A SYSTEM CHARACTERIZATION/IDENTIFICATION LABORATORY TEACHING TOOL FOR INTERNET

S. Chatfield, D. Cochran, M. Sadaka, and D. Sinno

Telecommunications Research Center
Arizona State University; Tempe, AZ 85287-7206

ABSTRACT

This paper describes the design and implementation of a simple internet-based "virtual systems laboratory" teaching tool. The tool is intended to provide a mechanism for assignment of system characterization/identification exercises in courses where students have access to internet electronic mail, but not necessarily to a common computing environment or more sophisticated internet resources such as Mosaic or Netscape. Specific exercises developed for use with this software in classes at Arizona State University are described.

1. INTRODUCTION

A standard textbook mechanism in the teaching of Signals and Systems at an undergraduate level is to present one or more input signals and their corresponding output signals, generally in graphical form, and ask the student to deduce information about the system (i.e., Is it linear? Time-invariant? What is its impulse response? Frequency response? etc.) In courses with laboratory components, students can be presented with the highly beneficial experience of designing their own tests to answer such questions about a system. In this setting, students themselves determine appropriate "test signals" to use as the inputs and measure the outputs with instruments. In contrast to most textbook exercises, there is no guarantee the test will provide the answer sought unless the test signals are well designed. More advanced courses can also benefit from signal laboratory exercises, where the unknown system might be a simulation of a noisy communication channel or an image blurring phenomenon which students must identify.

Computer-based packages can provide design laboratory experience of a similar nature, with some sacrifice of genuine "hands-on" experience using lab equipment and instruments. There are many good packages to support this kind of computer laboratory exercise, some of which are commonly available on university computer systems for student use and some of which are inexpensive enough for students to acquire for their own computers. The availability of options presents its own difficulties, however, by making it hard for instructors and students to agree on a standard (I don't know MATLAB, so can I write my own simulated systems in C? or I have trouble getting into the lab that has Maple, so could you prepare a Mathematica version of the exercise that I could do on the computer at work?)

This paper describes the design and implementation of a "virtual systems laboratory" accessible to any student having access to internet electronic mail. Students design test signals using any of their favorite software tools and email them to the system in a simple ASCII format. The tool parses the incoming message, passes the input signal data to the appropriate system module (i.e., a program that produces the output signal data), then generates a reply email message containing the output signal (or possibly an error message). This is all typically completed within a minute. The student receives the output signal by email and can then display or analyze it using any desired software and repeat the process as necessary to determine the desired properties of the system.

This approach is considerably less ambitious than highly impressive distance learning experiments for signal processing than have been undertaken recently (see, for example, [2, 3]). Its feature is simplicity: using the framework described in this paper, an instructor should be able to design and implement a good "laboratory" project for a distance learning graduate course in a single afternoon.

2. ARCHITECTURAL OVERVIEW

The software described in this paper runs on a Unix™ workstation computer. The overall architecture is as depicted in figure 1. The *laboratory manager* is invoked by the Unix cron utility at regular intervals.[1] When started, the laboratory manager checks its mail queue. If no messages are waiting, it quits; otherwise, it (i) reads the first message in the queue, (ii) parses the *subject* field to get the name of the module to be used in processing the input signal data, (iii) reads the *body* field and formats the input signal data in a file, (iv) invokes the appropriate module. When the module exits, the laboratory manager reads the output signal data from a file written by the module, formats a reply message, and sends it to the originator. This process is repeated until the message queue is empty, at which point the laboratory manager quits until restarted by cron.

*This work was supported by an Arizona State University CIEE/CEAS Educational Improvement Grant.

[1] An interval of one minute is the minimum supported by cron on many systems. This seems to be satisfactory to provide acceptable turn-around time for students who are working on exercises in "real time."

0-7803-3192-3/96 $5.00©1996 IEEE

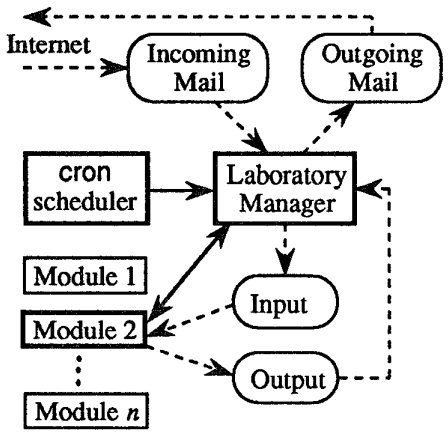

Figure 1. Architecture of the systems laboratory teaching tool. Solid arrows indicate program control and dashed arrows show data flow.

3. USER INTERACTION

The user's (student's) interaction with the laboratory is entirely via electronic mail. The student's message to the laboratory serves to identify which of possibly many available system modules are to process the input signal data and to specify the input signal data. It also implicitly provides a return address to which the processed data and/or any error messages are to be sent. The laboratory has an internet address to which messages having the following format are sent:

- The *subject* field specifies the system by which the signal data are to be processed. The subject line is scanned for the first occurrence of a valid processing module name (described below). If one is found, it is invoked to process the signal data; otherwise, a message is returned to the student indicating that the module name provided is not valid and containing a list of valid module names.
- The *body* field contains a specification of the input signal to be processed by the module. Mechanisms for signal specification are described below.

3.1. Input signal descriptors

An input signal may be given explicitly in the body field or specified by one of a set of pre-defined *signal descriptors*. In the former case, the body field consists of a simple list of numbers; e.g.,

```
3.49
-2.00
19
 ⋮
```

In the latter case, a valid *signal descriptor* should appear on the first line of the body field. Currently available signal descriptors are:

1. `Delta` offset amplitude length — The input signal is a string of length length consisting of offset zeros followed by a single sample of value amplitude and then length − offset − 1 more zeros.
2. `Step` offset amplitude length — The input signal is a string of length length consisting of offset zeros followed by length − offset samples having value amplitude.
3. `Ramp` offset slope length — The input signal is a string of length length whose n^{th} sample is zero for $0 \leq n <$ offset and is slope $\times$ (n − offset) for $n =$ offset, ..., length − 1.
4. `Constant` amplitude length — The input signal is a string of length length in which every sample has value amplitude.
5. `Cosine` frequency amplitude length — The input signal is a string of length length whose n^{th} sample is amplitude $\times \cos(2\pi n \text{frequency})$.
6. `Sine` frequency amplitude length — The input signal is a string of length length whose n^{th} sample is amplitude$\times$ $\sin(2\pi n \text{frequency})$.

The length of the input signal data is arbitrary. The processing module has the responsibility for ensuring the data provided are appropriate (in length, amplitude, sign, etc.) and for deciding how to handle cases where the data are unsuitable for processing.

3.2. Processing modules

The systems that act on the signals specified in the ways described above are implemented as modules; i.e., they are independent executable programs invoked as subprocesses when needed. The laboratory manager writes input data into a file (named input), invokes a processing module, and reads output signal data from another file output after the module completes execution. There are several implications of this approach:

- System modules may be written in any language and use any available software and hardware resources to process the input signal data. The laboratory manager will return whatever is in the file output when the module completes execution.
- The module can impose any desired restrictions and/or assumptions on the input data; e.g.,
  - A module that implements a causal FIR filter by convolution with a sequence of length n might assume that $n-1$ zeros follow the last data value in input.
  - In exercises where physical parameters of the input signal (e.g., sampling rate, start time reference, frequency bin spacing) are important, the exercise assignment and module must agree on a convention since the laboratory manager has no means of determining this information. In such cases, students should be warned to exercise care in using the built-in Sine and Cosine functions.
  - Complex and multidimensional data can be handled by any number of indexing conventions.

- Modules are responsible for checking that the data in input is appropriate to the processing to be performed. If a module "hangs" and does not complete execution because of corrupted or inappropriate data, the current system will not recover.

3.3. Output signal formats

Although output signals will typically be lists of numbers, the laboratory manager will return the contents of the ASCII file output written by the processing module regardless of its format. This allows modules to write error messages and other comments into the returned message. It also allows the module to return data in specially formatted forms (e.g. MATLAB M-files).

4. EXAMPLE EXERCISES

To illustrate the use of the laboratory, two exercises are summarized briefly in this section. The first is a routine system characterization problem suitable for an undergraduate "Signals and Systems" course. The second is a more involved project exercise suitable for a first-year graduate course in stochastic signal processing.

4.1. Deterministic system characterization

Two modules, problem1a and *problem1b* are prepared and installed in the laboratory's module library. The module problem1a implements the system $y(k) = \cos(\pi k)x(k)$ and the module problem1b implements the system $y(k) = 2x(k) - x(k-1)$.

Assignment

The modules problem1a and problem1b implement causal discrete-time systems. One of the systems is LTI and the other is not.

a. Which system is LTI?

b. Explain how you determined which system is not LTI. Which property, linearity or time-invariance, are you certain it does not have?

c. Test the LTI system's impulse response $h(k)$ and use it to predict its unit step response $s(k)$. Verify your prediction by testing the system's unit step response.

Approach

To test for time invariance, the student measures the first system's response a test signal:

```
% mail eee303@yamuna.eas.asu.edu
Subject: problem1a
1
1
1
```

The reply is received:

```
% Mail from eee303@yamuna.eas.asu.edu
1.00
-1.00
1.00
```

The student then measures the system's response to a time-shifted replicate of the same signal.

```
% mail eee303@yamuna.eas.asu.edu
Subject: problem1a
0
1
1
1
```

The reply shows the system is not time-invariant:

```
% Mail from eee303@yamuna.eas.asu.edu
0.00
-1.00
1.00
-1.00
```

Proceeding in a logical way, the problem can be solved completely in approximately 15 minutes. One might use the problem1b module later in the course in a problem on frequency response, comparing the DTFT of the measured impulse response to the system's response to a collection of cosine waves at various frequencies.

Note that the instructor's preparation for this exercise is minimal. An implementation (in C with no error checking) of the module problem1b is only a few lines:

```
/* Module title: problem1b
   Course:       EEE 303 - Signals and Systems
   Function:     y(k) = 2x(k) - x(k-1)
*/
# include <stdio.h>
# include <math.h>
main()
{
FILE *infile, *outfile, *fopen();
int count=0, k=0;
float xold, xnew, y;
infile=fopen("input","r");
outfile=fopen("output","w");
xold=0;
for(k=0; fscanf(infile,"%f",&xnew)!=EOF; k++)
   {y=2*xnew-xold;
   fprintf(outfile,"%f \n",y);
   xold=xnew;
   }
fprintf(outfile,"%f \n",-xold);
fclose(outfile); fclose(infile);
}
```

4.2. Characterization of a noisy channel

A more involved example has the goal of predicting the bit error rate in decoding a BPSK communication signal transmitted over a noisy channel. The channel is realized by a module that adds colored (highpass) gaussian noise to the input signal. For the students' convenience, another module is provided that accepts ASCII text (one character per line) and produces the corresponding antipodal binary representation of the text (i.e., writes a list of 1's and -1's into output).

Assignment

The module channel implements a communication channel in which transmitted signals are corrupted by additive WSS gaussian noise.

a. Characterize the noise. Estimate its mean, autocovariance function, and spectral density.

b. Assume the information bits in a binary signal are independent and that zeros and ones are equally likely. Based on your results in part (a), determine the optimal decision threshold for a decoding a BPSK representation of such a signal transmitted through the channel. Calculate the SNR that should be necessary to obtain a bit error rate (BER) of 10^{-1}. Test your conclusion by decoding your own signal after transmitting it through the channel.

c. If each bit is transmitted four times and the four received values corresponding to each bit are averaged before comparing to the decision threshold, what BER would you expect using the SNR determined in part (b)? Test your conclusions. In view of the spectral density of the noise, explain why this averaging technique works so well.

An important aspect of this assignment is that the laboratory allows student's to work with a "channel" that is genuinely unknown rather than being constrained to use pre-canned data.

5. SOFTWARE AVAILABILITY

All of the software described in this paper, a postscript version of this paper, some additional documentation, and the processing modules used in the exercises presented in the preceding section are available by anonymous ftp from the host trcsun3.eas.asu.edu. Interested parties should retrieve all of the files from the directory /pub/blackbox.

6. CONCLUSIONS

While the structure of the teaching tool described in this paper is simple, it allows high-quality computer exercises to be made accessible to a very broad pool of students using network infrastructure that is generally available to university instructors, students, and employees of technology oriented companies at little or no cost. This is particularly beneficial in view of the increasing fraction of working part-time students at many institutions for whom frequent campus visits are difficult, but who have access to computing capabilities at home or work.

Though increasing instructional use of the internet is being made at many institutions, the notion of simple "smart servers" on internet to support laboratory-type student exercises seems to be relatively unexplored. The authors welcome comments, recommendations for improvements, and improvements implemented by others.

REFERENCES

[1] T.E. Downing, L.C. Schooley, and E.M. Matz, "Improving instructor/student interaction with electronic mail," *Engineering Education*, vol. 78, pp. 247-50, January 1988.

[2] D.M. Etter, G.C. Orsak, and D.H. Johnson, "A distance learning laboratory design experiment in undergraduate digital signal processing," *Proceedings of the 1995 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, pp. 2885-2888, May 1995.

[3] G.C. Orsak, "Teaching signal processing on the information superhighway," *Proceedings of the Sixth IEEE Digital Signal Processing Workshop*, October 1994.

[4] T.N. Trick, "Educating electrical and computer engineers for the global renaissance," *Engineering Education*, vol. 83, pp. 57-62, 1994.