

DESIGNING A RELEVANT LAB FOR INTRODUCTORY SIGNALS AND SYSTEMS

Edward A. Lee

eal@eecs.berkeley.edu
Electrical Engineering & Computer Science
University of California, Berkeley

*Proc. of the First Signal Processing Education Workshop,
Hunt, Texas, October 15 - 18, 2000*

ABSTRACT

Contemporary reality in a digital, networked, computational world suggests a different approach to the hands-on component of an engineering curriculum. At Berkeley, we have introduced a new course that introduces signals and systems to EECS majors. A major objective of the course is to introduce applications early, well before the students have built up enough theory to fully analyze the applications. This helps to motivate the students to learn the theory. Indeed, a major theme of this course is the connection between a mathematical (declarative) and a computational (imperative) view of systems. In particular, we avoid using software as merely a more convenient way to do calculations that could otherwise be done by hand. Instead, we emphasize the use of software to perform operations that could not possibly be done by hand, operations on real signals such as sounds and images. We use Matlab and Simulink, and introduce them as complementary tools with distinct models of computation. Simulink, however, has some distinct limitations that compromise its utility. We offer some suggestions for future tools that would better match our objectives.

1. INTRODUCTION

We have developed a set of laboratory exercises based on Matlab and Simulink for a new introductory signals and systems course at Berkeley. The purpose of these exercises is to help reconcile the declarative (what is) and imperative (how to) points of view on signals and systems. The mathematical treatment that dominates in the lecture and textbook is declarative, in that it asserts properties of signals and studies the relationships between signals that are implied by systems. The labs focus on an imperative style, where signals and systems are constructed procedurally. The labs are included in a forthcoming text [2].

Matlab and Simulink, distributed by The MathWorks, Inc., are chosen as the basis for these exercises because they are widely used by practitioners in the field, and because they are capable of realizing interesting systems. Why use both Matlab and Simulink? Although they are integrated

into a single package, Matlab and Simulink are two very different pieces of software with radically different approaches to modeling of signals and systems. Matlab is an imperative programming language, whereas Simulink is a block diagram language. In Matlab, one specifies the sequence of steps that construct a signal or operate on a signal to produce a new signal. In Simulink, one connects blocks that implement elementary systems to construct more interesting systems. The systems we construct are aggregates of simpler systems.

Matlab fundamentally operates on matrices and vectors. Simulink fundamentally operates on discrete and continuous-time signals. Discrete-time signals, of course, can be represented as vectors, as long as they are finite. However, the signals of interest in this course are rarely finite, so Simulink provides a closer approximation. Continuous-time signals, of course, can only be approximated in software. Simulink approximates continuous-time signals by discretizing time in an adaptive and largely transparent way. The user (the model builder) can pretend that he or she is operating directly on continuous-time signals.

There is considerable value in becoming adept with these software packages. Matlab and Simulink are often used in practice for “quick-and-dirty” prototyping of concepts, and increasingly, with code generation, to produce production embedded software. In a matter of a few hours, very elaborate models can be constructed. This contrasts with the weeks or months that would often be required to build a hardware prototype to test the same concept.

Of course, a conventional programming language such as C++ or Java could also be used to construct prototypes of systems. However, these languages lack the rich libraries of built-in functions that Matlab and Simulink have. A task as conceptually simple as plotting a waveform can take weeks of programming in Java to accomplish well. Algorithms, such as the FFT or filtering algorithms, are also built in, saving considerable effort.

One hindrance to using these software environments for beginning students is that Matlab and Simulink both have capabilities that are much more sophisticated than anything covered in the course. This may be a bit intimidating at first,

particularly with Simulink, where it is hard to avoid stumbling across facilities much more sophisticated than what the students need. Their reaction is likely to be “what the heck is singular-value decomposition!?!?!?”. Students need to learn to ignore what they don’t understand, and focus on building up their abilities gradually.

For students with no background in programming, these exercises are difficult at first. Matlab, at its root, is a fairly conventionally programming language, and it requires a clear understanding of programming concepts such as variables and flow of control (for loops, while loops). As programming languages go, it is an especially easy one to learn, however. Its syntax is straightforward and close to that of the mathematical concepts that it emulates. Moreover, since it is an interpreted language (in contrast to a compiled language), students can easily experiment by just typing in commands at the console and seeing what happens. At Berkeley, only a tiny percentage of the students who take this class have no programming background, so there is not much of a problem.

1.1. Mechanics of the Labs

The labs are divided into two distinct sections, **in lab** and **independent**. This organization is inspired by that of the (excellent) laboratory exercises in *DSP First*[4]. The purpose of the in-lab section is to introduce concepts needed for later parts of the lab. Each in-lab section is designed to be completed during a scheduled lab time with an instructor present to clear up any confusing or unclear concepts. The in-lab section is completed by obtaining the signature of an instructor on a verification sheet.

The independent section begins where the in-lab section leaves off. It can be completed within the scheduled lab period, or may be completed on the students’ own time. They write a brief summary of their solution, following a supplied template, and turn it in at the beginning of the next scheduled lab period.

2. LAB CONTENT

There are 11 lab exercises, each designed to be completed in one week. In a 15 week semester, this leaves one week at the beginning devoted to purely organizational issues (setting up computer accounts, meeting the lab instructors, etc.), one week for gaining familiarity with the technology of the course (starting Matlab and Simulink, finding the tutorials, finding and printing the lab assignments, finding the lab report templates), and two weeks in the middle of the semester for midterm review. The lab exercises are designed to not require external instruction in Matlab or Simulink, nor use of supplementary materials such as books on Matlab or Simulink. The lab instructions and the on-line help are sufficient. We briefly describe each of the labs. The reader may wish to consult [3] to evaluate the alignment of the lab topics with those of the rest of the course.

2.1. Arrays and Sound (week 3)

The purpose of the first lab (which is assigned in week 3) is to explore arrays in Matlab and to use them to construct sound 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).

By the time the students do this lab, the course has discussed using sets and functions to represent signals, focusing on defining the domain and range of the functions. 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.

2.2. Images (week 4)

The second lab explores 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, getting results like those shown in figure 1.

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.

2.3. State Machines (week 5)

The third lab uses Matlab as a low-level programming language to construct state machines according to a systematic design pattern that will allow for easy composition. The theme of the lab is establishing the correspondence between pictorial representations of finite automata, mathematical functions giving the state update, and software realizations.

The main project in this lab exercise is to construct a virtual pet. This problem is inspired by the Tamagotchi virtual

pet made by Bandai in Japan. Tamagotchi pets, which translate as “cute little eggs,” were extremely popular in the late 1990’s, and had behavior considerably more complex than that described in this exercise. The pet is cat that behaves as follows:

It starts out *happy*. If you *pet* it, it *purrs*. If you *feed* it, it *throws up*. If *time passes*, it gets *hungry* and *rubs* against your legs. If you feed it when it is hungry, it purrs and gets happy. If you pet it when it is hungry, it *bites* you. If time passes when it is hungry, it *dies*.

The italicized words and phrases in this description should be elements in either the state space or the input or output alphabets. Students define the input and output alphabets and give a state transition diagram. They construct a function in Matlab that returns the next state and the output given the current state and the input. They then write a program to execute the state machine until the user types ‘quit’ or ‘exit.’

Next, the students design an open-loop controller that keeps the virtual pet alive. This illustrates that systematically constructed state machines can be easily composed.

This lab builds on the flow control constructs (for loops) introduced in the previous labs and introduces string manipulation and the use of M files.

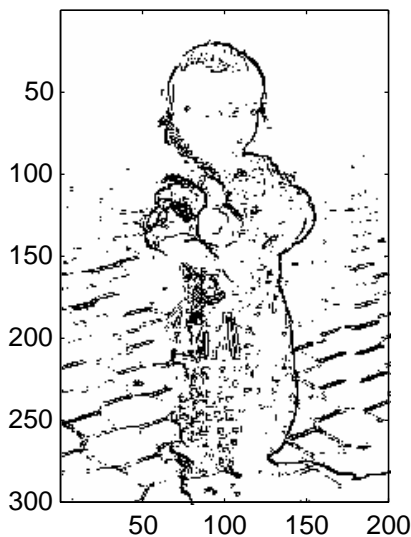
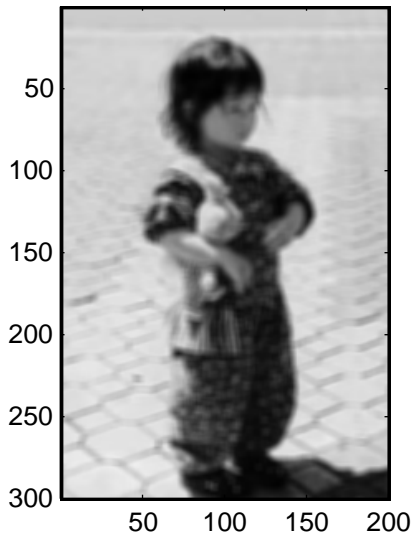


Figure 1: Blurred image constructed with a 5×5 moving average (above), and simple edge detection (below).

2.4. Control Systems (week 6)

In the previous lab, students were able to construct an open-loop controller that would keep their virtual pet alive. In this lab, they modify the pet so that its behavior is nondeterministic. In particular, they modify the cat’s behavior so that if it is hungry and they feed it, it sometimes gets happy and purrs (as it did before), but it sometimes stays hungry and rubs against your legs. They then attempt to construct an open-loop controller that keeps the pet alive, but of course no such controller is possible without some feedback information. So they are asked to construct a state machine that can be composed in a feedback arrangement such that it keeps the cat alive.

The semantics of feedback in this course are consistent with tradition in signals systems. Computer scientists call this style “synchronous composition,” and define the behavior of the feedback system as a (least or greatest) fixed point of a monotonic function on a partial order. In a course at this level, we cannot go into this theory in much depth, but we can use this example to explore the subtleties of synchronous feedback.

In particular, the controller composed with the virtual pet does not, at first, seem to have enough information available to start the model executing. The input to the controller, which is the output of the pet, is not available until the input to the pet, which is the output of the controller, is available. There is a bootstrapping problem. The (better) students learn to design state machines that can resolve this apparent paradox.

Most students find this lab quite challenging, but also very gratifying when they figure it out. The concepts behind it are deep, and the better students realize that. The weaker students, however, just get by, getting something working without really understanding how to do it systematically.

2.5. Difference Equations (week 8)

After a one week break to prepare for the first midterm, the lab reconvenes after the course has shifted themes towards linear time invariant systems. In this lab, the students build on the previous exercise by constructing state machine models (now with infinite states and linear update equations). They build stable, unstable, and marginally stable state machines, describing them as difference equations.

The prime example of a stable system yields a sinusoidal signal with a decaying exponential envelope. The corresponding state machine is a simple approximate model of the physics of a plucked string instrument, such as a guitar. It is also the same signal that the students generated in the first lab by more direct (and more costly) methods. They compare the complexity of the state machine model with that of the sound generators that they constructed in the first lab, finding that the state machine model yields sinusoidal outputs with considerably fewer multiplies and adds than direct calculation of trigonometric and exponential functions.

The prime example of a marginally stable system is an oscillator. The students discover that an oscillator is just a boundary case between stable and unstable systems.

2.6. Differential Equations (week 9)

The purpose of this lab is to experiment with models of continuous-time systems that are described as differential equations. The exercises aim to solidify state-space concepts while giving some experience with software that models continuous-time systems.

The lab uses Simulink, a companion to Matlab. The lab is self contained, in the sense that no additional documentation for Simulink is needed. Instead, we rely on the on-line help facilities. However, these are not as good for Simulink as for Matlab. The lab exercise have to guide the students extensively, trying to steer clear of the more confusing parts. As a result, this lab is bit more “cookbook-like” than the others.

Simulink is a block-diagram modeling environment. As such, it has a more declarative flavor than Matlab, which is imperative. You do not specify exactly how signals are computed in Simulink. You simply connect together blocks that represent systems. These blocks declare a relationship between the input signal and the output signal. One of the reasons for using Simulink is to expose students to this very different style of programming.

Simulink excels at modeling continuous-time systems. Of course, continuous-time systems are not directly realizable on a computer, so Simulink must discretize the system. There is quite a bit of sophistication in how this is done, but

the students are largely unaware of that. The fact that they do not specify how it is done underscores the observation that Simulink has a declarative flavor.

2.7. Spectrum (week 10)

The purpose of this lab is to learn to examine the frequency domain content of signals. Two methods are used. The first method is to plot the discrete Fourier series coefficients of finite signals. The second is to plot the Fourier series coefficients of finite segments of time-varying signals, creating a spectrogram.

The students have, by this time, done quite a bit with Fourier series, and have established the relationship between finite signals and periodic signals and their Fourier series.

Matlab does not have any built-in function that directly computes Fourier series coefficients, so an implementation using the FFT is given to the students. The students construct a chirp, listen to it, study its instantaneous frequency, and plot its Fourier series coefficients. They then compute a time-varying discrete-Fourier series using short segments of the signal, and plot the result in a waterfall plot. Finally, they render the same result as a spectrogram, which leverages their study of color maps in lab 2. The students also render the spectrogram of a speech signal.

The lab concludes by studying beat signals, created by summing sinusoids with closely spaced frequencies. A single Fourier series analysis of the complete signal shows its structure consisting of two distinct sinusoids, while a spectrogram shows the structure that corresponds better with what the human ear hears, which is a sinusoid with a low-frequency sinusoidal envelope.

2.8. Comb Filters (week 11)

The purpose of this lab is to use a comb filter to deeply explore concepts of impulse response and frequency response, and to lay the groundwork for much more sophisticated musical instrument synthesis done in the next lab. The “sewer pipe” effect of a comb filter is distinctly heard, and the students are asked to explain the effect in physical terms by considering sound propagation in a cylindrical pipe.¹ The comb filter is analyzed as a feedback system, making the connection to the virtual pet.

The lab again uses Simulink, this time for discrete-time processing. Discrete-time processing is not the best part of Simulink, so some operations are awkward. Moreover, the blocks in the block libraries that support discrete-time processing are not well organized. It can be difficult to discover how to do something as simple as an N -sample delay or an impulse source. The lab has to identify the blocks that the students need, which again gives it a more “cookbook-like” flavor. The students cannot be expected to wade through the extensive library of blocks, most of which will seem utterly incomprehensible.

¹Only a small percentage of the students do this successfully.

2.9. Plucked String Instrument (week 13)

The purpose of this lab is to experiment with models of a plucked string instrument, using it to deeply explore concepts of impulse response, frequency response, and spectrograms. The methods discussed in this lab were invented by Karplus and Strong [1]. The design of the lab itself was inspired by the excellent book of Steiglitz [5].

The lab uses Simulink, modifying the comb filter of the previous lab in three ways. First, the comb filter is initialized with random state, leveraging the concept of zero-input state response, studied previously with state-space models. Then it adds a lowpass filter to the feedback loop to create a dynamically varying spectrum, and it uses the spectrogram analysis developed in previous labs to show the effect. Finally, it adds an allpass filter to the feedback loop to precisely tune the resulting sound by adjusting the resonant frequency.

2.10. Modulation and Demodulation (week 14)

The purpose of this lab is to use frequency domain concepts to study amplitude modulation. This is motivated, of course, by talking about AM radio, but also about digital communication systems, including digital cellular telephones, voice-band data modems, and wireless networking devices.

The students are given the following problem scenario:

Assume we have a signal that contains frequencies in the range of about 100 to 300 Hz, and we have a channel that can pass frequencies from 700 to 1300 Hz. The task is to modulate the first signal so that it lies entirely within the channel passband, and then to demodulate to recover the original signal.

The test signal is a chirp. The frequency numbers are chosen so that every signal involved, even the demodulated signal with double frequency terms, is well within the audio range at an 8 kHz sample rate. Thus, students can reinforce the visual spectral displays with sounds that illustrate clearly what is happening.

A secondary purpose of this lab is to gain a working (users) knowledge of the FFT algorithm. In fact, they get enough information to be able to fully understand the algorithm that they were previously given to compute discrete Fourier series coefficients.

In this lab, the students also get an introductory working knowledge of filter design. They construct a specification and a filter design for the filter that eliminates the double frequency terms. This lab requires the Signal Processing Toolbox of Matlab for filter design.

2.11. Sampling and Aliasing (week 15)

The purpose of this lab is to study the relationship between discrete-time and continuous-time signals by examining sampling and aliasing. Of course, a computer cannot

directly deal with continuous-time signals. So instead, we construct discrete-time signals that are defined as samples of continuous-time signals, and then operate entirely on them, downsampling them to get new signals with lower sample rates, and upsampling them to get signals with higher sample rates. The upsampling operation is used to illustrate oversampling, as commonly used in digital audio players such as compact disk players. Once again, the lab is carefully designed so that all phenomena can be heard.

3. TEAMWORK

An engineer rarely works alone. Cooperation and collaboration are a very real part of the working world. Learning to collaborate effectively is important. In view of this, and the reality that they will do it anyway, students are encouraged to work cooperatively with up to two other students on lab reports. A team of three or fewer may turn in a single lab report with up to three names.

There are distinct problems with this approach. One is that even with this policy, students were found to copy the work of others in the class and include it unattributed in their own lab reports. Moreover, some teams partitioned the work (usually by lab, since each lab exercise is difficult to partition effectively). As a consequence, there were students who, based on their performance in the labs, appeared to fully understand the material, and yet were unable to perform in the exams.

4. CONCLUSION

We did not conduct a rigorous study of the impact of the lab, nor did we survey the students for their opinions about it. So all I can offer is general impressions. First, the students appeared to find the lab relatively easy, with the possible exception of the Plucked String modeling. The grades were consistently high, and questions on exams that were based primarily on lab material consistently showed strong performance.

The integration of the labs with the course is very careful, and very tightly scheduled. So much so that slipping by one or two lectures can create problems with the labs. We were trying for a careful balance of having covered the necessary background material, but not so long ago that the students have forgotten.

A key question that we can address is what we might change in the future. In my opinion, the weakest part of this lab is the use of Simulink. Although Simulink can model discrete-time systems, and mixed discrete and continuous-time systems, it does not do so well. The discrete-time models in Simulink are not truly discrete-time models. In Simulink semantics, discrete-time signals are piecewise constant continuous-time signals. This becomes evident when constructing mixed signal models. Confusion can be avoided by carefully avoiding this mixture. However, the

block library organization and the solver configuration does not make it easy to avoid this.

Although this interpretation of discrete-time signals might seem innocuous enough, unfortunately, it undermines some of the basic principles of the course, where domains and ranges of functions are carefully defined to get models of signals. It seems to say that precise definition of the domain is not important.

In addition, when doing discrete-time modeling, it is regrettable that the students have to specify mysterious “solver options” that are quite difficult to explain to them. We are forced to tell them “just do it ... don’t ask why.”

Simulink has other weaknesses. As discussed above, the on-line documentation and block organization are not good enough for the students to be able to find solutions on their own. The labs that use Simulink are forced to have a more “cookbook-like” flavor, clearly leading the students through the widely scattered block libraries to find what they need. Students need to follow our instructions closely, or they are likely to discover very puzzling behavior. We could probably improve this situation considerably by constructing custom libraries and on-line documents.

Finally, Simulink is more limited and awkward than Matlab in its ability to read and write audio files. We find that it is easier to simply write signals to the workspace and use `soundsc` to listen to them.

References

- [1] K. Karplus and A. Strong, “Digital Synthesis of Plucked-String and Drum Timbres,” *Computer Music Journal*, vol. 7, no. 2, pp. 43-55, Summer 1983.
- [2] Edward A. Lee & Pravin Varaiya, *Structure and Interpretation of Signals and Systems*, textbook draft, 2000 (<http://www.eecs.berkeley.edu/eal/eecs20>).
- [3] Edward A. Lee and Pravin Varaiya, “Introducing Signals and Systems – The Berkeley Approach,” *Proc. of the First Signal Processing Education Workshop*, Hunt, Texas, October 15 - 18, 2000 (this volume).
- [4] James H. McClellan, Ronald W. Schafer, and Mark A. Yoder, *DSP First: A Multimedia Approach*, Prentice-Hall, 1998.
- [5] Ken Steiglitz, *A DSP Primer: With Applications to Digital Audio and Computer Music*, Addison-Wesley, 1996.