# 5

# INTERACTIVE SIGNAL PROCESSING DOCUMENTS

• Malcolm Slaney
*Apple Computer, Inc.*

## 5.1 INTRODUCTION

During the last few years, microcomputers have become powerful and inexpensive enough so that every technical person has access to significant computer power on his or her desk.[1] These machines have changed the way many facets of business and research are conducted because users can easily interact with the computer.

Spreadsheets are a common example of how a relatively simple computer program has enhanced personal productivity by allowing users to harness the available computer power. Spreadsheet programs allow the user to develop a model and ask many "what-if" questions, thus gaining a better understanding of the system being modeled. High-level languages and spreadsheets do not make the most efficient use of the machine's computational power, but their speed and ease of use make them valuable to users.

A similar revolution is now possible in the world of signal processing. Recently it has become possible to combine a powerful mathematical program with a word processor and thus create interactive scientific documents. An interactive document includes text and a computer model so that it is easier for readers to understand the

[1] This chapter is an expanded version of [1].

material. These documents can be read like a normal technical paper, but since the document includes computer models, the reader can ask it questions. In this way, the material is learned much more quickly. I believe that interactive documents will eventually change publishing as much as did Gutenberg's invention of movable type. This chapter will describe the important features of an interactive document and why it helps make research and learning more efficient.

An important characteristic of an interactive signal processing document is the ability of the reader to simulate, change, and inquire about properties of the system being described. This chapter describes how a symbolic manipulation program can be used to model a DSP system and present it to a reader in a highly interactive form. Many programs have been written to aid portions of a DSP problem, but symbolic math programs allow a system to be modeled at any number of levels. Symbolic manipulation programs use their mathematical knowledge to automate common mathematical operations such as manipulating polynomials, calculating integrals, and finding limits. The resulting interactive signal processing document, or notebook, can be a very efficient method to teach DSP concepts.

The ideas expressed in this chapter evolved as I wrote an electronic notebook describing a cochlear model developed by R. F. Lyon [2]. This technical report began as a modest notebook for my own use as I wrote a conventional paper. As the work evolved, I realized that the examples in the notebook would also be useful to the reader. The original notebook was not comprehensible to other readers, but by combining the original paper with electronic models and interactive examples, a new type of document was created.

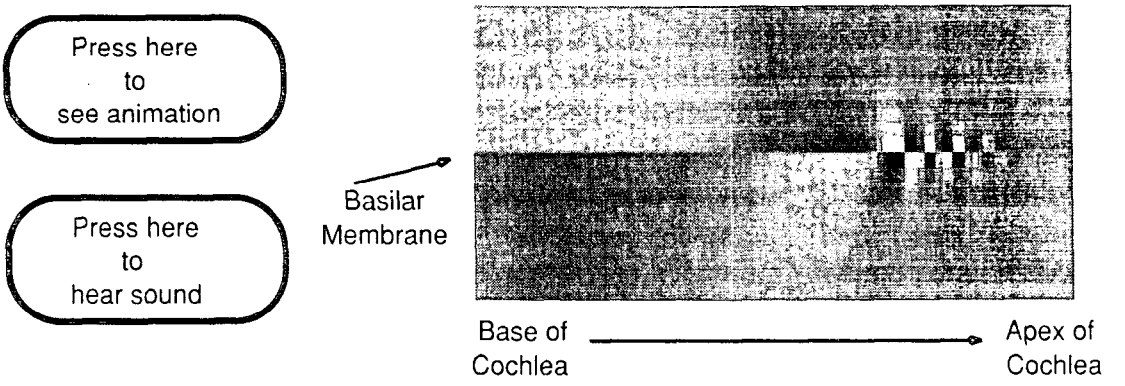
This electronic notebook was created using *Mathematica*. As suggested in Chapter 3, *Mathematica* notebooks exemplify many of the desirable characteristics of a system for research in signal processing. The resulting document is a powerful tool for research and development and an effective tool for teaching. Not only is *Mathematica* an example of a tool for symbolic mathematics, but it includes elements of hypermedia, interactive modeling, and literate programming. The ability to perform symbolic manipulations is important for reasoning about the models, and its other characteristics make it easier for readers to learn the material. How *Mathematica* uses these ideas is discussed in Chapter 3 and section 5.2.

This chapter is not a review of a software product. That function has been ably covered in many articles [3–5]. Instead, *Mathematica* notebooks are used to illustrate many useful features and as a framework to describe additional functionality. Some of the needs of an interactive signal processing document are met by *Mathematica* and related software, while others are not.

Figure 5.1 shows a portion of the type of interactive document this chapter describes. In this case, an animation of wave motion in the cochlea is included as part of the document. The reader can see the animation by clicking on the button with a mouse. Most important, the animation is not just a pretty picture, or even just a pretty animation. The computational model is defined elsewhere in the document, but it is available to the user to change. The reader can modify the parameters of the model and see how the animation changes. If the reader has a

This animation shows a gray scale representation of the pressure in the cochlea due to a single tone at 1000Hz. Darker and lighter regions correspond to pressures above and below the average. The basilar membrane is shown along the horizontal center line.

```
In[42]:
ShowCochlearGrayScaleAnimation[1000];
```




The change in propagation speed means that the long-wave, or one-dimensional, approximation is only valid in the early part of the wave's travel. Near the response peak the wavelength is short enough that energy can flow both down the cochlea and perpendicular to the membrane, so a two-dimensional model is needed.

**Figure 5.1** An electronic notebook combines many features to make it easier for the reader. This example combines text, a mathematical model, graphics, and sound with a user interface that helps guide the reader. One frame of a real *Mathematica* animation computed by R. F. Lyon showing fluid pressure in the cochlea (inner ear) is shown here. The reader can press the button with the mouse and see an animated representation of the fluid flow. The user can modify the frequency of the tone or the parameters of the model and see how the animations change.

favorite model of the middle ear, it can be added to the model. It is important to realize that, except for the buttons, this example is possible with commercially available software. Examples of similar notebooks will be presented throughout this chapter, and I hope it will encourage other researchers to build the necessary software and prepare additional electronic notebooks.

Today, many signal processing problems are easily described using such an interactive scientific notebook. Recent work described in other chapters of this book demonstrates the ability to automatically reason at a high level about a signal processing problem. In addition, personal computers are now powerful enough to implement the digital signal processing algorithms with real data. The problem that remains is, how can authors describing signal processing results take advantage of these technologies to better interact with their readers? This question is the primary focus of this chapter.

There are two concerns in the design of an interactive signal processing document: the content and the form. Certainly the most important part of such a document is the intellectual content. This chapter describes the techniques an author can use to write an interactive document. The second concern, the form of an interactive document, is set by the available tools. This chapter describes the ideal

tool and how the tools available today can be used to describe real signal processing problems.

First, the form of an interactive signal processing document is discussed. Section 5.2 describes some of the important properties of an electronic document and some of the characteristics of an ideal tool for researching and teaching signal processing. It is important that the system have powerful tools for modeling and also that this power is easily available to casual readers. Section 5.3 describes the state of DSP tools today. It describes conventional tools for DSP research and development and explains the use of *Mathematica* to create signal processing notebooks, and is illustrated with two *Mathematica* notebooks. The first example shows features of *Mathematica*; the second notebook shows how *Mathematica* can be used for signal processing research.

Given the tools that are currently available for writing an interactive signal processing document, how should the intellectual content be structured? Section 5.4 presents some of the design issues that must be faced by an author of an electronic document. All writing is difficult, and making it interactive adds another dimension. Some problems become easier to explain, but the additional flexibility can be difficult to manage. Finally, section 5.5 talks about research issues that should be addressed in the future.

## 5.2 PROPERTIES OF AN INTERACTIVE DOCUMENT

Communicating results is an important part of research. While developing the solution to a problem, it is often useful to share the results with colleagues. Later, when the results are polished, the report will be copied and more widely distributed.

An important characteristic of an interactive system for signal processing is that the work can be developed and documented in the same system. Results can be discovered and easily presented to colleagues. If changes are necessary, they can be made quickly without having to transfer the results between a symbolic math system and a word processor. Most important, the reader has the same tools available and can modify or extend the model as desired.

Signal processing is an ideal subject for an interactive document. It is hard to imagine using a computer to teach a carpenter, for example, how to hammer a nail, but computers are a necessary part of most signal processing work. Signal processing researchers already use computers to calculate and simulate algorithms. An interactive signal processing document can provide test data and an evaluation function so students can design a filter and automatically verify that it meets the design goals.

An interactive system is the most effective means to teach and disseminate signal processing results. Much is learned from passive documents, like books, but the learning process is more effective when the reader can ask questions. With interactive documents, readers effectively ask the document questions.

During the last ten years much effort has been expended to create interactive learning environments. The results have been mixed. Perhaps the biggest impedi-

ment to their success has been their goal to create an all-encompassing environment. This goal has brought with it the need for specialized and expensive computer systems, which are only available to a small number of users.

Interactive notebooks are an important intermediate step between paper and fully interactive environments. If written well a notebook can be printed on paper and read without any special technology. But if the reader does have the appropriate hardware and software, he or she will benefit from a much richer interaction.

This section describes the ideal properties of an interactive signal processing document; the current state of the art will be described in section 5.3. A well-designed interactive document will have several features. It will be an example of hypermedia; the user will be able to explore the document, study the sections that are new, and dig deeper into those that are at first not understood. It will be easy to refer to other parts of the document where references are first explained. In addition, computer models will allow the reader to ask questions and more easily integrate the new material with what is already known. Finally, the algorithm must be explained in a way that is easy for both a reader and a computer to understand. This is known as Literate Programming, or the creation of a document that is both a well-written program and a well-written paper.

The features of an interactive signal processing document as previously described will be discussed in the remainder of this section. But, these features are only part of a complete system. A researcher will also need drawing programs to create graphics to explain the work and spelling checkers to help get the descriptions right. These other components of a complete system are vital but are not discussed here.

### 5.2.1 Symbolic Manipulation

Perhaps the most important characteristic of an interactive signal processing document is the ability to reason symbolically about a system or a design. Certainly, many very important signal processing problems have been solved without symbolic manipulation programs, but their use allows much of the drudgery involved in the mathematics to be automated. This makes it easier for the casual reader to verify the results and to explore new ideas.

Many tools are good at numerical math. These tools might be used to calculate the eigenvalues of a matrix or to design a filter with a specified pass-band. The software might be supplied as a subroutine library that is used with a conventional language (such as IMSL [6]) or as a complete environment with a customized language and graphical output (such as MATLAB [7]).

There are many problems, however, where numerical answers do not provide much insight into the solution. Numerically integrating an equation to discover that the maximum transmission rate of a channel is 29 kbits/sec is useful, but an answer in terms of the bandwidth of the system and the antenna gain provides more insight. Symbolic math software allows the user to perform algebraic manipulations, do symbolic calculus, and find the solutions to equations. In addition, symbolic math packages include numerical routines to deal with problems that cannot be done

symbolically. This makes symbolic math packages a superset of numerical software, but often this generality makes strictly numerical calculations slower.[2]

Although symbolic math tools can be used to perform numerical calculations similar to those done by conventional programming tools, their real power is in manipulating the algebraic expressions that describe the system's behavior. An introduction to *Mathematica* as it relates to our goals in this chapter, follows shortly.

A symbolic system allows a filter to be studied in many different forms. For example, an expression that makes clear the poles and zeros of a filter can be expanded into the direct-form polynomial often used in a digital filter. This is how it can be done with *Mathematica* (Note: *Mathematica* uses `In` and `Out` to represent what the user typed and *Mathematica*'s response):

```
In[1]:=Expand[(z - 1/2 - 1/2I)(z - 1/2 + 1/2I)(z + 1/2)]
```

$$\text{Out[1]}= \frac{1}{4} - \frac{z^2}{2} + z^3$$

The filter's characteristics can then be shown graphically in any number of forms. This chapter shows several examples: the magnitude of the frequency response, pole-zero plots, and rubber sheet diagrams of the filter's $z$-domain response.

A symbolic math package allows a user to reason about a system in ways that are not possible with conventional tools. One question that came up when preparing this chapter is: where did the algorithm for designing a second-order digital filter with a given center frequency and bandwidth come from? A reader, unsure about the source of an algorithm in a paper, might want to do a bit of exploration. There will be false starts, but eventually the user should be able to find the correct answer. Note that a symbolic manipulation program does not answer the question automatically, but it does provide some of the necessary mathematical knowledge and expertise to allow the question to be answered.

Symbolic math programs do not normally contain domain-specific knowledge about fields such as DSP, but they can be easily extended. One such knowledge base for use with *Mathematica* is described in Chapter 3; their packages allow *Mathematica* to perform Fourier, Laplace, and $z$-transforms [8]. It is easy to add other packages containing, for example, knowledge about acoustics or speech synthesis.

Other types of common DSP operation that are difficult with symbolic math programs are the algorithm optimization built into ADE [9] and described in Chapter 2, and the morphological algorithm manipulation described by Richardson in Chapter 4. ADE allows a user to describe a signal processing algorithm in such a way that ADE can reason about the design, try alternate implementations of the algorithm, and show an equivalent design that is more efficient. Symbolic math programs will often rearrange an equation to put it in a standard form for display

[2] On the other hand, a symbolic package can evaluate a definite integral by first performing symbolic integration and then substituting the numerical limits. This might be faster than numerically evaluating the integrand and summing the results.

to the user, but the standard form is defined more by mathematical convention than by computational complexity. Thus, a calculation might produce the result

$$ax + b + cx^2$$

which will be displayed to the user as

$$cx^2 + ax + b$$

yet a potentially more efficient computational form is

$$(cx + a)x + b$$

As far as I know, there is no symbolic math program that allows users to specify the relative costs of different representations, let alone suggest the changes that might be made to optimize the algorithm as is done in ADE.

### 5.2.2 Hypermedia

At one time, information was passed from generation to generation via the story teller. This was inherently an interactive process, but it also limited the speed at which information could be conveyed. By the middle ages, publishing was well established. This increased the rate at which information could be disseminated, but it lost its interactive nature.

Most papers and books are designed to be read in a linear fashion. (Dictionaries and encyclopedias are examples of works that are meant to be accessed randomly.) The order of presentation is determined by the author, and the reader is expected to make the best of it. Some browsing is possible, but the paper medium makes this inconvenient. The reader is a passive part of the learning process.

Readers have questions to be answered. Often readers do not have the same goals as the original author and might want to skip around in the material. Sometimes they will look in the index for the subject in which they are interested and then work backwards until they understand enough to solve their problem. This sometimes involves just paging back through a few pages of material, but at other times the necessary preliminary information spans many chapters of the book, often with intervening material the reader does not need to understand to solve the problem at hand.

Hypermedia is often described as a solution to this problem [10]. Using a computer, the reader can browse through a document and then quickly move to other sections based on what is read. Thus, if a reader encounters a new concept, it might be accompanied by a button on the screen that will display more detailed information. In this way, the reader can fashion his or her own path through the material.

Hypermedia has many forms, but they all represent enhancements of the paper world. There are four hypermedia features of an electronic notebook that will be considered here. The simplest is using multiple media: sound, animations, and other ways of presenting information that are hard to print on paper. Second, the notebook

can be easily changed by the reader, to emphasize the points that are more interesting, or to add additional notes. Third, the most traditional form of hypermedia is represented by links, or the ability to move quickly from one topic to another in the paper. Finally, an electronic notebook has many functional links based on the mathematical definitions and algorithms in the paper. Each of these ideas will be considered in turn.

### Multimedia

Hypermedia can mean the use of more than one type of media in a document. There are many examples of papers that would be much more effective by adding audio and visual demonstrations. Why should a paper on sound perception not include audio examples so readers can make their own judgments about what is heard? A paper on speech coding should have audio examples so the results are meaningful to readers not familiar with intelligibility scores. A paper on acoustics is more readable with simple animations showing the propagation modes. A discussion of video compression algorithms should include a sequence of images so the reader can modify the algorithms and see how their changes affect performance.

### Modifiable

An electronic notebook can be easily changed. Users can rearrange the report to make the presentation more natural for their background. In addition, users can add their own material. Since there does not have to be any difference in appearance between the original text and the "margin" notes, the document becomes personalized for the reader.

### Links

The most conventional form of hypermedia allows the reader to browse through a document in any desired order. In its ideal form, the reader should be able to follow ideas anywhere they might go within a hypermedia document. But this leads to a multidimensional web of links that is difficult to organize on paper.

Instead, an electronic notebook has a hierarchical organization which can then be flattened when rendered on paper. Text, equations, output, and graphs are grouped into sections, and the entire paper organized into a hierarchy. Most papers have a tree-like structure, but the electronic version of a paper's most detailed sections can be hidden from the user so as to make the presentation easier to follow. These hidden sections can be easily opened by the user and can be used to hide details of the presentation, attempts that did not work, or test code that is not strictly necessary for the presentation.

Most writing efforts include unpublished examples that were used to test and refine the material in the paper. This material is probably not interesting to most readers, but in some cases it is. Inquisitive readers might want to know where a

derivation came from or would like to double check a result. For instance, an interactive signal processing document that I wrote describing the classic algorithms for filter design includes many examples that were used to test the algorithms in the paper. Some of these examples are interesting to the casual reader and are shown in the main body of the paper. Other examples were used to explore more difficult cases or to compare my solution to published designs. These examples are hidden, yet still available to the interested reader. The purpose of writing is to communicate a result to the reader. Allowing the reader to peek behind the scenes, so to speak, is beneficial to the communication process.

On top of this hierarchy, a separate set of links is represented by the functions and algorithms that are defined. This set of links might take the form of a help facility to make it easier for the user to understand the paper. Since the help facility includes information about any function defined in the paper, it is a very simple form of hypermedia. A user can select any function name in the paper, select help, and read a short description of the function. When more information is needed about a function, the system can take the reader to the point in the paper where the function is first defined.

#### Smart Links

The links defined by the functions in an interactive notebook are not just navigational aids. Instead, these functions have mathematical meaning, and their links often include dependencies on other mathematical definitions. For example, my report describing a cochlea model includes a relatively simple model of the outer and middle ears. This model is used when showing the neural firings due to a particular sound. But if a reader has a better model, it can be substituted in the report and the new graphs can be computed.

Thus, a scientific notebook extends the hypermedia concept because a function is not just a collection of symbols, but, more important, has a mathematical meaning. Using a function in a notebook not only implies a link back to the original definition, but its usage implies a specific mathematical operation. These "smart links" are an important part of an interactive signal processing notebook.

### 5.2.3 Interactive Models

An interactive signal processing document extends the hypermedia concept by casting each of the new results as an equation or computer model with which the user can interact. Since the system includes a computation engine, readers can change the model and see the effect. The results are shown graphically. By controlling the parameters of the model or system, the user can gain a better understanding of how it works.

This book describes several tools for signal processing research and development. Just as an interactive system for signal processing is a useful research tool, such a system can also be valuable to a reader trying to understand the results. A good

instructor or piece of writing should guide the student to the same conclusions that were reached by the research. Hopefully, the learning process will be more efficient than the original research, but the same tools are useful in both cases.

An interactive signal processing document contains equations and computer models that the reader can manipulate. In some cases, the reader will be content to change the parameters of a model and see how the results change. Other readers might want to study the model from a different angle. Perhaps due to a different background, the reader will want to analyze the system response in the time domain instead of a frequency domain approach more natural to the original author. With a symbolic math package, readers can apply the appropriate transformation and study the result in their preferred domain.

Users should be able to interact with a signal processing document in two ways. First, a mathematical model can be modified to extend it into the reader's own domain. For example, a reader of a paper on reconstruction theory might want to try the algorithm using data from his or her own research problems. This makes the solution described in the notebook more realistic to the reader.

A second, more important, aid to learning is direct manipulation. In many systems, the behavior is controlled by a numerical parameter. A paper describing such a system will probably include an equation describing the system's behavior as a function of this parameter. But the user would have a much better feel for the behavior of the system if there was a knob (or slider) that could be manipulated with a mouse and would immediately vary the system's output. A simple example of this behavior in an interactive signal processing document is shown in Figure 5.2.

A prototype of a system like this is distributed with version 1.0 of the NeXT workstation software. Unfortunately, many problems cannot be recomputed at rates fast enough to be interactive. Designing a simulation that can be solved with easily accessible hardware is a problem that is addressed in section 5.4.

Most symbolic math packages are interactive. An electronic notebook is unique, though, because the writer can guide the reader by suggesting areas to

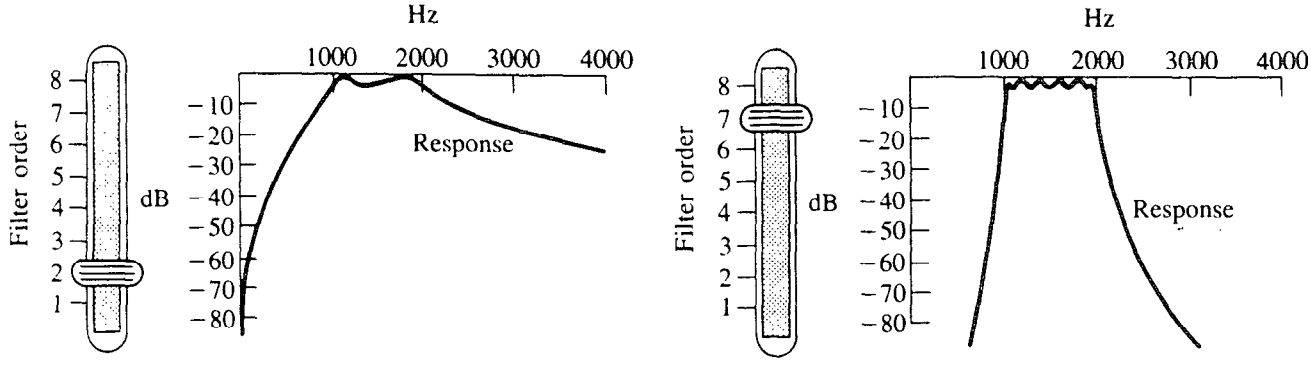



**Figure 5.2** Animated Figures. Two samples are shown of a proposed scheme for allowing the reader to modify a figure in an interactive signal processing document. As the reader uses a mouse to change the position of the slider, the order of the band-pass filter is changed and a new response curve is calculated.

explore. The notebook should encourage the reader to try new ideas. The symbolic math package can show the result as an equation, a figure, or even an animation.

It is unfortunate that a *Mathematica* notebook does not support direct graphical manipulation like that shown in Figure 5.2. Currently, notebooks have to be carefully written so that the reader does not have to understand much of *Mathematica* to know what parts of an expression to change. A graphical control, such as a slider, would make it possible for users, both novice and experienced users, to directly control an electronic notebook in a very intuitive fashion. Even the ability in a notebook to place a mousable button that would perform some action would be useful. An example of this capability is shown in Figure 5.1.

### 5.2.4 Literate Programming

Documenting a computer model or a signal processing algorithm is a useful way to transfer knowledge about a new result. Conventionally, this has been done using a simplified form of a language like Algol or Fortran. It is difficult within the constraints of these languages to eloquently express the ideas that went into the model.

An alternate style of expressing an algorithm is known as Literate Programming. Denning's introduction to Van Wyk's column on *Literate Programming* says [11]:

> A literate program contains not only the needed statements in a programming language, but also a precise problem statement, a summary of trade-offs between the running time and space, or between running time and programming time, and suggestions on how to modify the program. Program code segments are inserted in the text at points logical to the intellectual development of the algorithm. A literate program pays careful attention to lucidity of presentation and presents all arguments needed to understand why the program will actually work as intended.

The combination of an interactive computer model and literate programming is a very powerful tool for learning. Before a reader can intelligently change a model, the description must be read and understood. Expecting the reader to understand the program from just the embedded comments is not very practical. Instead, a literate program should include graphics, examples, and even interactive controls. All these techniques will help the reader to better understand the results.

Changing the focus of the programming effort can have a great benefit. A computer language is designed to make it easy to implement algorithms, not to explain material to another reader. Comments and sometimes even pictures are added to a program to describe an algorithm, but the text of the program is still one-dimensional. In a literate program the primary goal should be to describe an algorithm for a reader, but to do it in such a way that the computer can also understand it. This is especially true when describing highly mathematical material such as signal processing algorithms.

## 5.3 DSP TOOLS

Unfortunately, there is no software that possesses all the features needed to build the interactive signal processing document described in section 5.2. Most conventional tools for signal processing are libraries or environments that are designed for programmers. Although these tools are powerful, they are difficult for casual readers to use.

There are many programs that can perform symbolic manipulations, and one of them, *Mathematica*, has an electronic notebook interface. *Mathematica* and its notebooks come closest to the ideal interactive signal processing environment described here. Using a symbolic math program to do signal processing problems is not new (MACSYMA, the grandfather of all symbolic math programs, has built-in support for Fourier and Laplace transforms), but adding a notebook interface allows an author to make the DSP knowledge more accessible to a reader.

It is not possible to describe all the tools designed to help solve DSP problems. This section describes several of the conventional tools and then describes the use of *Mathematica* to research and teach signal processing ideas. The purpose of this chapter is to discuss the use of a symbolic manipulation program to describe in an interactive manner the solution of a signal processing problem. To provide some context, several conventional tools will first be reviewed.

### 5.3.1 Conventional Tools

Perhaps the most commonly used tools for signal processing are subroutine libraries. The two best-known libraries for signal processing are IMSL [6] and the IEEE Signal Processing Library [12]. These libraries include code to perform many common signal processing operations in a user's program. The user must still do much of the programming, but the difficult numerical work is handled by these subroutine libraries. Much research has gone into these algorithms and they represent a significant step in the use of structured programming techniques. The routines in these libraries, especially in IMSL, are highly optimized, and their numerical stability is well documented. Most of the work required to use these libraries consists of reading in the data and putting it in the proper form for the appropriate subroutine.

Subroutine libraries eventually led to complete programming environments for signal processing. SRL (the Signal Representation Language) [13] and SPLICE [14, 15] are two examples of specialized programming environments for signal processing. These systems are built on top of the Lisp programming language, and use object-oriented techniques to make it easy for researchers to extend the environment. Two extensions for computing sine wave signals in SRL are shown in Figure 5.3.

Systems such as SRL and SPLICE are very powerful, but their nonstandard programming methodology (object-oriented Lisp) requires much effort to learn. In a sense, these tools are programmer friendly but not necessarily user friendly. This has limited their success.

```
(defsigtype sine-wave-signal-type
  :a-kind-of basic-signal-type
  :parameters (ncycles length phase)
  :finder signal-sine-wave
  :init (setq-my dimensions (list length))
  :fetch ((i) (sin(*3.141592 2.0 ncycles (/ i length )))))

(defsigtype sine-wave-with-zero-phase-signal-type
  :a-kind-of sine-wave-signal-type
  :parameters (ncycles length)
  :finder signal-sine-wave-with-zero-phase
  :init (setq-my phase 0.0))
```

**Figure 5.3** Two examples of signal definitions in Kopec's Signal Representation Language (SRL). The first example, `sine-wave-signal-type`, defines the scheme to calculate a sine wave with `ncycles` in `length` samples. The second example, `sine-wave-with-zero-phase-signal-type`, further refines this signal type to include a default `phase` of 0 degrees. The rest of the behavior of the `sine-wave-with-zero-phase` is inherited from the signal's parents, `sine-wave-signal-type` and `basic-signal-type`.

As the number of people wanting to solve signal processing problems has grown, the need for systems that do not require any programming has also increased. These environments include a large number of specialized signal processing algorithms that can be applied in a cookbook fashion to solve a problem. With a high-level tool the language becomes more specialized, making it easier to express some algorithms. A drawback of such specialization is that concepts that do not fit within the tool's model are much harder to program.

*MATLAB* is another example of a specialized signal processing environment [7]. *MATLAB* includes a large number of routines for linear algebra and signal processing which can be used interactively or combined by the user into new functions. *MATLAB* also includes functions to import data and to plot the results. A large number of common signal processing operations are part of *MATLAB*'s libraries, and it is easy to add new functions. *MATLAB* provides a simple command line interface for the user and a single window for graphics.

An even more specialized tool that can be used for image processing is Adobe's *Photoshop* [16]. *Photoshop* is designed to make it easy to perform many common operations on images and see the results immediately. This tool is very powerful but is not programmable.

Other tools have been designed to deal with specific signal processing problems. For example, there are tools for speech analysis, all sorts of filter designs, and even VLSI layout for signal processing algorithms. The price paid for this power is the limited domain. For example, it would be difficult to design a filter using one of the filter design programs and then use the results in one of the speech analysis tools.

### 5.3.2 About *Mathematica*

*Mathematica* is an example of a system for creating interactive scientific documents [17]. It is first and foremost a program for doing mathematics on a computer. The program allows a user to pose both symbolic and numerical mathematics questions. Thus, a user can symbolically integrate an expression, and then find its numerical solution over any domain. *Mathematica* includes a programming language to allow more complicated models to be described and graphical functions to allow the user to visualize the results more easily. When mathematical definitions, graphics, and words are all combined, the resulting document is called a notebook. *Mathematica* is available for most of the popular scientific and personal computers, but the notebook feature is only currently available on the Apple Macintosh and the NeXT Machine. Notebook support for other machines has been promised.

Like its predecessors (MACSYMA, SMP, Maple, Reduce, etc.), *Mathematica* includes many facilities for doing symbolic mathematics and numerical calculations. The fact that these systems can easily work with polynomial equations makes them useful, for example, when designing filters. A family of filters can be designed and then analyzed for their behavior at DC. This section will talk about some of the features of *Mathematica* and how it can be used to create an interactive signal processing document.

The *Mathematica* system is divided into two halves. The user interacts with a front end while a back-end kernel provides the computational engine. The front end is unique for each type of machine, defines the behavior due to typed commands and mouse actions, and provides an interface to the host's window system. The back end is relatively machine independent and does all the calculations.

One useful feature of *Mathematica* is that the user interface (front end) and the kernel (back end) do not have to be on the same machine. Thus, a user can interact with a relatively inexpensive graphics machine on the desktop, while a more powerful shared back-end machine does all the calculations. The communications between the front end and the back end can be carried out over a serial line or a network connection.

On some machines, the *Mathematica* front end allows the user to create what is called a notebook. A *Mathematica* notebook is much like a scientist's notebook since it can contain data and thoughts about work in progress. However, these notebooks are unique in that they also contain computer models and even animations. The ability to create a live mathematical notebook is probably the feature that most distinguishes *Mathematica* from other systems.

Notebooks contain text, equations, and graphics. When a problem is first proposed, the notebook will reflect the steps actually used to carry out the calculation. It might contain false starts and notes understood only by the author. As the theory and calculations are refined, the notebook becomes more polished. Eventually, the notebook is cleaned up and the necessary text written so it can be distributed to colleagues. The finished notebook will contain explanatory text, symbolic and numerical models, and graphs and animations to explain the system and its solution.

Figures 5.4 and 5.5, my own technical report [2], and a recent book by Gray [18] are examples of notebooks in their polished form.

A notebook showing some of the capabilities of *Mathematica* is shown in Figure 5.4. *Mathematica* can be used as a calculator, even with arbitrary precision, but its real power comes from the symbolic functions. Equations can be integrated and differentiated, and algebraic manipulations can be performed to put the result into a simpler form. The results can be plotted to help understand how the system works. A second, more complete example showing the use of *Mathematica* in a signal processing application is shown in Figure 5.5.

The signal processing example in Figure 5.5 uses a special data structure to represent each filter. Ratios of polynomials are supported by *Mathematica*, but then any filter operation that needed the location of the poles and zeros would have to factor a polynomial with floating point coefficients. This can be done but is prone to errors and is time consuming.

Instead of representing filters as ratios of polynomials, the filter design functions shown here use a special structure that contains the gain, zero locations, and pole locations. This structure is called a GZP (Gain, Zeros, and Poles). The GZP structure is represented in *Mathematica* as a three-element list with the zeros and poles each being represented as a list of points in the complex plane. Choosing the appropriate data structure to model a system is important both for computational efficiency and for literate programming. One advantage a symbolic environment has over a purely numeric system such as *MATLAB* is that arrays and lists can be combined in arbitrary ways to represent the data at hand.

Both the GZP and ratio of polynomial representations of filters have their advantages. The GZP is used in Figure 5.5 because it is more accurate (roots of polynomials are already known) and it is easy to transform a filter from the GZP form to a ratio of polynomials. Consider the following filter design. The call to `ChebychevLp` returns an eighth-order filter with a 1 dB pass-band ripple. The gain of this filter has been adjusted so that it has unity gain at DC. Here, the GZP structure returned by the `ChebychevLp` function, the Laplace domain representation, and finally an expanded version are shown.

```
In[1]:=
      ChebychevLp[8,1]

Out[1]=
        .01720755
      {-----------, {} .
         1/20
        10
      {-0.0350082 - 0.996451 I, -0.099695 - 0.844751 I,
       -0.149204 - 0.564444 I, -0.175998 - 0.198206 I,
       -0.175998 + 0.198206 I, -0.149204 + 0.564444 I,
       -0.099695 + 0.844751 I, -0.0350082 + 0.996451 I}}
```

### *Mathematica* Introduction

This notebook is a short introduction to the features of *Mathematica*. In the two notebooks accompanying this chapter, the *Mathematica* input is shown in a **bold face courier font** and the *Mathematica* output is shown in a normal Courier font. See Wolfram's *Mathematica* book for more examples.

### Numerical Calculations

*Mathematica* can be used like a calculator to do numerical arithmetic. Here is Pi calculated to 50 decimal places.

```
N [Pi, 50]
```

```
3.14159265358979323846264338327950288419716
        93993751
```

Or *Mathematica* can be used to numerically integrate an expression which can't be integrated symbolically.

```
NIntegrate [Sin [Sin [x]], {x, 0, 1.0}]
```

```
0.43060610312069060 45
```

### Polynomial Manipulation

Manipulating algebraic expressions is easy for *Mathematica*. Here are some examples. First we multiply out the terms of an expression.

```
Expand [(x + 1) (x + 2) (x + 3) ^3 (x + 4)]
```

$$216 + 594\,x + 639\,x^2 + 350\,x^3 + 104\,x^4 + 16\,x^5 + x^6$$

We can then factor this equation to find the original expression.

```
Factor [216 + 594*x + 639*x^2 + 350*x^3 +
104*x^4 + 16*x^5 + x^6]
```

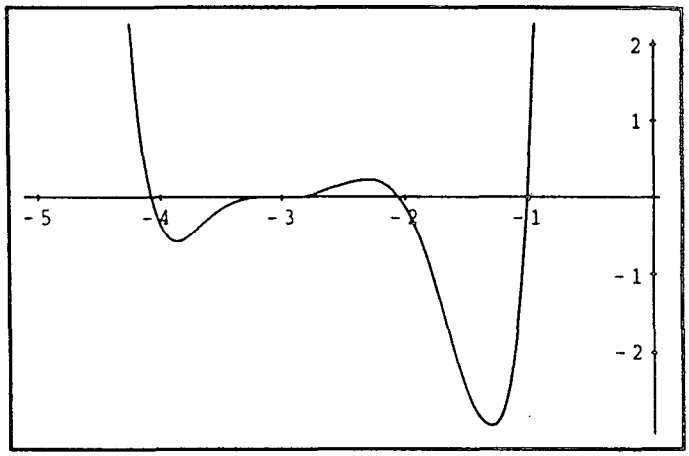
$$(1 + x)\,(2 + x)\,(3 + x)^3\,(4 + x)$$

```
Solve [216 + 594*x + 639*x^2 + 350*x^3 +
104*x^4 + 16*x^5 + x^6 = = 0, x]
```

```
{{x -> -1}, {x -> -2}, {x -> -4},
{x -> -3},{x -> -3}, {x -> -3}}
```

Here is a graph showing the behavior of this function as $x$ varies between −5 and 0.

```
Plot[(x + 1)(x + 2)(x + 3)^3(x + 4),{x,-5,0}];
```




### Calculus

*Mathematica* knows a lot about calculus. After reading in *Mathematica*'s integration rules, we can easily find the integral of $x/(1-x^3)$.

```
<<IntegralTables.m;
Integrate [x/(1-x^3), x]
```

$$-\frac{\sqrt{3}\,\text{ArcTan}\left[\frac{1+2x}{\sqrt{3}}\right]}{3} - \frac{\text{Log}[1-x]}{3} + \frac{\text{Log}[1+x+x^2]}{6}$$

Now let's differentiate this result and see if we get the original expression.

```
Simplify [D [- (3^ (1/2) *ArcTan [(1 +
      2*x) / 3^ (1/2)]/
      3 - Log [1 - x]/3 +
      Log [1 + x + x^2]/6, x ]]
```

$$\frac{x}{1-x^3}$$

We can also find the series expansion of an expression.

```
Series [Exp [x] Cos [4x], {x, 0, 6} ]
```

$$1 + x - \frac{15\,x^2}{2} - \frac{47\,x^3}{6} + \frac{161\,x^4}{24} + \frac{1121\,x^5}{120} - \frac{11\,x^6}{16} + O[x]^7$$

**Figure 5.4** An example of a *Mathematica* notebook showing elementary numerical, symbolic manipulation, graphing, analysis, and programming features.

### Solving Equations

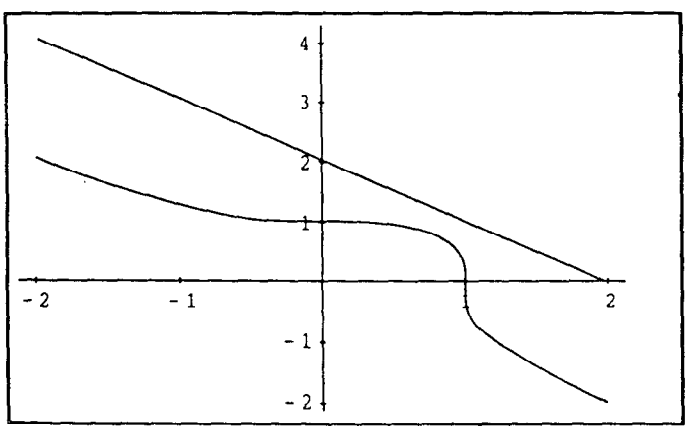
*Mathematica* can be used to solve simultaneous equations. Here is a simple example.

```
Solve [{x^3 + y^3 == 1, x + y == 2},{x, y}]

          6 - Sqrt [-6]
{ {x -> ---------------- ,
                6

          12 + 2 Sqrt [-6]
    y -> ----------------- } ,
                12

          6 + Sqrt [-6]
    {x -> ---------------- ,
                6

          12 - 2 Sqrt [-6]
    y -> ----------------- }}
                12
```

### Graphics

*Mathematica* can graphically show you the results of your calculations. Here is a plot showing the previous two equations. Note: there are no solutions for real values of $x$ and $y$.

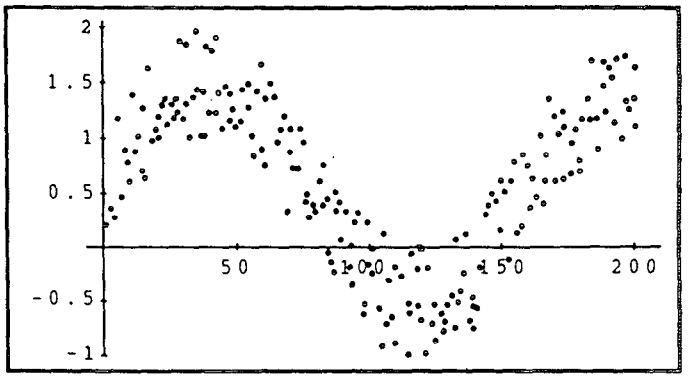
```
Plot [{(1 - x^3) ^ (1/3), 2 - x},
        {x, -2, 2}]
```




### Data Analysis

*Mathematica* can be used to analyze the results of your experiments. Let's create a sample data set by adding random noise (uniform between 0 and 1) to a sine wave.

```
data = Table [N [Sin [1/25] +
       Random []] , {1, 200}];

ListPlot [data];
```




Now let's fit the data to a constant term (to get the mean of the random variable) and a sine and cosine of the appropriate frequency. Note that the term multiplying the cosine in the result is small compared to the factor multiplying the sine.

```
Fit [data,{1, Sin [x/25], Cos [x/25]},x]

                                    x
0.499471 - 0.0565294 Cos [--] +
                                   25

                  x
  0.944494 Sin [--]
                 25
```

### Programming by Example

Rules can be added to *Mathematica* to specialize it for your own problem domain. Here is an alternate definition of factorial. The first rule defines the stop condition. The second rule is the basic recursion to solve the problem.

```
Fact [0] = 1
Fact [x_] := x Fact [x - 1]
Fact [6]
720
```

Here is an example of defining rules for simplifying logarithms in *Mathematica*.

```
log [a_ b_] := log [a] log [b]
log [x y^2 z]
log[x] log[y^2] log[z]
```

Now we can tell *Mathematica* about powers

```
log [x_^n_] := n log [x]
log [x y^2 z]
2 log[x] log [y] log [z]
log [x/y]
-(log[x] log[y])
```

**Figure 5.4** (continued)

### Signal Processing Example

This notebook is an example of using *Mathematica* to describe continuous filter design. This notebook is fully functional. It includes some *Mathematica* notation, but readers who are not *Mathematica* users should have no problem understanding the notebook by just reading the text. See [Wolfram88] for an explanation of the special notation used by *Mathematica*. Thanks to Ray DeCarlo at Purdue University for providing the original motivation to write this notebook.

There are a number of design techniques for high-order filter design. This notebook will show how to design a Chebychev low-pass filter, and then how to transform the original lowpass poles into a bandpass filter. Section 1 of this example defines a number of functions used to work with filter polynomials. Section 2 describes the techniques to design Chebychev lowpass filters with a corner frequency of 1 radian per second (rps), and Section 3 shows how to transform these generic lowpass filters into bandpass filters with arbitrary passbands.

### 1 Continuous Filter Functions

Continuous-time filters are described using polynomials of complex frequency s. A filter's response function is evaluated along the imaginary axis by making the substitution **s->I w** (or jω in conventional EE notation.) The following function is used to evaluate the complex response of a filter radians. Additional functions compute the gain, magnitude, and phase response of the filter. The expression **filter** can be an arbitrary function of the complex frequency s.

```
FilterGain[filter_, w_] :=
    ReplaceAll[filter, s -> I w];

FilterMag[filter_, w_] :=
    Abs[FilterGain[filter,w]]

FilterPhase[filter_, w_] :=
    Arg[FilterGain[filter,w]]

FilterDb[filter_, w_] :=
    20  Log[10,FilterMag[filter,w]]
```

The following function is used to display the frequency response of a continuous filter. (The plot starts at 0.01 Hz to avoid any problems with filters that have a zero at DC.)

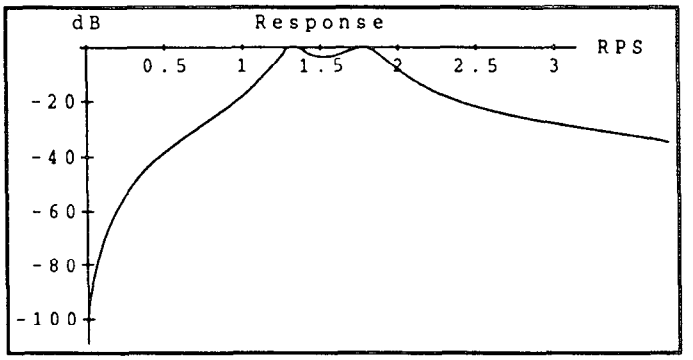
```
FreqResponse[filter_, maxf_,
      opts_:{}] :=
  Block[{response},
    response = N[FilterDb[filter, 2 Pi f]];
    Plot[response,{f,.01,maxf},
      AxesLabel->{" Hz","dB"},
      PlotLabel->"Response",
      opts]];
```

We define a similar function for displaying the frequency response of a filter as a function of radian frequency (ω or radians per second, rps).

```
FreqResponseRadians[filter_, maxw_,
      opts_:{}] :=
  Block[{response},
    response =
N[FilterDb[filter,w]];
      Plot [response,{w,.01,maxw},
      AxesLabel->{" RPS", "dB"},
       PlotLabel->"Response",
      opts]];
```

Note that for each of these functions there is a third optional argument that allows additional options to be set. We use this feature to pass special parameters to the **Plot** function. The frequency response of a fourth-order filter is shown below.

```
FreqResponseRadians[.197 s^2/
((0.09 - 1.3I + s)(0.09 + 1.3I + s)
  (.12-1.8I + s) (.12 + 1.8I + s)),4];
```




The **AdjustGain** function is used to modify a filter so that it has unity gain at any desired frequency.

```
AdjustGain[filter_,f_] :=
     filter/FilterMag[filter,f]
```

Higher order filters could be designed with *Mathematica* using either rational polynomials or lists of poles and zeros. Rational polynomials would be nice because all intermediate results would look like filters. Unfortunately, we sometimes need to talk about individual poles and zeros, for example when doing partial-fraction expansions. This is difficult if the filter is described as a polynomial. If a filter is described by its poles, zeros, and gain, we can always regenerate the polynomial.

A list of polynomial roots is turned into a polynomial in s using this *Mathematica* expression.

```
PolynomialFromRoots[roots_] :=
  If[Length[roots] == 0,
    1,
    First[Apply[Times,
      Map[(s-#)&,roots]]]]
PolynomialFromRoots[{4,2,1}]
```

```
(-4 + s) (-2 + s) (-1 + s)
```

**Figure 5.5** A sample notebook shows the use of *Mathematica* to design and document a filter design paper.

In this notebook we use a list to keep track of the zeros, poles, and gain of a filter. Functions that transform filters will take as input a list of these three items and return a similar structure. We abbreviate the name of this structure to just GZP (Gain, Zeros, and Poles). The following function is then used to take one of these lists and transform it into a filter in the s domain. Note that we have used the pattern matching facilities of *Mathematica* to pick out the three elements of the input list.

```
FilterFromGZP[{gain_, zeros_,
        poles_}] :=
    gain*PolynomialFromRoots[zeros]/
        PolynomialFromRoots[poles]//N
FilterFromGZP[{2.4, {4,2,1},
                {12,10,7}}]
```

```
 2.4 (-4. + s) (-2.+ s) (-1. + s)
---------------------------------
  (-12. + s) (-10. + s) (-7. + s)
```

## 2 Chebychev Filters

The simplest high-order filters to design are the Butterworth and the Chebychev. The poles of a Butterworth low-pass filter are arrayed so that the filter's response is flat through most of its passband. As the frequency approaches the corner frequency, the gain quickly falls off. In some cases this characteristic is an advantage because the gain between DC and the corner frequency is nearly flat.

For a given stopband or transition band specification, filters with a much smaller variation in gain in the passband can be designed using the Chebychev polynomials. Chebychev filters do not have a flat response in the passband, but, as in Butterworth filters, the passband error can be made arbitrarily small.

The poles of the Chebychev polynomials are given by the following expression [Daryanani76]. This expression is a function of the desired order of the polynomial (**n**) and the maximum error (**amax**) in dB in the passband.

```
ChebychevPoles[n_,amax_] :=
  Block [{e},
    e = Sqrt[10^(amax/10)-1];
    Table[Sin[Pi/2(1 + 2k)/n]
        Sinh[ArcSinh[1/e]/n] +
      I Cos[Pi/2(1 + 2k)/n]
        Cosh[ArcSinh[1/e]/n],
        {k,n,2n - 1}]]
```

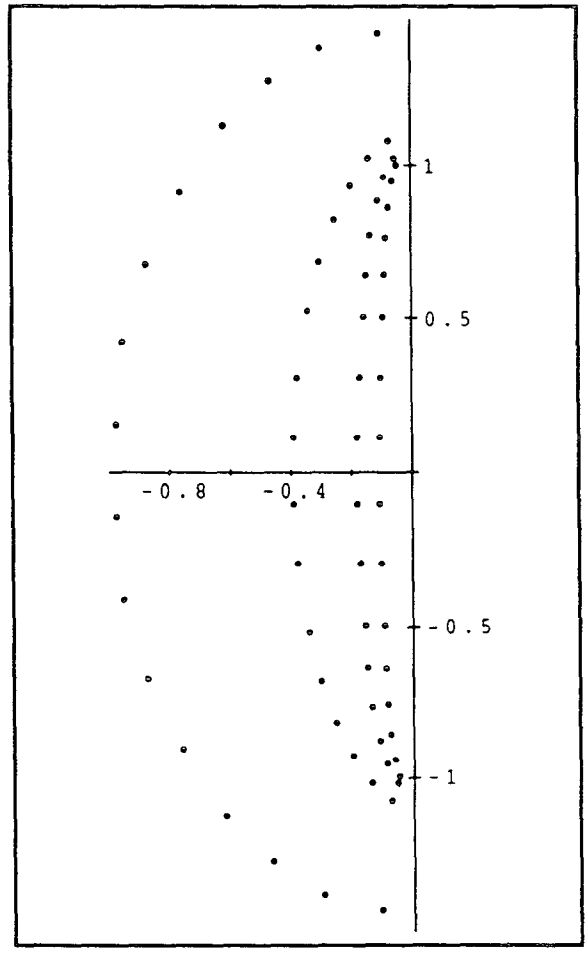
The **ChebychevPoles** function returns the location of the poles of a **n**-th order low-pass Chebychev filter with a cutoff frequency of 1 rps and a maximum pass-band error of **amax** dB.

```
ChebychevPoles[6,2]//N
```

```
{-0.0469732 - 0.981705 I,
   -0.128333 - 0.718658 I,
   -0.175306 - 0.263047 I,
   -0.175306 + 0.263047 I,
   -0.128333 + 0.718658 I,
   -0.0469732 + 0.981705 I}
```

As can be seen from the pole plot below, the roots of a Chebychev polynomial fall on an ellipse. This plot shows the roots as the maximum error in the passband is varied from $10^{-10}$ (the ones that look most like a circle) to a passband error of 1 dB (the rightmost arc).

```
PlotPoles[Flatten[Map[N[
    ChebychevPoles[16,#]]&,
    (10^-10,10^-4,.1,1}]]];
```




The next function computes a Chebychev low-pass filter and returns a list with the gain, zeros, and poles. Note that a Chebychev low-pass filter has only poles so the list of zeros is empty. The resulting GZP list can be passed to the filter transform routines to realize other types of filters (band-pass, band-reject, and high-pass). In this filter design function the gain at the corner frequency (1 rps) is adjusted so that it has a

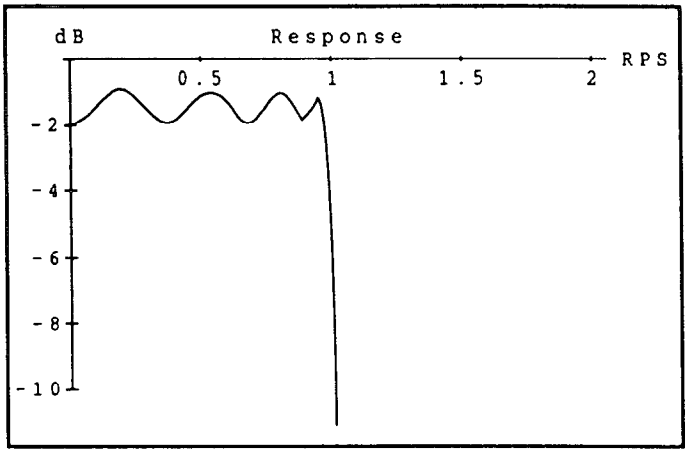
**Figure 5.5** (continued)

loss of **amax**. As will be seen in the plots to follow, this will set the maximum gain of the filter (at the peaks in the passband) to 0 dB.

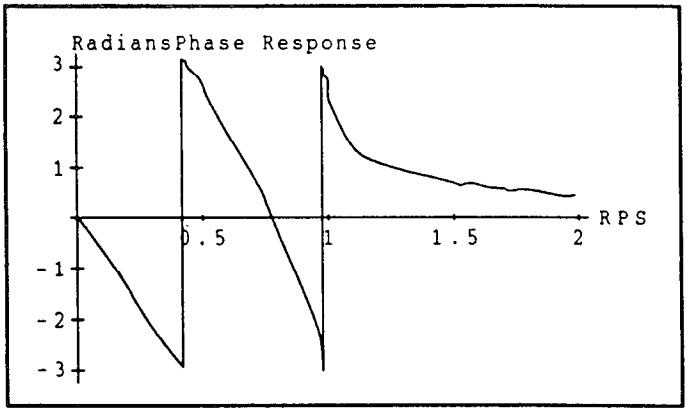
```
ChebychevLp[n_,amax_] :=
  Block[{poles, gain},
    poles=ChebychevPoles[n,amax]// N;
    gain = FilterMag[
      PolynomialFromRoots[poles],
                        1/ (2 Pi)]// N;
    gain = 10^(-amax/20) * gain;
    Return[{gain, {}, poles}]]
```

The following plot shows the magnitude and phase response of an eighth-order Chebychev low-pass filter with a pass-band error of 1 dB.

```
ft=FilterFromGZP[ChebychevLp[8,1]];
FreqResponseRadians[ft,2,
                    PlotRange->{-10,0}];
```

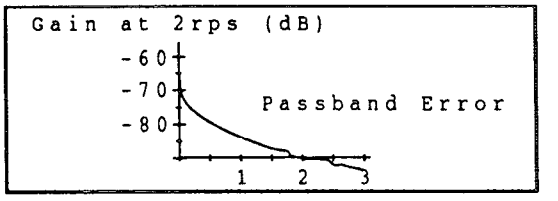



```
Plot[FilterPhase[ft,r],{r,0,2},
    AxesLabel->{" RPS", "Radians"},
    PlotLabel->"Phase Response"];
```




Chebychev filters can have an arbitrarily small error in the passband but this does not come for free. The following plot shows the gain at twice the corner frequency as a function of pass-band error. In each case an eighth-order Chebychev low-pass filter was designed. Note that if more error in the passband can be tolerated then a much sharper cutoff can be realized.

```
Plot[FilterDb[
FilterFromGZP[ChebychevLp[8,e]],2],
    {e,.01,3},
    AxesLabel->{"Passband  Error",
                "Gain at 2rps (dB)"}];
```




## 3 Band-pass Filters

Section 2 showed how to design a generic Chebychev low-pass filter. These low-pass filters can then be transformed into low-pass, high-pass, band-pass, and band-reject filters with arbitary cutoff frequencies. This section will show how to transform a low-pass filter into a band-pass filter. We use the gain, zero, pole structure to keep track of the filter parameters.

A low-pass filter is transformed into a band-pass by specifying the location of the two corner frequencies. We make this transform by substituting the following expression for s into the normalized low-pass filter [Daryanani76]:

$$S = \frac{s^2 + w0^2}{B\ s}$$

In these expressions **B** is the difference (in radians) between the two edges of the passband and **w0** is the geometric mean of the frequencies at the edges of the passband. The function **BpTransform** is used to transform a single root of the normalized filter into two new roots due to the substitution above. (The extra root at zero is ignored for now.)

```
BpTransform [roots_, w0_, B_] :=
  N[Flatten[Map[{B # / 2 +
  Sqrt [B^2#^2-4w0^2]/2, B # / 2 -
  Sqrt [B^2#^2-4w0^2]/2}&, roots]]]

BpTransform [ButterworthPoles [3],
      2Pi Sqrt[1000 2000],
      2Pi 1000]

{-1104.8 - 6450.39 I,
  -2036.79 + 11891.8 I,
  -3141.59 + 8311.87 I,
  -3141.59 - 8311.87 I,
  -1104.8 + 6450.39 I,
  -2036.79 - 11891.8 I}
```

**Figure 5.5** (continued)

The function **LpToBp** transforms each of the poles and zeros in the original low-pass filter according to the **BpTransform** function. In addition, each zero in the original low-pass filter contributes a pole at zero, and, likewise, the original poles contribute a zero at DC. The difference between the number of poles and zeros tell us the number of roots at zero to add, and extra factors of **B** to add to the gain.

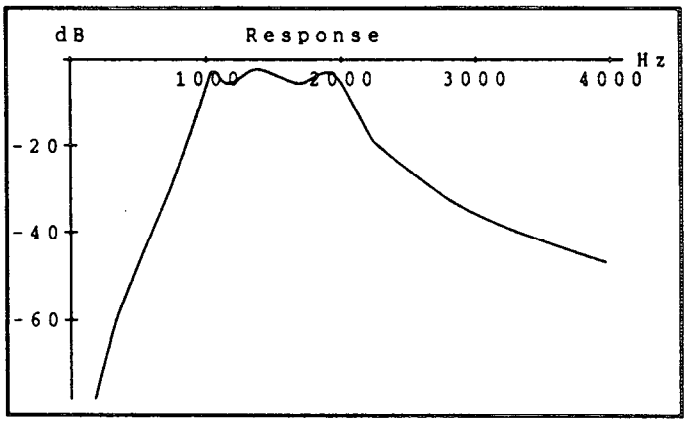
```
LpToBp [{gain_, zeros_, poles_},
        fp1_, fp2_] :=
  Block[{w0, B, RootDiff,
         ExcessPoles, ExcessZeros},
       w0 = 2 Pi Sqrt [fp1 fp2];
       B = 2 Pi (fp2 - fp1);
       RootDiff = Length[zeros] -
           Length[poles];
       If[RootDiff > 0,
         ExcessZeros = RootDiff;
           ExcessPoles = 0,
         ExcessPoles = -RootDiff;
           ExcessZeros = 0];
      {gain/B^RootDiff,
     Join[BpTransform[zeros, w0, B],
     Table[0,{ExcessPoles}]],
     Join[BpTransform[poles, w0, B],
     Table[0, {ExcessZeros}]]}]
```

This transform is applied to a third-order low-pass filter to determine a sixth-order band-pass filter with a passband between 1 kHz and 2 kHz and a maximum pass-band error of 3dB.

```
LpToBp [ChebychevLp [3, 3], 1000,2000]
                                  //N

           10
(4.8443 10  , {0., 0., 0.},
    {-326.13 + 6478.28 I,
        -612.013 - 12157.1 I,
        -938.143 + 8836.1 I,
        -938.143 - 8836.1 I,
        -326.13 - 6478.28 I,
```

The frequency and phase response of this sixth-order band-pass is shown below.

```
flt = FilterFromGZP [LpToBp[
    ChebychevLp[3, 3], 1000, 2000]];
FreqResponse[flt, 4000]
```




[Wolfram88] S. Wolfram, *Mathematica* (Redwood City, Calif.: Addison-Wesley, 1988).

[Daryanani76] G. Daryanani, *Principles of Active Network Synthesis and Design*, (New York: John Wiley and Sons, 1976).

**Figure 5.5** (continued)

```
In[2]:=
        FilterFromGZP[ChebychevLp[8,1]]

Out[2]=
        .01535187 /
        ((0.350082 + 0.996451 I + s) (0.0350082 - 0.996451 I + s)
         (0.099695 - 0.844751 I + s) (0.099695  + 0.844751 I + s)
         (0.149204 - 0.564444 I + s) (0.149204  + 0.564444 I + s)
         (0.175998 + 0.198206 I + s) (0.175998  - 0.198206 I + s))

In[3]:=
    Chop[ExpandDenominator[FilterFromGZP[ChebychevLp[8,1]]]]

Out[3]=
    .01535187 /
        (0.0172267 + 0.107345 s  + 0.447826 s^2  + 0.846824 s^3  +
              1.8369 s^4  + 1.65516 s^5  + 2.42303 s^6  + 0.919811 s^7  + s^8)
```

The `Chop` function is used here to drop the very small imaginary terms that are caused by roundoff error in the floating point calculations.

One of the more useful features of a symbolic math system is that it can be extended. Rules can be added or programs written to specialize the behavior of the system. In *Mathematica*, rules are defined using a pattern matching language much like Prolog. On the left-hand side of a rule the underscore character (_) indicates a wildcard position where any quantity can be substituted. Furthermore, the underscore character can be appended to a variable name to make a named wildcard variable. The pattern matching capability built into *Mathematica* is very powerful. The simple expression

```
foo[a_, b_, c_]
```

on the left side of a rule matches the function `foo` called with three arguments. The expression on the right-hand side of the rule will be used with the appropriate substitutions whenever the variables `a`, `b`, and `c` are used. A more complicated expression like

```
factorial[n_Integer]
```

matches any time the `factorial` function is called with an integer argument. The matching expression can include arbitrary *Mathematica* notation. For example

```
diff[a_ + b_]
```

can be used to pick apart a sum and define a new differentiation rule.

*Mathematica* notebooks also include the ability to display animations. This is useful as a way to show how simple parameter changes affect the solution, or to rotate a three-dimensional graph around its origin so the reader can more easily perceive

its form. In my own work we have used *Mathematica* animations to show wave propagation solutions.

Readers of this chapter can get a sense of the readability of a notebook from the examples in Figures 5.4 and 5.5. A problem with *Mathematica* is that the language is new and probably unfamiliar to many readers. Consequently, notebooks should be written, much like mathematical papers, so that the general flow can be understood by skipping the equations [19]. A brief description of unusual syntax might be given the first time it is used. In this sense, a notebook is no different from a normal paper.

*Mathematica* includes elements of all the important characteristics of a system for creating and exploring interactive signal processing documents. First, *Mathematica* notebooks are organized hierarchically. Within a notebook, equations, paragraphs, mathematical results, and graphics are each cells that can be grouped into larger cells. Cells can be hidden (or closed) in such a way that only the first line of a cell is visible. From this information the reader can decide whether the rest of the cell needs further attention. The first line of large, grouped cells is typically used as a section title.

The help system in *Mathematica* is a simple example of hypermedia. A user can select a function in a notebook and ask for more information. A new window appears describing the usage of the function. These simple usage statements are handy, but the original definition will include a more complete description of the algorithm. The system would be even more useful if the user could ask about a function and immediately move to the part of the notebook where the function is first defined.

All of these features of a *Mathematica* notebook would not be interesting if notebooks could not be distributed. The essential information in a notebook is conventional ASCII text, which is easily moved through the email and computer networks. While only some computer systems support the complete notebook concept, all *Mathematica* systems can understand the data and the mathematics contained in a notebook. Thus, a reader with a version of *Mathematica* without the full notebook capability can study the printed version and still try the examples.

One disadvantage of systems like *Mathematica* is that strictly numerical calculations are not efficient. Symbolic manipulation programs are designed to work with any type of mathematical quantity. Thus, when performing a multiplication, the terms could be symbols, arbitrary precision integers (bignums), high-precision floating point numbers, or simple integers or floats in the machine's native format. Of these, only the native format calculations are fast. Still, a symbolic manipulation program must check for all of these possibilities each time it does an operation, and this type checking can be more expensive than the mathematical operation. Conventional programming languages do not pay this penalty because all types are known at compile time and the proper machine instructions generated ahead of time.

Finally, *Mathematica* is a commercial product that not everybody will be able to afford. Wolfram Research has put into the public domain a *Mathematica* notebook reader. This notebook reader does not have any of *Mathematica*'s mathematical

ability, but it does allow people to view a notebook and play the animations on any Macintosh computer. My own cochlear notebook [2] has been published on paper and with a floppy disc containing the *Mathematica* notebook and the notebook reader to allow the material to have the widest possible distribution.

## 5.4 DESIGN ISSUES

Sections 5.2 and 5.3 have described the form an interactive document might use to describe a signal processing algorithm. How should the author structure his or her writing to make the best use of the technology available to describe a signal processing problem and its solution? There is no question that an interactive signal processing document requires new skills from an author. Some of these skills are the subject of this section.

The benefits of interactive signal processing documents described here do not come for free. Certainly the design and writing of such a document takes more thought and care than conventional papers do. When the interactive document is done, there is no easy way to disseminate its electronic form. Both of these problems should diminish as people become more familiar with this new medium for research and publishing.

This section describes some of the factors that make an interactive signal processing document a success. Certainly the biggest factor is writing the document so that it invites the reader to interact with the material. Fortunately, this is easily addressed by the author. Other factors, for example distribution and notation, are more difficult. Each of these difficulties are addressed in the remainder of this section.

### 5.4.1 How to Write an Interactive Signal Processing Document

Designing an interactive document is not easy, but the effort is worthwhile since writing an interactive mathematical document becomes as much a learning experience as reading it. Teaching new material is often the best way to learn it. By preparing an interactive document, one is forced to study the material as a reader, and by having a tool such as *Mathematica* it is possible to explore more of the subject area. In addition, when a common system is used to research and present a new result, the effort in creating an interactive document is minimized.

Making a document interactive adds another dimension to the writing task. In some ways this makes the task more difficult, but in other ways the task becomes simpler. Just as a picture is worth a thousand words, an interactive example showing how frequency response changes with pole location can be worth a thousand figures.

The interactive dimension might require extra work by the author of a signal processing document. One can always use the tools described in this chapter to write a conventional paper; it would not be any more or less efficient than using a word

processor. Fortunately, some of the interactive features described here make the writing process easier. It is often easier to show a reader an interactive graphic than it is to explain it in words. Other parts of an interactive signal processing document, such as working models and simulations that would otherwise never leave the lab, require more effort to polish and make ready for publication. As always, it is up to the author to decide on the proper amount of effort to apply. A short note explaining a new algorithm for a research group probably doesn't need as much polish as an undergraduate signal processing text.

One might think that this extra work would slow the rate of research. This is not necessarily true. Preliminary signal processing ideas are already exchanged within one's research or development group as small code samples and in interactive discussions. The point of this chapter is to describe the benefits of allowing a reader to more easily benefit from this rich form of interaction.

There are four skills and practices that should be remembered when writing an interactive signal processing document. They are:

1. Use good writing and graphics.
2. Iterate the design with real readers.
3. Guide readers to interactions.
4. Limit interactions to fit the reader and available computer power.

That good writing and graphics are important for an interactive document should go without saying. It is especially important that the ideas expressed by Tufte [20] should be applied to the interactive display and models. The remaining techniques for designing an interactive signal processing document will be discussed next.

Adding an interactive component to a signal processing paper is not a panacea. Just as there are bad papers, there will be bad notebooks. Fortunately the technology encourages a closer collaboration between the author and the reader. A successful interactive document will often go through several iterations. When first writing an interactive document it is hard to know how much detail to include or what kinds of models are useful to any particular reader. At successive stages, observation of how readers interact with the document will help guide its evolution.

The key task is to design the interactive document so that the reader can profit from the information and the changes can be shown at interactive rates. The first part of this problem, inviting the reader to play with the model, is easily solved with words. Telling the reader, "Here's some equations, play around with them," will only help the most motivated readers. Instead, the introduction of an interactive document could guide the reader to those parts of the document that can be modified, such as is done in this example [2]:

> The best way to interact with this notebook is to read the description, study the examples, and then modify an example to see how different parameters give different results. For example, an appendix to this report describes digital filtering and provides

functions to design first and second order filters. Much can be learned about digital filtering by combining these filters and studying the resulting frequency response or pole-zero plots.

Readers might also want to modify this model to better fit their own experience or ideas. For example this notebook describes a relatively simple model of the effects of the outer and middle ears on the sound. A reader might be interested in providing a better model or removing the outer and middle ear filters completely and studying the change in response. As another example, this report describes a simple Automatic Gain Control (AGC) to compensate for the large range of sounds produced by humans. This notebook explores several variations on the basic AGC but readers might want to try their own.

Simple examples, spread liberally through the text, encourage the reader to "kick the tires." It is probably not important that every reader understand the details of a filter design algorithm. But labeling a simple figure showing a Butterworth filter response with the *Mathematica* text

```
PlotFilterResponse[DesignButterworthLowPass[8,1000]]
```

makes it clear to casual readers that this is a low-pass filter with a cutoff frequency of 1,000 Hz. If the `1000` is changed to `2000`, the cutoff frequency should change by an octave. The reader might not appreciate exactly what an eighth-order filter is but should see that the filter attenuates more quickly with higher order.

Readers also need to be guided toward those portions of the document that are interactive. Not every part of a document can be changed in a meaningful way, but those parts that can be changed or perform an action for the user (play a sound or display an animation) should be marked. If the reader changes the title of the paper it probably will not automatically change the contents. Highlighting the input text in a special font or with graphics tells the reader which parts of the document can be changed.

It is easy for both the reader and the computer to be overwhelmed by an interactive document. Without adequate guidance the reader might wander down paths where there is no hope that any meaningful conclusion can be reached. In addition, an all-encompassing simulation would model many details that are not interesting to the reader. Limiting the domain over which the reader can change the simulation helps control the amount of processing power that is needed to answer a reader's question. If the problem is well defined it might be possible to precompute all of the interesting results and then use interpolation to display the correct simulation result.

Figure 5.6(a) shows a simulation where the reader has too much freedom. In this example, the reader can place the poles of a filter at any place on the *s*-plane and see the resulting frequency response. This gives the reader too much freedom and it is unlikely, for example, that the special properties of the classical filter design techniques will be found. In addition, it will be hard to find an easily affordable

document reader that will have the computational horsepower to keep up with the user's requests.

Figure 5.6(b) shows a modified version of this example where the reader is limited to studying the relationship between the Butterworth and Chebychev filters. By moving a knob that controls the eccentricity of the pole locations, the reader can see the effect on the pass-band ripple and the filter attenuation rolloff. This is now simple enough that even a dozen precomputed frequency responses would show the concept to the reader, without the reader knowing that the computations were done ahead of time. It would even be possible to precompute audio examples so that the reader can listen to the effect of each filter.

I do not mean to say that a paper including an example like that shown in Figure 5.6(a) is not useful. There will always be readers who will understand the basics of such an example and will want to explore the effect that quantization has on pole location or any number of ideas that never occurred to the original author. It is up to the author to carefully draw the line between guiding the reader and allowing the reader to become lost in the details.

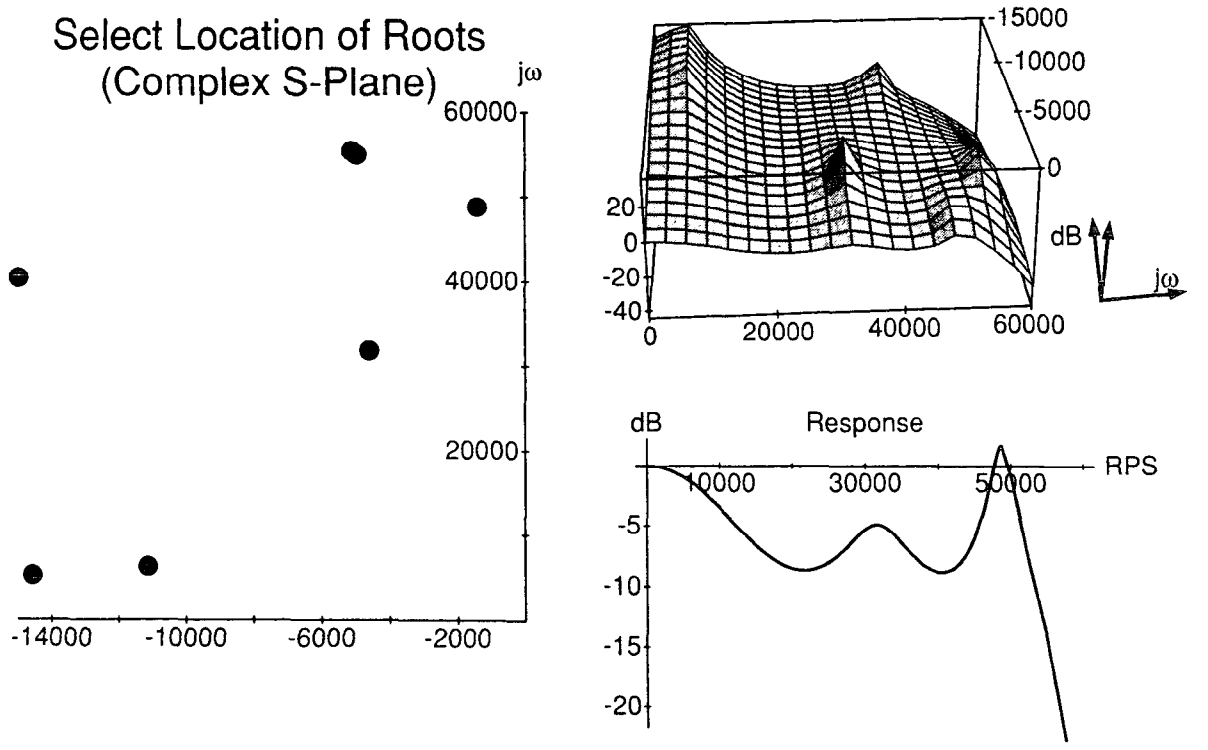
### 5.4.2 Problems with Interactive Signal Processing Documents

Other parts of the problem are not as easy for the author to address. These problems include choosing a system, publishing the electronic document, and picking a notation. Each of these problems will be addressed in the remainder of this section. Other issues, such as version control or keeping track of what has changed between versions, and maintaining correctness in the face of changes by the reader, are secondary problems and are not discussed here.

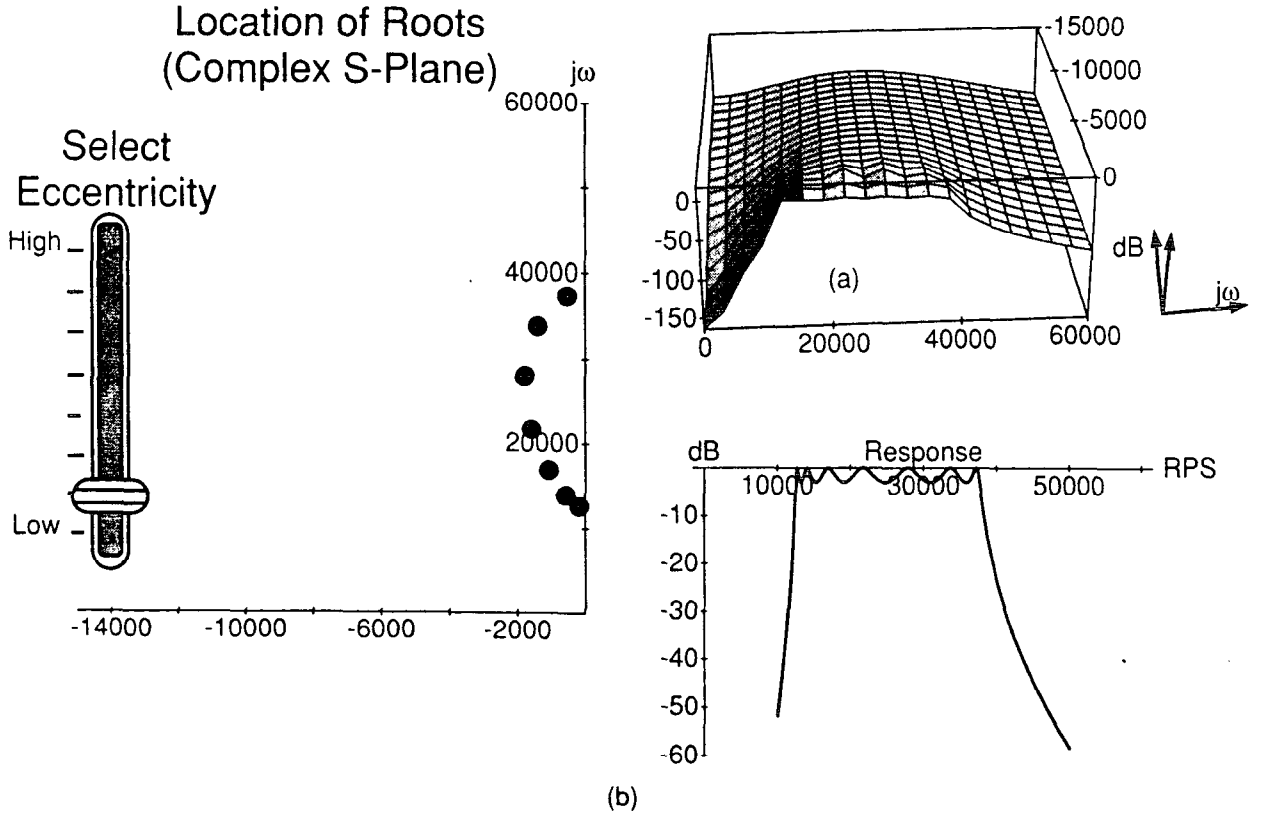
Choosing a system for writing an interactive signal processing document is not easy; the ideal system does not exist yet. I have used *Mathematica* for writing several interactive signal processing documents. It has many of the desired characteristics but is lacking in other areas. *Mathematica* would be a much better environment for creating electronic DSP notebooks if it had better multimedia support, was more efficient at strictly numerical calculations, and if it had better text formatting and graphics support. Hopefully these and other problems will be addressed in future releases of the software.

On the other hand, it is hard to believe that any one system will solve everybody's problems. Large, all-encompassing tools tend to be unwieldy and not solve anybody's problem very well. Instead it will probably be best if a user can mix a tool for filter design from one vendor with a special-purpose accelerator from another to make the most efficient research and learning environment.

Publishing notebooks and other forms of electronic documents is not easy. Magazines and journals usually used to disseminate research results have evolved efficient mechanisms and designs to effectively communicate the printed word. But


Select Location of Roots
(Complex S-Plane)
jω
60000
40000
20000
-14000
-10000
-6000
-2000
-15000
-10000
-5000
0
20
0
-20
-40
dB
jω
0
20000
40000
60000
dB
Response
RPS
10000
30000
50000
-5
-10
-15
-20

(a)


Location of Roots
(Complex S-Plane)
jω
Select
Eccentricity
High
Low
60000
40000
20000
-14000
-10000
-6000
-2000
-15000
-10000
-5000
0
0
-50
-100
-150
dB
jω
(a)
0
20000
40000
60000
dB
Response
RPS
10000
30000
50000
-10
-20
-30
-40
-50
-60

(b)

a single floppy, or other form of electronic media, can cost as much as the magazine it accompanies.[3]

One of the more successful schemes for electronic publishing is based on the international computer networks. Dongarra at Argonne Labs maintains an electronic mail system for distributing many large numerical software packages [21]. Users can send electronic requests to a special address and receive more information or the software by return mail. Another scheme is to broadcast the software on one of the computer bulletin boards. This is commonly done, for example, on the Usenet bulletin board comp.sources [22].

Unfortunately, not everybody has access to the computer networks. Instead, electronic material is often made available on floppy disks that can be read on a user's own computer. For instance, my own report on the implementation of a cochlear model [2] was published as a technical report so it could be accompanied by a floppy disk containing the *Mathematica* notebook. A recent issue of the *Communications of the ACM* [10] included an advertisement for floppy disks containing hypermedia examples.

An additional problem is that an electronic document is not as convenient as a magazine or a book. Curling up with a computer will probably never have the same appeal as curling up with a good book, but the next generations of portable computers should make this easier. For example, my own notebook was designed so that it can be read as a normal paper but without all the benefits of an electronic document.

Finally, there is the issue of notation. Within any one technical area, for example, signal processing or high-energy physics, the notation is well established, but it often differs widely between areas. Even a concept as simple as an integral is written in many different ways with marks to indicate different flavors of integration.

*Mathematica* solves this problem by defining a new language based on the ASCII alphabet. Wolfram has exchanged the rich notation that scientists and math-

[3] There are more and more examples of printed works accompanied by electronic media. Often books on microcomputer programming include a floppy disc with programming examples [for example, *Programming with MacApp* by David Wilson]. *develop*, a magazine that Apple publishes for its software developers, includes a CD-ROM with every issue. Each CD-ROM contains the complete text for all the issues of the magazine published to date and source code. According to the editor, the magazine and the CD-ROM each cost approximately $2 in 1990. The *Mathematica Journal* distributes a floppy disk with source code with each issue, but the articles are written like a conventional paper accompanied by source code.

**Figure 5.6** Two examples are shown of interactive signal processing models. The first example (a) allows the user to pick any location for the filter poles. It probably provides too much freedom for most users to gain any useful insight. The second example (b) allows the user to change the eccentricity of the pole locations and compare the Butterworth and Chebychev methods. Each figure shows the locations of the poles in the upper left quadrant of the $s$-plane, the magnitude of the response in the same quadrant, and the conventional frequency response.

ematicians have evolved through the ages for a very precise functional notation. For example, one writes

```
Laplace[ f[t], t, s]
```

to represent the Laplace transform of a function of $t$ in terms of the complex variable $s$.

Other programs, such as *Milo*, by a company called Paracomp, use a more conventional mathematical notation, but their knowledge of mathematics is limited. A research system called *CaminoReal* [23] gives authors an interactive interface to a writing program and symbolic algebra programs. The result is a nicely formatted paper document without the hypermedia and interactive features that are part of an electronic notebook. Perhaps the best solution is to allow users of symbolic math programs to define graphical templates, which are used when the system wants to translate its internal representation into something to be displayed to the reader.

## 5.5 RESEARCH ISSUES

Electronic notebooks are possible today and the resulting document, for example [2], can successfully communicate a signal processing idea. Currently available software, however, limits the topics that can be covered readily. A notebook on filter design is relatively straightforward. Graphics and animations can show most of the ideas, but a notebook on audio compression would be frustrating without the ability to include high-quality audio examples in the notebook.

Some needs of future electronic notebooks go without saying. There will always be a need for more computational horsepower to enable more realistic models to be built. Higher quality audio and easier ways to integrate video will allow more signal processing topics to be described.

Other needs, such as providing instructional directions and tuning the human interface, are difficult issues. It is important that an electronic notebook encourage the user to interact with the material by making it easy for the reader to navigate through the notebook and ask reasonable questions.

Before concluding, two areas of future work are worth noting. These ideas have been alluded to in other parts of this chapter but they are worth repeating. First, more powerful symbolic tools will make it easier for researchers and readers to explore difficult signal processing problems. Second, there are many software engineering problems in designing a system that allow users to move between different types of simulations and include the necessary tools for a complete system.

Several improvements to the mathematical symbolic manipulation world would be nice to see. More powerful symbolic tools will allow more difficult DSP problems to be solved without resorting to brute force. Allowing the user to specify the cost of mathematical operations and then automatically optimize an algorithm as is done with ADE would make it easier to realize the solution to a DSP problem using the available hardware. Finally, better support for strictly numerical calcula-

tions will allow a system to be designed and the resulting algorithm applied to real data. All of these improvements would make it easier to write an interactive notebook to describe the solution to a signal processing problem.

The remaining problem is one of software engineering. Symbolically manipulating mathematical symbols is only part of the problem. An author of an interactive signal processing document will need other tools. Tools such as text formatters, spelling checkers, and drawing programs are needed but probably not within the domain of expertise of most people designing signal processing environments. A software component system is one solution to this problem.

## ACKNOWLEDGMENTS

I would like to thank a number of people for help with this chapter. Theo Gray, Nancy Blachman, and Paul Abbott at Wolfram Research have been invaluable in helping me master *Mathematica*. They were very understanding when I tried to use *Mathematica* in ways that were never envisioned. I would also like to thank Richard F. Lyon, Robert Hon, Neenie Billawala, Monica Ertel, Pam Lau, and Nancy Tague for their help in producing the notebook [2] that led to this chapter. Finally, Richard Lyon, Michele Covell, Richard Fateman, Brian Evans, James McClellan, Norm Carter, and Dennis Arnon have all made many useful comments about the ideas expressed in this chapter.

## REFERENCES

[1] M. Slaney, "Interactive Signal Processing Documents," *IEEE ASSP Magazine*, 7 (1990), 8–20.

[2] M. Slaney, "Lyon's Cochlear Model," *Apple Computer Technical Report #13*, Corporate Library, 20525 Mariani Avenue, Cupertino, Calif. (1988).

[3] "Enter *Mathematica*," *MacUser* (November 1988), 199–216.

[4] J. Barwise, "Computers and Mathematics," *Notices of the American Math Monthly*, 35 (1988), 1333–49.

[5] R. J. Fateman, "A Review of *Mathematica*," *Journal of Symbolic Computation*, to appear.

[6] *IMSL Library Reference Manual*, IMSL Inc., Houston, Tex. (1980).

[7] *MATLAB for Macintosh Computers*, The MathWorks, Inc., 24 Prime Park Way, South Natick, Mass. (1989).

[8] B. Evans and J. H. McClellan, "Symbolic Transforms with Application to Signal Processing," *Mathematica Journal*, 1 (Fall 1990), 70–80.

[9] M. Covell, "An Algorithm Design Environment for Signal Processing," RLE Technical Report 549, Cambridge, Mass.: MIT, 1989.

[10] *Special Issue on Hypertext*, in *Comm. of the ACM* (July 1988).

[11] P. J. Denning, "Announcing Literate Programming," *Comm. of the ACM*, 30 (1987), 593.

[12] *Programs for Digital Signal Processing* (New York: IEEE Press, 1979).

[13] G. Kopec, "The Signal Representation Language SRL," *IEEE Trans. on ASSP*, ASSP-33 (1985), 921–32.

[14] C. S. Myers, "Signal Representation for Symbolic and Numerical Processing," RLE Technical Report 521, Cambridge, Mass.: MIT, 1986.

[15] W. Dove, C. S. Myers, and E. E. Milios, "An Object-Oriented Signal Processing Environment: The Knowledge-Based Signal Processing Package," RLE Technical Report 502, Cambridge, Mass.: MIT, 1984.

[16] *Photoshop User's Guide*, Adobe Systems, Inc., Mountain View, Calif. (1990).

[17] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer* (Redwood City, Calif.: Addison-Wesley, 1988).

[18] T. W. Gray and J. Glynn, *Exploring Mathematics with Mathematica* (Redwood City, Calif.: Addison-Wesley, 1991).

[19] D. E. Knuth, T. Larrabee, and P. M. Roberts, "Mathematical Writing," MAA Notes No. 14, The Mathematical Association of America (1989).

[20] E. R. Tufte, *Envisioning Information* (Cheshire, Conn.: Graphics Press, 1990).

[21] J. J. Dongarra and E. Grosse, "Distribution of Mathematical Software via Electronic Mail," *Comm. of the ACM*, 30 (1987), 403–07.

[22] J. S. Quarterman and J. C. Hoskins, "Notable Computer Networks," *Comm. of the ACM*, 29 (1986), 932–71.

[23] D. Arnon, R. Beach, K. McIsaac, and C. Waldspurger, "CaminoReal: An Interactive Mathematical Notebook," in *Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography*, ed. J. C. van Vliet (New York: Cambridge University Press, 1988).

# SYMBOLIC AND KNOWLEDGE-BASED SIGNAL PROCESSING

EDITORS

Alan V. Oppenheim

*Massachusetts Institute of Technology*

S. Hamid Nawab

*Boston University*

Prentice Hall

Englewood Cliffs, New Jersey 07632