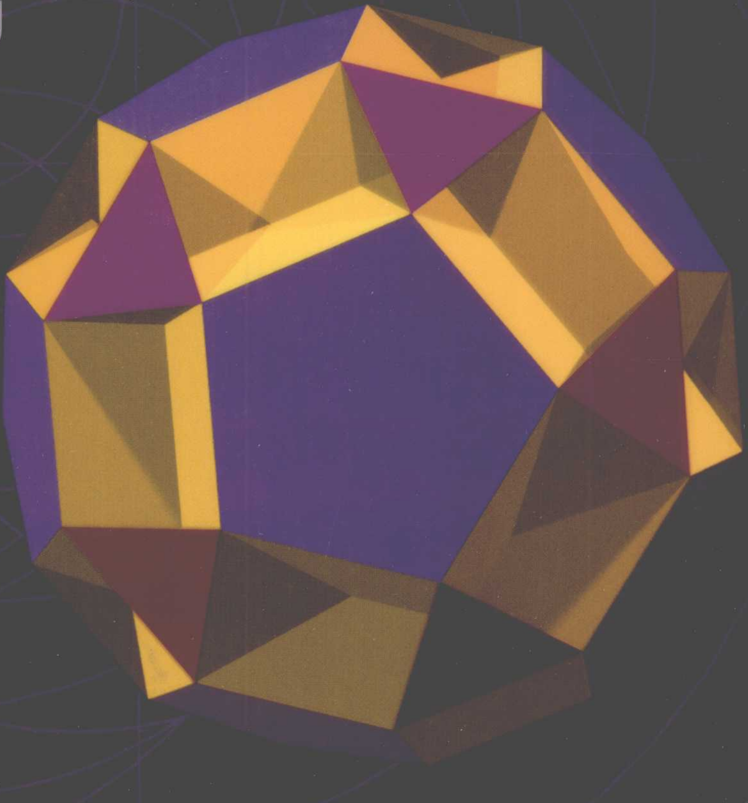
Computer Science with *Mathematica*®

Theory and Practice for Science, Mathematics, and Engineering



Roman E. Maeder

VERSION 4

Computer Science with *Mathematica*

Theory and Practice for Science, Mathematics, and Engineering

Roman E. Maeder

CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK   http://www.cup.cam.ac.uk
40 West 20th Street, New York, NY 10011-4211, USA   http://www.cup.org
10 Stamford Road, Oakleigh, Melbourne 3166, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain

First published 2000

Printed in the United States of America

Typeset by the author using *Mathematica* 4.0 and the TEX typesetting language on a Sun ULTRAsparc II computer.

A catalog record for this book is available from the British Library.

Library of Congress Cataloging in Publication Data
Maeder, Roman.
Computer science with Mathematica : theory and practice for
science, mathematics, and engineering / Roman E. Maeder.
p.   cm.
Includes index.
ISBN 0-521-63172-6. – ISBN 0-521-66395-4 (pbk.)
1. Mathematica (Computer file) 2. Mathematics – Data processing.
3. Mathematica (Computer programming language) I. Title.
QA76.95.M34   1999
510′.285′5369–dc21   99-38932
CIP

ISBN 0 521 63172 6   hardback
ISBN 0 521 66395 4   paperback

Preface

This book provides an introduction to computer science, and shows how modern computer-based tools can be used in science, mathematics, and engineering. Computer-aided mathematics has reached a level where it can support effectively many computations in science and engineering. In addition to treating traditional computer-science topics, an introductory book should show scientists and engineers how these computer-based tools can be used to do scientific computations. Students must get to know these possibilities, and they must gain practical experience. Learning a traditional programming language becomes less important, just as learning arithmetic is not a main topic of mathematics education. In an introductory book, it is clearly necessary to limit ourselves to a small part of the huge field of computer science. We emphasize topics that are related to possible applications in mathematics and the sciences. Technical and practical computer science have therefore been neglected.

It is certainly worthwhile to combine an introductory computer-science course with exercises. In the same way as we learn a foreign language by speaking the language and by studying literature in that language, we should apply algorithmic knowledge by studying programs and writing our own. If we can solve an interesting problem from mathematics or the sciences at the same time, all the better! Traditionally, such introductory courses use languages such as Pascal, C, or FORTRAN. These languages have in common that the effort to develop even a small program (one that adds two numbers, for example) is considerable. One has to write a main program that deals with input and output, and to compile the program. Furthermore, these languages cannot be used easily to solve nonnumerical problems. Leaving aside these practical difficulties gives us room to look at other topics in computer science, an extension that is not offered in traditional programming courses. In this way, we gain insight into computer *science*, which consists of much more than writing small programs.

Another disadvantage of traditional languages is that they support only *procedural* programming. This style is an important one, but it is not the only option and it is not always the best approach. I prefer a language that does not force this programming style on programmers. The programming style should be chosen to fit the problem to be solved, rather than vice versa. The language should be interactive, to encourage experimentation and to allow us to call individual functions without having to write a whole program.

Mathematica was first released in 1988, and it is being used with increasing frequency in teaching, research, and industry. A by-product of the symbolic computation system, it is a programming language that differs from traditional languages in many important ways.

Conventional languages are not well suited to expressing mathematical formulae and algorithms. LISP and other functional languages showed alternatives. An important aspect of scientific computation is an easy way to express mathematical rules. Application of rules by machine requires good pattern-matching capabilities of the kind found in Prolog. Another prerequisite is that it be simple to manipulate structured data. Such structural operations have been pioneered by APL. Object-oriented elements and modularization are important tools for developing larger projects. Ideas were taken from Simula, Smalltalk, and C++. We also want to support traditional procedural programming in the style of Pascal and C. All these objectives lead to a large language with many built-in functions. It nevertheless has a consistent and uniform style, made possible through the use of *rewrite rules,* which underly all other programming constructs. Such a language is also interactive and therefore easy to use. It is not necessary to compile functions or to embed them into a main program to use them. The additional step of compilation increases the difficulty of program development and requires special tools (debuggers) to study the behavior of programs.

Because *Mathematica* also contains most operations needed in mathematics and physics, it is especially well suited for an introductory course in computer science for readers interested primarily in the sciences and engineering. It allows us to treat interesting examples easily. There is no good reason, for example, to restrict the range of integers to $2,147,483,647$, as is done in most programming languages. This restriction makes no sense in mathematics. Programming with recursively defined functions is often treated as extraordinary and difficult. We can express naturally many mathematical algorithms, however, by using recursion, and it should be possible to formulate recursion easily in a language. For example, the properties of the greatest common divisor of two integers leading directly to Euclid's algorithm,

$$\begin{aligned}\gcd(a,b) &= \gcd(b, a \bmod b)\\ \gcd(a,0) &= a\,,\end{aligned}$$

can be expressed verbatim in *Mathematica* and tried out immediately. As in LISP, the technique of tail-recursion elimination in *Mathematica* ensures that the corresponding program runs as fast as the loop that is normally used (which is not the case in most procedural languages). Deriving the loop invariant and programming the same function as a loop leads naturally to systematic programming and considerations of program correctness.

Mathematica is helpful in all areas of computer use in mathematics, in the sciences, and in engineering:

- Its numerical part, which allows arithmetic to arbitrary precision, can be used to treat numerical mathematics, including traditional floating-point arithmetic.
- Its symbolic part does computations with formulae, solves equations, performs series expansions and transformations, and knows calculus to the level required for an undergraduate degree.

- The programming language supports all traditional programming styles, including procedural programming. The language can therefore be used for traditional computer-science classes (algorithms and data structures) as well.
- The rule-based programming system allows a natural expression of scientific facts.
- Graphics allows the meaningful presentation of results and experimental data. It is also useful for showing how algorithms work.
- We can call external programs and exchange results, so we can use external software libraries and even control laboratory experiments.

This book grew out of class notes for a course given at the Department of Mathematics and Physics at the Swiss Federal Institute of Technology, Zurich. It was originally published in my native German language [48], and I am glad to present now my own English translation and adaptation.

I am thankful to Erwin Engeler, John Gray, and Stephen Wolfram for their inspiration and many interesting discussions. Helpful suggestions on particular topics came from R. Marti and H. Mössenböck. Lyn Dupré proofread an early version of the manuscript, and Karen Tongish copyedited the final version. The publishers of the German and English editions, Ekkehard Hundt and Alan Harvey, helped me to keep going. Many thanks to the anonymous reviewer whose favorable comments and useful suggestions motivated me to finish this project.

R. E. M.
Wollerau, March 1999

About This Book

The emphasis of this introduction to computer science is *algorithmics* – that is, the study of algorithms. We do not want this activity to become a dry exercise, so we shall try out all algorithms as soon as possible. Our programs will often consist of only a few lines of code. Such simplicity allows us to concentrate on the essentials and to ignore peripheral matters such as input, output, and driver programs. Often, however, we shall develop whole packages, collections of various procedures grouped around a topic. The methods for writing such packages will be explained in Chapter 4. After all, computer science is not about writing small, throwaway programs but rather developing larger applications. In addition to finding suitable algorithms, this entails techniques of documentation and maintenance of software. We shall present some of these techniques.

Mathematica does have a major advantage over traditional programming languages: It is *interactive.* Interactivity encourages experimentation and allows us to test each function separately and to study its behavior. In the first section we shall study recursively defined functions, a topic often considered difficult and therefore treated with caution. We also have at our disposal a symbolic, numerical, and graphic computation system – an added benefit that we shall use in many ways.

Overview of Contents

Each chapter after the first two introductory ones presents a topic from computer science together with its applications and examples in mathematics, the sciences, and engineering. You can choose from the many applications presented those that correspond to your background. Because only one system (*Mathematica*) is used for all programs and all calculations, the extra work of learning about practical matters such as editing or working with the application is minimized. My experiences have shown that *Mathematica* is rather easy to learn; you will be able to work with it quite soon, after overcoming any initial difficulties you might encounter.

Chapter 1 is not a prerequisite for the rest of the text, if you already know something about computers. It shows how computers can be used in the sciences, explains the history and current state of computers, and discusses what computer science is all about.

The quick introduction to *Mathematica*'s syntax in Chapter 2 should be studied with a computer at hand, so you can try out the calculations for yourself and get a feeling for what it is like to work with *Mathematica*. The elements of programming presented in Sections 2.1–2.3 are the foundation of our programs.

In Chapter 3, we use two simple examples to show how mathematical questions can be turned into computer programs. The most important concepts are iteration and recursion. The section on loop invariants gives a method for proving programs correct.

Chapter 4 explains how programs in *Mathematica* are structured. We start with simple commands, which we turn into a program by defining a few functions. We will give guidelines for turning a program into a package. Packages allow for easier use of programs and prevent unwanted side effects on other programs, which might have similar function names. The tools we use are modularization and separation of the interface (for the user of our program) and the implementation (for the program developer). You can use these techniques as recipes, even if you do not know how they work in detail. You can use our template package as a starting point.

Abstract data types, presented in Chapter 5, constitute one of the most important tools for the design of programs. These methods allow a clean separation of design and implementation. We shall use them in most of our programs in this book.

Algorithms for searching and sorting are the basic building blocks of many programs. The algorithms presented in Chapter 6 are part of basic computer-science knowledge.

Problems can be solved in many ways. One aspect to consider when choosing a method is the complexity of the resulting algorithm. Chapter 7 provides an introduction to algorithmic complexity. As an example, we look at the computation of large Fibonacci numbers, optimization problems, and arbitrary-precision arithmetic.

Vectors and matrices are important data structures for mathematical applications. We present several important operations on them and look at a few algorithms from linear algebra in Chapter 8.

In Chapter 9, we program in LISP, a language that we can interpret in *Mathematica* easily. Recursion is the most important tool for solving problems in LISP, where it replaces iteration.

For many scientific problems, *rule-based programming* is the simplest method of solution. It is also the foundation of *Mathematica*'s programming language. In Chapter 10, we shall look at the important concepts of simplification and normal forms, as well as at some applications.

Functions are of central importance in mathematics. They play a lesser role in computer science, because many programming languages have only rudimentary means of dealing with them. An important exception are the functional languages, including *Mathematica*. Functions are the topic of Chapter 11. That chapter highlights the differences between the symbolic computation system *Mathematica* and ordinary languages.

In Chapter 12, we give a short introduction to theoretical computer science. There we see that this topic is not necessarily as "theoretical" as is often feared. We answer the question of what the fundamental limits of computers are and show that some problems cannot be solved by machine, even disregarding the practical matters of limited memory and computing time.

Databases are the most important commercial application of computers. Managing large volumes of data demands reliable and powerful programs. A precise mathematical model of collections of data provides the tools for their easy manipulation. We treat these concepts in Chapter 13.

Chapter 14 introduces an important programming style: object-oriented programming. It is especially useful for larger applications and for the design of reusable software.

Appendix A is an annotated bibliography on the topics programming methods, teaching with *Mathematica*, and literature about *Mathematica*; it includes a section with references for the topics treated in this book, followed by the bibliographical data.

The more detailed explanations about the structure of *Mathematica* given in Appendix B are useful for self-study and are also meant as a reference. For a complete reference to *Mathematica*, you should consult *The Mathematica Book* [74]. The appendix of that manual contains an alphabetical listing of all built-in functions, commands, and other objects. This listing, as well as the complete manual, is available on-line in *Mathematica* (in the *Help Browser*). Looking up an item there is much easier than is looking it up in a heavy book. Studying the *Mathematica* manual is not a prerequisite for reading this book.

Appendix B also contains a section that demonstrates *Mathematica*'s more advanced capabilities. Finally, we give the programs used to generate the chapter-opener pictures.

Certain sections are labeled "Advanced Topic." They presume that the reader has a more complete mathematical background than is required for the rest of the book; they are optional.

Sections marked "Special Topic" are independent from the rest of the book. Sections marked "Example" or "Application" develop a topic using a larger example that is of interest in its own right.

At the end of most sections, there is a review list, entitled "Key Concepts," of new concepts that have been introduced. At the end of the chapters, you will find numerous exercises.

The verso page following a chapter title contains a brief overview of the sections in the chapter, and an explanation of the graphic illustration on the title page. The programs for generating these pictures are in the package Pictures.m; see Section B.2.

Comments on Exercises

We assume that you already know how to work with your computer. The installation of *Mathematica* on your machine is explained in the documentation that comes with the software. This documentation includes a manual that explains the machine-specific features of *Mathematica*. The best way to learn *Mathematica* is to do practical exercises at the machine. In the beginning, you may want to look at one of the included demonstration documents before moving on to your own small examples. You can also find simple examples in the section titled "A Tour of *Mathematica*" in *The Mathematica Book*. We recommend that you work through such examples.

There are two ways to use *Mathematica* on a computer: the Notebook frontend and a simple dialog with the kernel of *Mathematica* (the kernel is the part that does the actual computations; the frontend serves as a user interface to the kernel). The Notebook frontend is more comfortable to use, but is not required for the examples in this book, which have all been computed by direct interaction with the kernel. All examples have been tested with Version 4.0 of *Mathematica*.

If you use the Notebook frontend, your interaction with *Mathematica* will look a bit different from the way it is presented in the book, but the results will be the same. Numbering of your inputs happens only after they have been sent to kernel for evaluation (with SHIFT-RETURN or ENTER), because the number is given out by the kernel, rather than by the frontend. An example Notebook is reproduced on page 95.

Please note that each example has been computed in a fresh *Mathematica* session. We recommend that you begin new sessions to avoid any influences from previous computations whenever the numbering of the input lines restarts at 1. Under the Notebook frontend, you can choose the menu command Quit Kernel to start a fresh kernel.

The frontend allows you to store your programs and your sample computations in the same document (the Notebook) and to open them again in the future. We recommend, however, that you store *packages* in separate files, and read them into *Mathematica* using `<<`*file*`` ` ``. This command to read in a package is often not shown in the dialogs in this book. If you want to reproduce the examples, you must read the appropriate programs into *Mathematica* first.

Electronic Resources

All programs mentioned in this book are available in machine-readable form from the book's Web site, located at `http://www.mathconsult.ch/CSM/`. There, you will find compressed archives of all files ready to download. Packages have the extension .m; Notebooks have the extension .nb. Both kinds of files can be opened with the frontend. Packages can be read into the kernel directly (using `` <<CSM` ``*file*`` ` ``) and can also be opened with any text editor (in ASCII mode).

The archive should be extracted into the AddOns/Applications subdirectory of your *Mathematica* installation directory. Extraction will create a subdirectory CSM inside the Applications directory.

Mathematica can display its own installation directory. The value of `$TopDirectory` will reflect the actual place where you installed *Mathematica* on your computer.

```
In[1]:= $TopDirectory
Out[1]= /usr/local/Mathematica
```

If you installed the files correctly, this simple test should give the result shown here. Note the use of the backquote ` as a machine-independent way to specify directories and files.

```
In[2]:= << CSM`Test`
The CSM packages are correctly installed in
/usr/local/Mathematica/AddOns/Applications/CSM
```

All packages mentioned in this book can be loaded by prefixing their name with the directory, CSM.

```
In[3]:= << CSM`ComplexParametricPlot`
```

Please refer to the book's Web site for up-to-date information on available archive formats and detailed installation instructions.

In addition to the programs, the book's Web site contains other information, such as notebooks, updates, a list of errata, and the archive of the mailing list intended for readers of this book. I encourage you to join the mailing list. Please see the Web site for details.

Notation and Terminology

Mathematica input and output is typeset in a typewriterlike style (in the Courier font): `Expand[(x+y)^9]`. Parts of *Mathematica* expressions not to be entered verbatim, but denoting (meta) variables, are set in italic: `f[`*var*`_] :=` *body*.

Functions or *commands* are denoted by their name, followed by an empty argument list in square brackets: `Expand[]`. Program listings are delimited by horizontal lines:

```
a[1] = a[2] = 1

a[n_Integer?Positive] := a[n] = a[a[n-1]] + a[n-1-a[n-1]]
```

A sequence by John H. Conway.

A program package is identified by name (the context name, as we shall see) – for example, `Complex`. The files used for storing successive versions of this package will be named Complex1.m, Complex2.m, and so on. The final version will be called Complex.m.

Mathematica dialog is set in two columns. The left column contains explanations; the right column contains input and output, including graphics. This form of presentation is derived from *The Mathematica Book*.

As usual, we will clarify program structure by indentation. *Mathematica* allows writing deeply nested expressions. It is, therefore, often necessary to break such expressions into multiple lines.

Here is an example of such a dialog. You would enter only the input set in boldface. The prompt `In[1]:=` is printed by *Mathematica*. If you work with the Notebook frontend, this prompt will appear *after* you evaluate your input with ENTER.

```
In[1]:= Factor[ x^34 - 1 ]

Out[1]= (-1 + x) (1 + x)

                 2    3    4    5    6    7    8    9
    (1 - x + x  - x  + x  - x  + x  - x  + x  - x  +

      10    11    12    13    14    15    16
     x   - x   + x   - x   + x   - x   + x  )

                 2    3    4    5    6    7    8    9
    (1 + x + x  + x  + x  + x  + x  + x  + x  + x  +

      10    11    12    13    14    15    16
     x   + x   + x   + x   + x   + x   + x  )
```

In most programming languages, you can define procedures, functions, or subroutines. *Mathematica* uses only one mechanism, called *definitions*, which look like f`[`x`_] :=` *def*. Chapter 2 provides a short explanation of the elements of *Mathematica*'s programming language. A more in-depth presentation is given in Appendix B.

The table on page xx lists the mathematical notations that we use. Equations, figures, program listings, and tables are numbered by section. For example, Equation 3.1–1 is the first equation in Section 3.1.

Colophon

Mathematica dialogs were computed on a Sun ULTRAsparc II with Version 4.0 of *Mathematica* using the initialization file init.m reproduced here.

```
Format[Continuation[_]] := ""

SeedRandom[10000]

Off[ General::spell, General::spell1 ]

Unprotect[Short]
Short[e_] := Short[e, 2]        (* lines are very short *)
Protect[Short]

SetOptions[ Plot3D, AspectRatio -> Automatic, PlotPoints -> 35 ]
SetOptions[ Graphics3D, AspectRatio -> Automatic ]
SetOptions[ ParametricPlot, AspectRatio -> Automatic ]
SetOptions[ ParametricPlot3D, Axes -> None ]

Needs["ProgrammingInMathematica`Options`"]
SetAllOptions[ ColorOutput -> GrayLevel ]

$DefaultFont = {"Times-Roman", 9.0} (* font in graphics *)

SetOptions["stdout", PageWidth->56] (* line width *)
```

init.m: *Mathematica* initialization for this book.

The manuscript is written in LaTeX [40] (with many custom macros). It contains only the input of the sample computations. The results were computed by *Mathematica* and were inserted

automatically into the file. The bibliography was produced with BibTeX [59], and the index was sorted with makeindex [41]. Those figures not produced with *Mathematica* were designed with FrameMaker and included in PostScript form. The reproductions of Notebooks and help screens were taken from the computer's screen. Finally, the output of LaTeX was converted into PostScript and phototypeset.

$\lg x$	logarithm to base 2, $\log_2 x$
$\log x$	natural logarithm (base e)
$\gcd(a, b)$	greatest common divisor
$a \mid b$	a divides b
$a \bmod b$	remainder when a is divided by b
$a \operatorname{div} b$	integer part of the quotient a/b
$\operatorname{sign} x$	sign of x
$n!$	n factorial, $n! = n(n-1)(n-2)\cdots 1$; $0! = 1$
$\mathbf{N}$	set of nonnegative integers $\{0, 1, 2, \ldots\}$
$\mathbf{Z}$	ring of integers $\{0, \pm 1, \pm 2, \ldots\}$
$\mathbf{Z}_p$	residue classes modulo p
$\mathbf{R}$	field of real numbers
$\mathbf{C}$	field of complex numbers
i	imaginary unit, $i = \sqrt{-1}$
$\lfloor r \rfloor$	largest integer $\leq r$
$\lceil r \rceil$	smallest integer $\geq r$
$x \approx y$	approximate equality of x and y
$\pi(x)$	number of primes $\leq x$
a^t	transpose of matrix a
$v.w$	dot product of vectors v and w
$a \otimes b$	outer product of tensors a and b
$\operatorname{div} v$	divergence of vector field v
$\operatorname{grad} s$	gradient of scalar field s
$\nabla^2 s$	Laplace operator, $\nabla^2 s = \operatorname{div} \operatorname{grad} s$
$\frac{d}{dx}$	total derivative w.r.t. x
$\frac{\partial}{\partial x}$	partial derivative w.r.t. x
$x \mapsto y$	mapping of x to y
$\lambda x.t(x)$	lambda expression (pure function)
$[x \to a]e$	substitution of x by a in e
$p \wedge q$	p AND q
$p \vee q$	p OR q
$p \to q$	p implies q
$r \cup s$	union of sets r and s
$r \cap s$	intersection of sets r and s
$r - s$	difference of sets r and s
$r \bowtie s$	join of relations r and s

Mathematical Notation Used in This Book.

Contents

Preface ix

About This Book xiii

1 Computers and Science

1.1 From Problems to Programs 3
1.2 Computers 14
1.3 Programming Languages 18
1.4 Computer Science 23

2 *Mathematica*'s Programming Language

2.1 Arithmetic and Logic 27
2.2 Definitions 30
2.3 Simple Program Structures 34
2.4 Structure of Expressions 44
2.5 Help with Problems 49
2.6 Exercises 52

3 Iteration and Recursion

3.1 The Greatest Common Divisor 57
3.2 The $3x + 1$ Problem 60
3.3 Advanced Topic: Loop Invariants 64
3.4 Application: Differential Equations 69
3.5 Exercises 76

4 Structure of Programs

4.1 Complex Parametric Lines 81
4.2 The First Package 85
4.3 Optional Arguments 90
4.4 A Template Package 94
4.5 Exercises 96

5 Abstract Data Types

5.1 Definition of Abstract Data Types 99
5.2 Example: Modular Numbers 102
5.3 Design of Abstract Data Types 106
5.4 Exercises 110

6 Algorithms for Searching and Sorting

6.1 Searching Ordered Data 115
6.2 Sorting Data 121
6.3 Binary Trees 129
6.4 Exercises 138

7 Complexity of Algorithms

7.1 Complexity of Computations 143
7.2 Example: Computing the nth Fibonacci Number 147
7.3 Special Topic: Dynamic Programming 156
7.4 Long-Integer Arithmetic and Fast Multiplication 162
7.5 Exercises 167

8 Operations on Vectors and Matrices

8.1 Vectors and Matrices 171
8.2 Inner and Outer Products 175
8.3 Linear Algebra 185
8.4 Programs with Arrays 194
8.5 Application: Aggregation 202
8.6 Exercises 207

9 List Processing and Recursion

9.1 Symbolic Expressions and Lists 213
9.2 List Processing 217
9.3 Exercises 223

10 Rule-Based Programming

10.1 Pattern Matching 227
10.2 Rules and Term Rewriting 231
10.3 Simplification Rules and Normal Forms 235
10.4 Application: Trigonometric Simplifications 241
10.5 Exercises 248

11 Functions

11.1 A Notation for Functions (λ Calculus) 255
11.2 Functions as Values 260
11.3 Example: Simulation of Shift Registers 266
11.4 Exercises 268

12 Theory of Computation

12.1 Computable Functions 273
12.2 Models of Computation 280
12.3 Turing Machines 282
12.4 Recursive Functions Are Turing Computable 293
12.5 Exercises 299

13 Databases

13.1 Database Design . . . 303
13.2 Relational Databases . . . 308
13.3 Data Entry and Queries . . . 314
13.4 Commercial Databases . . . 319
13.5 Exercises . . . 321

14 Object-Oriented Programming

14.1 Introduction . . . 325
14.2 Example: Bank Accounts . . . 327
14.3 Principles of Object-Oriented Programming . . . 330
14.4 Application: Collections . . . 332
14.5 Exercises . . . 341

Appendix A Further Reading

A.1 A Guide to the Literature . . . 345
A.2 References . . . 347

Appendix B More Information About *Mathematica*

B.1 Computations You Can Do with *Mathematica* . . . 353
B.2 The Code for the Illustrations in this Book . . . 364
B.3 *Mathematica*'s Evaluation Method . . . 368
B.4 Syntax of Operators . . . 374

Index . . . 377