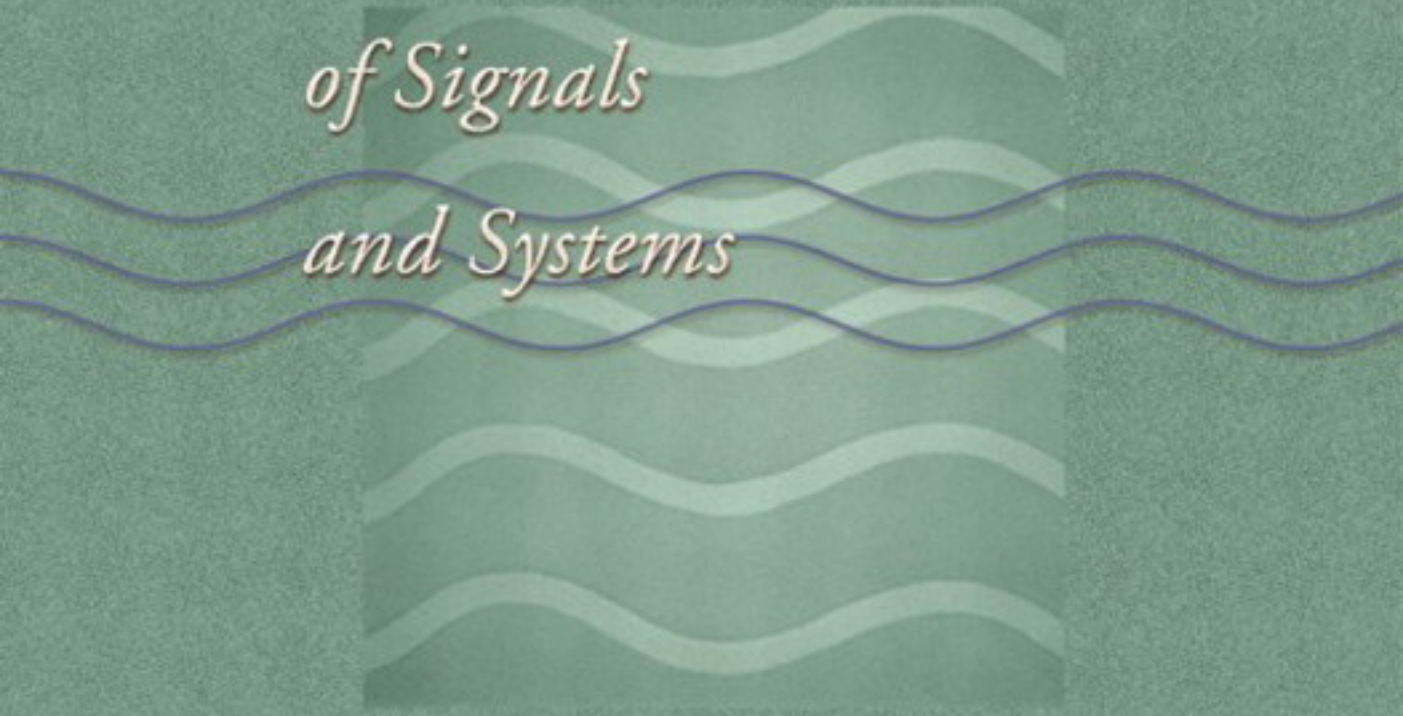
Structure and Interpretation of Signals and Systems



Edward A. Lee / Pravin Varaiya

Structure and Interpretation of Signals and Systems

Edward A. Lee and Pravin Varaiya
eal@eecs.berkeley.edu, varaiya@eecs.berkeley.edu
Electrical Engineering & Computer Science
University of California, Berkeley

July 4, 2000

Contents

Preface

This book is a reaction to a trauma in our discipline. We have all been aware for some time that "electrical engineering" has lost touch with the "electrical." Electricity provides the impetus, the pressure, the *potential*, but not the body. How else could microelectromechanical systems (MEMS) become so important in EE? Is this not a mechanical engineering discipline? Or signal processing. Is this not mathematics? Or digital networking. Is this not computer science? How is it that control system are applied equally comfortably to aeronautical systems, structural mechanics, electrical systems, and options pricing?

Like so many engineering schools, Berkeley used to have an introductory course entitled "Introduction to Electrical Engineering" that was about analog circuits. This quaint artifact spoke more about the origins of the discipline that its contemporary reality. Like engineering topics in schools of mining (which Berkeley's engineering school once was), ours has evolved more rapidly than the institutional structure around it.

Abelson and Sussman, in *Structure and Interpretation of Computer Programs* (MIT Press), a book that revolutionized computer science education, faced a similar transition in their discipline.

> "Underlying our approach to this subject is our conviction that 'computer science' is not a science and that its significance has little to do with computers."

Circuits used to be the heart of electrical engineering. It is arguable that today it is the analytical techniques that emerged from circuit theory that are the heart of the discipline. The circuits themselves have become an area of specialization. It is an important area of specialization, to be sure, with high demand for students, who command high salaries. But it is a specialization nonetheless.

Before Abelson and Sussman, computer programming was about getting computers to do your bidding. In the preface to *Structure and Interpretation of Computer Programs*, they say

> "First, we want to establish the idea that a computer language is not just a way of getting a computer to perform operations but rather that it is a novel formal medium for expressing ideas about methodology. Thus, programs must be written for people to read, and only incidentally for machines to execute."

In the origins of our discipline, a *signal* was a voltage that varies over time, an electromagnetic waveform, or an acoustic waveform. Now it is likely to be a sequence of discrete messages sent

over the internet using TCP/IP. The *state* of a system used to be adequately captured by variables in a differential equation. Now it is likely to be the registers and memory of a computer, or more abstractly, a process continuation, or a set of concurrent finite state machines. A *system* used to be well-modeled by a linear time-invariant transfer function. Now it is likely to be a computation in a Turing-complete computational engine. Despite these fundamental changes in the medium with which we operate, the methodology remains robust and powerful. It is the methodology, not the medium, that defines our field. Our graduates are more likely to write software than to push electrons, and yet we recognize them as electrical engineers.

Fundamental limits have also changed. Although we still face thermal noise and the speed of light, we are likely to encounter other limits before we get to these, such as complexity, computability, chaos, and, most commonly, limits imposed by other human constructions. A voiceband data modem, for example, faces the telephone network, which was designed to carry voice, and offers as immutable limits such non-physical constraints as its 3 kHz bandwidth. DSL modems face regulatory constraints that are more limiting than their physical constraints. Computer-based audio systems face latency and jitter imposed by the operating system.

The mathematical basis for the discipline has also shifted. Although we still use calculus and differential equations, we frequently need discrete math, set theory, and mathematical logic. Indeed, a major theme of this book is to illustrate that formal techniques can be used in a very wide range of contexts. Whereas the mathematics of calculus and differential equations evolved to describe the physical world, the world we face as system designers often has non-physical properties that are not such a good match to this mathematics. Instead of abandoning formality, we need to broaden the mathematical base.

Again, Abelson and Sussman faced a similar conundrum.

> "... we believe that the essential material to be addressed by a subject at this level is not the syntax of particular programming language constructs, nor clever algorithms for computing particular functions efficiently, nor even the mathematical analysis of algorithms and the foundations of computing, but rather the techniques used to control the intellectual complexity of large software systems."

This book is about signals and systems, not about large software systems. But it takes a computational view of signals and systems. It focuses on the methods "used to control the intellectual complexity," rather than on the physical limitations imposed by the implementations of old. Appropriately, it puts emphasis on discrete-time modeling, which is pertinent to the implementations in software and digital logic that are so common today. Continuous-time models describe the physical world, with which our systems interact. But fewer of the systems we engineer operate directly in this domain.

If imitation is the highest form of flattery, then it should be obvious whom we are flattering. Our title is a blatant imitation of *Structure and Interpretation of Computer Programs*. The choice of title reflects partly a vain hope that we might (improbably) have as much influence as they have. But more realistically, it reflects a sympathy with their cause. Like us, they faced an identity crisis in their discipline.

> "The computer revolution is a revolution in the way we think and in the way we express what we think. The essence of this change is what might best be called *procedural epistemology* – the study of the structure of knowledge from an imperative point of view, as opposed to the more declarative point of view taken by classical mathematical subjects. Mathematics provides a framework for dealing precisely with notions of 'what is.' Computation provides a fram-work for dealing precisely with notions of 'how to'."

Indeed, a major theme in our book is the connection between imperative (computational) and declarative (mathematical) descriptions of signals and systems. The laboratory component of the text, in our view, is an essential part of a complete study. The web content, with its extensive applets illustrating computational concepts, is an essential complement to the mathematics. Traditional electrical engineering has emphasized the declarative view. Modern electrical engineering has much closer ties to computer science, and has to complement this declarative view with an imperative one.

Besides being a proper part of the intellectual discipline, an imperative view has another key advantage. It enables a much closer connection with "real signals and systems," which are often too messy for a complete declarative treatment. While a declarative treatment can easily handle a sinusoidal signal, an imperative treatment can easily handle a voice signal. One of our objectives in designing this text and the course on which it is based is to illustrate concepts with real signals and systems *at every step*.

This is quite hard to do in a textbook. The print medium biases authors towards the declarative view simply because its static nature is better suited to the declarative view than to the imperative view. Our solution to this problem has been heavily influenced by the excellent and innovative textbooks by Steiglitz, *A Digital Signal Processing Primer – with Applications to Digital Audio and Computer Music* (Addison-Wesley), and McClellan, Schafer and Yoder, *DSP First – A Multimedia Approach*. Steiglitz leverages natural human interest in the very human field of music to teach, spectacularly gently, very sophisticated concepts in signals and systems. McClellan, Schafer and Yoder beautifully integrate web-based content and laboratory exercises, complementing the traditional mathematical treatment with accessible and highly motivational manipulation of real signals. If you are familiar with these books, you will see their influence all over the laboratory exercises and the web content.

Notes to Instructors

The course begins by describing signals as functions, focusing on characterizing the domain and the range. Systems are also described as functions, but now the domain and range are sets of signals. Characterizing these functions is *the* topic of this course. Sets and functions provide the unifying notation and formalism.

We begin by describing systems using the notion of state, first using automata theory and then progressing to linear systems. Frequency domain concepts are introduced as a complementary toolset, different from that of state machines, and much more powerful when applicable. Frequency decomposition of signals is introduced using psychoacoustics, and gradually developed until all four Fourier transforms (the Fourier series, the Fourier transform, the discrete-time Fourier transform, and the DFT) have been described. We linger on the first of these, the Fourier series, since it is conceptually the easiest, and then quickly present the others as simple generalizations of the Fourier series. Finally, the course closes by using these concepts to study sampling and aliasing, which helps bridge the computational world with the physical world.

This text has evolved to support a course we now teach regularly at Berkeley to about 500 students per year. An extensive accompanying web page is organized around Berkeley's 15 week semester, although we believe it can be adapted to other formats. The course organization at Berkeley is as follows:

Week 1 – Signals as Functions – Chapters 1 and 2. The first week motivates forthcoming material by illustrating how signals can be modeled abstractly as functions on sets. The emphasis is on characterizing the domain and the range, not on characterizing the function itself. The startup sequence of a voiceband data modem is used as an illustration, with a supporting applet that plays the very familiar sound of the startup handshake of V32.bis modem, and examines the waveform in both the time and frequency domain. The domain and range of the following signal types is given: sound, images, position in space, angles of a robot arm, binary sequences, word sequences, and event sequences.

Week 2 – Systems as Functions – Chapters 1 and 2. The second week introduces systems as functions that map functions (signals) into functions (signals). Again, it focuses not on how the function is defined, but rather on what is its domain and range. Block diagrams are defined as a visual syntax for composing functions. Applications considered are DTMF signaling, modems, digital voice, and audio storage and retrieval. These all share the property that systems are required to convert domains of functions. For example, to transmit a digital signal through the telephone system, the digital signal has to be converted into a signal in the domain of the telephone system

(i.e., a bandlimited audio signal).

Week 3 – State – Chapter 3. Week 3 is when the students start seriously the laboratory component of the course. The first lecture in this week is therefore devoted to the problem of relating declarative and imperative descriptions of signals and systems. This sets the framework for making the intellectual connection between the labs and the mathematics.

The purpose of this first lab exercise is to explore arrays in Matlab and to use them to construct audio signals. The lab is designed to help students become familiar with the fundamentals of Matlab, while applying it to synthesis of sound. In particular, it introduces the vectorization feature of the Matlab programming language. The lab consists of explorations with sinusoidal sounds with exponential envelopes, relating musical notes with frequency, and introducing the use of discrete-time (sampled) representations of continuous-time signals (sound).

Note that there is some potential confusion because Matlab uses the term "function" somewhat more loosely than the course does when referring to mathematical functions. Any Matlab command that takes arguments in parentheses is called a function. And most have a well-defined domain and range, and do, in fact, define a mapping from the domain to the range. These can be viewed formally as a (mathematical) functions. Some, however, such as `plot` and `sound` are a bit harder to view this way. The last exercise in the lab explores this relationship.

The rest of the lecture content in the third week is devoted to introducing the notion of state and state machines. State machines are described by a function *update* that, given the current state and input, returns the new state and output. In anticipation of composing state machines, the concept of *stuttering* is introduced. This is a slightly difficult concept to introduce at this time because it has no utility until you compose state machines. But introducing it now means that we don't have to change the rules later when we compose machines.

Week 4 – Nondeterminism and Equivalence – Chapter 3. The fourth week deals with nondeterminism and equivalence in state machines. Equivalence is based on the notion of simulation, so simulation relations and bisimulation are defined for both deterministic and nondeterministic machines. These are used to explain that two state machines may be equivalent even if they have a different number of states, and that one state machine may be an abstraction of another, in that it has all input/output behaviors of the other (and then some).

During this week, students do a second set of lab exercises that explore the representation of images in Matlab, relating the Matlab use of color maps with a formal functional model. It discusses the file formats for images, and explores the compression that is possible with colormaps and with more sophisticated techniques such as JPEG. The students construct a simple movie, reinforcing the notions of sampling introduced in the previous lab. They also blur an image and create a simple edge detection algorithm for the same image. This lab also reinforces the theme of the previous one by asking students to define the domain and range of mathematical models of the relevant Matlab functions. Moreover, it begins an exploration of the tradeoffs between vectorized functions and lower-level programming constructs such as for loops. The edge detection algorithm is challenging (and probably not practical) to design using only vectorized functions. As you can see, the content of the labs lags the lecture by about one week so that students can use the lab to reinforce material that they have already had some exposure to.