Preface

About This Book

This book is designed to be used by mathematicians, engineers, and computer scientists as a graduate-level introduction to numerical analysis and its methods. Readers are expected to have had courses or experience in calculus, linear algebra, complex variables, differential equations, and programming. Of course, many students will be missing some of that material, and we encourage generalized review, especially of linear algebra.

The book is intended to be suitable both for one-semester and for two-semester courses. It gathers important and recent material from floating-point arithmetic, numerical linear algebra, polynomials, interpolation, numerical differentiation and integration, and numerical solutions of differential equations. Our guiding principle for the selection of material and the choice of perspective is that numerical methods should be discussed as a part of a more general practice of mathematical modeling as is found in applied mathematics and engineering. Once mostly absent from texts on numerical methods, this *desideratum* has become an integral part of much of the active research in various fields of numerical analysis (see, e.g., Enright 2006a). However, because the intended audience is so broad that we cannot really presume a common background in application material, while we focus on applicable computational mathematics, we will not present many actual applications. We believe that the best-compromise approach is to use a perspective on the quality of numerical solution known as *backward error analysis*, together with the theory of *conditioning* or *sensitivity of a problem*, already known to Turing and widely practiced and written on by J. H. Wilkinson, W. Kahan, and others.[1] These ideas, very important although not a panacea, will be introduced progressively. The basic underpinning of the backward error idea, that a numerical method's errors should be analyzable in

[1] The first explicit use of backward error analysis is credited by Wilkinson (1971) to Wallace Givens, and indeed, it is already present in Von Neumann and Goldstine (1947) (see also Grcar 2011), but it is broadly agreed that it was Wilkinson himself who began the systematic exploitation of the idea in a broad collection of contexts.

the same terms as whatever physical (or chemical or biological or social or what-have-you) modeling errors, is readily understandable across all fields of application. As Wilkinson (1971 p. 554) pointed out, backward error analysis

> has the advantage that rounding errors are put on the same footing as errors in the original data and the effect of these has usually to be considered in any case.

The notion of the sensitivity of the problem to changes in its data is also one that is easy to get across to application-oriented students. As Chap. 1 explains, this means that we favor a residual-based a posteriori type of backward error analysis that provides numerical solutions that are readily interpretable in the broader context of mathematical modeling.

The pedagogical problem that (we hope!) justifies the existence of this book is that even though many excellent numerical analysis books exist, no single one of them that we know of is suitable for such a broad introductory graduate course—at least, not one that provides a unifying perspective based on the concept of backward error analysis, which we think is the most valuable aspect of this present book. Some older books *do* hold this perspective, most notably Henrici (1982), but that book is dated in other respects nowadays.

Other differences between this book and the general numerical analysis literature is that it uses the Lagrange and Hermite interpolational bases heavily, with a complex-variable focus, both because of the recent recognition of the superiority of this approach, and in order to introduce topics in an example-based format. Our objective is to provide the reader with a perspective on scientific computing that provides a systematic method for thinking about numerical solutions and about their interpretation and assessment.

The closest existing texts to our book in this outlook might be Quarteroni et al. (2007), or perhaps the pair of books Deuflhard and Bornemann (2002) and Deuflhard and Hohmann (2003), but our book differs from those in several other respects. We believe, for one, that our relatively informal treatment is less demanding on the mathematical and analytical prerequisites of the students; our students in particular have a very wide range of backgrounds. The topics we cover are also slightly different from those in the aforementioned books—for example, we cover delay differential equations and they do not, whereas their coverage of the numerical solution of PDEs is more complete than ours. But for us, the most important thing about a graduate-level introduction is to show the essential unity of the subject, and we feel that aim of this present work is worth pursuing.

Thus, our objective is to present a unified view of numerical computation, insofar as that is possible. The book cannot, therefore, be self-contained, or anything like complete; it can only hit some of the highlights and point to more extensive discussions of specific points. This is, unfortunately, a necessary tradeoff for such a book, and in partial compensation the list of references is substantial. Consequently, the book is not a "standard" numerical analysis text, in several respects. The topic selection is intended to introduce the reader to important components of a graduate students' toolbox, but more on the *analysis* side than the *methods* side. It is not intended to be a book of recipes.

This brings up the "elephant in the room," the massively popular and useful book (Press et al. 1986), which has been cited more than 33,000 times according to Google Scholar, as we write this. That book is intended for "quick use," we believe. If you have a numerical problem to solve, and only a weekend to do something about it, that book should probably be your first choice of reference. One thing we certainly do not criticize that book for is its attempt at comprehensive coverage. However, it is not a textbook and does not serve the purpose of a course in numerical analysis, which we believe includes a unified theoretical view of all numerical methods. Hence, this present book attempts a complementary view to that of Press et al. (1986).

Finally, even though a unified view is attempted here, many important topics in numerical analysis had to be left out altogether. This includes optimization, integral equations, parallel and high-performance computing, among others. We regret that,[2] but we make no promises to remedy this deficit any time soon. Instead, it is our hope that the reader of this book will have acquired a framework for assessing and understanding numerical methods generally.

Another difference in perspective of this book is the following. As the reader might know (or will know very soon!), there tends to be a tension between computation time, on the one hand, and accuracy and reliability, on the other hand. There are two points of view in scientific computing nowadays, which are paraphrased below:

1. I don't care how correct your answer is if it takes 100 years to get it.
2. I don't care how quickly you give me the wrong answer.

Of these two blunders, we tend to think the first is worse: Hence, this book concentrates on reliability. Therefore, we will not focus on cost very much, nor will we discuss vectorization of algorithms and related issues.

There are more schemes for computation than just IEEE standard fixed-precision floating-point arithmetic, which is the main tool used in this book (and, without much doubt, the main tool used in scientific and engineering computing). There is also *arbitrary*-precision floating-point arithmetic, which is used in computer algebra systems such as MAPLE. This is comparatively slow but occasionally of great interest; some examples will be given in this book. There is also *interval* arithmetic, which is discussed concisely, with references, on the Wikipedia page of the same name: The principle of interval arithmetic is to compute not just answers, but also bounds for the errors in the answers. Again, this is slower than standard fixed-precision floating-point arithmetic, but not solely for the reason that more computation is done with the bounds, but also for the somewhat surprising reason that for many algorithms of practical interest as implemented in floating-point, the *rounding errors usually cancel,* leaving an accurate answer but with overly wide error bounds in interval arithmetic. As a consequence, other algorithms (usually iterative) have to be developed specifically for use with intervals, and while this has been done, particularly for many problems in optimization, and is valuable especially in cases where

[2] In particular, we regret not covering the finite-element method; or multigrid; or …; you get the idea.

the consequences of rounding errors are disastrously expensive, interval arithmetic is not as widely used as floating-point arithmetic is.

A prominent computer algebra researcher asks, "Why not compute the answer exactly?" This researcher knows full well that in the vast majority of cases, exact computation is either impossible outright or impossibly expensive. However, for *some* problems, particularly some linear algebra problems, the data are indeed known exactly and the algorithms for computing the exact rational answer have now been developed to a high degree of efficiency, making it possible nowadays to really get the exact answer (what we will call the *reference* answer in this book). We do not discuss such algorithms here, in part because they are specialized, but really because this course is about numerical methods with approximate arithmetic.

There are yet other arithmetics that have been proposed: significance arithmetic (which is similar to interval arithmetic but less rigorous), and "rounded rational" arithmetic, and others. Some of these are discussed in Knuth (1981). A recent discussion of computational arithmetic can be found in Brent and Zimmermann (2011).

Finally, we underline the fact that the background theoretical ideas from analysis and algebra used in this book are explained in a rather informal way, focusing more on helping visualization and intuition than precise theoretical understanding. This is justified by the fact that if the reader knows the material already, then it serves as a good refresher and also introduces the perspective on it that is relevant to the matter at hand. If the reader does not know the necessary material, or does not know it well, then it should provide just enough guidance to have a feel of what is going on, while at the same time give precise indications as to what and where to look to acquire the required concepts. Just pointing at a book wouldn't do if we can't say what to look for. In this way, we expect to be able to reach the vastly different kinds of reader who need the course this book was designed to support.

On Programming

Computations in the book are carried out almost exclusively in MATLAB (but we also use MAPLE on some occasions). Readers not familiar with MATLAB are encouraged to acquire the wonderful book Higham and Higham (2005) to help them to learn MATLAB. We have no commercial commitment to MATLAB, and if the reader wishes to use SCILAB or OCTAVE instead, then other than some of the advanced techniques available in MATLAB but not in those two for the numerical solution of sparse matrices or ordinary differential equations, the substitution might be all right (but we have not tried). Similarly, the reader may wish to use SAGE or some other freely available computer algebra package to help get through the more formulaic aspects of this book.

This book is *not* a book that teaches programming skills, even by example (our programs are all short and intended to illustrate one or two numerical techniques only). The programs in this book are not even intended as good examples of programming style, although we hope they meet minimal goals in that respect, with

an emphasis on readability over efficiency. The elegant little book Johnson (2010) is a useful guide to a consistent MATLAB style. The style of the programs in this present book differs slightly from that advocated there, in part because our aesthetic tastes differ slightly and in part because the purpose of numerical computing, being more limited than computing that includes, for example, data management, can bear a simpler style without loss of readability or maintainability. However, we emphatically agree with Johnson that a consistent style is a great help, both to the readers of the code (which might include the writer, three months later) and to the users of the code. We also agree that attention to stylistic issues while writing code is a great help in minimizing the number and severity of bugs.

In this book, MATLAB commands will be typeset in the `lstlisting` style and are intended to be typed as shown (with the exception of the line numbers to the left, when any, which are added for pedagogical purposes). For example, the commands

```
1 x = linspace( -1, 1, 21 );
2 y = sin( pi*x );
3 plot( x, y, 'k--' )
```

produce a black dashed-line plot of $\sin(\pi x)$ on the interval $-1 \le x \le 1$. One difference to the style advocated in Johnson (2010) is that spaces are introduced after each opening parenthesis and before each closing parenthesis; similarly, spaces are used after commas. These spaces have no significance to MATLAB but they significantly improve readability for humans (especially in the small font in which this book is typeset). The programs written for this book are all intended to be made available over the web, so longer bits of code need not be typed. The code repository can be accessed at http://www.nfillion.com/coderepository. Similarly, MAPLE commands will also be typeset in the `lstlisting` style; since the syntaxes for the two languages are similar but *not* identical, this has a risk of causing confusion, for which we apologize in advance. However, there are not that many pieces of MAPLE code in the book, and each of them is marked in the text surrounding it, so any confusion will not last long. For example, a similar plot to that created above can be done in MAPLE by the single command

```
plot( sin( Pi*x ), x=-1..1, linestyle=3, color=BLACK );
```

Moreover, we *request* the reader to minimize the use of `sym` in MATLAB. If you are going to do symbolic computation, fire up a computer algebra system (MAPLE, Sage, MuPAD, whatever you like) and use it and its features separately. Yes, the Symbolic Toolbox (which uses MuPAD or MAPLE), if you have it, can be helpful and professionals often do use it for small symbolic computations. In a numerical course, however, `sym` can be very confusing and requires more care in handling than we want to discuss here. This book will not use it at all, and the problems and exercises have been designed so that you need not use it. If you *do* choose to use it, do so on your own recognizance.

Scientific programming is, in our view, seriously underrated as a discipline and given nowhere near the attention in the curriculum that it deserves or needs. Many people view the course that this book is intended to support, namely, an introductory course in numerical analysis for graduate students, as "the" course that a graduate

student takes in order to learn how to program. *This is a serious mistake.* If this is the only course that you take that has programming in it, you are in trouble. It takes more than a few weekends to learn how to program (and given the amount of material here, you won't have many weekends available, even).

However, you can make a *start* on programming at the same time as you read this book if you are willing to really put in some effort. Both MATLAB and MAPLE are easier to learn than many scientific programming languages, at least for people with a high level of mathematical maturity and background knowledge. You will need substantial guidance, though, in addition to this book. The aforementioned book Higham and Higham (2005) is highly recommended. The older book Corless (2002), while dated in some respects, was intended to teach MAPLE to numerical analysts, and since the *programming language* for MAPLE has not changed much since then (although the GUI has), it remains potentially useful. Our colleague Dhavide Aruliah also recommends the Software Carpentry project by Greg Wilson http://software-carpentry.org/, which we were delighted to learn about—there seems to be a wealth of useful material there, including a section on MATLAB. See also Aruliah et al. (2012).

Large scientific programs require a serious level of discipline and mathematical thought; this is the discipline nowadays called software engineering. This book does not teach software engineering at all. For those wishing to have a glimpse, we highly recommend the (ancient, in computer terms) books by Leo J. Brodie, which use the curiously lovely computer language Forth.[3] In some sense, Forth is natural to teach those concepts: It is possible to write arbitrarily unreadable code in Forth entirely by accident, and you *need* to learn some discipline to write it well; it's harder to write unreadable code in MAPLE (although for sure it can be done!).

Writing software that is robust, readable, maintainable, usable, and efficient, and overall does what it was intended to do is a humbling activity. The first thing that one learns is—a true scientific lesson—that one's thought processes are not as reliable as one had believed. Numerical analysis and scientific computing (along with computer programming generally) have overturned many things that were thought mathematically to be true, and computer programs have had a profound influence on how we view the world and how we think about it. Indeed, one of us has coined the term "computer-mediated thinking" to cover some aspects of that profound change (see Corless 2004, for a discussion of this in a pedagogical context). Put simply, there is *no other way* to think about some complex systems than to combine the power of the mind with the power of the computer. We will see an example due to Turing, shortly.

[3] The book *Starting Forth* is now available free online at http://www.forth.com/starting-forth/, and although Forth has very little in common with MATLAB or MAPLE, the programming concepts and discipline begun in that book will transfer easily. The second book, *Thinking Forth*, is also available online at http://thinking-forth.sourceforge.net/ and is one of the most useful introductions to software engineering, even though it, like its predecessor, is focused on Forth.

How to Use This Book

We believe, with Trefethen (2008b p. 606), that

> the main business of numerical analysis is designing algorithms that converge quickly; rounding-error analysis, while often a part of the discussion, is rarely the central issue.

Why, then, are the first chapter and the first appendix of this book so heavy on floating-point arithmetic? The answer is that the material is logically first, not that it is of the first importance didactically. In fact, when RMC teaches this course, he begins with Chap. 4 and looks back on the logically prior material when needed: His approach is "leap ahead, back fill." But the students' needs may vary considerably, and there are those who decidedly prefer an abstract presentation first, filled in with examples later: For them, they may begin with Chap. 1 and proceed in the order of Fig. 1a.

The instructor should find that the book can be used in many ways. Following the linear order is an option, provided you have enough time (a one-semester course certainly isn't enough time). With the time constraint in mind, Fig. 1a follows the same theoretical order, but it shows what should be considered optional. As stated before, at Western we start with Chap. 4. That way the course starts with the material on the QR and SVD factoring, culminating in a definition of condition number. This approach brings the student to immediately engage a problem that stimulated the development of numerical analysis in the first place. Then we come

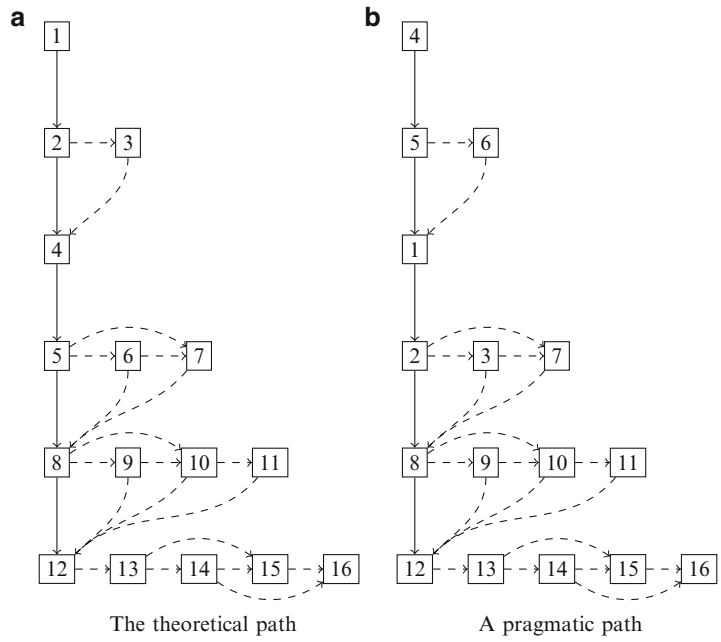

Fig. 1 Suggested teaching paths for this book, where *dashed lines* denote options. (**a**) The theoretical path. (**b**) A pragmatic path

back to Chap. 1 (and Appendix A) for a necessary examination of theoretical issues in finite-precision computation and approximation. We then return to Part II to discuss eigenvalue problems, sparse systems, and structured systems. We then proceed to polynomials, function evaluation, and then rootfinding. The course closes with material covering numerical integration and numerical solution of differential equations, followed by delay differential equations or partial differential equations, as the tastes of the students indicate and as time permits. A curriculum closely related to this pragmatic orientation is in Fig. 1b.

Experience has shown that the material in Chap. 8 is used heavily in almost all later chapters. Experience has also shown that the later chapters *always* get shortchanged in a one-semester course: Probably at most one of Chaps. 14, 15, or 16 can be covered, and Chap. 9, though short and important, is in some danger of being omitted too. In any case, perhaps that is because RMC is personally focused more on Chap. 12 and its sequels, not because of the students' needs. In any case, the linear algebra topics can (and should!) always be covered.

Some of the chapters may be used for reading only. Good candidates are Chap. 1, Chap. 3 on the evaluation of functions, and Chap. 7 on iterative methods. Chapter 14, on delay DE, seems quite popular and goes quickly after the work on IVP and on interpolation.

Exercises

This book contains many exercises. They are identified as belonging to one of these categories:

1. Theory and Practice;
2. Investigations and Projects.

The first type of problem will include simple tasks that amount to "getting the go of it" or to make sure that one understands the basic notions that are assumed in the various manipulations. This includes practice with basic MATLAB and MAPLE tricks. It may also involve proofs—either from scratch, sometimes with hints, or completing proof sketches, including some error analyses (although not too many). The final type of problem—namely, investigations and projects—typically involves more time and effort from the students. Typically, these problems will involve exploring various numerical methods in depth, that is, doing analytic work and then implementing it, usually in MATLAB. That is, this type of problem is to some extent a programming assignment although the course we teach is not intended to teach programming skill.

The instructor may find it convenient to combine problems of different categories as well as different degrees of difficulty following this scheme. Students thus have the chance to feel the pleasant breeze of review, get their hands dirty with the tedious but very important practical work, and feel the ecstatic frustration of working on a problem of some *envergure*. For the more challenging projects, the student should refrain from being frustrated, remembering the words of J. S. Mill (1873 p. 45):

> A pupil from whom nothing is ever demanded which [s]he cannot do, never does all [s]he can.

One of the authors (NF), upon whom the teaching method used in this book was tested, has to agree that some of the difficult problems in this book are among those from which he learned most. We hope the reader will feel the same.

London, ON

Robert M. Corless
Nicolas Fillion

http://www.springer.com/978-1-4614-8452-3

A Graduate Introduction to Numerical Methods
From the Viewpoint of Backward Error Analysis

Corless, R.; Fillion, N.

2013, XXXIX, 869 p. 194 illus., 10 illus. in color.,
Hardcover
ISBN: 978-1-4614-8452-3