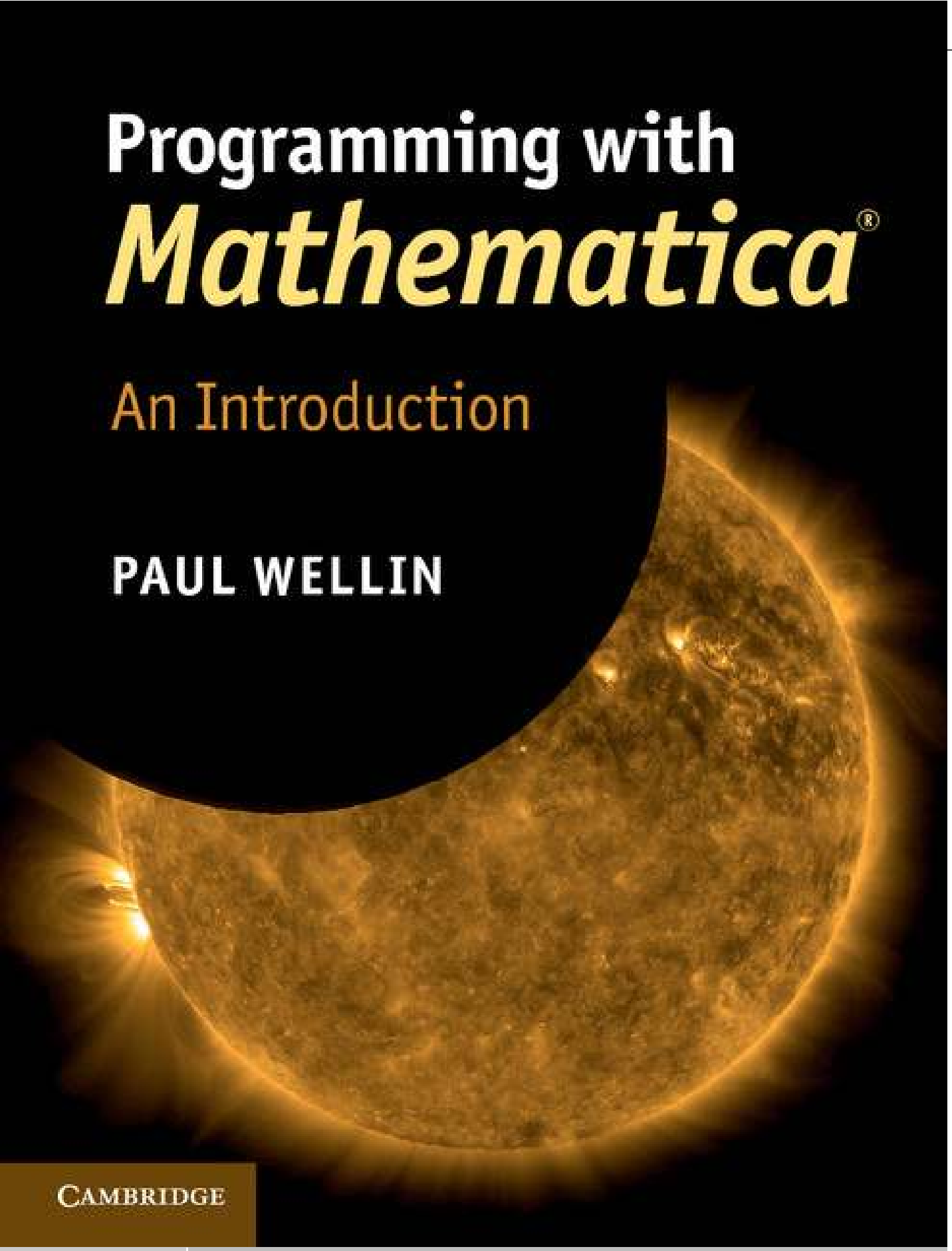
Programming with Mathematica®

An Introduction

PAUL WELLIN



CAMBRIDGE

Programming with *Mathematica*®

An Introduction

Starting from first principles, this book covers all of the foundational material needed to develop a clear understanding of the *Mathematica* language, with a practical emphasis on solving problems. Concrete examples throughout the text demonstrate how *Mathematica* can be used to solve problems in science, engineering, economics/finance, computational linguistics, geoscience, bioinformatics, and a range of other fields.

- Assumes no formal knowledge of programming.
- Over 285 exercises give the reader plenty of practice using the language to solve problems.
- Ideal for self-study, or for anyone wishing to further their understanding of *Mathematica*.
- *Mathematica* notebooks containing examples, programs and solutions to exercises are available from www.cambridge.org/wellin.

Paul Wellin worked for Wolfram Research from the early-1990s through 2011, directing the *Mathematica* training efforts with the Wolfram Education Group. He has taught mathematics at both public schools and at the university level for over 12 years. He has given talks, workshops, and seminars around the world on the integration of technical computing and education and he has served on numerous government advisory panels on these issues. He is the author of several books on *Mathematica*.

Programming with *Mathematica*®

An Introduction

PAUL WELLIN

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town,
Singapore, São Paulo, Delhi, Mexico City

Cambridge University Press
The Edinburgh Building, Cambridge CB2 8RU, UK

Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9781107009462

Text set in DTL Albertina 11/13 pt; captions set in Syntax LT System *Mathematica*®.
Designed and typeset by the author.

First published 2013

Printed and bound in the United Kingdom by the MPG Books Group

A catalogue record for this publication is available from the British Library

ISBN 978-1-107-00946-2 Hardback

Additional resources for this publication at www.cambridge.org/wellin

Contents

Solutions to exercises

Preface

Programming with *Mathematica*

Well-designed tools are not simply things of beauty to be admired. They are, above all, a joy to use. They seem to have their own consistent and readily apparent internal logic; using them seems natural – intuitive even – in that it is hard to imagine using any other tool, and, typically, a minimal amount of effort is required to solve the problem for which those tools were designed. You might even begin to think that your problems were designed for the tool rather than the other way around.

Programming with *Mathematica* is, first and foremost, a joy. Having used various programming languages throughout my life (starting with ALGOL and FORTRAN), it is now hard for me to imagine using a tool other than *Mathematica* to solve most of the computational problems that I encounter. Having at my fingertips an extremely well-thought-out language, combined with tools for analysis, modeling, simulation, visualization, interface creation, connections to other technologies, import and export, seems to give me everything I might need.

Ultimately though, no tool can solve every problem you might encounter; what really makes *Mathematica* the indispensable tool for many computational scientists, engineers, and even artists and musicians, is its capability for infinite extension through programming. As a language, built upon the shoulders of such giants as LISP, PROLOG, APL and C++, *Mathematica* has extended some of the best ideas from these languages and created some new ones of its own. A powerful pattern matching language together with a rule-based paradigm for transforming expressions provides for a natural approach to writing programs to solve problems. By "natural" I mean a quick and direct implementation, one that mirrors as closely as possible the statement of the problem to be solved. From there, it is just a short path to prototyping and eventually a program that can be tested for correctness and efficiency.

But there are tools, and there are tools! Some tools are very domain-specific, meaning that they are designed for a narrow set of tasks defined by a certain discipline or framework and are inappropriate for tasks outside of their domain. But *Mathematica* has taken a different approach. It provides broadly useful tools by abstracting the computational tasks (through symbolic expression manipulation) in such a way that it has found wide use in fields as varied as genomics and bioinformatics, astronomy, image processing, social networks, linguistics, and much more.

In addition to the breadth of fields that can be addressed with *Mathematica*, the variety and extent of the computational tasks that now challenge us have greatly expanded since the turn of the millennium. This is due to the explosion in the sheer amount of information and data that people study. This expansion mirrors the rapid growth in computer hardware capabilities of the 1990s and 2000s which saw speed and storage grow exponentially. Now the challenge is to find software solutions that are up to the task of managing this growth in information and data. Given the variety of data objects that people are interested in studying, tools that provide generality and avoid domain-specific solutions will be the most broadly useful across disciplines and across time. *Mathematica* has been around now for over two decades and it continues to find application in surprising places.

Using this book

This book is designed for anyone who wants to learn how to write *Mathematica* programs to solve problems. It does not presuppose a formal knowledge of programming principles as taught in a modern course on a language such as C or JAVA, but there is quite a bit of overlap between this material and what you would expect in such a formal course. You will learn about the basic building blocks of the *Mathematica* language: expressions; the syntax of that language; and how to put these objects together to make more complicated expressions. But it is more than just a primer on the language. The focus is on solving problems and, as such, this is an example-driven book. The approach here is practical. Programming is about solving problems and besides the obvious necessity of learning the rules of the language, many people find it instructive and concrete to see concepts put into action. The book is packed with examples both in the text proper and in the exercises. Some of these examples are quite simple and straightforward and can be understood with a modicum of understanding of *Mathematica*. Other examples and exercises are more involved and may require a bit more study before you feel that you have mastered the underlying concepts and can apply them to related problems. Since this book is written for readers with various backgrounds in programming languages and using *Mathematica*, I think it best to not identify "levels of difficulty" with the examples and exercises.

Becoming a proficient programmer requires not only a clear understanding of the language but also practice using it. As such, one of the aims of this book is to provide the novice with examples of good programming style and practice. Many of the examples in the chapters are, by design, concise, in order to focus on a concept that is being developed. More involved examples drawing together several different conceptual ideas appear in the examples and applications sections at the end of many of the chapters. Depending upon your needs and level of expertise, you can either start with first principles, move on to basic examples, and then to more involved applications of these concepts, or you might find yourself looking at interesting examples and then, as the need arises, jumping back into the discussion of syntax or usage earlier in a chapter.

The exercises (over 290 of them) are designed to extend and expand upon the topics discussed in the chapters in which they occur. You cannot learn how to program by simply reading a book; the old maxim, "you learn by doing" is as true of learning how to speak a foreign (natural) language as it is true of learning a computer programming language. Try to do as many exercises as you can; create and solve problems that interest you; "life is not a spectator sport" and neither is learning how to program.

Due to resource limitations, all the solutions could not be included in the printed book. Fortunately, we live in an age of easily disseminated information, and so you will find an extended set of solutions to most of the exercises in both notebook and PDF format at www.cambridge.org/wellin. In addition, many of the programs developed in the sections and exercises are included as packages at the same website.

Scope of this book

This book evolved from an earlier project, *An Introduction to Programming with Mathematica*, the third edition of which was also published by Cambridge University Press. As a result of several factors, including a long time between editions, much new material due to major upgrades in *Mathematica*, the original authors traveling different paths – it seemed as if a new title was in order, one that both reflects and builds upon this history while incorporating the latest elements of *Mathematica* itself.

The several versions of *Mathematica* that have been released since the third edition of *An Introduction to Programming with Mathematica* was published now include extensive coverage in new application areas, including image processing, control systems, wavelets, graphs and networks, and finance. The present book draws from many of these areas in the never-ending search for good examples that not only help to illustrate conceptual problems, but also serve as interesting and enlightening material on their own. The examples, exercises, and applications draw from a variety of fields, including:

- *textual analysis and natural language processing*: corpus linguistics, word stemming, stop words, comparative textual analysis, scraping websites for data, sorting strings, bigrams and *n*-grams, word games (anagrams, blanagrams, palindromes), filtering text;
- *bioinformatics*: analysis of nucleotide sequences, computing GC ratios, displaying blocks of genetic information, searching for subsequences, protein-protein interaction networks, dot plots;
- *computer science*: hashing (checksums), encoding/encryption, sorting, adjacency structures, triangular numbers, Hamming numbers, Fibonacci numbers, Euler numbers, root finders, random number generation algorithms, sieving;
- *finance and economics*: time-series analysis, trend plots, stock screens;

- *data analysis*: filtering signals, cleaning data, stem plots, statistical tests, lag plots, correlograms, visualizing spread of data;
- *geometry*: convex hull, diameter of pointsets, point-in-polygon problems, traveling salesman-type problems, hypocycloids and epicycloids, Apollonius' circle;
- *image processing*: resizing, filtering, segmentation;
- *graphs and networks*: random graphs, regular graphs, bond percolation, connected components.

Chapter 1 is designed as a brief tour of the current version of *Mathematica* as of the publication of this book. The examples give a sense of the scope of *Mathematica*'s usage in science, engineering, and other analytic fields. Included is a basic introduction to the syntax of *Mathematica* expressions, working with the *Mathematica* interface, and also pointers to the documentation features.

Several important topics are introduced in Chapter 2 that are used throughout the book, in particular, structure of expressions, evaluation of expressions, various aspects of function definitions, predicates, relational and logical operators, and attributes.

Lists are an essential data type in *Mathematica* and an understanding of how to work with them provides a practical framework for the generalization of these ideas to arbitrary expressions. Chapter 3 focuses on structure, syntax, and tools for working with lists. These topics are all extended in later chapters in the context of various programming tasks. Included in this chapter are discussions of functions for creating, displaying, testing, measuring lists, various visualization tools, arrays (sparse and otherwise), list component assignment, and using `Span` to extract ranges of elements.

Patterns and rules are introduced in Chapter 4. Even though pattern-based programming may be new to many, patterns are so essential to all programming in *Mathematica*, that it seems most natural to introduce them at this point and then use them in later chapters on functional and procedural programming. Topics include a discussion of structured patterns, conditional patterns, sequence pattern matching, using data types to match an expression, repeated patterns, replacement rules, and numerous examples of functions and programs that make heavy use of pattern matching.

The chapter on functional programming (Chapter 5) introduces the many functions built into *Mathematica* associated with this programming paradigm: `Map`, `Apply`, `Thread`, `Outer`, `Select`, `Pick`, and many others. Scoping constructs are explicitly called out in a separate section. A section on pure functions includes numerous examples to help understand this important construct in the context of concrete problems. Adding options, error trapping and messaging, so important for well-designed functions and programs, are discussed in this chapter so that they can be used in all that follows. Numerous applied examples are included such as protein

interaction networks, Hamming distance, defining new graphics objects, creating palettes for project files, and much more.

Procedural programming may be most familiar to those who learned programming in a more traditional language such as FORTRAN or C. The syntax of procedural programming in *Mathematica* is quite similar to that in C and Chapter 6 is designed to help you transition to using *Mathematica* procedurally but also mixing it with other programming styles when and where appropriate. Looping constructs and their syntax are discussed in terms of basic examples which are then built upon and extended in the remainder of the book. Included are piecewise-defined functions, flow control, and several classical examples such as sieving for primes and sorting algorithms.

The chapter on recursion, Chapter 7, gives a basic introduction to programming recursively-defined functions. The main concepts – base cases, recursion on the tail, recursion with multiple arguments, and so on – are introduced through illustrative examples. The chapter concludes with a discussion of dynamic programming, a technique for greatly speeding up recursive computations by automatically creating definitions at runtime.

Chapter 8 introduces the various types of number you can work with in *Mathematica* – exact, machine-precision, arbitrary-precision as well as different number types and arrays of numbers. It includes an extended discussion of random number generators and functions for sampling and choosing random numbers. The examples and applications section includes a program to compute the radius of gyration tensor of a random walk as well as material on statistical tests, both built-in and user-defined tests for checking the randomness of sequences of numbers.

The chapter on strings, Chapter 9, is included in recognition of the ubiquity of these objects in broad areas of science, engineering, linguistics, and many other fields. Topics include an introduction to the structure and syntax of strings, basic operations on strings including those that mirror similar operations on lists, an extensive discussion on string patterns including regular expressions such as are found in languages like PERL and PYTHON, and many applications and examples drawn from linguistics, computer science, and bioinformatics.

Chapter 10 on visualization is designed to give you a good sense of the symbolic graphics language so that you can both create your own graphics scenes and functions and also make your objects as efficient as possible. Included is a discussion of primitives, directives, and options, all of which is mirrored in the section on sound. A section on efficient graphics structures is included that discusses multi-objects such as multi-points and multi-lines, as well as material on `GraphicsComplex`, a compact way to represent a graphical object with many repeated primitive elements. Many extended examples are included for functions to plot points in space connected by lines, economic or financial trend plots, space-filling molecule plots for proteins and other chemicals, and root plotting functions.

Dynamic objects were introduced in *Mathematica* 6, and there have, sadly, been few resources for learning the ins and outs of dynamic programming. Dynamic objects provide tools to create

interactive elements in your documents from as simple as an animation to as complex as…well, as complex as you can imagine. In Chapter 11 we introduce dynamic objects, starting with top-level functions `Animate` and `Manipulate`, moving on to viewers and various control objects that can be used to control changing parameters. The primitive elements that lie underneath all these top-level functions are `Dynamic` and `DynamicModule`, which are the foundations of the entire interactive machinery now built into *Mathematica*. The chapter closes with several applications including building up interfaces to work with multi-dimensional data, extending work earlier in the book on palettes for file openers, event handlers to interact more with your mouse, and a simple geometry demonstration due to Apollonius.

As a result of the many comments and suggestions from people in the broad *Mathematica* community, I have included a chapter on writing efficient programs, Chapter 12. Although there are many approaches you might take to solve a problem, it is often difficult for the novice to tell which is the most appropriate, or the most efficient, or which scales best. Several "good practices" are considered, including choosing the right function, choosing the right algorithm, listability, pure functions, packed arrays, and so on. Sections on parallel computation and on compiling are also included. These issues are discussed through the use of concrete examples drawn from earlier parts of the book.

The chapter on applications, Chapter 13, builds upon much of the work in the rest of the book but extends it for those who wish to turn their code into programs and applications that can be shared with colleagues, students, or clients. The focus is on making your *Mathematica* programs as much like built-in functions as possible, thereby taking advantage of the interface elements that a user of your code would already know and expect from working in *Mathematica*, things like writing modular functions, usage messages, overloading, and creating and working with packages.

In trying to keep this book both introductory and concise, many topics had to be left out. Some of these topics include: creation of new data types; the internals for ordering of rules; upvalues, downvalues and other internal transformation rules; tuning and debugging; connecting to external programs and databases; interacting with web servers. All of these topics are both interesting and important but there was simply not enough room in the present volume to include them.

Colophon

This book was written and developed in *Mathematica*. Stylesheets were created to the page specifications designed by the author while adhering to the constraints of the publisher's production department. Pages were output to PostScript and then distilled to PDF with Adobe Distiller using a configuration file supplied by the publisher to set such parameters as resolution, font embeddings, as well as color and image conversions.

The text for this book, including mathematical formulas, is set in *Albertina*, a humanist font designed by the Dutch calligrapher Chris Brand (1921–1999), and digitized by the Dutch Type Library (DTL). Captions and labels use the fairly animated sans serif *Syntax*, designed by the Swiss typographer Hans Eduard Meier (1922–).

Acknowledgments

Although writing a book may appear to others as a solitary project, authors know better. I consider myself very fortunate to have had wonderful colleagues to work with and have benefited in innumerable ways from their expertise. The following people provided concrete help in discussing various topics and answering my many questions: Darren Glosemeyer on date plotting functions, statistical tests, and statistical plots; Harry Calkins on graphics and general language issues; Charles Pooh on graphs and networks; Dan Lichtblau on internal algorithms and numerous language issues; Michael Kelly for some suggestions on trend lines implementation; Adriano Pascoletti for permission to use and modify his code for computing points in nonconvex polygons; Tom Sherlock and Faisal Whepley for help on front-end related issues; Oyvind Tafjord for various questions and issues with string manipulation and regular expressions; Andre Kuzniarek and Larry Adelston for layout and production questions.

In addition, I am grateful to the reviewers who provided valuable feedback on early drafts of this book: Harry Calkins, Darren Glosemeyer, Mariusz Jankowski, Dan Lichtblau, and Oyvind Tafjord. Any mistakes that remain are mine and mine alone. If you think you have found one, please let me know so that I can update an errata page on the publisher's website as well as in any future printings of this book.

The entire editorial and production stages of this project have been miraculously smooth, in no small part due to the team at Cambridge University Press. In particular, my editor, David Tranah and his team, have been both supportive and encouraging throughout the project, providing all that an author can ask for. Clare Dennison and Abigail Jones were most helpful on the innumerable editorial and production details that accompany a book project such as this.

Loved ones are the unnamed partners in writing a book. Although unrecognized to the reader, they nonetheless play a critical role for the author. They provide nourishment (in its many guises), support, feedback, and that all-too-critical element, time. I have been blessed with a supportive family throughout this project. In particular, my wife Sheri has lovingly provided all these things and more.

Finally, I would like to dedicate this book to the memory of a very special friend, Bob Johnson. Bob was the person most responsible for getting me involved with *Mathematica* when, back in 1989, as chair of the mathematics department at Sonoma State University, he asked me to join him in the basement (computers were always in basements in those days!?) at Sonoma State and we took our first look at a strange new program called *Mathematica* running on a strange new

computer housed in a strange black magnesium cube. The excitement of realizing that the worlds of mathematics, science, and engineering would be dramatically changed by this new program was matched by the joy Bob and I experienced in learning how to incorporate this tool into our research and teaching. Bob was that unusual person who knew how to keep his eyes on the prize and his encouragement of my efforts made a huge difference in my life and in the lives of others as well. Thanks Bob.

Paul R. Wellin

I

An introduction to *Mathematica*

Overview of basic operations · Numerical computation · Symbolic computation · Graphics and visualization · Data import and analysis · Dynamic and interactive computation · Programming · Starting up Mathematica · Notebook interface · Entering input · Mathematical expressions · Syntax of functions · Lists · Dealing with errors · Help and documentation

Mathematica is a very large and seemingly complex system. It contains thousands of functions for performing various tasks in science, mathematics, engineering, and many other disciplines. These tasks include numerical and symbolic computation, programming, data analysis, knowledge representation, and visualization of information. In this introductory chapter, we give a sense of its breadth and depth by looking at some computational and programming examples drawn from a variety of fields. The last part of the chapter covers basic topics in getting started, including how to enter and evaluate expressions, how to deal with errors, and how to get help, with pointers to the documentation system. Users already familiar with *Mathematica* could lightly skim this chapter.

1.1 Overview of basic operations

Numerical and symbolic computation

On a very basic level, *Mathematica* can be thought of as a sophisticated calculator. With it you can enter mathematical expressions and compute their values.

```
In[1]:= Sqrt[2.0 * 10 Pi] (10/E)^10
Out[1]= 3.5987 * 10^6
```

You can store values in memory to be used in subsequent computations. For example, the following three inputs compute the Lorentz factor for an object moving at half the speed of light.

```
In[2]:= c = 299 792 458 Meter/Second;

In[3]:= v = c/2

Out[3]= 149 896 229 Meter/Second

In[4]:= N[Sqrt[1 - v^2/c^2]]

Out[4]= 0.866025
```

Yet *Mathematica* differs from calculators and simple computer programs in its ability to calculate exact results and to compute to an arbitrary degree of precision.

```
In[5]:= 1/2 + 1/3 + 1/5 + 1/7 + 1/11 + 1/13

Out[5]= 40 361/30 030

In[6]:= 2^1024

Out[6]= 179 769 313 486 231 590 772 930 519 078 902 473 361 797 697 894 230 657 273 ⁖
        430 081 157 732 675 805 500 963 132 708 477 322 407 536 021 120 113 879 ⁖
        871 393 357 658 789 768 814 416 622 492 847 430 639 474 124 377 767 893 ⁖
        424 865 485 276 302 219 601 246 094 119 453 082 952 085 005 768 838 150 ⁖
        682 342 462 881 473 913 110 540 827 237 163 350 510 684 586 298 239 947 ⁖
        245 938 479 716 304 835 356 329 624 224 137 216

In[7]:= N[Sin[2017 × 2^(1/5)], 40]

Out[7]= -0.9999999999999999978567771261060983259068 5
```

One of the most significant features of *Mathematica* is its ability to manipulate and compute with symbolic expressions. For example, you can factor polynomials and simplify trigonometric expressions.

```
In[8]:= Factor[x^7 - 1]

Out[8]= (-1 + x) (1 + x + x^2 + x^3 + x^4 + x^5 + x^6)
```

```
In[9]:= TrigReduce[Sin[3 θ]^5]
```

$$\text{Out[9]}= \frac{1}{16}\left(10\,\text{Sin}[3\,\theta] - 5\,\text{Sin}[9\,\theta] + \text{Sin}[15\,\theta]\right)$$

You can simplify expressions using assumptions about variables contained in those expressions. For example, if k is assumed to be an integer, $\sin(2\pi k + x)$ simplifies to $\sin(x)$.

```
In[10]:= Assuming[k ∈ Integers, Simplify[Sin[2 π k + x]]]
Out[10]= Sin[x]
```

Functions are available for solving systems of equations, for example, this solves a symbolic 2×2 linear system.

```
In[11]:= LinearSolve[{{a11, a12}, {a21, a22}}, {x1, x2}]
```

$$\text{Out[11]}= \left\{\left\{\frac{a_{22}\,x_1 - a_{12}\,x_2}{-a_{12}\,a_{21} + a_{11}\,a_{22}}\right\}, \left\{\frac{a_{21}\,x_1 - a_{11}\,x_2}{a_{12}\,a_{21} - a_{11}\,a_{22}}\right\}\right\}$$

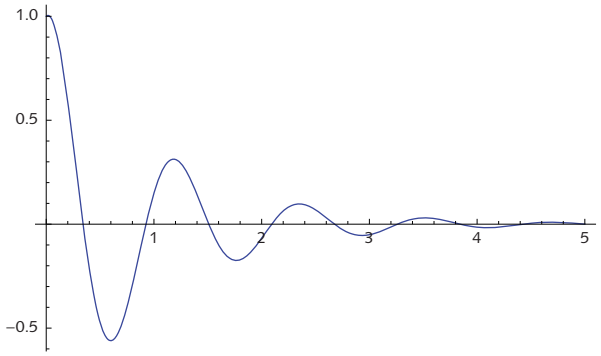
You can solve and plot solutions to differential equations, for example, a system representing a linear damped pendulum.

```
In[12]:= soln = DSolve[{y''[x] + 2 y'[x] + 30 y[x] == 0,
           y[0] == 1, y'[0] == 1 / 2}, y[x], x]
```

$$\text{Out[12]}= \left\{\left\{y[x] \to \frac{1}{58}\,e^{-x}\left(58\,\text{Cos}\left[\sqrt{29}\,x\right] + 3\sqrt{29}\,\text{Sin}\left[\sqrt{29}\,x\right]\right)\right\}\right\}$$

```
In[13]:= Plot[y[x] /. soln, {x, 0, 5}, PlotRange → All]
```

Out[13]=



You can create and then operate on functions that are defined piecewise.

```
In[14]:= sinc[x_] = Piecewise[ {{1, x == 0}}, Sin[x] / x]

Out[14]= { 1          x == 0
         { Sin[x]/x   True
```

```
In[15]:= Integrate[ sinc[x^2] / x, x]

Out[15]= CosIntegral[x^2]/2 - Sin[x^2]/(2 x^2)
```

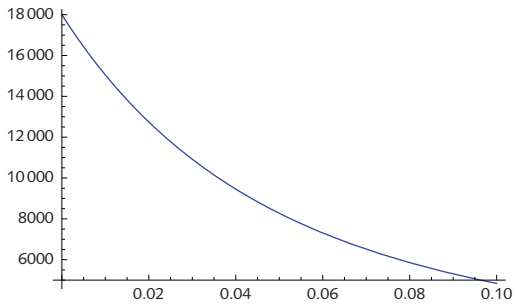
One of the advantages of working symbolically is that you can quickly see underlying formulas and algorithms at work. For example, this computes a present value for an annuity of 36 payments of $500 using a symbolic effective interest rate.

```
In[16]:= presentValue = TimeValue[Annuity[500, 36], r, 0]

Out[16]= 500 (-1 + (1 + r)^36) / (r (1 + r)^36)
```

A plot clearly shows the relationship between the interest rate and the present value of the annuity.

```
In[17]:= Plot[presentValue, {r, 0.0, 0.10}]
```



In fact, symbolic expressions are very general objects – you can work with them as you would any expression.

```
In[18]:= Factor[ ■^7 - 1]

Out[18]= (-1 + ■) (1 + ■ + ■^2 + ■^3 + ■^4 + ■^5 + ■^6)
```