INTERACTIVE SOLUTION SETS AS COMPONENTS OF FULLY ELECTRONIC SIGNALS AND SYSTEMS COURSEWARE

Brian L. Evans and *Steve X. Gu*

Dept. of Electrical Eng. and Comp. Sciences
211-105 Cory Hall
University of California
Berkeley, CA 94720-1772 USA
{ble,sgu}@ohm.eecs.berkeley.edu

Roberto H. Bamberger

School of Electrical Eng. and Comp. Science
Washington State University
Pullman, WA 99164-2752 USA
bamberg@eecs.wsu.edu

Abstract

The use of computers in teaching signals and systems has evolved. In understanding algorithms, computers were used first as simulation environments and then as real-time implementation environments. In learning the theory of signals and systems, passive tutorial systems have become interactive. Now, software is becoming more and more of an integral part of textbooks and may eventually include the textbook. This paper discusses two interdependent fundamental components of fully electronic courseware— the development of interactive problems and examples, and the supporting software to evaluate student solutions and to explain steps in a correct solution.

1. Introduction

Over the last two decades, many engineering educators have sought to integrate the computer into the daily lives of their undergraduate and graduate students. Not overwhelmed by the issues of resources and student apathy, those engineering educators have struggled with how to balance learning by lecture, computer, and laboratory. The difference in recent years is that many of the entering college students have been exposed to computers since they were of school age. That is, some of them are very comfortable learning concepts through interaction with the computer.

In the context of signals and systems, the four primary areas of current computer use in the curriculum are simulation, implementation, theoretical investigation, and tutorials. One emerging trend, however, is in way in which textbooks and laboratory manuals are being used. Now, software is being bundled as electronic companions to printed textbooks, but in the future, manuals will be bundled as printed companions to fully electronic courseware. This paper focuses on two aspects involved in making the transition to fully electronic courseware— the creation of interactive problems and examples, and the programming of computing environments to explain answers and evaluate student solutions. These two components are interrelated and require creativity.

Section 2 highlights the use of the computer in signals and systems education over the last two decades. Section 3 discusses recent trends in software for signals and systems that can benefit educational courseware. Section 4 introduces our current research in expanding the computer's role in evaluating student answers to questions and in explaining answers to students. The section focuses on our development of an interactive companion to a linear systems textbook [1]. Section 5 concludes the paper.

2. The Role of the Computer

In the context of signals and systems, the four primary areas of computer use in the curriculum are tutorials, theoretical investigation, simulation, and implementation. Of these areas, educators in the early 1980's focused on simulation, by providing either a graphical interface [2] or a collection of library routines [3], as well as "passive" tutorials via computer-aided instruction [4]. Passive tutorials were primarily adaptations of printed textbook material onto the computer screen. The supporting software was generally written in house. By the late 1980's, the use of locally

B. L. Evans and S. X. Gu were supported by the State of California MICRO Program. B. L. Evans can be reached by phone at +1 510 643-6686 and by fax at +1 510 642-2739.

R. H. Bamberger was supported by the NSF under contract MIP-9116683. R. H. Bamberger can be reached by phone at +1 509 335-4053 and by fax at +1 509 335-3818.

1058-6393/95 $4.00 © 1995 IEEE

authored software (e.g. [5]) was becoming prevalent. Commercially available environments were beginning to be used. For example, Matlab [6] was especially attractive because of its Signal Processing and Controls Toolboxes. During this time, implementing algorithms in hardware become more accessible to students with the advent of assemblers, debuggers, and other programming tools for digital signal processing (DSP) boards [7].

In the 1990's, educators have continued to place emphasis on all of the above areas. Passive tutorials [8] are still in use, and programming tools for DSP boards are seeing wider use. The use of the commercial simulation packages Matlab, Monarch, ILS, and Hyperception have overtaken locally authored tools [9]. Matlab, for example, benefits from its toolboxes, its successful student version, and the many signals and systems textbooks and laboratory manuals available for it, e.g. [1, 10, 11]. The next section shifts the focus away from simulation, implementation, and passive tutorial systems and towards recent trends in the use of theoretical investigation tools, interactive tutorial systems, and integrated computing environments.

3. Recent Trends in Signals and Systems Software

Recent trends in the software for signals and systems have included theoretical investigation tools, interactive tutorial systems, and integrated computing environments. Theoretical investigation tools have been primarily used for teaching signals and systems concepts at an mathematical or algebraic level. Interactive tutorial systems have served as front ends for theoretical investigation, simulation, and implementation tools. Integrated environments have been used heavily in the design, simulation, and prototyping of communications and signal processing systems. Properly interconnecting integrated environments with the theoretical investigation tools and interactive tutorial systems is a key in implementing fully electronic, interactive courseware for signals and systems.

3.1. Tools for Theoretical Investigation

Theoretical investigation refers to the design and analysis of signals and systems using algebraic (symbolic) transformations. In terms of users, the two leading symbolic mathematics environments are Maple [12] and Mathematica [13]. Both are being used to teach concepts such as z and Laplace transforms [14] and bilinear transformations.

Although Maple has more knowledge of linear transforms and piecewise functions, more third-party development for signals and systems has occurred for Mathematica. The signal processing packages [15] for Mathematica add a wide variety of signals and systems and implement common algebraic operations on them. The algebraic operations include convolution for discrete and continuous signals as well as the Laplace, Fourier, z, discrete-time Fourier, and discrete Fourier transforms. The convolution routines not only work for sampled signals, but also for signals represented by formulas. The packages can help redesign a given system using simplifications, rearrangements, and number theoretic transformations [16]. In terms of courseware, the key component of the signal processing packages is that they can explain their answers. For example, the linear transform routines can display the intermediate steps, and the convolution routines can animate the flip-and-slide approach to computing the convolution graphically. This justification ability will be discussed further in Section 4.

3.2. Interactive tutorials

Interactive tutorials combine passive information, such as text, tables, and graphs, with interactive examples involving sound, graphics, and animations. Several interactive tutorial systems have been developed for signals and systems. A multimedia self-study system [17] has been developed based on the HyperCard hypertext program by Claris Corp. The material in the course can be navigated sequentially, hierarchically, or randomly. HyperCard preserves links between related topics which the student can follow. To add real-time signal processing functionality, the system can be interfaced to a Motorola DSP board.

Two other interactive tutorial systems have been developed based on Mathematica. Although the Mathematica notebook interface does not support true hypertext because it does not preserve links between related topics, it does however identify keywords that can be followed in a document. The benefit in using Mathematica notebooks is that any given examples also can contain solutions written in Mathematica code that can be altered and evaluated. Both the Digital Signal Processing Learning Environment (DISIPLE) [18] and the Signal Processing Notebooks [19] are based on the Mathematica notebook front end. DISIPLE focuses on numerical processing of music and speech signals, and was written for composers, psychoacousticians, and other people without an engineering background. The Signal Processing Notebooks, which utilize the functionality of the signal processing packages mentioned above, provide interactive tutorials on the

conventional signal processing topics of analog filter design, convolution, discrete-time Fourier transform, and the z-transform.

3.3. Integrated Environments

Integrated environments combine two or more of the following areas: theoretical investigation, graphical layout, simulation, implementation, and tutorial explanations. Developing integrated environments is motivated by the fact that one tool cannot perform all of the specialized tasks in prototyping standalone signal processing systems. Instead, tools working at the same level of abstraction should be able to interchange system descriptions. To support top-down design, tools should also be able to convert their system descriptions to a form suitable for tools working at the next level. The converse should be true to support bottom-up design as well.

Several of the environments listed earlier in this section exhibit interconnectivity of two tools, e.g. HyperCard with a DSP board or the Mathematica interface to the Mathematica kernel. In the ideal classroom or laboratory, all software tools would be integrated with one another. For example, a student could verify the mathematics underlying a design, then convert the algebraic formulas to a graphical layout in blocks, then simulate the algorithm, and finally generate working DSP assembly code. Although this is an example of top-down design, the student should be able to make changes at any part in the exercise and have those changes propagated to higher levels.

Much of the current research effort in integrated systems is in the context of developing environments for the rapid prototyping of signal processors, e.g. Ptolemy [20, 21] and Using these tools, students would layout algorithms as block diagrams and then connect the algorithms together to form a complex system. The tools can simulate a complex system *and* generate equivalent source code and download the code onto DSP boards. At the level of simulation, Ptolemy supports an interface to several specialized simulators. Some simulators, called domains, are released with Ptolemy and provide a wide variety of signal generators, systems, and signal plotters implemented for different computational models. Other simulators include both freely distributable environments (e.g. the Thor register-transfer-level simulator from Stanford University) and commercial software (e.g. Matlab). Now, Ptolemy directly can access toolboxes, custom scripts, and exercises already written in Matlab (e.g. [10]).

4. Making Problems and Examples Interactive

In this section, we will discuss programming strategies to evaluate student answers to design and analysis questions and to explain steps in a solution. We use a symbolic algebra system for our underlying framework because solutions to questions in a first-year signals and systems sequence typically require algebraic manipulation, plotting functions, and small-scale numeric calculations, which fits the capabilities of a symbolic mathematics environment well. Furthermore, the student needs a user interface to the symbolic algebra environment to document steps taken in a solution. The interface should allow the solution set writer to explain the solution in terms of commands for the student to evaluate and and explanation as formatted text and graphics. From among the possible candidates, we chose Mathematica for three reasons. First, its notebook interface allows us to organize problems and their solutions in a natural way. Second, Mathematica supports pattern matching, so we can easily flag and diagnose common errors with very little programming effort. Third, the Signal Processing Packages for Mathematica already define commonly used signals and systems and compute transforms and convolution on sampled sequences and algebraic expressions with explanation.

Based on our experience writing interactive tutorials [19] and self-evaluation exercises [22] for signals and systems, we have created an interactive solution set for the linear systems textbook *Contemporary Linear Systems Using Matlab* by Robert D. Strum and Donald E. Kirk [1]. Although the textbook heavily uses Matlab, we found that Matlab was only applicable to roughly half of the problems in the textbook. Furthermore, students would use Matlab is at the command line by evaluating sequences of commands, and besides the online help information, students would be on their own. We saw an opportunity to program a system like Mathematica to give students immediate feedback for their solution steps and their answers, as well as to bring a symbolic algebra environment to bear on those problems of an algebraic nature.

The companion has taken the form of two introductory notebooks, one reference notebook, and a pair of notebooks per chapter. For each chapter, a problems notebook explains how to solve the example problems in that chapter using Mathematica and provides a workspace for students to try to solve the problems with answer checking. Also for each chapter, a solutions notebook explains the solution to each problem in that chapter and where appropriate, gives a sequence

of Mathematica commands to compute the solution.

Our approach to evaluating student answers to design and analysis questions in the problems notebooks is to define a global function in Mathematica called, **CheckMyAnswer**, which takes the problem name and the student's solution as input and checks the solution against equivalent forms of the answer. If the student's solution matches one the forms of the answer, then the **CheckMyAnswer** routine returns True. Otherwise, the routine returns the appropriate False messages and some hints leading to the correct solution. If the question has more than one correct answer, then the routine will check several possibilities and return False messages if the student's solution does not match any one of the correct answers. To demonstrate the use of **CheckMyAnswer** in an interactive notebook environment, we present the following scenario in which a student tries to use Mathematica functions to attack two linear system questions from [1] and use **CheckMyAnswer** to verify the solutions.

4.1. Example Interaction #1

A student is working on part (e) of Problem 5 in Chapter 10 [1], which is to use circular convolution to find the linear convolution of the sequences $\{2, -1, 4\}$ and $\{3, -2, -1, 1, 5\}$. The correct answer to this question is 6, -7, 12, -5, 5, -1, 20, which can also be found by using our `DiscreteConvolution` function:

```
In[3]:=
  DiscreteConvolution[
     {2, -1, 4}, {3, -2, -1, 1, 5}, n  ]
Out[3]=
  {6, -7, 12, -5, 5, -1, 20}
```

This answer is programmed into **CheckMyAnswer** function for this problem when the problem set is initiated.

To find the circular convolution of two sequences with unequal length, the student must first determine the maximum number of non-zero values for the linear convolution of these two sequences, which is $3+5-1=7$, and then add zeros at the end of both sequences to achieve this length. Assume the student neglects the necessity of zero padding and applies our function `circularConvolution` directly to these two sequences. The function will generate an error message, give hint to help the student apply the function correctly, and return an empty sequence:

```
In[4]:= circularConvolution[
          {2, -1, 4}, {3, -2, 1, 1, 5}  ]
Error: Sequences {2, -1, 4} and
  {3, -2, 1, 1, 5} have unequal length.
Hint: Try applying zeroPadding to both inputs.
Out[4]:= {}
```

By following the instruction, the student would hopefully try to compute the maximum number of non-zero values and then apply zero padding technique. But, the student incorrectly calculates the output sequence length to be $3+5=8$, instead of $3+5-1=7$. So, the student proceeds with following command:

```
In[5]:=
  circularConvolution[
    zeroPadding[ {2, -1, 4}, 8 ],
    zeroPadding[ {3, -2, 1, 1, 5}, 8 ]  ]
Out[5]=
  {6., -7., 16., -7., 13., -1., 20., 0}
```

The student thinks that answer is correct and now asks **CheckMyAnswer** to verify it (note that **%** refers to the previous output):

```
In[6]:= CheckMyAnswer[ "P10.5e", % ]
Out[6]:= False: list exceeded by 1 element(s).
```

At this point, the student would realize that since the list is too long, the maximum number of non-zero values is incorrect. The student would then proceed with the correct number; but this time, after editing the previous **circularConvolution** command by changing 8 to 7 to get

```
In[7]:=
  circularConvolution[
    zeroPadding[ {2, -1, 4}, 7 ],
    zeroPadding[ {3, -2, 1, 1, 5}, 7 ]  ]
Out[7]=
  {6., -7., 16., -7., 13., -1., 20.}
```

Now, **CheckMyAnswer** will return a False message because the student typed 1 instead of -1 for the third element of the second sequence. This typo will manifest itself by **CheckMyAnswer** pointing out that the third, fourth and fifth elements of the solution are incorrect:

```
In[8]:= CheckMyAnswer[ "P10.5e", % ]

False: incorrect list element(s).

Out[8]= {6., -7., False[1], False[2],
         False[3], -1., 20.}
```

After closely examining the input command, the student finds the error and corrects the typographical error to obtain

```
In[9]:=
  circularConvolution[
    zeroPadding[ {2, -1, 4}, 7 ],
```

```
    zeroPadding[ {3, -2, -1, 1, 5}, 7 ]  ]
Out[9]=
  {6., -7., 12., -5., 5., -1., 20.}
```

Now, **CheckMyAnswer** confirms the solution.

```
In[10]:= CheckMyAnswer[ "P10.5e", % ]
Out[10]= True
```

4.2. Example Interaction #2

A student is working on problem P 8.2a in [1], which gives the transfer function of a first-order linear-time-invariant system

$$H(z) = \frac{(1+2z^{-2})}{(1-z^{-1})}, \quad |z| > 1.$$

and asks the student to find the unit sample response h(n).

The answer for the impulse (unit sample) response of this transfer function may take any one of many equivalent forms, depending on how the transfer function is manipulated before it is inverse transformed. Three of the possible forms that are returned by our **ZTransform** function are:

1. direct inversion gives
   $h(n) = 3u[n] - 2\delta[n-1] - 2\delta[n]$
2. expand transfer function and then invert gives
   $h(n) = 2u[n-2] - u[n]$
3. apply partial fractions decomposition and then invert gives
   $h(n) = 3u[n-1] - 2\delta[n-1] + \delta[n]$

We have programmed these forms into **CheckMyAnswer** for this problem.

Suppose the student leaves out the coefficient, 2, in front of the z^{-2} term in the numerator when typing in the command. **CheckMyAnswer** will check the consequently incorrect solution against all possible answers and point out where the errors are

```
In[11]:=
  InverseZTransform[
    ((1 + z^(-2)) / (1 - z^(-1))), z, n,
    RegionOfConvergence -> {1, Infinity} ]
Out[11]=
  2 DiscreteStep[n] - KroneckerDelta[-1 + n] -
    KroneckerDelta[n]

In[12]:=
  CheckMyAnswer[ "P8.2a", % ]

False: incorrect expression.
Here is how close you came to 3 forms
  of the answer, with False representing a
  mismatched term:

Out[12]=
1   DiscreteStep[n] False[1] +
    False[2] KroneckerDelta[-1 + n] +
    False[3] KroneckerDelta[n]

2   False[1]

3   DiscreteStep[False[1]] False[2] +
    False[3] +
    False[4] KroneckerDelta[-1 + n]
```

At any point, students can ask the inverse z-transform routine to explain how it arrived at an answer by enabling a **Justification** option. The transform, convolution, signal analysis routines, signal simplification, some signal plotting routines, and some other routines will justify their answers.

5. Conclusion

This paper focuses on one aspect involved in making the transition to fully electronic courseware— the development of computing environments to explain answers and evaluate student solutions. Our approach in this paper is based on Mathematica, an environment with a kernel to perform symbolic computations and a front end that supports the writing of interactive tutorials. We have taken advantage of both components in writing an interactive companion for a linear systems textbook [1].

Since any current implementation of interactive problems and examples is bound by the sophistication of computer software, we first identify the trends in the computer software. The last five years have seen an increasing number of students learning signals and systems concepts interactively on the computer, through investigation of theory, animated simulation, automated implementation, and multimedia tutorials. Authors have bundled software with more and more textbooks and laboratory books. Ultimately, this trend will lead to the merging of software environment and textbook to form fully electronic courseware.

In developing fully electronic courseware, however, no one computer environment can embody all of the specialized theoretical, simulation, and implementation knowledge of signal processing algorithms. Current research in rapid prototyping of signal processors is producing integrated environments in which software tools at the same level of abstraction can interchange system

descriptions and design information. Rapid prototyping research is also being conducted to support top-down and bottom-up design simultaneously, which is important when a student iteratively designs and tests an algorithm.

Finally, this paper talks about classes of signals and systems problems that can be made fully electronic. That is, the computer can pose the problem, explain the operations the students are applying to the problem, and evaluate student responses. Although only currently suitable for a restricted class of problems that are primarily of a theoretical nature, this sort of approach will be a necessary component in fully electronic courseware. Advances in hypertext and tutoring systems, especially in their diagnosis capabilities and interoperability with other tools, may hold the keys for a more robust framework for the development of fully electronic courseware.

6. References

[1] R. D. Strum and D. E. Kirk, *Contempory Linear Systems Using MATLAB.* Boston, MA: PWS Publishing, 1994.

[2] S. S. Rao, "Interactive computer graphics-based software to support undergraduate signals and systems," in *Proc. Frontiers in Education Conference*, pp. 247–254, Oct. 1984.

[3] H. E. Hanrahan, "An open-ended computer package for signal processing teaching and design," *IEEE Trans. on Education*, vol. 28, pp. 155–163, Aug. 1985.

[4] B. Onaral, D. Tashayyod, and J. M. Trosino, "Interactive computing and graphics in undergraduate digital signal processing," in *Proc. Frontiers in Education Conference*, (Worcester, MA), pp. 468–474, Oct. 1983.

[5] A. Kamas and E. A. Lee, *Digital Signal Processing Experiments.* Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1989.

[6] C. Moler, J. Little, and S. Bangert, *Matlab User's Guide.* Natick, MA: The MathWorks Inc., 1989.

[7] D. L. Jones and T. W. Parks, *A Digital Signal Processing Laboratory Using the TMS32010.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1988.

[8] S. Wood, "Tutorial visualization software for concept reinforcement in digital signal processing education," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. 4, (San Francisco, CA), pp. 77–80, Mar. 1992.

[9] J. D. Mellot and F. J. Taylor, "Signal Processing's education survey results," *IEEE Signal Processing Magazine*, vol. 9, pp. 16–19, Oct. 1992.

[10] C. S. Burrus, J. H. McClellan, A. V. Oppenheim, T. W. Parks, R. W. Schafer, and H. Schüssler, *Computer-Aided Exercises for Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1994.

[11] K. Ogata, *Solving Control Engineering Problems with MATLAB.* Englewood Cliffs, NJ: Prentice-Hall, 1994.

[12] B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, and S. Watt, *Maple Reference Manual.* Waterloo, Canada: WATCOM Publications, 1988.

[13] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer.* Redwood City, CA: Addison-Wesley, 1988.

[14] J. Mathews, "Using computer symbolic algebra in applied engineering mathematics to teach Laplace transforms," *Computers in Education Division of ASEE*, vol. 1, no. 1, pp. 71–76, 1991.

[15] B. L. Evans and J. H. McClellan, "Symbolic analysis of signals and systems," in *Symbolic and Knowledge-Based Signal Processing* (A. Oppenheim and H. Nawab, eds.), pp. 88–141, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1992.

[16] B. L. Evans, R. H. Bamberger, and J. H. McClellan, "Rules for multidimensional multirate structures," *IEEE Trans. on Signal Processing*, vol. 42, pp. 762–771, Apr. 1994.

[17] A. Sekey, "Multimedia self-study courses in dsp and speech processing," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. 4, (San Francisco, CA), pp. 73–76, Mar. 1992.

[18] S. Pointer, J. Wawrzynek, and D. Wessel, "A multimedia digital signal processing tutoring system," in *Proc. Int. Computers in Music Conf.*, (Glasgow, UK), pp. 35–8, Sept. 1990.

[19] B. L. Evans, L. J. Karam, K. A. West, and J. H. McClellan, "Learning signals and systems with Mathematica," *IEEE Trans. on Education*, vol. 36, pp. 72–78, Feb. 1993.

[20] E. A. Lee, "A design lab for statistical signal processing," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. 4, (San Francisco, CA), pp. 81–84, Mar. 1992.

[21] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A platform for heterogeneous simulation and prototyping," in *Proc. of the 1991 European Simulation Conf.*, (Copenhagen, Denmark), July 1991.

[22] B. L. Evans, J. H. McClellan, and H. J. Trussell, "Investigating signal processing theory with Mathematica," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. I, (Minneapolis, MN), pp. 12–15, Apr. 1993.