

Mathematica Programming – Start Here

Introduction to Programming with Mathematica (first edition) is highly effective in living up to its title. This review outlines its content in some detail, points out its many strengths, and identifies a few areas that could be strengthened, or with which the reviewer takes some exception. A postscript describes the changes we may expect to see in the forthcoming second edition.

Kirk A. Mathews

Introduction to Programming with Mathematica, by Richard J. Gaylord, Samuel N. Kamin, and Paul R. Wellin. TELOS/Springer-Verlag, 1993. 302 pages, hardcover, with disk. \$39.95. ISBN 0-387-94048-0. Second edition to be published in September 1995.

THIS IS THE BOOK I WISHED FOR WHEN I WAS READING THE *Mathematica* book for the first, second, and third times. This is the book I hoped for when I bought *Programming in Mathematica*, by Roman Maeder. It fits perfectly between these and the how-to books, such as *Mathematica by Example*, by Martha L. Abell and James P. Brasetton, and *Mathematica in Action*, by Stan Wagon.

The authors state that the book was written for those with no prior programming experience (including those with experience only in “low-level languages, like C and FORTRAN”) who would like to learn to program in a “friendly and useful language.” I’m not sure I fit in that group, having programmed in (or studied programming style in) FORTRAN, Pascal, Basic, ALGOL, Lisp, Logo, APL, PL/1, Forth, and Citran. In any case, I found this introduction to be clear, concise, and to draw together many of the diverse and important features of the *Mathematica* language. The exercises are useful in driving home the points made, and in opening the door to exploring them in a little more depth. As I read the book, I frequently thought how much easier the learning curve would have been if I’d had it a few years earlier, and there were several points that became clear to me for the first time.

I selected *Introduction to Programming with Mathematica* as the text for a four-week, two hour per day “computer review” for entering M.S. students in physics and nuclear engineering at the Air Force Institute of Technology. In class, I lectured and presented demonstrations using a 486 PC with overhead VGA display and *Mathematica* for Windows software. The students worked on Sun SparcStation2 workstations with the non-notebook interface. (They did not find

the difference in interfaces to be a problem.) The students found the text useful, but many would have preferred a “how-to” book, rather than a “programming” book. Those students found chapters 1 and 2 (Preliminaries and A Brief Overview) more to their liking. Those who went on to use *Mathematica* extensively (particularly in their thesis work) found the text especially useful.

The organization and balance of the text are appropriate to its uses. After the lead-in material, chapter 3 presents lists and list manipulations. These topics are essential to effective use of *Mathematica* functions and are important in their own right. Placing them here makes possible a more thorough treatment of the following material. Chapter 4 introduces functions, and develops a central concept: that programming consists of defining useful functions, and that problem solving consists of applying them to data. The practice provided here helps beginning *Mathematica* programmers, particularly those accustomed to a traditional, sequential language, to see “programming” in this new way.

The real power of *Mathematica* can’t be perceived until its underlying structure, as a system of rewrite rules applied recursively, is understood. So chapter 5 focuses on the evaluation of expressions, including rewrite rules, expressions, patterns, term rewriting, and transformation rules. Conditional function evaluation is the final prerequisite to recursive and iterative programming. Chapter 6 presents conditionals, both implicit (pattern matching conditions for conditional application of rules) and explicit (If, Which, Switch).

Chapter 7 examines recursive problem solving through recursive function definitions. The Fibonacci numbers are used as an introductory example. This view of recursion is rather like mathematical induction worked backwards. Recursive functions that take lists as arguments are considered next. Such functions work by calling themselves with structurally simpler arguments until a trivial case is reached. Sorting a list is a simple example of this idea: to sort a list of length greater than one, return the list formed by appending the largest element of the list to a sorted copy of the list with the largest element dropped (the simplified argument); to sort a list of length one, return the list unchanged. Several good examples are explored in the section titled “Thinking Recursively.” Developing a recursive system of rules for symbolic differentiation is another, more abstract application

Dr. Kirk A. Mathews is an Associate Professor of Nuclear Engineering at the Air Force Institute of Technology. He has used Mathematica, since Version 1.3 for MS-DOS, both in teaching and in his research, developing new algorithms for numerical modeling of radiation transport. He is the author of Exploring Physics with TK Solver. He holds degrees in physics, English literature, and nuclear engineering, and is retired from the US Navy, where he served in nuclear submarines and with the Defense Nuclear Agency, as well as at AFIT.

that is nicely presented. A recursive approach to Gaussian elimination is particularly effective for students who remember what their FORTRAN program for that task looked like. (I was especially pleased to see a traditional numerical computing task treated in this way. Engineering students tend to see number crunching as the only real computing, so if this topic were first seen in the iteration chapter, it would be interpreted as confirmation of their feelings that these other approaches aren't real programming.) Some classic computer science applications are treated recursively: binary trees and Huffman encoding. Sections on dynamic programming, higher-order functions, and debugging techniques round out the chapter.

Chapter 8, Iteration, finally reveals the secret of how to write FORTRAN programs in *Mathematica*. I applaud the authors' decision to hold off on this topic for as long as possible, but to include it. When we reached this point in the course (near the end), many of my students expressed relief that it actually was possible to "write programs" in *Mathematica*, and happily wrote FORTRAN-*Mathematica* from then on. But many others had gotten the messages I had intended: that although iteration is natural for some problems, recursion or list processing is more natural for others; and that all too often, "writing a program" solves only one problem, while functional programming provides a bag of tools that can be readily used to solve many problems. The coverage here is effective, including examples of Newton's method, vectors and matrices, passing arrays to functions, and another implementation of Gaussian elimination. Even here, however, the code doesn't really look like the typical numerical analysis textbook pseudocode. Rather than a monolithic structure of sequences of nested loops, a main function and several auxiliary functions are used, so that there is still the flavor of top-down analysis.

Chapter 9 covers numerics, including number types, random numbers, Precision and Accuracy, and examples of numerical difficulties. Gaussian elimination is revisited yet again. Partial pivoting is presented to ameliorate bad conditioning, and is shown to be somewhat successful for linear systems problems with Hilbert matrices. An exercise discusses scaled partial pivoting. (Full pivoting could be used here as a project assignment for a programming course, although it is not in the text. Exercises 5 and 6 of section 3.3 provide prerequisite work with permutation lists and their inverses.) As with the previous chapter, I find this material to be well placed. Too many numerical methods books start with floating point representation and catastrophic cancellation in chapter one, when the students need to learn first what will work, not what won't. Only later, after they've seen things go wrong in the output but can't find errors in their programs, are students ready to hear about bad conditioning.

Chapter 10 covers the inevitable topics of graphics programming and sound. Unlike many books, which are over half filled with graphics arcana, the coverage here is 40 pages out of 278. That seems like unusually good balance to me. For many users, a few good plot commands, perhaps with some optional settings, will suffice. The first three sections cover simple graphics primitives, directives and options, and plot commands. The fourth section uses these for graphics

programming, drawing simple closed paths and binary trees. The final section introduces the Sound command. This material is useful, and presented concisely enough to not be overwhelming.

One of the themes that is seen repeatedly in the book is that of avoiding name conflicts. Modules are extensively used for this purpose. The final chapter, Contexts and Packages, deals with the other mechanisms for controlling visibility of names. This chapter first covers the use of packages, including Get (<<) and Needs, and the context commands (Begin, End, Context) and variables (\$Context, \$ContextPath). Then the package commands (BeginPackage, EndPackage) are covered, including a clear exposition of their effects on those variables. An example package, summary, and related miscellaneous items close out the text. This material is often confusing to students, but I found the coverage here to be complete, concise, and helpful.

After working through much of the book with the review class, and reading the rest of it for my own edification, I find that I have made only a few marginal comments, to correct typos or disagree with the authors. Some of my comments are about explanations that could be clearer. For example, the use of the terms "inner iterator" and "outer iterator" is backwards, leading to unnecessary confusion (page 54). Using the authors' nomenclature, the action of the outer (meaning at the end of the command, on the right) iterator is nested inside the action of the inner (left) iterator. The resulting confusion among some of my students was difficult to correct. If one notes that the iterators in the Sum command are in the same (left to right) order as the symbols are normally written, with the outermost sum first and the innermost sum last, and uses the terms in that way, then the point the authors make would be self-evident. (See the discussion on page 91 of the *Mathematica* Book, 2nd ed.) Also, the presentation of the Partition function (pages 58–59) needs a prose explanation of the purposes of its various arguments; it's too hard to sort out only from examples of its use. The following example of the behavior of Accuracy (page 198) is explained in terms of rounding in the binary representation, which misses the point altogether.

```
In[1]:= {Accuracy[1.23], Accuracy[12.5]}
Out[1]= {16, 15}
```

A better example is

```
In[2]:= {Accuracy[1.25], Accuracy[12.5]}
Out[2]= {16, 15}
```

which returns the same result, although there is no rounding in binary for either number. The real point is that adding a digit to the left of the decimal point removes one from the right of the decimal point. The input ?Accuracy returns a concise and correct description:

```
In[3]:= ?Accuracy
Accuracy[x] gives the number of digits to the right of the
decimal point in the number x.
```

Other comments are about programming practice. For example, the authors imply that the ordering of conditional definitions is unimportant, and only a matter of convention: “We can clean up this code a little, ... we’ve also reordered the rules to put the more specific ones first, as is more conventional (though not required), ...” (page 135). In the given example, the entry order of the rules is not significant, but in general, entry order does determine the rule used when “more specific” is ambiguous. The convention is good programming practice because it helps avoid bugs when these ambiguities occur, and it helps the programmer recognize this situation (and change the conditions to eliminate ambiguities, if possible). But these are minor points, usually easily explained to students.

If there were one thing I could add to the text, it would be more coverage of data types and their use in conditional definitions and in overloading functions. This topic has many implications; I’ll suggest three of them.

First, an important behavior of built-in *Mathematica* functions is that they return unevaluated if the argument isn’t in a form for which the intended definitions would be meaningful. This is an elegant application of the general practice of validating input to a program. Further, it empowers the user to add new definitions when additional meanings become appropriate or are needed. Even more importantly, it can delay evaluation until the arguments become appropriate (as, for example, when symbolic arguments are later assigned numeric values), or the needed but overlooked definition is added. User-defined functions should exhibit this same behavior. The authors appear to have a different philosophy that puts the emphasis on efficiency or expediency. For example, a function `pointLoc`, which returns the number of the quadrant of a point in the x - y plane, is defined with six conditional definitions and one that is unconditional (page 123). The intended condition for this last one is entered as a comment. The authors state “(It is a good idea to include the last condition as a comment, rather than as a condition in the code, because *Mathematica* would not realize that the condition has to be true at that point and would check it anyway.)” The result of their coding is that

```
pointLoc[{2, 1}, {x^2, z}]
```

returns 4, even though this is meaningless. Worse,

```
n = pointLoc[a, b]; a = 1; b = 1; n
```

returns 4, rather than 1.

Second, proper restriction of the domain of definitions is especially important when recursion is used. The text defines a Fibonacci function recursively (page 130) by

```
F[0] := 0; F[1] := 1; F[n_] := F[n-2] + F[n-1] /; n>1
```

This code fails, with output that is confusing for a beginner, when evaluating `F[Pi]`, `F[3/2]`, or `F[1.5]`, or even `F[2.0]`. The definition is only meaningful for natural numbers, so the compound type recognizer and conditional `F[n_Integer?Positive]` should be used in place of the appended conditional (`/; n>1`).

Third, I found the exercises on norms and condition numbers to be a good place to explain definition overloading. I had the students define the functions `normInfinity[v_?VectorQ]` and `normOne[v_?VectorQ]` and use them in defining `normInfinity[m_?MatrixQ]` and `normOne[m_?MatrixQ]`.

In summary, *Introduction to Programming with Mathematica* serves its intended purpose well. The coverage is appropriate and well organized. The explanations are nearly always clear and helpful. The exercises are useful if the book is used as a course textbook or for serious self-study. (I found it necessary to expand upon them, particularly in providing content-area problems that would interest the students, but that is no fault of the text.) I used it in class and plan to do so again. I recommend this book as a good preparation for effective use of *Mathematica*.

Postscript on the Second Edition

The authors are working on a second edition, and I have been privileged to see a “nearly final” manuscript. The second edition will build on the foundation laid by the first edition in two major ways: There is a new chapter on applications, and solutions for most of the exercises are added. The final chapter, on packages, has been enlarged to include a section on options and defaults, and to work through a substantive example package, on random walks. Additionally, the authors state that they have made numerous small changes in details and organization, and corrected errors that had been brought to their attention. (I hope they will incorporate some of the suggestions made above, which were provided to them at an early stage in the writing, but I’ve seen little evidence of that yet.) An example of this sort of revision is the addition of a section introducing linear algebra functions in chapter 2.

The new chapter uses the techniques presented in the first 10 chapters to look at three applications in some depth. These projects cover a wide range of areas and techniques. The first considers the random walk on a two-dimensional lattice. It looks at data representations: as lists of steps, as lists of locations, graphical representations, and animations. Then numerical simulations are used to estimate statistical properties of the random walk process, including the critical exponent. The second project examines the Game of Life. First the authors provide a clear exposition of the method used in the notebook distributed with *Mathematica*. Then they present a simpler, more concise, and much faster approach. The third project is the most complex, the development of a simple language interpreter. The authors define a “picture description language” (PDL), both operationally and formally, then develop routines (which make extensive use of replacement rules) to perform lexical analysis of an input string, followed by parsing (construction of the parse tree), calculation of the locations of the picture primitives used in the picture, and finally, construction of the graphical representation of the picture as defined in PDL by the input string. This project seems well designed in that it is relatively simple, but requires all the major steps of language definition and implementation.

All in all, the new edition should be a significant enhancement of the original, while preserving its character and utility. 