Preface

> I've developed an obscene interest in computation, and I'll be returning to the United States a better and impurer man.
> —John von Neumann

It might seem that computing should simply be a matter of translating formulas from the page to the machine. But even when such formulas are known, applying them in a numerical fashion requires care. For instance, rounding off numbers at the sixteenth significant digit can lay low such stalwarts as the quadratic formula! Fortunately, the consequences of applying a numerical method to a mathematical problem are quite understandable from the right perspective. In fact, it is our mastery of what can go wrong in some approaches that gives us confidence in the rest of them.

If mathematical modeling is the process of turning real phenomena into mathematical abstractions, then numerical computation is largely about the transformation from abstract mathematics to concrete reality. Many science and engineering disciplines have long benefited from the tremendous value of the correspondence between quantitative information and mathematical manipulation. Other fields, from biology to history to political science, are rapidly catching up. In our opinion, a young mathematician who is ignorant of numerical computation in the 21st century has much in common with one who was ignorant of calculus in the 18th century.

To the student

Welcome! We expect that you have had lessons in manipulating power series, solving linear systems of equations, calculating eigenvalues of matrices, and obtaining solutions of differential equations. We also expect that you have written computer programs that take a nontrivial number of steps to perform a well-defined task, such as sorting a list. Even if you have rarely seen how these isolated mathematical and computational tasks interact with one another, or what they have to do with practical realities, you are part of the intended audience for this book.

Based on our experiences teaching this subject, our guess is that some rough seas may lie ahead of you. Probably you do not remember learning all parts of calculus, linear algebra, differential equations, and computer science with equal skill and fondness.

This book draws from all of these areas at times, so your weaknesses are going to surface once in a while. Furthermore, this may be the first course you have taken that does not fit neatly within a major discipline. Von Neumann's use of "impurer" in the quote above is a telling one: numerical computation is about solving problems, and the search for solution methods that work well can take us across intellectual disciplinary boundaries. This mindset may be unfamiliar and disorienting at times.

Don't panic! There is a great deal to be gained by working your way through this book. It goes almost without saying that you can acquire computing skills that are in much demand for present and future use in science, engineering, and mathematics—and, increasingly, in business, social science, and humanities, too. There are less tangible benefits as well. Having a new context to wrestle with concepts like Taylor series and matrices may shed new light on why they are important enough to learn. It can be rewarding to apply skills to solve relatable problems. Finally, the computational way of thought is one that complements other methods of analysis and can serve you well.

MATLAB

All of the coded functions and most of the examples are presented in MATLAB. MATLAB was invented by an academic numerical analyst who wanted better tools for teaching the subject. Today MATLAB is a behemoth with endless bells and whistles, and it is crucial to major industrial sectors. It has both fans and detractors. At its core, though, MATLAB remains an environment that is excellent for lightweight, easily understood expression and exploration of the concepts that are in this text.

While a student should have one standard programming course under his or her belt before starting this book, prior knowledge of MATLAB should not be necessary. We encourage you to download the examples and execute them yourself (details on accessing them are given below). Not only should you see on the screen exactly what is in the book, but you should try to make changes and, in the words of Mark Zuckerberg, "move fast and break things." As with a natural language, the easiest path to fluency is by imitating a proficient speaker and gradually building your vocabulary as the need arises.

If you prefer a more to-the-point introduction only to MATLAB, there are lots of videos and tutorial resources online (for example, search for "matlab academy"), and we can shamelessly recommend at least one book: *Learning MATLAB*, by one of us (Driscoll).

To the instructor

First of all, don't overlook the online content—you can find the details below.

The plausibly important introductory material on numerical computation for the majority of undergraduate students easily exceeds the capacity of two semesters—and of one textbook. As instructors and as authors, we face difficult choices as a result. We set aside the goal of creating an agreeable canon. Instead we hope for students to experience an echo of that "obscene interest" that von Neumann so gleefully described

and pursued. For while there are excellent practical reasons to learn about numerical computing, it also stands as a subject of intellectual and even emotional relevance. We have seen students excited and motivated by applications of their newly found abilities to problems in mechanics, biology, networks, finance, and more—problems that are of unmistakable importance in the jungle beyond university textbooks, yet largely impenetrable using only the techniques learned within our well-tended gardens.

In writing this book, we have not attempted to be encyclopedic. We're sorry if some of your favorite topics don't appear or are minimized in the book. (It happened to us too; many painful cuts were made from prior drafts.) But in an information-saturated world, the usefulness of a textbook lies with teaching process, not content. We have aimed not for a cookbook but for an introduction to the principles of cooking.

Still, there *are* lots of recipes in the book—it's hard to imagine how one could become a great chef without spending time in the kitchen! Our language for these recipes is MATLAB, for a number of reasons: it is precise, it is executable, it is as readable as could be hoped for our purposes, it rewards thinking at the vector and matrix level, and (at this writing) it is widespread and worth knowing. There are 46 standalone functions and over 150 example scripts, all of them downloadable exactly as seen in the text. Some of our codes are quite close to production quality, some are serviceable but lacking, and others still are meant for demonstration only. Ultimately our codes are mainly meant to be clear, not ideal. We try to at least be explicit about the shortcomings of our implementations.

Just as good coding and performance optimization are secondary objectives of the book, we cut some corners in the mathematics as well. We state and in some cases prove the most essential and accessible theorems, but this is not a theorem-oriented book, and in some cases we are content with less precise arguments. We have tried to make clear where solid proof ends and where approximation, estimation, heuristics, and other indispensable tools of numerical analysis begin.

The examples and exercises are meant to show and encourage a numerical mode of thought. As such they often focus on issues of convergence, experimentation leading to abstraction, and attempts to build upon or refine presented ideas. Some exercises follow the premise that an algorithm's most interesting mode of operation is failure. We expect that any learning resulting from the book is likely to take place mostly from careful study of the examples and working through the problems.

Contents

Chapter 1 explains how computers represent numbers and arithmetic, and what doing so means for mathematical expressions. Chapter 2 discusses the solution of square systems of linear equations and, more broadly, basic numerical linear algebra. Chapter 3 extends the linear algebra to linear least squares. These topics are the bedrock of scientific computing, because "everything" has multiple dimensions, and while "everything" is also nonlinear, our preferred starting point is to linearize.

Chapters 4 through 6 introduce what we take to be the rest of the most common problem types in scientific computing: roots and minimization of algebraic functions,

piecewise approximation of functions, numerical analogs of differentiation and integration, and initial-value problems for ordinary differential equations. We also explain some of the most familiar and reliable ways to solve these problems, effective up to a certain point of size and/or difficulty. Chapters 1 through 6 can serve for a single-semester survey course. If desired, Chapter 6 could be left out in favor of one of Chapters 7, 8, or 9.

The remaining chapters are intended for a second course in the material. They go into more sophisticated types of problems (eigenvalues and singular values, boundary value problems, and partial differential equations), as well as more advanced techniques for problems from the first half (Krylov subspace methods, spectral approximation, stiff problems, boundary conditions, and tensor-product discretizations).

Online content

We have created many downloadable materials for this book. There will be a maintained repository at `www.siam.org/books/ot154`. The materials include the following:

- All of the functions presented in the book.
- All of the computational examples, written as scripts that you can execute to reproduce the results.
- Slides and videos related to presenting the material in class.
- Laboratory exercises useful for hands-on problem solving with expert supervision. These draw from a wide variety of practical and mathematical applications. They have been used by the authors.
- Project ideas, typically motivated by real-world applications. These too have been used by the authors.

Acknowledgments

We are, of course, deeply indebted to all who taught us and inspired us about numerical computation over the years. We are thankful to Rodrigo Platte, who used the book before it was fully baked and offered numerous suggestions. We thank an enthusiastic group of grad students for proofreading help: Samuel Cogar, Shukai Du, Kristopher Hollingsworth, Rayanne Luke, Navid Mirzaei, Nicholas Russell, and Osman Usta. We are also grateful to Paula Callaghan and the publishing team at SIAM, whose dedication to affordable, high-quality books makes a real difference in the field.

We thank our families for their support and endurance. Last but not least, we are grateful to the many students at the University of Delaware who have taken courses based on iterations of this book. Their experiences are what convinced us that the project was worth finishing.

About the cover

- *(front cover, top left)* The Z4 digital computer was completed in Berlin in 1945 by Konrad Zuse, who hoped to sell it commercially. It could only perform 2.5 additions per second and used movie film punched with holes for input; the film could literally be looped for repeated instructions. The Z4 was hastily moved out of Berlin at the end of World War II to keep it out of Soviet hands, and it was stored in various houses and barns. In 1949 Eduard Stiefel (one of the co-discoverers of the conjugate gradients algorithm) had it brought to the ETH in Zürich.

- *(front cover, top right)* The UNIVAC 9400 was introduced by the Sperry Rand Corporation in 1967. It was programmed using then-standard 80-column punch cards. The 9400 had a maximum internal storage of 131,072 bytes and could perform over 10^5 additions per second.

- *(front cover, middle right)* The SWAC was built by the U. S. National Bureau of Standards (now the National Institute of Standards and Technology) in 1950. It was the fastest computer in the world, able to do 15,000 additions per second. In the words of Alexandra Forsythe, who, with her husband George, was among the early users and legendary pioneers in computer science, "you ran everything twice and if the results agreed you figured it was ok." It played a key role in the Nobel Prize in Chemistry awarded to Dorothy Hodgkin for discovering the structure of vitamin B_{12}.

- *(front cover, bottom right)* The ENIAC, completed in 1946 at the University of Pennsylvania, was one of the first general-purpose computers. It weighed over 27,000 kg and could add 5,000 times per second. The ENIAC took weeks to program, requiring the manual manipulation of switches and cables. In the photo are Betty Jennings (left) and Frances Bilas (right), two of the six all-female programmers.

- *(front cover, bottom left)* The TRADIC computer, created at Bell Labs in 1954, was among the first in the world to rely on transistors (684 of them) rather than vacuum tubes for computation. Its lead developer was Jean Felker, who promised the U. S. Air Force "a computer as reliable as a hammer!" It could perform 62,000 additions per second. The TRADIC was programmed using removable boards that could each hold up to 128 machine instructions, including a reusable subroutine.

- *(rear cover)* The authors are standing (TAD at right) at the University of Delaware with a sculpture called *Reconnect*, by Ronald Longsdorf. The six-foot-plus steel and resin column of computer parts is shrouded by connecting cables. (Photo by Andrew Bernoff.)