DSPA

CONNEXIONS
CNX.ORG

DSPA

Collection edited by: Janko Calic

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

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

DSPA

Table of Contents

Preface for Digital Signal Processing: A User's Guide

Digital signal processing (DSP) has matured in the past few decades from an obscure research discipline to a large body of practical methods with very broad application. Both practicing engineers and students specializing in signal processing need a clear exposition of the ideas and methods comprising the core signal processing "toolkit" so widely used today.

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

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

The text assumes a familiarity with basic signal processing concepts such as ideal sampling theory, continuous and discrete Fourier transforms, convolution and filtering. It evolved from a set of notes for a second signal processing course, ECE 451: Digital Signal Processing II, in Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, aimed at second-semester seniors or first-semester graduate students in signal processing. Over the years, it has been enhanced substantially to include descriptions of common applications, sometimes hard-won knowledge about what actually works and what doesn't, useful tricks, important extensions known to experienced engineers but rarely discussed in academic texts, and other relevant "know-how" to aid the real-world user. This is necessarily an ongoing process, and I continue to expand

and refine this component as my own practical knowledge and experience grows. The topics are the core signal processing methods that are used in the majority of signal processing applications; discrete Fourier analysis and FFTs, digital filter design, adaptive filtering, multirate signal processing, and efficient algorithm implementation and finite-precision issues. While many of these topics are covered at an introductory level in a first course, this text aspires to cover all of the methods, both basic and advanced, in these areas which see widespread use in practice. I have also attempted to make the individual modules and sections somewhat self-sufficient, so that those who seek specific information on a single topic can quickly find what they need. Hopefully these aspirations will eventually be achieved; in the meantime, I welcome your comments, corrections, and feedback so that I can continue to improve this text.

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

Publication of this text in Connexions would have been impossible without the help of many people. A huge thanks to the various permanent and temporary staff at Connexions is due, in particular to those who converted the text and equations from my original handwritten notes into CNXML and MathML. My former and current faculty colleagues at the University of Illinois who have taught the second DSP course over the years have had a substantial influence on the evolution of the content, as have the students who have inspired this work and given me feedback. I am very grateful to my teachers, mentors, colleagues, collaborators, and fellow engineers who have taught me the art and practice of signal processing; this work is dedicated to you.

Chapter 1. Background, Review, and Reference

1.1. Discrete-Time Signals and Systems*

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

Real- and Complex-valued Signals

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

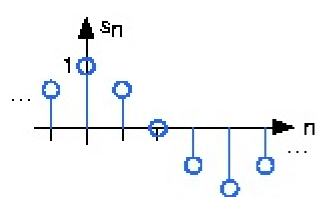

Figure 1.1. Cosine

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

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

Complex Exponentials

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

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

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.

$$\begin{aligned} e^{i2\pi(f+m)n} &= e^{i2\pi fn}e^{i2\pi mn} \\ &= e^{i2\pi fn} \end{aligned}$$ ()

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.

Sinusoids

Discrete-time sinusoids have the obvious form $s(n)=A\cos(2\pi fn+\varphi)$. As opposed to analog complex exponentials and sinusoids that can have their frequencies be any real value, frequencies of their discrete-time counterparts yield unique waveforms **only** when f lies in the interval $\left(-\left(\frac{1}{2}\right), \frac{1}{2}\right]$. 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.

Unit Sample

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

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

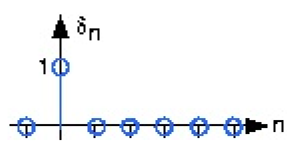

Figure 1.2. Unit sample

The unit sample.

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

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

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

Unit Step

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

$$u(n) = \begin{cases} 1 \text{ if } n \ge 0 \\ 0 \text{ if } n < 0 \end{cases} \quad ()$$

Symbolic Signals

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

Discrete-Time Systems

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

1.2. Systems in the Time-Domain*

A discrete-time signal $s(n)$ is **delayed** by n_0 samples when we write $s(n-n_0)$, with $n_0>0$. Choosing n_0 to be negative advances the signal along the integers. As opposed to analog delays, discrete-time delays can **only** be integer valued. In the frequency domain, delaying a signal corresponds to a linear phase shift of the signal's discrete-time Fourier transform: $(s(n-n_0) \leftrightarrow e^{-(i2\pi f n_0)} S(e^{i2\pi f}))$.

Linear discrete-time systems have the superposition property.

Superposition (1.1)

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

A discrete-time system is called **shift-invariant** (analogous to time-invariant analog systems) if delaying the input delays the corresponding output.

Shift-Invariant (1.2)

$$\text{If} \quad S(x(n))=y(n)\ , \text{Then} \quad S(x(n-n_0))=y(n-n_0)$$

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

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

The Difference Equation (1.3)

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

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

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

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