as the Sampling Theorem, in the 1920s. It found no real application back then. Claude Shannon, also at Bell Laboratories, revived the result once computers were made public after World War II.

The sampled version of the analog signal $s(t)$ is $s(nT_s)$, with T_s known as the **sampling interval**. Clearly, the value of the original signal at the sampling times is preserved; the issue is how the signal values between the samples can be reconstructed since they are lost in the sampling process. To characterize sampling, we approximate it as the product $x(t) = s(t)\,PT_s(t)$, with $PT_s(t)$ being the periodic pulse signal. The resulting signal, as shown in Figure 5.3 (Sampled Signal), has nonzero values only during the time intervals

$$\left(nT_s - \frac{\Delta}{2}, nT_s + \frac{\Delta}{2}\right), n \in \{\ldots, -1, 0, 1, \ldots\}.$$

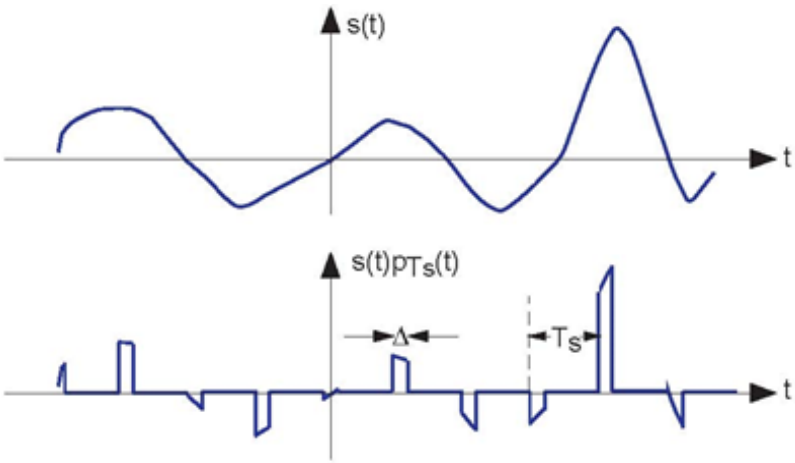

Figure 5.3 Sampled Signal The waveform of an example signal is shown in the top plot and its sampled version in the bottom.

For our purposes here, we center the periodic pulse signal about the origin so that its Fourier series coefficients are real (the signal is even).

$$PT_s(t) = \sum_{k=-\infty}^{\infty} \left(c_k e^{\frac{j2\pi k t}{T_s}}\right)$$

$$c_k = \frac{sin\left(\frac{\pi k \Delta}{T_s}\right)}{\pi k}$$

If the properties of ***s (t)*** and the periodic pulse signal are chosen properly, we can recover ***s (t)*** from ***x (t)*** by filtering.

To understand how signal values between the samples can be "filled" in, we need to calculate the sampled signal's spectrum. Using the Fourier series representation of the periodic sampling signal,

$$x(t) = \sum_{k=-\infty}^{\infty} \left(c_k e^{\frac{j2\pi kt}{T_s}} s(t) \right)$$

Considering each term in the sum separately, we need to know the spectrum of the product of the complex exponential and the signal. Evaluating this transform directly is quite easy.

$$\int_{-\infty}^{\infty} s(t) e^{\frac{j2\pi kt}{T_s}} e^{-(j2\pi kt)} dt = \int_{-\infty}^{\infty} s(t) e^{-(j2\pi (f - \frac{k}{T_s})t)} dt = S\left(f - \frac{k}{T_s}\right)$$

Thus, the spectrum of the sampled signal consists of weighted (by the coefficients ck) and delayed versions of the signal's spectrum (Figure 5.4 (aliasing)).

$$X(f) = \sum_{k=-\infty}^{\infty} \left(c_k S\left(f - \frac{k}{T_s}\right) \right)$$

In general, the terms in this sum overlap each other in the frequency domain, rendering recovery of the original signal impossible. This unpleasant phenomenon is known as **aliasing**.

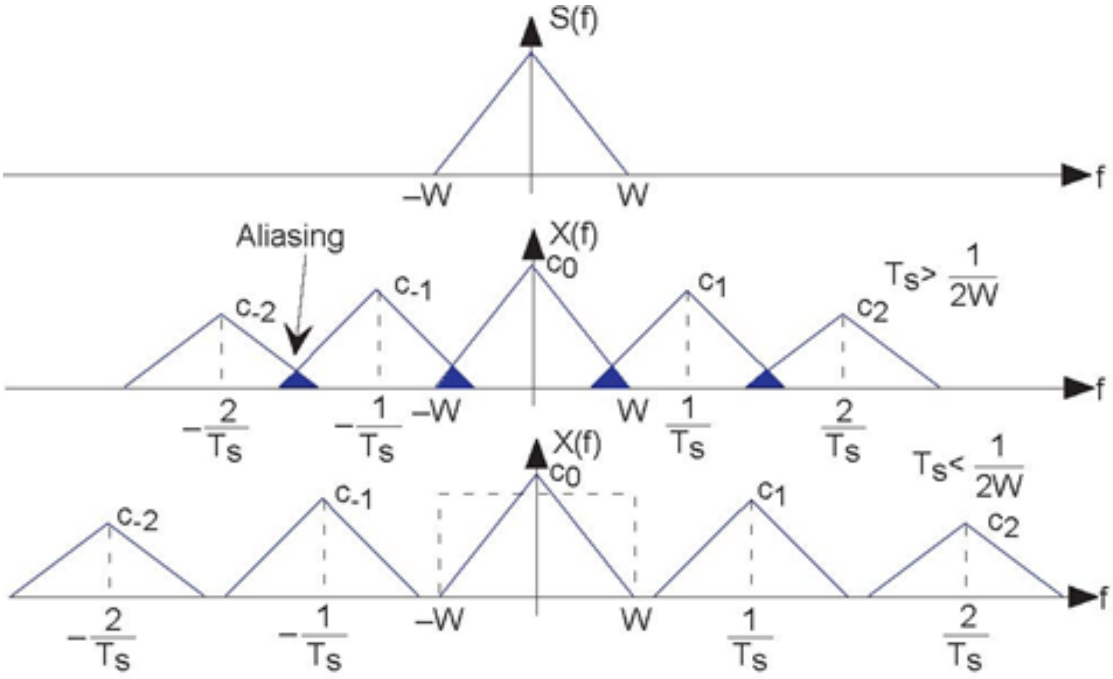

Figure 5.4 aliasing The spectrum of some bandlimited (to **W** Hz) signal is shown in the top plot. If the sampling interval $\mathbf{T}_S$ is chosen too large relative to the bandwidth **W**, aliasing will occur. In the bottom plot, the sampling interval is chosen sufficiently small to avoid aliasing. Note that if the signal were not bandlimited, the component spectra would always overlap.

If, however, we satisfy two conditions:

- The signal s (t) is **bandlimited** has power in a restricted frequency range to **W** Hz, and
- the sampling interval **Ts** is small enough so that the individual components in the sum do not overlap **Ts** < 1/2**W** ,

aliasing will not occur. In this delightful case, we can recover the original signal by lowpass filtering x (t) with a filter having a cutoff frequency equal to **W** Hz. These two conditions ensure the ability to recover a bandlimited signal from its sampled version: We thus have the **Sampling Theorem**.

Exercise 5.3.1

The Sampling Theorem (as stated) does not mention the pulse width Δ. What is the effect of this parameter on our ability to recover a signal from its samples (assuming the Sampling Theorem's two conditions are met)?

The frequency

$$\frac{1}{2T_s}$$

, known today as the **Nyquist frequency** and the **Shannon sampling frequency**, corresponds to the highest frequency at which a signal can contain energy and remain compatible with the Sampling Theorem. High-quality sampling systems ensure that no aliasing occurs by unceremoniously lowpass filtering the signal (cutoff frequency being slightly lower than the Nyquist frequency) before sampling. Such systems therefore vary the **anti-aliasing** filter's cutoff frequency as the sampling rate varies. Because such quality features cost money, many sound cards do **not** have anti-aliasing filters or, for that matter, post-sampling filters. They sample at high frequencies, 44.1 kHz for example, and hope the signal contains no frequencies above the Nyquist frequency (22.05 kHz in our example). If, however, the signal contains frequencies beyond the sound card's Nyquist frequency, the resulting aliasing can be impossible to remove.

Exercise 5.3.2

To gain a better appreciation of aliasing, sketch the spectrum of a sampled square wave. For simplicity consider only the spectral repetitions centered at

$$-\left(\frac{1}{T_s}\right),\ 0,\ \frac{1}{T_s}.$$

Let the sampling interval ***Ts*** be 1; consider two values for the square wave's period: 3.5 and 4. Note in particular where the spectral lines go as the period decreases; some will move to the left and some to the right. What property characterizes the ones going the same direction?

If we satisfy the Sampling Theorem's conditions, the signal will change only slightly during each pulse. As we narrow the pulse, making Δ smaller and smaller, the nonzero values of the signal ***s(t) pTs (t)*** will simply be ***s*** (***nTs***), the signal's **samples**. If indeed the Nyquist frequency equals the signal's highest frequency, at least two samples will

occur within the period of the signal's highest frequency sinusoid. In these ways, the sampling signal captures the sampled signal's temporal variations in a way that leaves all the original signal's structure intact.

Exercise 5.3.3

What is the simplest bandlimited signal? Using this signal, convince yourself that less than two samples/period will not suffice to specify it. If the sampling rate

$$\frac{1}{T_s}$$

is not high enough, what signal would your resulting undersampled signal become?

5.5 Amplitude Quantization

The Sampling Theorem says that if we sample a bandlimited signal s (t) fast enough, it can be recovered without error from its samples **s (nTs), n ∈{..., −1, 0, 1,... }.** Sampling is only the first phase of acquiring data into a computer: Computational processing further requires that the samples be **quantized**: analog values are converted into digital (Section 1.2.2: Digital Signals) form. In short, we will have performed **analog-to-digital (A/D) conversion**.

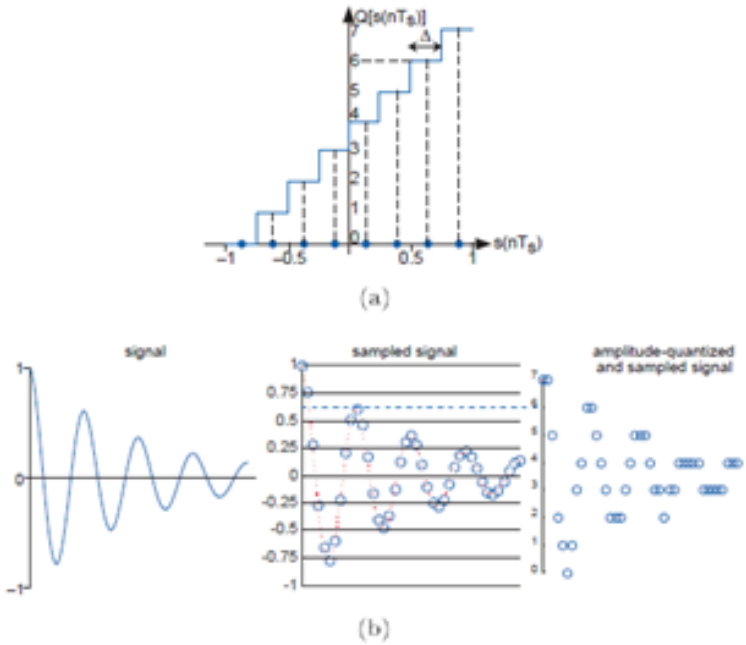

Figure 5.5 A three-bit A/D converter A three-bit A/D converter assigns voltage in the range [−1, 1] to one of eight integers between 0 and 7. For example, all inputs having values lying between 0.5 and 0.75 are assigned the integer value six and, upon conversion back to an analog value, they all become 0.625. The width of a single quantization interval Δ equals . The bottom panel shows a signal going through the analog-to-digital converter, where B is the number of bits used in the A/D conversion process (3 in the case depicted here). First it is sampled, then amplitude-quantized to three bits. Note how the sampled signal waveform becomes distorted after amplitude quantization. For example the two signal values between 0.5 and 0.75 become 0.625. This distortion is irreversible; it can be reduced (but not eliminated) by using more bits in the A/D converter.

A phenomenon reminiscent of the errors incurred in representing numbers on a computer prevents signal amplitudes from being converted with no error into a binary number representation. In analog-to-digital conversion, the signal is assumed to lie within a predefined range. Assuming we can scale the signal without affecting the information it expresses, we'll define this range to be [−1, 1]. Furthermore, the A/D converter assigns amplitude values in this range to a set of integers. A B-bit converter produces one of the integers **{0, 1,..., 2B− 1}**for each sampled input. Figure 5.5 shows how a three-bit A/D converter assigns input values to the integers. We define a **quantization interval** to be the range of values assigned to the same integer. Thus, for our example three-bit A/D converter, the quantization interval Δ is 0.25; in general, it is

$$\frac{2}{2^B}$$

.

Exercise 5.4.1

Recalling the plot of average daily highs in this frequency domain problem (Problem 4.5), why is this plot so jagged? Interpret this effect in terms of analog-to-digital conversion.

Because values lying anywhere within a quantization interval are assigned the same value for computer processing, **the original amplitude value cannot be recovered without error**. Typically, the D/A converter, the device that converts integers to amplitudes, assigns an amplitude equal to the value lying halfway in the quantization interval. The integer 6 would be assigned to the amplitude 0.625 in this scheme. The error introduced by converting a signal from analog to digital form by sampling and amplitude quantization then back again would be half the quantization interval for each amplitude value. Thus, the so-called **A/D** error equals half the width of a quantization interval:

$$\frac{1}{2^B}$$

. As we have fixed the input-amplitude range, the more bits available in the A/D converter, the smaller the quantization error.

To analyze the amplitude quantization error more deeply, we need to compute the signal-to-noise ratio, which equals the ratio of the signal power and the quantization error power. Assuming the signal is a sinusoid, the signal power is the square of the rms amplitude: **power**

$$power(s) = \left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}.$$

The illustration (Figure 5.6) details a single quantization interval.

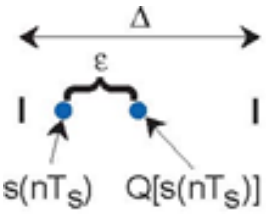

Figure 5.6 A single quantization interval A single quantization interval is shown, along with a typical signal's value before amplitude quantization s (nTs) and after Q (s (nTs)). E denotes the error thus incurred.

Its width is Δ and the quantization error is denoted by E. To find the power in the quantization error, we note that no matter into which quantization interval the signal's value falls, the error will have the same characteristics. To calculate the rms value, we must square the error and average it over the interval.

$$\begin{aligned} rms(\epsilon) &= \sqrt{\frac{1}{\triangle}\int_{-\left(\frac{\triangle}{2}\right)}^{\frac{\triangle}{2}} \epsilon^2 d\epsilon} \\ &= \left(\frac{\triangle^2}{12}\right)^{\frac{1}{2}} \end{aligned}$$

Since the quantization interval width for a *B*-bit converter equals

$$\frac{2}{2^B} = 2^{-(B-1)},$$

we find that the signal-tonoise ratio for the analog-to-digital conversion process equals

$$SNR = \frac{\frac{1}{2}}{\frac{2^{-(2(B-1))}}{12}} = \frac{3}{2}2^{2B} = 6B + 10log_{10}\ 1.5dB$$

Thus, every bit increase in the A/D converter yields a 6 dB increase in the signal-to-noise ratio. The constant term **10log1.5** equals 1.76.

Exercise 5.4.2

This derivation assumed the signal's amplitude lay in the range [−1, 1]. What would the amplitude quantization signal-to-noise ratio be if it lay in the range $[-A, A]$?

Exercise 5.4.3

How many bits would be required in the A/D converter to ensure that the maximum amplitude quantization error was less than 60 db smaller than the signal's peak value?

Exercise 5.4.4

Music on a CD is stored to 16-bit accuracy. To what signal-to-noise ratio does this correspond?

Once we have acquired signals with an A/D converter, we can process them using digital hardware or software. It can be shown that if the computer processing is linear, the result of sampling, computer processing, and unsampling is equivalent to some analog linear system. Why go to all the bother if the same function can be accomplished using analog techniques? Knowing when digital processing excels and when it does not is an important issue.

5.6 Discrete-Time Signals and Systems

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

5.6.1 Real-and Complex-valued Signals

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

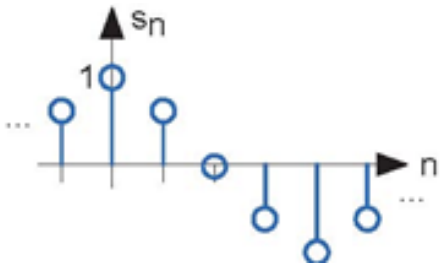

Figure 5.7 Cosine The discrete-time cosine signal is plotted as a stem plot. Can you find the formula for this signal?

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

5.6.2 Complex Exponentials

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

$$s(n) = e^{j2\pi fn}$$

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

$$\begin{aligned} e^{j2\pi(f+m)n} &= e^{j2\pi fn}e^{j2\pi mn} \\ &= e^{j2\pi fn} \end{aligned}$$

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

5.6.3 Sinusoids

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

$$\left(-\frac{1}{2}, \frac{1}{2}\right]$$

This choice of frequency interval is arbitrary; we can also choose the frequency to lie in the interval [0,1). How to choose a unit-length interval for a sinusoid's frequency will become evident later.

5.6.4 Unit Sample

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

$$\delta(n) = \begin{cases} 1 \; if \; n = 0 \\ 0 \; otherwise \end{cases}$$

(5.14)

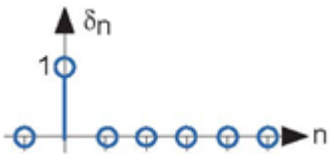

Figure 5.8 The unit sample.

Examination of a discrete-time signal's plot, like that of the cosine signal shown in Figure 5.7(Cosine), reveals that all signals consist of a sequence of delayed and scaled unit samples. Because the value of a sequence at each integer m is denoted by s(*m*) and the unit sample delayed to occur at m is written δ(n - m), we can decompose **any** signal as a sum of unit samples delayed to the appropriate location and scaled by the signal value.

$$s(n) = \sum_{m=-\infty}^{\infty} (s(m)\delta(n-m))$$

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

5.6.5 Unit Step

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

$$u(n) = \begin{cases} 1 \; if \; n \geq 0 \\ 0 \; if \; n < 0 \end{cases}$$

(5.16)

5.6.6 Symbolic Signals

An interesting aspect of discrete-time signals is that their values do not need to be real numbers. We do have real-valued discrete-time signals like the sinusoid, but we also have signals that denote the sequence of characters typed on the keyboard. Such characters certainly aren't real numbers, and as a collection of possible signal values, they have little mathematical structure other than that they are members of a set. More formally, each element of the **symbolic-valued** signal *s(n)* takes on one of the values $\{a_1,\ldots,a_K\}$ which comprise the **alphabet *A***. This technical terminology does not mean we restrict symbols to being members of the English or Greek alphabet. They could represent keyboard characters, bytes (8-bit quantities), integers that convey daily temperature. Whether controlled by software or not, discrete-time systems are ultimately constructed from digital circuits, which consist **entirely** of analog circuit elements. Furthermore, the transmission and reception of discrete-time signals, like e-mail, is accomplished with analog signals and systems. Understanding how discrete-time and analog signals and systems intertwine is perhaps the main goal of this course.

5.6.7 Discrete-Time Systems

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

5.7 Discrete-Time Fourier Transform (DTFT)

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

$$S(e^{j2\pi f}) = \sum_{n=-\infty}^{\infty} \left(s(n)e^{-(j2\pi fn)}\right)$$

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

$$S(e^{-(j2\pi f)}) = S(e^{j2\pi f})^*.$$

Exercise 5.6.1

A special property of the discrete-time Fourier transform is that it is periodic with period one:

$$S\left(e^{j2\pi(f+1)}\right) = S\left(e^{j2\pi f}\right).$$

Derive this property from the Definition of the DTFT.

Because of this periodicity, we need only plot the spectrum over one period to understand completely the spectrum's structure; typically, we plot the spectrum over the frequency range

$$\left[-\left(\frac{1}{2}\right), \frac{1}{2}\right].$$

When the signal is real-valued, we can further simplify our plotting chores by showing the spectrum only over

$$\left[0, \frac{1}{2}\right];$$

the spectrum at negative frequencies can be derived from positive-frequency spectral values. When we obtain the discrete-time signal via sampling an analog signal, the Nyquist frequency (p. 176) corresponds to the discrete-time frequency

$$\frac{1}{2}.$$

To show this, note that a sinusoid having a frequency equal to the Nyquist frequency

$$\frac{1}{2T_S}$$

has a sampled waveform that equals

$$\cos\left(2\pi \times \frac{1}{2Ts} nTs\right) = \cos(\pi n) = (-1)^n$$

The exponential in the DTFT at frequency

$$\frac{1}{2}$$

equals

$$e^{-\left(\frac{j2\pi n}{2}\right)} = e^{-(j\pi n)} = (-1)^n$$

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

$f_D = f_A T_S$ (5.18)

f_D and f_A represent discrete-time and analog frequency variables, respectively. The aliasing figure (Figure 5.4) provides another way of deriving this result. As the duration of each pulse in the periodic sampling signal $PT_S(t)$ narrows, the amplitudes of the signal's spectral repetitions, which are governed by the Fourier series coefficients (4.10) of $PT_S(t)$, become increasingly equal. Examination of the periodic pulse signal (Figure 4.1) reveals that as Δ decreases, the value of c_0, the largest Fourier coefficient, decreases to zero:

$$|C_0| = \frac{A\Delta}{T_s}.$$

Thus, to maintain a mathematically viable Sampling Theorem, the amplitude A must increase as

$$\frac{1}{\Delta},$$

becoming infinitely large as the pulse duration decreases. Practical systems use a small value of Δ, say $0.1 \cdot T_s$ and use amplifiers to rescale the signal. Thus, the sampled signal's spectrum becomes periodic with period

$$\frac{1}{T_s}$$

. Thus, the Nyquist frequency

$$\frac{1}{2T_s}$$

corresponds to the frequency

$$\frac{1}{2}.$$

Example 5.1

Let's compute the discrete-time Fourier transform of the exponentially decaying sequence $s(n)=a^n u(n)$, where $u(n)$ is the unit-step sequence. Simply plugging the signal's expression into the Fourier transform formula,

$$\begin{aligned} S(e^{j2\pi f}) &= \sum_{n=-\infty}^{\infty} \left(a^n u(n) e^{-(j2\pi fn)}\right) \\ &= \sum_{n=0}^{\infty} \left(\left(ae^{-(j2\pi f)}\right)^n\right) \end{aligned}$$

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

$$\sum_{n=0}^{\infty} (a^n) = \frac{1}{1-\alpha}, \quad |\alpha| < 1$$

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

$$S(e^{j2\pi f}) = \frac{1}{1-ae^{-(j2\pi f)}}$$

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

$$\begin{aligned} \left|s(e^{j2\pi f})\right| &= \frac{1}{\sqrt{(1-a\cos(2\pi f))^2 + a^2\sin^2(2\pi f)}} \\ \angle\left(S\left(e^{j2\pi f}\right)\right) &= -\left(\tan^{-1}\left(\frac{a\sin(2\pi f)}{1-a\cos(2\pi f)}\right)\right) \end{aligned}$$

No matter what value of a we choose, the above formulae clearly demonstrate the periodic nature of the spectra of discrete-time signals. Figure 5.9 (Spectrum of exponential signal) shows indeed that the spectrum is a periodic function. We need only consider the spectrum between

$$-\frac{1}{2}$$

and

$$\frac{1}{2}$$

to unambiguously define it. When a> 0, we have a lowpass spectrum the spectrum diminishes as frequency increases from 0 to $\frac{1}{2}$ with increasing a leading to a greater low frequency content; for a< 0, we have a highpass spectrum (Figure 5.10 (Spectra of exponential signals)).

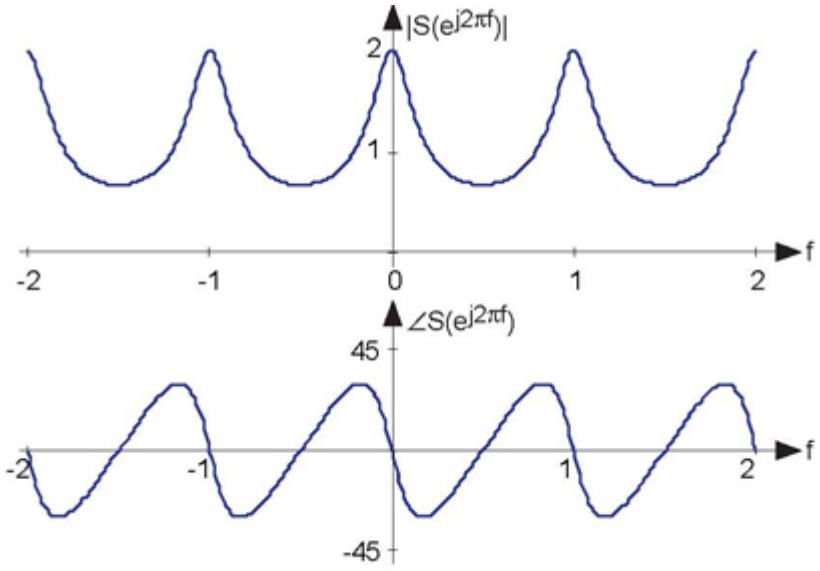

Figure 5.9 Spectrum of exponential signall

The spectrum of the exponential signal (a = 0:5) is shown over the frequency range [-2,2], clearly demonstrating the periodicity of all discrete-time spectra. The angle has units of degrees.

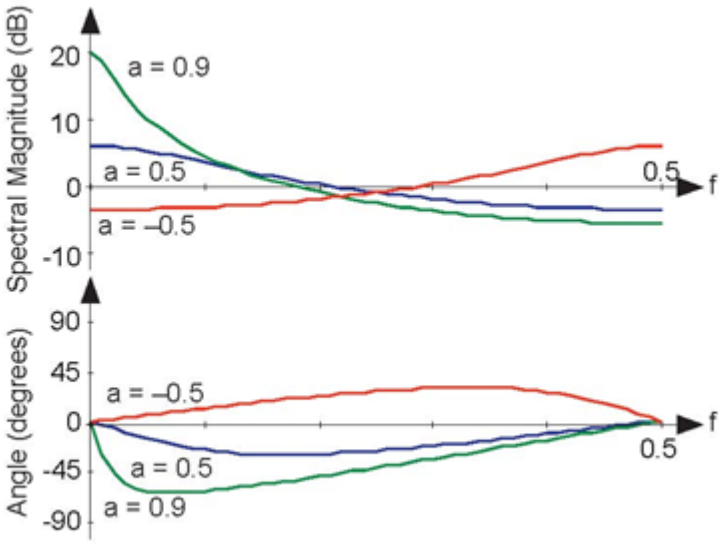

Figure 5.10 Spectra of exponential signals

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

Example 5.2

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

$$s(n) = \begin{cases} 1 \; if \; 0 \leq n \leq N-1 \\ 0 \; otherwise \end{cases}$$

(5.24)

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

$$S(e^{j2\pi f}) = \sum_{n=0}^{N-1} \left(e^{-(j2\pi fn)}\right)$$

(5.25)

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

$$\sum_{n=n_0}^{N+n_0-1} (\alpha^n) = \alpha^{n_0} \frac{1-\alpha^N}{1-\alpha}$$

(5.26)

for **all** values of α.

Exercise 5.6.2

Derive this formula for the finite geometric series sum. The "trick" is to consider the difference between the series' sum and the sum of the series multiplied by α. Applying this result yields (Figure 5.11 (Spectrum of length-ten pulse).)

$$\begin{aligned} S(e^{j2\pi f}) &= \frac{1-e^{-(j2\pi fN)}}{1-e^{-(j2\pi f)}} \\ &= e^{-(j\pi f(N-1))} \frac{\sin(\pi fN)}{\sin(\pi f)} \end{aligned}$$

The ratio of sine functions has the generic form of

$$\frac{\sin(Nx)}{\sin(x)},$$

which is known as the **discrete-time sinc function dsinc (x)**. Thus, our transform can be concisely expressed as

$$S\left(e^{j2\pi f}\right) = e^{-(j\pi f(N-1))} dsinc(\pi f),$$

The discrete-time pulse's spectrum contains many ripples, the number of which increase with *N*, the pulse's duration.

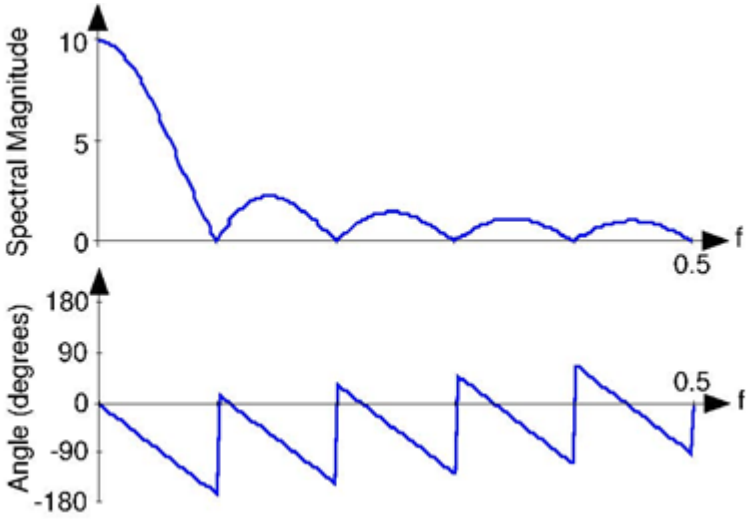

Figure 5.11 Spectrum of length-ten pulse

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

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

Invalid Equation

Therefore, we find that

$$\begin{aligned}\int_{-\left(\frac{1}{2}\right)}^{\frac{1}{2}} S(e^{j2\pi f})e^{j2\pi fn}df &= \int_{-\left(\frac{1}{2}\right)}^{\frac{1}{2}} \sum_m \left(s(m)e^{-(j2\pi fm)}e^{j2\pi fn}\right) df \\ &= \sum_m \left(s\left(m\right) \int_{-\left(\frac{1}{2}\right)}^{\frac{1}{2}} e^{(-(j2\pi f))(m-n)}df \right) \\ &= s\left(n\right)\end{aligned}$$

The Fourier transform pairs in discrete-time are

$$S(e^{j2\pi f}) = \sum_{n=-\infty}^{\infty} \left(s(n)e^{-(j2\pi fn)}\right)$$

$$s(n) = \int_{-\left(\frac{1}{2}\right)}^{\frac{1}{2}} S(e^{j2\pi f})e^{j2\pi fn}df$$

The properties of the discrete-time Fourier transform mirror those of the analog Fourier transform. The DTFT properties table ("Properties of the DTFT" <http://legacy.cnx.org/content/m0506/latest/>) shows similarities and diferences. One important common property is Parseval's Theorem.

$$\sum_{n=-\infty}^{\infty} \left((|s(n)|)^2\right) = \int_{-\left(\frac{1}{2}\right)}^{\frac{1}{2}} \left(\left|S(e^{j2\pi f})\right|\right)^2 df$$

To show this important property, we simply substitute the Fourier transform expression into the frequency-domain expression for power.

$$\int_{-\left(\frac{1}{2}\right)}^{\frac{1}{2}} \left(\left|S(e^{j2\pi f})\right|\right)^2 df = \int_{-\left(\frac{1}{2}\right)}^{\frac{1}{2}} \left(\sum_n \left(s(n)e^{-(j2\pi fn)}\right)\right) \sum_m \left(s(n)^* e^{j2\pi fm}\right) df$$
$$= \sum_{n,m} \left(s(n)s(n)^* \int_{-\left(\frac{1}{2}\right)}^{\frac{1}{2}} e^{j2\pi f(m-n)} df\right)$$

Using the orthogonality relation (5.28), the integral equals δ (m − n)**,** where δ (n) is the unit sample (Figure 5.8: Unit sample). Thus, the double sum collapses into a single sum because nonzero values occur only when n = m, giving Parseval's Theorem as a result. We term

$$\sum_n \left(s^2(n)\right)$$

the energy in the discrete-time signal *s(n)* in spite of the fact that discrete-time signals don't consume (or produce for that matter) energy. This terminology is a carry-over from the analog world.

Exercise 5.6.3

Suppose we obtained our discrete-time signal from values of the product $s(t)p_{T_S}(t)$, where the duration of the component pulses in $p_{T_S}(t)$ is Δ. How is the discrete-time signal energy related to the total energy contained in *s(t)*? Assume the signal is bandlimited and that the sampling rate was chosen appropriate to the Sampling Theorem's conditions.

5.8 Discrete Fourier Transforms (DFT)

The discrete-time Fourier transform (and the continuous-time transform as well) can be evaluated when we have an analytic expression for the signal. Suppose we just have a signal, such as the speech signal used in the previous chapter, for which there is no formula. How then would you compute the spectrum? For example, how did we compute a spectrogram such as the one shown in the speech signal example (Figure 4.17: spectrogram)? The Discrete Fourier Transform (DFT) allows the computation of spectra from discrete-time data. While in discrete-time we can exactly calculate spectra, for analog signals no similar exact spectrum computation exists. For analog-signal spectra, use must build special devices, which turn out in most cases to consist of A/D converters and discrete-time computations. Certainly discrete-time spectral analysis is more flexible than continuous-time spectral analysis.

The formula for the DTFT (5.17) is a sum, which conceptually can be easily computed save for two issues.

- **Signal duration**. The sum extends over the signal's duration, which must be fnite to compute the signal's spectrum. It is exceedingly difcult to store an infnite-length signal in any case, so we'll assume that the signal extends over [0,*N* − 1].
- **Continuous frequency**. Subtler than the signal duration issue is the fact that the frequency variable is continuous: It may only need to span one period, like

$$\left[-\left(\frac{1}{2}\right),\frac{1}{2}\right]$$

or [0, 1], but the DTFT formula as it stands requires evaluating the spectra at all frequencies within a period. Let's compute the spectrum at a few frequencies; the most obvious ones are the equally spaced ones

$$f=\frac{k}{K},\ k\in\left\{0,\cdots,K-1\right\}$$

We thus define the **discrete Fourier transform** (DFT) to be

$$S(k)=\sum_{n=0}^{N-1}\left(s(n)e^{-\left(\frac{j2\pi nk}{K}\right)}\right)\ ,\ k\in\{0,...,K-1\}$$

Here, *S(k)* is shorthand for

$$S\left(e^{j2\pi\frac{k}{K}}\right).$$

We can compute the spectrum at as many equally spaced frequencies as we like. Note that you can think about this computationally motivated choice as **sampling** the spectrum; more about this interpretation later. The issue now is how many frequencies are enough to capture how the spectrum changes with frequency. One way of answering this question is determining an inverse discrete Fourier transform formula: given *S(k)*, *k = {0,...,K − 1}* how do we find *s(n)*, n = {0,...,N − 1}? Presumably, the formula will be of the form

$$s(n)=\sum_{k=0}^{K-1}\left(S(k)e^{\frac{j2\pi nk}{K}}\right).$$

Substituting the DFT formula in this prototype inverse transform yields

$$s(n)=\sum_{k=0}^{K-1}\left(\sum_{m=0}^{N-1}\left(s(m)e^{-(j\frac{2\pi mk}{K})}e^{j\frac{2\pi nk}{K}}\right)\right)$$

Note that the orthogonality relation we use so often has a different character now.

$$\sum_{k=0}^{K-1}\left(e^{-(j\frac{2\pi km}{K})}e^{j\frac{2\pi kn}{K}}\right)=\begin{cases}K\ if\ m=\{n,(n\pm K),(n\pm 2K),...\}\\0\ otherwise\end{cases}$$

We obtain nonzero value whenever the two indices difer by multiples of *K*. We can express this result as

$$K\sum_{l}\left(\delta(m-n-lK)\right).$$

Thus, our formula becomes

$$s(n)=\sum_{m=0}^{N-1}\left(s(m)K\sum_{l=-\infty}^{\infty}\left(\delta(m-n-lK)\right)\right)$$

The integers *n* and *m* both range over {0,...,*N* − 1}. To have an inverse transform, we need the sum to be a **single** unit sample for m, n in this range. If it did not, then *s (n)* would equal a sum of values, and we would not have a valid transform: Once going into the frequency domain, we could not get back unambiguously! Clearly, the term *l* =0 always provides a unit sample (we'll take care of the factor of *K* soon). If we

evaluate the spectrum at **fewer** frequencies than the signal's duration, the term corresponding to *m* = *n* + *K* will also appear for some values of m, n = {0,...,*N* - 1}. This situation means that our prototype transform equals *s (n) + s (n + K)* for some values of n. The only way to eliminate this problem is to require *K* ≥ *N*: We **must** have at least as many frequency samples as the signal's duration. In this way, we can return from the frequency domain we entered via the DFT.

Exercise 5.7.1

When we have fewer frequency samples than the signal's duration, some discrete-time signal values equal the sum of the original signal values. Given the sampling interpretation of the spectrum, characterize this effect a different way.

Another way to understand this requirement is to use the theory of linear equations. If we write out the expression for the DFT as a set of linear equations,

$$s(0) + s(1) + ... + s(N-1) = S(0)$$

$$s(0) + s(1)e^{(-j)\frac{2\pi}{K}} + ... + s(N-1)e^{(-j)\frac{2\pi(N-1)}{K}} = S(1)$$

.....

$$s(0) + s(1)e^{(-j)\frac{2\pi(K-1)}{K}} + ... + s(N-1)e^{(-j)\frac{2\pi(N-1)(K-1)}{K}} = S(K-1)$$

we have *K* equations in *N* unknowns if we want to find the signal from its sampled spectrum. This require ment is impossible to fulfll if *K* < *N*; we must have *K* ≥ *N*. Our orthogonality relation essentially says that if we have a sufcient number of equations (frequency samples), the resulting set of equations can indeed be solved.

By convention, the number of DFT frequency values *K* is chosen to equal the signal's duration *N*. The discrete Fourier transform pair consists of

Discrete Fourier Transform Pair

$$S(k) = \sum_{n=0}^{N-1} \left(s(n) e^{-(j\frac{2\pi nk}{N})} \right)$$

$$s(n) = \sum_{k=0}^{N-1} \left(S(k) e^{j\frac{2\pi nk}{N}} \right)$$

Example 5.3

Use this demonstration to perform DFT analysis of a signal.

This media object is a LabVIEW VI. Please view or download it at

<DFTanalysis.llb>

Example 5.4

Use this demonstration to synthesize a signal from a DFT sequence.

This media object is a LabVIEW VI. Please view or download it at

<DFT_Component_Manipulation.llb>

5.9 DFT: Computational Complexity

We now have a way of computing the spectrum for an arbitrary signal: The Discrete Fourier Transform (DFT) (5.33) computes the spectrum at N equally spaced frequencies from a length-N sequence. An issue that never arises in analog "computation," like that performed by a circuit, is how much work it takes to perform the signal processing operation such as filtering. In computation, this consideration translates to the number of basic computational steps required to perform the needed processing. The number of steps, known as the **complexity**, becomes equivalent to how long the computation takes (how long must we wait for an answer). Complexity is not so much tied to specific computers or programming languages but to how many steps are required on any computer. Thus, a procedure's stated complexity says that the time taken will be **proportional** to some function of the amount of data used in the computation and the amount demanded.

For example, consider the formula for the discrete Fourier transform. For each frequency we choose, we must multiply each signal value by a complex number and add together the results. For a real-valued signal, each real-times-complex multiplication requires two real multiplications, meaning we have $2N$ multiplications to perform. To add the results together, we must keep the real and imaginary parts separate. Adding N numbers requires $N - 1$ additions. Consequently, each frequency requires $2N + 2(N - 1) = 4N - 2$ basic computational steps. As we have N frequencies, the total number of computations is $N(4N - 2)$.

In complexity calculations, we only worry about what happens as the data lengths increase, and take the dominant term here the $4N^2$ term as reflecting how much work is involved in making the computation. As multiplicative constants don't matter since we are making a "proportional to" evaluation, we find the DFT is an $O(N^2)$ computational procedure. This notation is read "order N-squared". Thus, if we double the length of the data, we would expect that the computation time to approximately quadruple.

Exercise 5.8.1

In making the complexity evaluation for the DFT, we assumed the data to be real. Three questions emerge. First of all, the spectra of such signals have conjugate symmetry, meaning that negative frequency components (

$$k = \left\{ \frac{N}{2} + 1, \ldots, N + 1 \right\}$$

in the DFT (5.33)) can be computed from the corresponding positive frequency components. Does this symmetry change the DFT's complexity? Secondly, suppose the data are complex-valued; what is the DFT's complexity now? Finally, a less important but interesting question is suppose we want K frequency values instead of N; now what is the complexity?

5.10 Fast Fourier Transform (FFT)

One wonders if the DFT can be computed faster: Does another computational procedure an **algorithm** exist that can compute the same quantity, but more efficiently. We could seek methods that reduce the constant of proportionality, but do not change the DFT's complexity $O(N^2)$. Here, we have something more dramatic in mind: Can the computations be restructured so that a **smaller** complexity results?

In 1965, IBM researcher Jim Cooley and Princeton faculty member John Tukey developed what is now known as the Fast Fourier Transform (FFT). It is an algorithm for computing that DFT that has order $O(N\log N)$ **for certain length inputs**. Now when the length of data doubles, the spectral computational time will not quadruple as with the DFT algorithm; instead, it approximately doubles. Later research showed that no algorithm for computing the DFT could have a smaller complexity than the FFT. Surprisingly, historical work has shown that Gauss2l in the early nineteenth century developed the same algorithm, but did not publish it! After the FFT's rediscovery, not only was the computation of a signal's spectrum greatly speeded, but also the added feature of **algorithm** meant that computations had flexibility not available to analog implementations.

Exercise 5.9.1

Before developing the FFT, let's try to appreciate the algorithm's impact. Suppose a short-length transform takes 1 ms. We want to calculate a transform of a signal that is 10 times longer. Compare how much longer a straightforward implementation of the DFT would take in comparison to an FFT, both of which compute exactly the same quantity.

To derive the FFT, we assume that the signal's duration is a power of two: $N=2^L$. Consider what happens to the even-numbered and odd-numbered elements of the sequence in the DFT calculation.

$$\begin{aligned} S(k) &= s(0) + s(2)e^{(-j)\frac{2\pi 2k}{N}} + \ldots + s(N-2)e^{(-j)\frac{2\pi(N-2)k}{N}} \\ &\quad + s(1)e^{(-j)\frac{2\pi k}{N}} + s(3)e^{(-j)\frac{2\pi(2+1)k}{N}} + \cdots + s\left(N-1\right)e^{(-j)\frac{2\pi(N-(2-1))k}{N}} \\ &= \left[s(0) + s(2)e^{(-j)\frac{2\pi k}{\frac{N}{2}}} + \ldots + s(N-2)e^{(-j)\frac{2\pi\left(\frac{N}{2}-1\right)k}{\frac{N}{2}}} \right] \\ &\quad + \left[s(1) + s(3)e^{(-j)\frac{2}{\pi}} + \ldots + s(N-1)e^{(-j)\frac{2\pi\left(\frac{N}{2}-1\right)}{\frac{N}{2}}} \right] e\frac{-(j2\pi k)}{N} \end{aligned}$$

(5.39)

Each term in square brackets has the form of a

$$\frac{N}{2} - length$$

DFT. The first one is a DFT of the even-numbered elements, and the second of the odd-numbered elements. The first DFT is combined with the second multiplied by the complex exponential

$$e^{-\left(\frac{j2\pi k}{N}\right)}.$$

The half-length transforms are each evaluated at frequency indices *k* = 0, ..., *N* - 1. Normally, the number of frequency indices in a DFT calculation range between zero and the transform length minus one. The **computational advantage** of the FFT comes from recognizing the periodic nature of the discrete Fourier transform. The FFT simply reuses the computations made in the half-length transforms and combines them through additions and the multiplication by e, which is not periodic over

$$\frac{N}{2}$$

Figure 5.12 (Length-8 DFT decomposition) illustrates this decomposition. As it stands, we now compute

$$two\ length - \frac{N}{2}$$

transforms

$$(complexity\ 2O(\frac{N^2}{4})),$$

multiply one of them by the complex exponential (complexity *O(N)*), and add the results (complexity *O(N)*). At this point, the total complexity is still dominated by the half-length DFT calculations, but the proportionality coefficient has been reduced.

Now for the fun. Because $N=2^L$, each of the half-length transforms can be reduced to two quarter-length transforms, each of these to two eighth-length ones, etc. This decomposition continues until we are left with length-2 transforms. This transform is quite simple, involving only additions. Thus, the first stage of the FFT has

$$\frac{N}{2}$$

length-2 transforms (see the bottom part of Figure 5.12 (Length-8 DFT decomposition)). Pairs of these transforms are combined by adding one to the other multiplied by a complex exponential. Each pair requires 4 additions and 2 multiplications, giving a total number of computations equaling

$$6 \cdot \frac{N}{4} = \frac{3N}{2}$$

This number of computations does not change from stage to stage. Because the number of stages, the number of times the length can be divided by two, equals $\log_2 N$, the number of arithmetic operations equals

$$\frac{3N}{2}\log_2 N$$

which makes the complexity of the FFT $O(N\log_2 N)$.

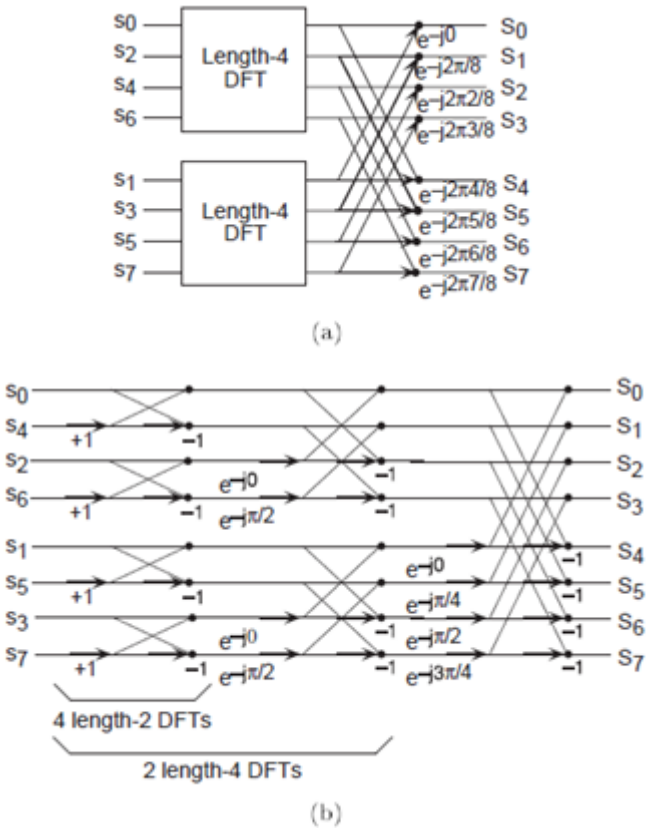

Figure 5.12 Length-8 DFT decomposition

Doing an example will make computational savings more obvious. Let's look at the details of a length-8 DFT. As shown on Figure 5.13 (Butterfy), we first decompose the DFT into two length-4 DFTs, with the outputs added and subtracted together in pairs. Considering Figure 5.13 (Butterfy) as the frequency index goes from 0 through 7, we recycle values from the length-4 DFTs into the fnal calculation because of the periodicity of the DFT output. Examining how pairs of outputs are collected together, we create the basic computational element known as a **butterfy** (Figure 5.13 (Butterfy)).

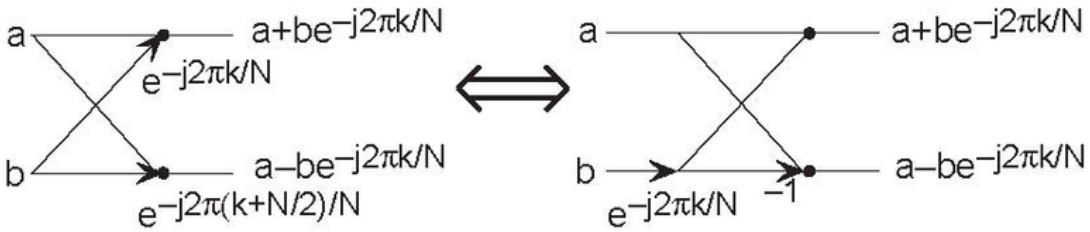

Figure 5.13 Butterfy

The basic computational element of the fast Fourier transform is the butterfly. It takes two complex numbers, represented by a and b, and forms the quantities shown. Each butterfly requires one complex multiplication and two complex additions.

By considering together the computations involving common output frequencies from the two half-length DFTs, we see that the two complex multiplies are related to each other, and we can reduce our computational work even further. By further decomposing the length-4 DFTs into two length-2 DFTs and combining their outputs, we arrive at the diagram summarizing the length-8 fast Fourier transform (Figure 5.12

(Length-8 DFT decomposition)). Although most of the complex multiplies are quite simple (multiplying by

$$e - (j\pi)$$

means swapping real and imaginary parts and changing their signs), let's count those for purposes of evaluating the complexity as full complex multiplies. We have

$$\frac{N}{2} = 4$$

complex multiplies and N =8 complex additions for each stage and $\log_2 N$ = 3 stages, making the number of basic computations

$$\frac{3N}{2} \log_2 N$$

as predicted.

Exercise 5.9.2

Note that the ordering of the input sequence in the two parts of Figure 5.12 (Length-8 DFT decomposition) aren't quite the same. Why not? How is the ordering determined?

Other "fast" algorithms were discovered, all of which make use of how many common factors the transform length ***N*** has. In number theory, the number of prime factors a given integer has measures how **composite** it is. The numbers 16 and 81 are highly composite (equaling 2^4 and 3^4 respectively), the number 18 is less so ($2^1 \cdot 3^2$), and 17 not at all (it's prime). In over thirty years of Fourier transform algorithm development, the original Cooley-Tukey algorithm is far and away the most frequently used. It is so computationally efficient that power-of-two transform lengths are frequently used regardless of what the actual length of the data.

Exercise 5.9.3

Suppose the length of the signal were 500? How would you compute the spectrum of this signal using the Cooley-Tukey algorithm? What would the length N of the transform be?

5.11 Spectrograms

We know how to acquire analog signals for digital processing (pre-filtering), sampling (The Sampling Theorem (Page 201)), and A/D conversion (Analog-to-Digital Conversion (Page 201)) and to compute spectra of discrete-time signals (using the FFT algorithm (Fast Fourier Transform (FFT) (Page 221)), let's put these various components together to learn how the spectrogram shown in Figure 5.14 (Speech Spectrogram), which is used to analyze speech (Modeling the Speech Signal (Page 165)), is calculated. The speech was sampled at a rate of 11.025 kHz and passed through a 16-bit A/D converter.

POINT OF INTEREST: Music compact discs (CDs) encode their signals at a sampling rate of 44.1 kHz. We'll learn the rationale for this number later. The 11.025 kHz sampling rate for the speech is 1/4 of the CD sampling rate, and was the lowest available

sampling rate commensurate with speech signal bandwidths available on my computer.

Exercise 5.10.1

Looking at Figure 5.14 (Speech Spectrogram) the signal lasted a little over 1.2 seconds. How long was the sampled signal (in terms of samples)? What was the datarate during the sampling process in bps (bits per second)? Assuming the computer storage is organized in terms of bytes (8-bit quantities), how many bytes of computer memory does the speech consume?

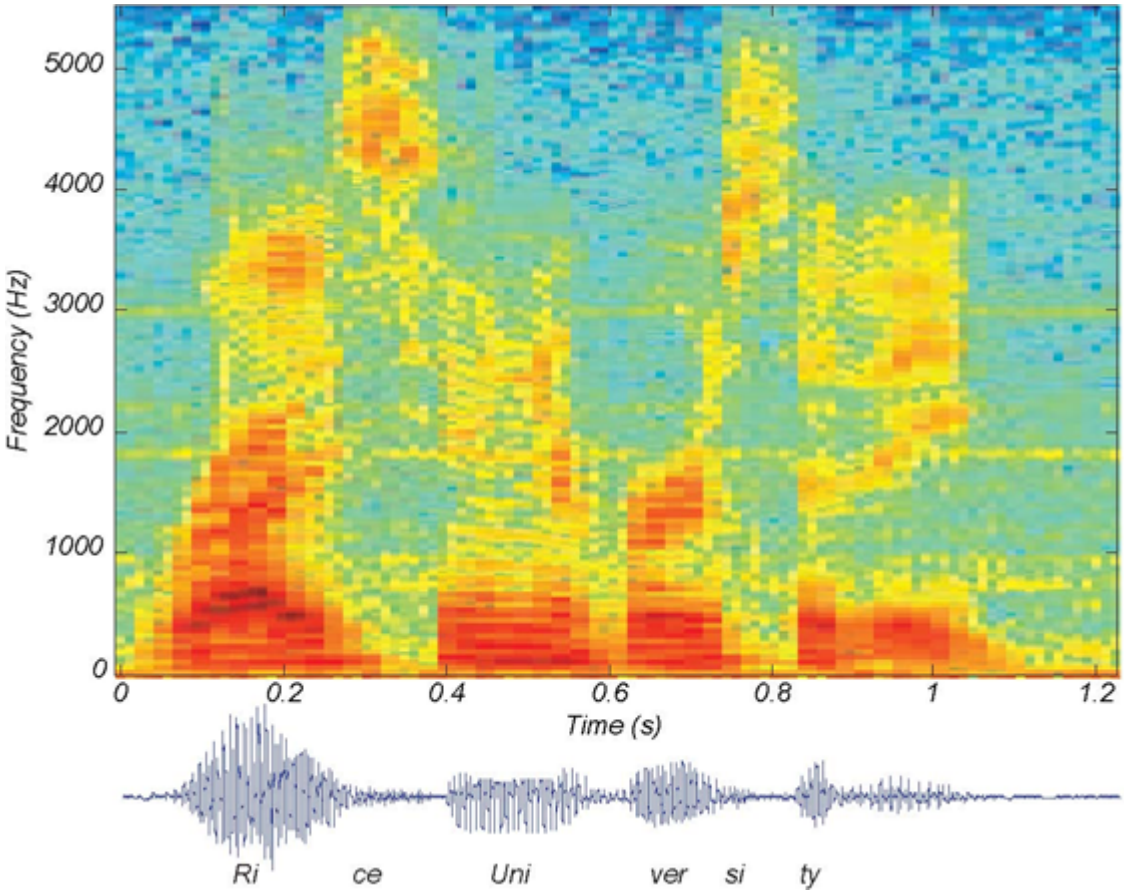

Figure 5.14 Speech Spectrogram

The resulting discrete-time signal, shown in the bottom of Figure 5.14 (Speech Spectrogram), clearly changes its character with time. To display these spectral changes, the long signal was sectioned into **frames**: comparatively short, contiguous groups of samples. Conceptually, a Fourier transform of each frame is calculated using the FFT. Each frame is not so long that signifcant signal variations are retained within a frame, but not so short that we lose the signal's spectral character. Roughly speaking, the speech signal's spectrum is evaluated over successive time segments and stacked side by side so that the x-axis corresponds to time and the y-axis frequency, with color indicating the spectral amplitude.

An important detail emerges when we examine each framed signal (Figure 5.15 (Spectrogram Hanning vs. Rectangular)).

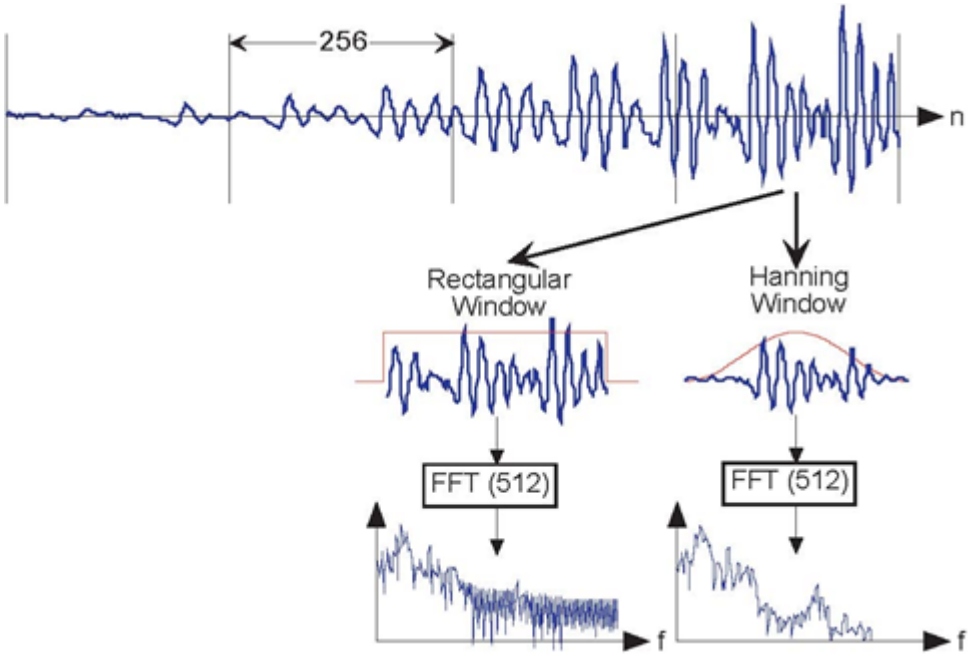

Figure 5.15 Spectrogram Hanning vs. Rectangular

The top waveform is a segment 1024 samples long taken from the beginning of the "Rice University" phrase. Computing Figure 5.14 (Speech Spectrogram) involved creating frames, here demarked by the vertical lines, that were 256 samples long and _nding the spectrum of each. If a rectangular window is applied (corresponding to extracting a frame from the signal), oscillations appear in the spectrum (middle of bottom row). Applying a Hanning window gracefully tapers the signal toward frame edges, thereby yielding a more accurate computation of the signal's spectrum at that moment of time.

At the frame's edges, the signal may change very abruptly, a feature not present in the original signal. A transform of such a segment reveals a curious oscillation in the spectrum, an artifact directly related to this sharp amplitude change. A better way to frame signals for spectrograms is to apply a **window**: Shape the signal values within a frame so that the signal decays gracefully as it nears the edges. This shaping is accomplished by multiplying the framed signal by the sequence w (n). In sectioning the signal, we essentially applied a rectangular **window**: ***w (n)=1, 0 ≤ n ≤ N −1***. A much more graceful window is the **Hanning window**; it has the cosine shape

$$w(n) = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{N}\right)\right).$$

As shown in Figure 5.15 (Spectrogram Hanning vs. Rectangular), this shaping greatly reduces spurious oscillations in each frame's spectrum. Considering the spectrum of the Hanning windowed frame, we find that the oscillations resulting from applying the rectangular window obscured a formant (the one located at a little more than half the Nyquist frequency).

Exercise 5.10.2

What might be the source of these oscillations? To gain some insight, what is the length-*2N* discrete Fourier transform of a length-*N* pulse? The pulse emulates the rectangular window, and certainly has edges. Compare your answer with the length-*2N* transform of a length-*N* Hanning window.

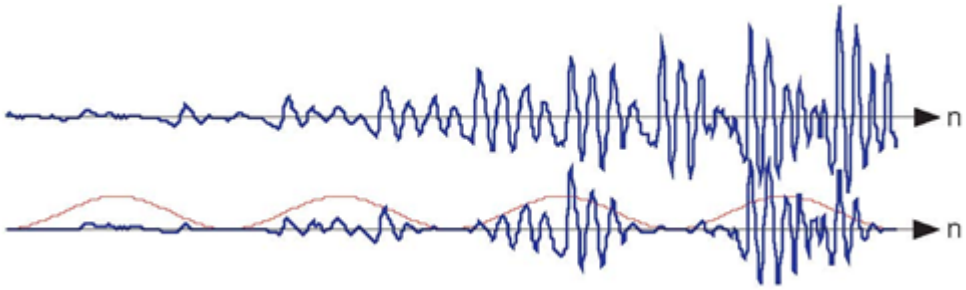

Figure 5.16 Non-overlapping windows

In comparison with the original speech segment shown in the upper plot, the non-overlapped Hanning windowed version shown below it is very ragged. Clearly, spectral information extracted from the bottom plot could well miss important features present in the original.

If you examine the windowed signal sections in sequence to examine windowing's effect on signal amplitude, we see that we have managed to amplitude-modulate the signal with the periodically repeated window (Figure 5.16 (Non-overlapping windows)). To alleviate this problem, frames are overlapped (typically by half a frame duration). This solution requires more Fourier transform calculations than needed by rectangular windowing, but the spectra are much better behaved and spectral changes are much better captured.

The speech signal, such as shown in the speech spectrogram (Figure 5.14: Speech Spectrogram), is sectioned into overlapping, equal-length frames, with a Hanning window applied to each frame. The spectra of each of these is calculated, and displayed in spectrograms with frequency extending vertically, window time location running horizontally, and spectral magnitude color-coded. Figure 5.17 (Overlapping windows for computing spectrograms) illustrates these computations.

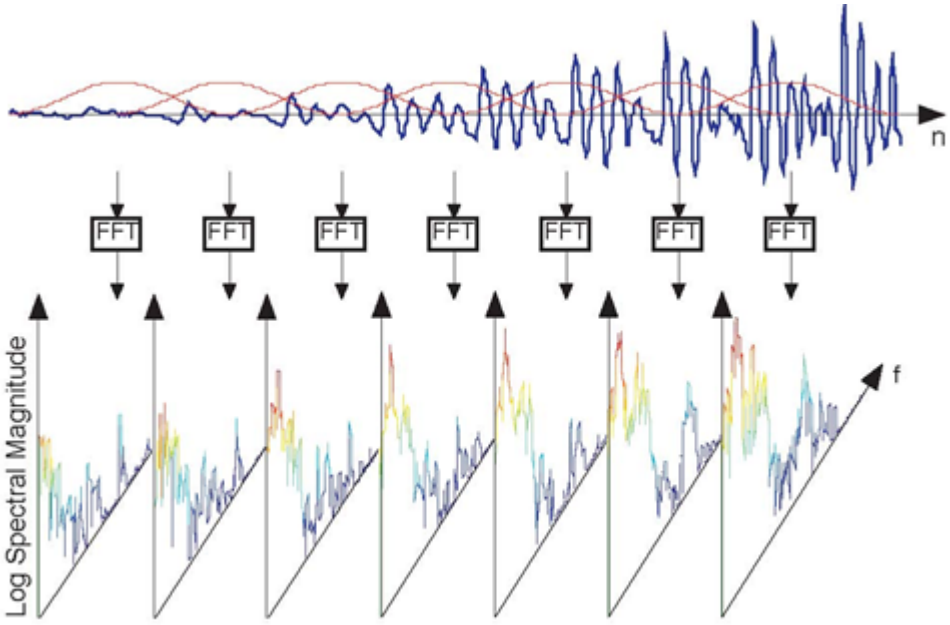

Figure 5.17 Overlapping windows for computing spectrograms

The original speech segment and the sequence of overlapping Hanning windows applied to it are shown in the upper portion. Frames were 256 samples long and a

Hanning window was applied with a half-frame overlap. A length-512 FFT of each frame was computed, with the magnitude of the first 257 FFT values displayed vertically, with spectral amplitude values color-coded.

Exercise 5.10.3

Why the specific values of 256 for *N* and 512 for *K*? Another issue is how was the length-512 transform of each length-256 windowed frame computed?

5.12 Discrete-Time Systems

When we developed analog systems, interconnecting the circuit elements provided a natural starting place for constructing useful devices. In discrete-time signal processing, we are not limited by hardware considerations but by what can be constructed in software.

Exercise 5.11.1

One of the first analog systems we described was the amplifier (Amplifiers (Page 27): Amplifiers). We found that implementing an amplifier was difficult in analog systems, requiring an op-amp at least. What is the discrete-time implementation of an amplifier? Is this especially hard or easy?

In fact, we will discover that frequency-domain implementation of systems, wherein we multiply the input signal's Fourier transform by a frequency response, is not only a viable alternative, but also a computationally efficient one. We begin with discussing the underlying mathematical structure of linear, shift-invariant systems, and devise how software filters can be constructed.

5.13 Discrete-Time Systems in the Time-Domain

A discrete-time signal *s(n)* is delayed by n_0 samples when we write *s*(*n* - n_0), with $n_0 > 0$. Choosing n_0 to be negative advances the signal along the integers. As opposed to analog delays (Delay (Page 28) : Delay), discrete-time delays can **only** be integer valued. In the frequency domain, delaying a signal corresponds to a linear phase shift of the signal's discrete-time Fourier transform :

$$\left(s(n-n_0) \leftrightarrow e^{-(j2\pi f n_0)} S(e^{j2\pi f})\right).$$

Linear discrete-time systems have the superposition property.

$$S(a_2 x_1(n) + a_2 x_2(n)) = a_1 S(x_1(n)) + a_2 S(x_2(n))$$

(5.40)

A discrete-time system is called **shift-invariant** (analogous to time-invariant analog systems (Time-Invariant Systems (Page 30)) if delaying the input delays the corresponding output. If *S (x (n)) = y (n)*, then a shift-invariant system has the property

$$S(x(n-n_0)) = y(n-n_0)$$

(5.41)

We use the term shift-invariant to emphasize that delays can only have integer values in discrete-time, while in analog signals, delays can be arbitrarily valued.

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

$$y(n) = a_1 y(n-1) + \ldots + a_p y(n-p) + b_0 x(n) + b_1 x(n-1) + \ldots + b_q x(n-q)$$

(5.42)

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

ASIDE: There is an asymmetry in the coefficients: where is a_0? This coefficient would multiply the *y(n)* term in (5.42). We have essentially divided the equation by it, which does not change the input-output relationship. We have thus created the convention that a_0 is always one.

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

Difference equations are usually expressed in software with for loops. A MATLAB program that would compute the first 1000 values of the output has the form

```
for n=1:1000
    y(n) = sum(a.*y(n-1:-1:n-p)) + sum(b.*x(n:-1:n-q));
end
```

An important detail emerges when we consider making this program work; in fact, as written it has (at least) two bugs. What input and output values enter into the computation of *y (1)*? We need values for *y (0), y (−1), ...,* values we have not yet computed. To compute them, we would need more previous values of the output, which we have not yet computed. To compute these values, we would need even earlier values, ad infinitum. The way out of this predicament is to specify the system's **initial conditions**: we must provide the p output values that occurred before the

input started. These values can be arbitrary, but the choice does impact how the system responds to a given input. **One** choice gives rise to a linear system: Make the initial conditions zero. The reason lies in the Definition of a linear system (Section 2.6.6: Linear Systems): The only way that the output to a sum of signals can be the sum of the individual outputs occurs when the initial conditions in each case are zero.

Exercise 5.12.1

The initial condition issue resolves making sense of the difference equation for inputs that start at some index. However, the program will not work because of a programming, not conceptual, error. What is it? How can it be "fixed?"

Example 5.5

Let's consider the simple system having p =1 and q =0.

$$y(n) = ay(n-1) + bx(n)$$

(5.43)

To compute the output at some index, this difference equation says we need to know what the previous output $y(n-1)$ and what the input signal is at that moment of time. In more detail, let's compute this system's output to a unit-sample input: $x(n) = \delta(n)$. Because the input is zero for negative indices, we start by trying to compute the output at n = 0.

$$y(0) = ay(-1) + b$$

(5.44)

What is the value of $y(-1)$? Because we have used an input that is zero for all negative indices, it is reasonable to assume that the output is also zero. Certainly, the difference equation would not describe a linear system (Section 2.6.6: Linear Systems) if the input that is zero for **all** time did not produce a zero output. With this assumption, $y(-1) = 0$, leaving y (0) = b. For n> 0, the input unit-sample is zero, which leaves us with the difference equation $y(n) = ay(n-1)$, $n > 0$. We can envision how the filter responds to this input by making a table.

$$y(n) = ay(n-1) + b\delta(n)$$

(5.45)

n	$x\,(n)$	$y\,(n)$
−1	0	0
0	1	b
1	0	ba
2	0	ba^2
:	0	:
n	0	ba^n

Coefficient values determine how the output behaves. The parameter b can be any value, and serves as a gain. The effect of the parameter a is more complicated (Table). If it equals zero, the output simply equals the input times the gain b. For all non-zero values of a, the output lasts forever; such systems are said to be **IIR** (**I**nfinite **I**mpulse **R**esponse). The reason for this terminology is that the unit sample also known as the impulse (especially in analog situations), and the system's response to the "impulse" lasts forever. If a is positive and less than one, the output is a decaying exponential. When a =1, the output is a unit step. If a is negative and greater than −1, the output oscillates while decaying exponentially. When a = −1, the output changes sign forever, alternating between b and $-b$. More dramatic effects when $|a| > 1$; whether positive or negative, the output signal becomes larger and larger, growing exponentially.

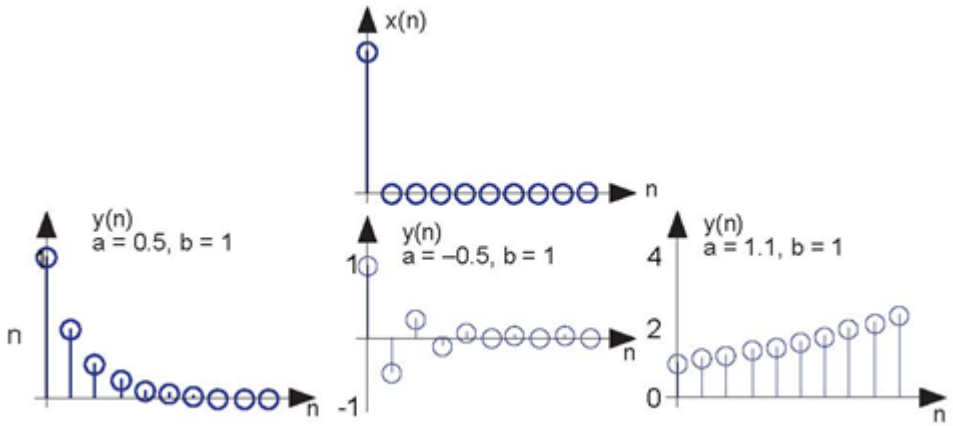

Figure 5.18

The input to the simple example system, a unit sample, is shown at the top, with the outputs for several system parameter values shown below.

Positive values of *a* are used in population models to describe how population size increases over time. Here, *n* might correspond to generation. The diference equation says that the number in the next generation is some multiple of the previous one. If this multiple is less than one, the population becomes extinct; if greater than one, the population fourishes. The same diference equation also describes the effect of compound interest on deposits. Here, n indexes the times at which compounding occurs (daily, monthly, etc.), *a* equals the compound interest rate plus one, and *b* =1 (the bank provides no gain). In signal processing applications, we typically require that the output remain bounded for any input. For our example, that means that we restrict $|a| < 1$ and choose values for it and the gain according to the application.

Exercise 5.12.2

Note that the difference equation (5.42),

$$y(n) = a_1 y(n-1) + \ldots + a_p y(n-p) + b_0 x(n) + b_1 x(n-1) + \ldots + b_q x(n-q)$$

does not involve terms like *y (n + 1)* or *x (n + 1)* on the equation's right side. Can such terms also be included? Why or why not?

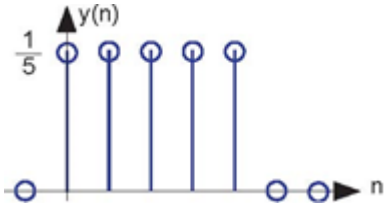

Figure 5.19 The plot shows the unit-sample response of a length-5 boxcar filter.

Example 5.6

A somewhat different system has no "*a*" coefficients. Consider the difference equation

$$y(n) = \frac{1}{q}(x(n) + \ldots + x(n-q+1))$$

(5.46)

Because this system's output depends only on current and previous input values, we need not be concerned with initial conditions. When the input is a unit-sample, the output equals

$$\frac{1}{q}$$

for $n = \{0,\ldots,q-1\}$, then equals zero thereafter. Such systems are said to be **FIR** (**F**inite **I**mpulse **R**esponse) because their unit sample responses have finite duration. Plotting this response (Figure 5.19) shows that the unit-sample response is a pulse of width q and height

$$\frac{1}{q}.$$

This waveform is also known as a boxcar, hence the name **boxcar filter** given to this system. We'll derive its frequency response and develop its filtering interpretation in the next section. For now, note that the difference equation says that each output value equals the **average** of the input's current and previous values. Thus, the output

equals the running average of input's previous q values. Such a system could be used to produce the average weekly temperature (q =7) that could be updated daily.

[MEDIA OBJECT] (This media object is a LabVIEW VI. Please view or download it at <DiscreteTimeSys.llb>)

5.14 Discrete-Time Systems in the Frequency Domain

As with analog linear systems, we need to find the frequency response of discrete-time systems. We used impedances to derive directly from the circuit's structure the frequency response. The only structure we have so far for a discrete-time system is the difference equation. We proceed as when we used impedances: let the input be a complex exponential signal. When we have a linear, shift-invariant system, the output should also be a complex exponential of the same frequency, changed in amplitude and phase. These amplitude and phase changes comprise the frequency response we seek. The complex exponential input signal is $x(n) = Xe^{j2\pi fn}$. Note that this input occurs for **all** values of n. No need to worry about initial conditions here. Assume the output has a similar form: $y(n) = Ye^{j2\pi fn}$. Plugging these signals into the fundamental difference equation (5.42), we have

$$Ye^{j2\pi fn} = a_1 Ye^{j2\pi f(n-1)} + \ldots + a_p Ye^{j2\pi f(n-p)} + b_0 Xe^{j2\pi fn} + b_1 Xe^{j2\pi f(n-1)} + \ldots + b_q Xe^{j2\pi f(n-q)}$$

(5.47)

The assumed output does indeed satisfy the diference equation if the output complex amplitude is related to the input amplitude by

$$Y = \frac{b_0 + b_1 e^{-(j2\pi xf)} + \ldots + b_q e^{-(j2\pi qf)}}{1 - a_1 e^{-(j2\pi f)} - \ldots - a_p e^{-(j2\pi pf)}} X$$

This relationship corresponds to the system's frequency response or, by another name, its transfer function. We find that any discrete-time system defined by a diference equation has a transfer function given by

$$H(e^{j2\pi f}) = \frac{b_0 + b_1 e^{-(j2\pi f)} + \ldots + b_q e^{-(j2\pi qf)}}{1 - a_1 e^{-(j2\pi f)} - \ldots - a_p e^{-(j2\pi pf)}}$$

(5.48)

Furthermore, because **any** discrete-time signal can be expressed as a superposition of complex exponential signals and because linear discrete-time systems obey the Superposition Principle, the transfer function relates the discrete-time Fourier transform of the system's output to the input's Fourier transform.

$$Y(e^{j2\pi f}) = X(e^{j2\pi f})H(e^{j2\pi f})$$

(5.49)

Example 5.7

The frequency response of the simple IIR system (difference equation given in a previous example (Discrete-Time Systems in the Time-Domain (Page 228))) is given by

$$H(e^{j2\pi f}) = \frac{b}{1 - ac^{-(j2\pi f)}}$$

(5.50)

This Fourier transform occurred in a previous example; the exponential signal spectrum (Figure 5.10: Spectra of exponential signals) portrays the magnitude and phase of this transfer function. When the filter coefficient *a* is positive, we have a lowpass filter; negative *a* results in a highpass filter. The larger the coefficient in magnitude, the more pronounced the lowpass or highpass filtering.

Example 5.8

The length-*q* boxcar filter (difference equation found in a previous example (Discrete-Time Systems in the Time-Domain (Page 228))) has the frequency response

$$H(e^{j2\pi f}) = \frac{1}{q}\sum_{m=0}^{q-1}\left(e^{-(j2\pi fm)}\right)$$

(5.51)

This expression amounts to the Fourier transform of the boxcar signal (5.13). There we found that this frequency response has a magnitude equal to the absolute value of **dsinc(π*f*)**; see the length-10 filter's frequency response (Figure 5.11: Spectrum of length-ten pulse). We see that boxcar filters length-***q*** signal averages have a lowpass behavior, having a cutoff frequency of

$$\frac{1}{q}.$$

.

Exercise 5.13.1

Suppose we multiply the boxcar filter's coefficients by a sinusoid:

$$b_m = \frac{1}{q}\cos\left(2\pi f_0 m\right)$$

Use Fourier transform properties to determine the transfer function. How would you characterize this system: Does it act like a filter? If so, what kind of filter and how do you control its characteristics with the filter's coefficients?

These examples illustrate the point that systems described (and implemented) by difference equations serve as filters for discrete-time signals. The filter's **order** is given by the number *p* of denominator coefficients in the transfer function (if the system is IIR) or by the number q of numerator coefficients if the filter is FIR. When a system's transfer function has both terms, the system is usually IIR, and its order equals *p* regardless of *q*. By selecting the coefficients and filter type, filters having virtually any frequency response desired can be designed. This design flexibility can't be found in analog systems. In the next section, we detail how analog signals can be filtered by computers, offering a much greater range of filtering possibilities than is possible with circuits.

5.14.1 Filtering in the Frequency Domain

Because we are interested in actual computations rather than analytic calculations, we must consider the details of the discrete Fourier transform. To compute the length-*N* DFT, we assume that the signal has a duration less than or equal to *N*. Because frequency responses have an explicit frequency-domain specification (5.47) in terms of filter coefficients, we don't have a direct handle on which signal has a Fourier transform equaling a given frequency response. Finding this signal is quite easy. First of all, note that the discrete-time Fourier transform of a unit sample equals one for all frequencies. Since the input and output of linear, shift-invariant systems are related to each other by

$$Y(e^{j2\pi f}) = H(e^{j2\pi f})x(e^{j2\pi f}),$$

a unit-sample input, which has $X(e^{j2\pi f}) = 1$, results in the output's Fourier transform equaling the system's transfer function.

Exercise 5.14.1

This statement is a very important result. Derive it yourself. In the time-domain, the output for a unit-sample input is known as the system's **unit-sample response**, and is denoted by *h (n)*. Combining the frequency-domain and time-domain interpretations of a linear, shift-invariant system's unit-sample response, we have that *h (n)* and the transfer function are Fourier transform pairs **in terms of the discrete-time Fourier transform.**

$$\left(h(n) \leftrightarrow H(2^{j2\pi f})\right)$$

(5.52)

Returning to the issue of how to use the DFT to perform filtering, we can analytically specify the frequency response, and derive the corresponding length-*N* DFT by sampling the frequency response.

$$H(k) = H(e^{\frac{j2\pi k}{N}}) \ , \ k = \{0, ..., N-1\}$$

(5.53)

Computing the inverse DFT yields a length-*N* signal **no matter what the actual duration of the unit-sample response might be**. If the unit-sample response has a duration less than or equal to N (it's a FIR filter), computing the inverse DFT of the sampled frequency response indeed yields the unit-sample response. If, however, the duration exceeds *N*, errors are encountered. The nature of these errors is easily explained by appealing to the Sampling Theorem. By sampling in the frequency domain, we have the potential for aliasing in the time domain (sampling in one domain, be it time or frequency, can result in aliasing in the other) unless we sample fast enough. Here, the duration of the unit-sample response determines the minimal sampling rate that prevents aliasing. For FIR systems they by Definition have finite-duration unit sample responses the number of required DFT samples equals the unit-sample response's duration: N ≥ q.

Exercise 5.14.2

Derive the minimal DFT length for a length-q unit-sample response using the Sampling Theorem. Because sampling in the frequency domain causes repetitions of the unit-sample response in the time domain, sketch the time-domain result for various choices of the DFT length N.

Exercise 5.14.3

Express the unit-sample response of a FIR filter in terms of difference equation coefficients. Note that the corresponding question for IIR filters is far more difficult to answer: Consider the example (Discrete-Time Systems in the Time-Domain (Page 228)).

For IIR systems, we cannot use the DFT to find the system's unit-sample response: aliasing of the unit-sample response will always occur. Consequently, we can only implement an IIR filter accurately in the time domain with the system's difference equation. **Frequency-domain implementations are restricted to FIR filters.**

Another issue arises in frequency-domain filtering that is related to time-domain aliasing, this time when we consider the output. Assume we have an input signal having duration N_x that we pass through a FIR filter having a length-q +1 unit-sample response. What is the duration of the output signal? The difference equation for this filter is

$$y(n) = b_0 x(n) + \ldots + b_q x(n-q)$$

(5.54)

This equation says that the output depends on current and past input values, with the input value q samples previous defining the extent of the filter's **memory** of past input values. For example, the output at index $\boldsymbol{N_x}$ depends on $\boldsymbol{x(N_x)}$ (which equals zero), $\boldsymbol{x(N_x - 1)}$, through $\boldsymbol{x(N_x - q)}$. Thus, the output returns to zero only after the last input value passes through the filter's memory. As the input signal's last value occurs at index $\boldsymbol{N_x - 1}$, the last nonzero output value occurs when $\boldsymbol{n - q = N_x - 1}$ or $\boldsymbol{n = q + N_x - 1}$. Thus, the output signal's duration equals $\boldsymbol{q + N_x}$.

Exercise 5.14.4

In words, we express this result as "The output's duration equals the input's duration plus the filter's duration minus one.". Demonstrate the accuracy of this statement.

The main theme of this result is that a filter's output extends longer than either its input or its unit-sample response. Thus, to avoid aliasing when we use DFTs, the dominant factor is not the duration of input or of the unit-sample response, but of the output. Thus, the number of values at which we must evaluate the frequency response's DFT must be at least $\boldsymbol{q + N_x}$ and we must compute the same length DFT of the input. To accommodate a shorter signal than DFT length, we simply zero-pad the input: Ensure that for indices extending beyond the signal's duration that the signal is zero. Frequency-domain filtering, diagrammed in Figure 5.20, is accomplished by storing the filter's frequency response as the DFT H (k), computing the input's DFT $X(k)$, multiplying them to create the output's DFT $Y(k) = H(k)X(k)$, and computing the inverse DFT of the result to yield $y(n)$.

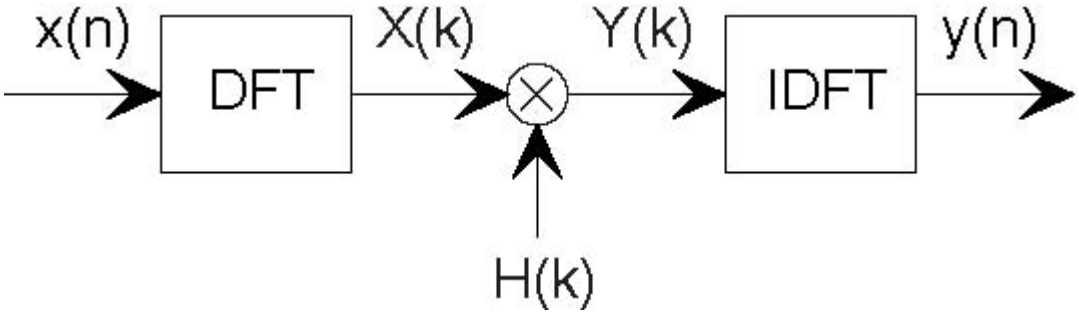

Figure 5.20

To filter a signal in the frequency domain, first compute the DFT of the input, multiply the result by the sampled frequency response, and fnally compute the inverse DFT of the product. The DFT's length must be at least the sum of the input's and unit-sample response's duration minus one. We calculate these discrete Fourier transforms using the fast Fourier transform algorithm, of course.

Before detailing this procedure, let's clarify why so many new issues arose in trying to develop a frequency-domain implementation of linear filtering. The frequency-domain relationship between a filter's input and output is **always** true:

$$Y(e^{j2\pi f}) = H(e^{j2\pi f})X(e^{j2\pi f})$$

The Fourier transforms in this result are discrete time Fourier transforms; for example,

$$X(e^{j2\pi f}) = \sum_{n} \left(x(n)e^{-(j2\pi fn)}\right) .$$

Unfortunately, using this relationship to perform filtering is restricted to the situation when we have analytic formulas for the frequency response and the input signal. The reason why we had to "invent" the discrete Fourier transform (DFT) has the same origin: The spectrum resulting from the discrete-time Fourier transform depends on the **continuous** frequency variable *f*. That's fine for analytic calculation, but computationally we would have to make an uncountably infinite number of computations.

> **Note:** Did you know that two kinds of infinities can be meaningfully defined? A **countablyinfinite** quantity means that it can be associated with a limiting process associated with integers. An **uncountablyinfinite** quantity cannot be so associated. The number of rational numbers is countably infinite (the numerator and denominator correspond to locating the rational by row and column; the total number so-located can be counted, voila!); the number of irrational numbers is uncountably infinite. Guess which is "bigger?"

The DFT computes the Fourier transform at a finite set of frequencies samples the true spectrum which can lead to aliasing in the time-domain unless we sample sufficiently fast. The sampling interval here is

$$\frac{1}{K}$$

for a length-*K* DFT: faster sampling to avoid aliasing thus requires a longer transform calculation. Since the longest signal among the input, unit-sample response and output is the output, it is that signal's duration that determines the transform length. We simply extend the other two signals with zeros (zero-pad) to compute their DFTs.

Example 5.9

Suppose we want to average daily stock prices taken over last year to yield a running weekly average (average over five trading sessions). The filter we want is a length-5 averager (as shown in the unit-sample response (Figure 5.19), and the input's duration is 253 (365 calendar days minus weekend days and holidays). The output duration will be 253 + 5 - 1 = 257, and this determines the transform length we need to use. Because we want to use the FFT, we are restricted to power-of-two transform lengths. We need to choose any FFT length that exceeds the required DFT length. As it turns out, 256 is a power of two ($2^8 = 256$), and this length just undershoots our required length. To use frequency domain techniques, we must use length-512 fast Fourier transforms.

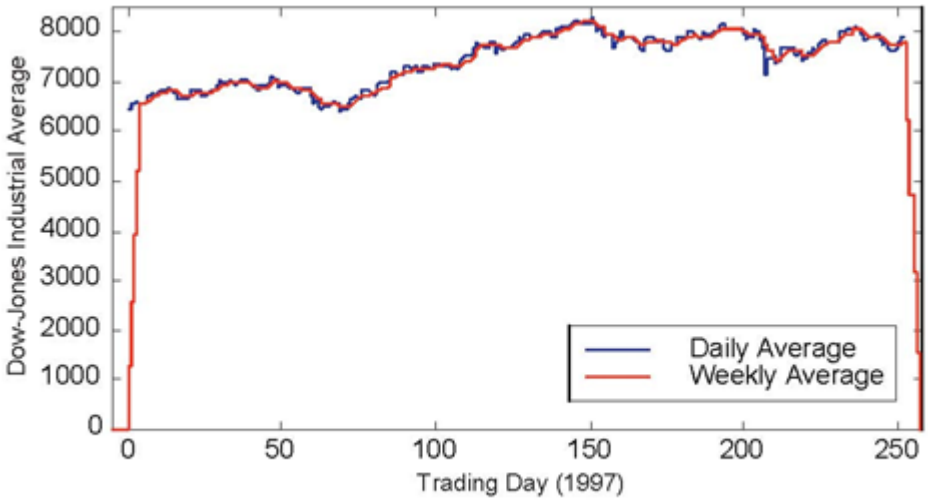

Figure 5.21 .

The blue line shows the Dow Jones Industrial Average from 1997, and the red one the length-5 boxcar-filtered result that provides a running weekly of this market index. Note the "edge" effects in the filtered output

Figure 5.21 shows the input and the filtered output. The MATLAB programs that compute the filtered output in the time and frequency domains are

```
Time Domain
h = [1 1 1 1 1]/5;
y = filter(h,1,[djia zeros(1,4)]);

Frequency Domain
h = [1 1 1 1 1]/5;
DJIA = fft(djia, 512);
H = fft(h, 512);
Y = H.*X;
y = ifft(Y);
```

Note: The filter program has the feature that the length of its output equals the length of its input. To force it to produce a signal having the proper length, the program zero-pads the input appropriately.

MATLAB's `fft` function automatically zero-pads its input if the specified transform length (its second argument) exceeds the signal's length. The frequency domain result will have a small imaginary component largest value is 2.2×10^{-11} because of the inherent finite precision nature of computer arithmetic. Because of the unfortunate misfit between signal lengths and favored FFT lengths, the number of arithmetic operations in the time-domain implementation is far less than those required by the frequency domain version: 514 versus 62,271. If the input signal had been one sample shorter, the frequency-domain computations would have been more than a factor of two less (28,696), but far more than in the time-domain implementation.

An interesting signal processing aspect of this example is demonstrated at the beginning and end of the output. The ramping up and down that occurs can be traced to assuming the input is zero before it begins and after it ends. The filter "sees" these initial and final values as the difference equation passes over the input. These artifacts can be handled in two ways: we can just ignore the edge effects or the data from previous and succeeding years' last and first week, respectively, can be placed at the ends.

5.15 Efficiency of Frequency-Domain Filtering

To determine for what signal and filter durations a time-or frequency-domain implementation would be the most efficient, we need only count the computations required by each. For the time-domain, difference equation approach, we need N_x (2 (q) + 1). The frequency-domain approach requires three Fourier transforms, each requiring

$$\left(\frac{K}{2}\right)(\log_2 K)$$

computations for a length-K FFT, and the multiplication of two spectra ($6K$ computations). The output-signal-duration-determined length must be at least $N_x + q$. Thus, we must compare

$$\left(N_x\left(2q+1\right) \leftrightarrow 6\left(N_x+q\right)+\frac{3}{2}\left(N_x+q\right) log_2\left(N_x+q\right)\right)$$

Exact analytic evaluation of this comparison is quite difficult (we have a transcendental equation to solve). Insight into this comparison is best obtained by dividing by N_x.

$$\left(2q+1 \leftrightarrow 6\left(1+\frac{q}{N_x}\right)+\frac{3}{2}\left(1+\frac{q}{N_x}\right) log_2\left(N_x+q\right)\right)$$

With this manipulation, we are evaluating the number of computations per sample. For any given value of the filter's order q, the right side, the number of frequency-domain computations, will exceed the left if the signal's duration is long enough. However, for filter durations greater than about 10, as long as the input is at least 10 samples, the frequency-domain approach is faster **so long as the FFT's power-of-two constraint is advantageous.**

The frequency-domain approach is not yet viable; what will we do when the input signal is infinitely long? The difference equation scenario fits perfectly with the envisioned digital filtering structure (Figure 5.24), but so far we have required the input to have limited duration (so that we could calculate its Fourier transform). The solution to this problem is quite simple: Section the input into frames, filter each, and add the results together. To section a signal means expressing it as a linear combination of length-N_x non-overlapping "chunks." Because the filter is linear, filtering a sum of terms is equivalent to summing the results of filtering each term.

$$x(n) = \sum_{m=-\infty}^{\infty} (x(n - mN_x)) \Rightarrow y(n) = \sum_{m=-\infty}^{\infty} (y(n - mN_x))$$

(5.55)

As illustrated in Figure 5.22, note that each filtered section has a duration longer than the input. Conse quently, we must literally add the filtered sections together, not just butt them together.

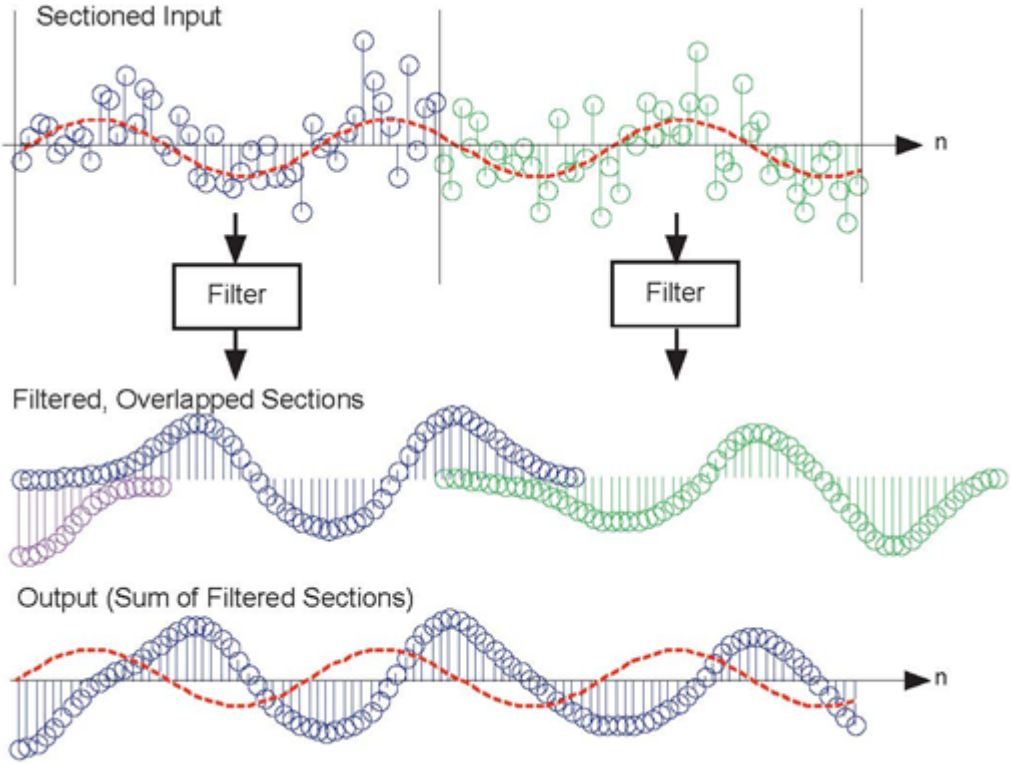

Figure 5.22

The noisy input signal is sectioned into length-48 frames, each of which is filtered using frequency-domain techniques. Each filtered section is added to other outputs that overlap to create the signal equivalent to having filtered the entire input. The sinusoidal component of the signal is shown as the red dashed line.

Computational considerations reveal a substantial advantage for a frequency-domain implementation over a time-domain one. The number of computations for a time-domain implementation essentially remains constant whether we section the input or not. Thus, the number of computations for each output is 2(q)+1. In the frequency-domain approach, computation counting changes because we need only compute the filter's frequency response H (k) once, which amounts to a fixed overhead. We need

only compute two DFTs and multiply them to filter a section. Letting N_x denote a section's length, the number of computations for a section amounts to

$$(N_x + q)\log_2(N_x + q) + 6(N_x + q).$$

In addition, we must add the filtered outputs together; the number of terms to add corresponds to the excess duration of the output compared with the input (q). The frequency-domain approach thus requires

$$\left(1 + \frac{q}{N_x}\right)\log_2(N_x + q) + 7\frac{q}{N_x} + 6$$

computations per output value. For even modest filter orders, the frequency-domain approach is much faster.

Exercise 5.15.1

Show that as the section length increases, the frequency domain approach becomes increasingly more efficient. Note that the choice of section duration is arbitrary. Once the filter is chosen, we should section so that the required FFT length is precisely a power of two: Choose N_x so that $N_x + q = 2^L$.

Implementing the digital filter shown in the A/D block diagram (Figure 5.24) with a frequency-domain implementation requires some additional signal management not required by time-domain implementations. Conceptually, a real-time, time-domain filter could accept each sample as it becomes available, calculate the difference equation, and produce the output value, all in less than the sampling interval T_s. Frequency-domain approaches don't operate on a sample-by-sample basis; instead, they operate on sections. They filter in real time by producing N_x outputs for the same number of inputs faster than N_xT_s. Because they generally take longer to produce an output section than the sampling interval duration, we must filter one section while accepting into memory the **next** section to be filtered. In programming, the operation of building up sections while computing on previous ones is known as **buffering**. Buffering can also be used in time-domain filters as well but isn't required.

Example 5.10

We want to lowpass filter a signal that contains a sinusoid and a significant amount of noise. The example shown in Figure 5.22 shows a portion of the noisy signal's waveform. If it weren't for the overlaid sinusoid, discerning the sine wave in the signal is virtually impossible. One of the primary applications of linear filters is **noise removal**: preserve the signal by matching filter's passband with the signal's spectrum and greatly reduce all other frequency components that may be present in the noisy signal.

A smart Rice engineer has selected a FIR filter having a unit-sample response corresponding a

period-17 sinusoid:

$$h(n) = \frac{1}{17}\left(1 - \cos\left(\frac{2\pi n}{17}\right)\right), \quad n = \{0, \ldots, 16\},$$

which makes $q = 16$. Its frequency response (determined by computing the discrete Fourier transform) is shown in Figure 5.23. To apply, we can select the length of each section so that the frequency-domain filtering approach is maximally efficient: Choose

the section length N_x so that $N_x + q$ is a power of two. To use a length-64 FFT, each section must be 48 samples long. Filtering with the difference equation would require 33 computations per output while the frequency domain requires a little over 16; this frequency-domain implementation is over twice as fast! Figure 5.22 shows how frequency-domain filtering works.

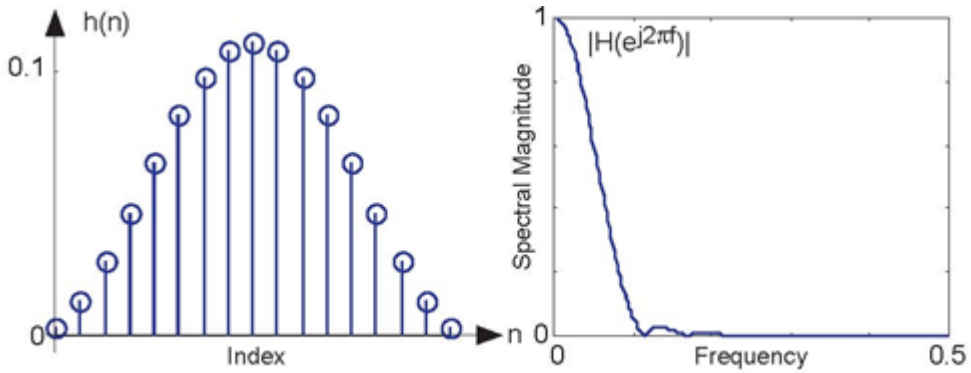

Figure 5.23

The figure shows the unit-sample response of a length-17 Hanning filter on the left and the frequency response on the right. This filter functions as a lowpass filter having a cutoff frequency of about 0.1

We note that the noise has been dramatically reduced, with a sinusoid now clearly visible in the filtered output. Some residual noise remains because noise components within the filter's passband appear in the output as well as the signal.

Exercise 5.15.2

Note that when compared to the input signal's sinusoidal component, the output's sinusoidal component seems to be delayed. What is the source of this delay? Can it be removed?

5.16 Discrete-Time Filtering of Analog Signals

Because of the Sampling Theorem (The Sampling Theorem (Page 202)), we can process, in particular filter, analog signals "with a computer" by constructing the system shown in Figure 5.24. To use this system, we are assuming that the input signal has a lowpass spectrum and can be bandlimited without afecting important signal aspects. Bandpass signals can also be filtered digitally, but require a more complicated system. Highpass signals cannot be filtered digitally. Note that the input and output filters must be analog filters; trying to operate without them can lead to potentially very inaccurate digitization.

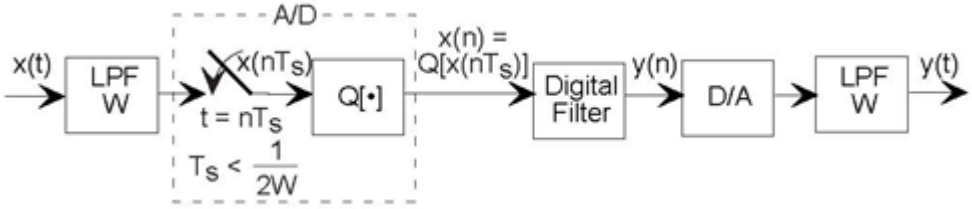

Figure 5.24 A system

To process an analog signal digitally, the signal x (t) must be filtered with an anti-aliasing filter (to ensure a bandlimited signal) before A/D conversion. This lowpass filter (LPF) has a cutoff frequency of **W** Hz, which determines allowable sampling intervals T_S. The greater the number of bits in the amplitude quantization portion Q [·] of the A/D converter, the greater the accuracy of the entire system. The resulting digital signal ***x (n)*** can now be filtered in the time-domain with a difference equation or in the frequency domain with Fourier transforms. The resulting output y (n) then drives a D/A converter and a second anti-aliasing filter (having the same bandwidth as the first one).

Another implicit assumption is that the digital filter can operate in **real time**: The computer and the filtering algorithm must be sufficiently fast so that outputs are computed faster than input values arrive. The sampling interval, which is determined by the analog signal's bandwidth, thus determines how long our program has to compute **each** output *y (n)*. The computational complexity for calculating each output with a difference equation (5.42) is *O (p + q)*. Frequency domain implementation of the filter is also possible. The idea begins by computing the Fourier transform of a length-N portion of the input *x* (*n*), multiplying it by the filter's transfer function, and computing the inverse transform of the result. This approach seems overly complex and potentially inefficient. Detailing the complexity, however, we have *O*(*N*log*N*) for the two transforms (computed using the FFT algorithm) and O (N) for the multiplication by the transfer function, which makes the total complexity *O*(*N*log*N*) for N input values**.** A frequency domain implementation thus requires *O*(*N*log*N*) computational complexity for each output value. The complexities of time-domain and frequency-domain implementations depend on different aspects of the filtering: The time-domain implementation depends on the combined orders of the filter while the frequency-domain implementation depends on the logarithm of the Fourier transform's length.

It could well be that in some problems the time-domain version is more efficient (more easily satisfies the real time requirement), while in others the frequency domain approach is faster. In the latter situations, it is the FFT algorithm for computing the Fourier transforms that enables the superiority of frequency-domain implementations. Because complexity considerations only express how algorithm running-time increases with system parameter choices, we need to detail both implementations to determine which will be more suitable for any given filtering problem. Filtering with a difference equation is straightforward, and the number of computations that must be made for each output value is 2 (*p*+*q*).

Exercise 5.16.1

Derive this value for the number of computations for the general difference equation (5.42).

5.17 Digital Signal Processing Problems

Problem 5.1: Sampling and Filtering

The signal *s (t)* is bandlimited to 4 kHz. We want to sample it, but it has been subjected to various signal processing manipulations.

1. What sampling frequency (if any works) can be used to sample the result of passing *s (t)* through an RC highpass filter with $R = 10k\Omega$ and $C = 8nF$?
2. What sampling frequency (if any works) can be used to sample the **derivative** of *s (t)*?
3. The signal *s (t)* has been modulated by an 8 kHz sinusoid having an unknown phase: the resulting signal is $s(t)\sin(2\pi f_0 t + \phi)$, with f_0 = 8kHz and ϕ =? Can the modulated signal be sampled so that the **original** signal can be recovered from the modulated signal regardless of the phase value φ? If so, show how and find the smallest sampling rate that can be used; if not, show why not.

Problem 5.2: Non-Standard Sampling

Using the properties of the Fourier series can ease finding a signal's spectrum.

1. Suppose a signal *s (t)* is periodic with period T . If c_k represents the signal's Fourier series coefficients, what are the Fourier series coefficients of
$$s\left(t-\frac{T}{2}\right)?$$
2. Find the Fourier series of the signal *p (t)* shown in Figure 5.25 (Pulse Signal).
3. Suppose this signal is used to sample a signal bandlimited to
$$\frac{1}{T}Hz.$$
Find an expression for and sketch the spectrum of the sampled signal.
4. Does aliasing occur? If so, can a change in sampling rate prevent aliasing; if not, show how the signal can be recovered from these samples.

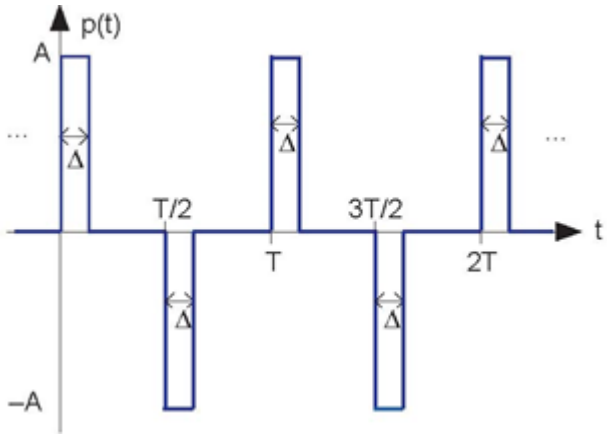

Figure 5.25 Pulse Signal

Problem 5.3: A Different Sampling Scheme

A signal processing engineer from Texas A&M claims to have developed an improved sampling scheme. He multiplies the bandlimited signal by the depicted periodic pulse signal to perform sampling (Figure 5.26).

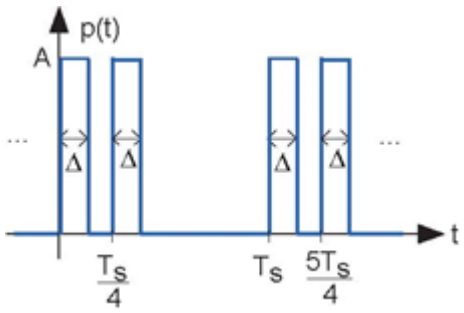

Figure 5.26

1. Find the Fourier spectrum of this signal.
2. Will this scheme work? If so, how should T_S be related to the signal's bandwidth? If not, why not?

Problem 5.4: Bandpass Sampling

The signal $s(t)$ has the indicated spectrum.

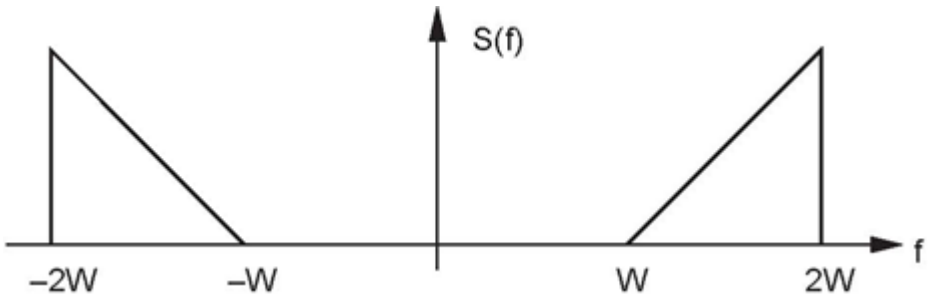

Figure 5.27

1. What is the minimum sampling rate for this signal suggested by the Sampling Theorem?
2. Because of the particular structure of this spectrum, one wonders whether a lower sampling rate could be used. Show that this is indeed the case, and find the system that reconstructs $s(t)$ from its samples.

Problem 5.5: Sampling Signals

If a signal is bandlimited to W Hz, we can sample it at any rate

$$\frac{1}{T_s} > 2W$$

and recover the waveform exactly. This statement of the Sampling Theorem can be taken to mean that all information about the original signal can be extracted from the samples. While true in principle, you do have to be careful how you do so. In addition to the rms value of a signal, an important aspect of a signal is its peak value, which equals $max\{|s(t)|\}$.

1. Let $s(t)$ be a sinusoid having frequency W Hz. If we sample it at precisely the Nyquist rate, how accurately do the samples convey the sinusoid's amplitude? In other words, find the worst case example.
2. How fast would you need to sample for the amplitude estimate to be within 5% of the true value?

   Another issue in sampling is the inherent amplitude quantization produced by A/D converters. Assume the maximum voltage allowed by the converter is V_{max} volts and that it quantizes amplitudes to b bits. We can express the quantized sample
   $$Q(s(nT_s))$$
3. as
   $$s(nT_s) + \epsilon(t),$$
   where
   $$\epsilon(t)$$
   represents the quantization error at the n^{th} sample. Assuming the converter rounds, how large is maximum quantization error?
4. We can describe the quantization error as noise, with a power proportional to the square of the maximum error. What is the signal-to-noise ratio of the quantization error for a full-range sinusoid? Express your result in decibels.

Problem 5.6: Hardware Error

An A/D converter has a curious hardware problem: Every other sampling pulse is half its normal amplitude (Figure 5.28).

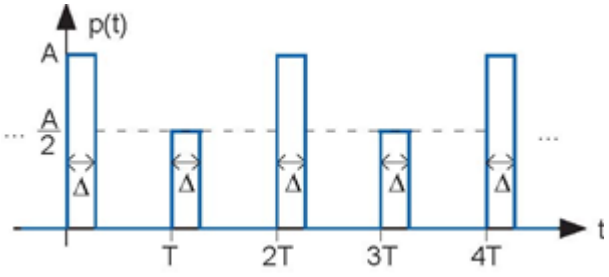

Figure 5.28

1. Find the Fourier series for this signal.
2. Can this signal be used to sample a bandlimited signal having highest frequency
   $$W = \frac{1}{2T}?$$

Problem 5.7: Simple D/A Converter

Commercial digital-to-analog converters don't work this way, but a simple circuit illustrates how they work. Let's assume we have a B-bit converter. Thus, we want to convert numbers having a *B*-bit representation into a voltage proportional to that