
Preface

There are so many excellent books on finite difference methods for ordinary and partial differential equations that writing yet another one requires a different view on the topic. The present book is not so concerned with the traditional academic presentation of the topic, but is focused at teaching the practitioner how to obtain reliable computations involving finite difference methods. This focus is based on a set of learning outcomes:

1. understanding of the ideas behind finite difference methods,
2. understanding how to transform an algorithm to a well-designed computer code,
3. understanding how to test (verify) the code,
4. understanding potential artifacts in simulation results.

Compared to other textbooks, the present one has a particularly strong emphasis on computer implementation and verification. It also has a strong emphasis on an intuitive understanding of constructing finite difference methods. To learn about the potential non-physical artifacts of various methods, we study exact solutions of finite difference schemes as these give deeper insight into the physical behavior of the numerical methods than the traditional (and more general) asymptotic error analysis. However, asymptotic results regarding convergence rates, typically truncation errors, are crucial for testing implementations, so an extensive appendix is devoted to the computation of truncation errors.

Why finite differences? One may ask why we do finite differences when finite element and finite volume methods have been developed to greater generality and sophistication than finite differences and can cover more problems. The finite element and finite volume methods are also the industry standard nowadays. Why not just those methods? The reason for finite differences is the method's simplicity, both from a mathematical and coding perspective. Especially in academia, where simple model problems are used a lot for teaching and in research (e.g., for verification of advanced implementations), there is a constant need to solve the model problems from scratch with easy-to-verify computer codes. Here, finite differences are ideal. A simple 1D heat equation can of course be solved by a finite element package, but a 20-line code with a difference scheme is just right to the point and provides an

understanding of all details involved in the model and the solution method. Everybody nowadays has a laptop and the natural method to attack a 1D heat equation is a simple Python or Matlab program with a difference scheme. The conclusion goes for other fundamental PDEs like the wave equation and Poisson equation as long as the geometry of the domain is a hypercube. The present book contains all the practical information needed to use the finite difference tool in a safe way.

Various pedagogical elements are utilized to reach the learning outcomes, and these are commented upon next.

Simplify, understand, generalize The book's overall pedagogical philosophy is the three-step process of first *simplifying* the problem to something we can *understand* in detail, and when that understanding is in place, we can *generalize* and hopefully address real-world applications with a sound scientific problem-solving approach. For example, in the chapter on a particular family of equations we first simplify the problem in question to a 1D, constant-coefficient equation with simple boundary conditions. We learn how to construct a finite difference method, how to implement it, and how to understand the behavior of the numerical solution. Then we can generalize to higher dimensions, variable coefficients, a source term, and more complicated boundary conditions. The solution of a compound problem is in this way an assembly of elements that are well understood in simpler settings.

Constructive mathematics This text favors a constructive approach to mathematics. Instead of a set of definitions followed by popping up a method, we emphasize how to think about the construction of a method. The aim is to obtain a good intuitive understanding of the mathematical methods.

The text is written in an easy-to-read style much inspired by the following quote.

Some people think that stiff challenges are the best device to induce learning, but I am not one of them. The natural way to learn something is by spending vast amounts of easy, enjoyable time at it. This goes whether you want to speak German, sight-read at the piano, type, or do mathematics. Give me the German storybook for fifth graders that I feel like reading in bed, not Goethe and a dictionary. The latter will bring rapid progress at first, then exhaustion and failure to resolve.

The main thing to be said for stiff challenges is that inevitably we will encounter them, so we had better learn to face them boldly. Putting them in the curriculum can help teach us to do so. But for teaching the skill or subject matter itself, they are overrated. [18, p. 86]
Lloyd N. Trefethen, Applied Mathematician, 1955-.

This book assumes some basic knowledge of finite difference approximations, differential equations, and scientific Python or MATLAB programming, as often met in an introductory numerical methods course. Readers without this background may start with the light companion book “Finite Difference Computing with Exponential Decay Models” [9]. That book will in particular be a useful resource for the programming parts of the present book. Since the present book deals with partial differential equations, the reader is assumed to master multi-variable calculus and linear algebra.

Fundamental ideas and their associated scientific details are first introduced in the simplest possible differential equation setting, often an ordinary differential equation, but in a way that easily allows reuse in more complex settings with partial differential equations. With this approach, new concepts are introduced with a

minimum of mathematical details. The text should therefore have a potential for use early in undergraduate student programs.

All nuts and bolts Many have experienced that “vast amounts of easy, enjoyable time”, as stated in the quote above, arises when mathematics is implemented on a computer. The implementation process triggers understanding, creativity, and curiosity, but many students find the transition from a mathematical algorithm to a working code difficult and spend a lot of time on “programming issues”.

Most books on numerical methods concentrate on the mathematics of the subject while details on going from the mathematics to a computer implementation are less in focus. A major purpose of this text is therefore to help the practitioner by providing *all nuts and bolts* necessary for safely going from the mathematics to a well-designed and well-tested computer code. A significant portion of the text is consequently devoted to programming details.

Python as programming language While MATLAB enjoys widespread popularity in books on numerical methods, we have chosen to use the Python programming language. Python is very similar to MATLAB, but contains a lot of modern software engineering tools that have become standard in the software industry and that should be adopted also for numerical computing projects. Python is at present also experiencing an exponential growth in popularity within the scientific computing community. One of the book’s goals is to present an up-to-date Python eco system for implementing finite difference methods.

Program verification Program testing, called *verification*, is a key topic of the book. Good verification techniques are indispensable when debugging computer code, but also fundamental for achieving reliable simulations. Two verification techniques saturate the book: exact solution of discrete equations (where the approximation error vanishes) and empirical estimation of convergence rates in problems with exact (analytical or manufactured) solutions of the differential equation(s).

Vectorized code Finite difference methods lead to code with loops over large arrays. Such code in plain Python is known to run slowly. We demonstrate, especially in Appendix C, how to port loops to fast, compiled code in C or Fortran. However, an alternative is to vectorize the code to get rid of explicit Python loops, and this technique is met throughout the book. Vectorization becomes closely connected to the underlying array library, here *numpy*, and is often thought of as a difficult subject by students. Through numerous examples in different contexts, we hope that the present book provides a substantial contribution to explaining how algorithms can be vectorized. Not only will this speed up serial code, but with a library that can produce parallel code from *numpy* commands (such as *Numba*¹), vectorized code can be automatically turned into parallel code and utilize multi-core processors and GPUs. Also when creating tailored parallel code for today’s supercomputers, vectorization is useful as it emphasizes splitting up an algorithm into plain and simple

¹ <http://numba.pydata.org>

array operations, where each operation is trivial to parallelize efficiently, rather than trying to develop a “smart” overall parallelization strategy.

Analysis via exact solutions of discrete equations Traditional asymptotic analysis of errors is important for verification of code using convergence rates, but gives a limited understanding of how and why a correctly implemented numerical method may give non-physical results. By developing exact solutions, usually based on Fourier methods, of the discrete equations, one can obtain a physical understanding of the behavior of a numerical method. This approach is favored for analysis of methods in this book.

Code-inspired mathematical notation Our primary aim is to have a clean and easy-to-read computer code, and we want a close one-to-one relationship between the computer code and mathematical description of the algorithm. This principle calls for a mathematical notation that is governed by the natural notation in the computer code. The unknown is mostly called u , but the meaning of the symbol u in the mathematical description changes as we go from the exact solution fulfilling the differential equation to the symbol u that is naturally used for the associated data structure in the code.

Limited scope The aim of this book is not to give an overview of a lot of methods for a wide range of mathematical models. Such information can be found in numerous existing, more advanced books. The aim is rather to introduce basic concepts and a thorough understanding of how to think about computing with finite difference methods. We therefore go in depth with only the most fundamental methods and equations. However, we have a multi-disciplinary scope and address the interplay of mathematics, numerics, computer science, and physics.

Focus on wave phenomena Most books on finite difference methods, or books on theory with computer examples, have their emphasis on diffusion phenomena. Half of this book (Chap. 1, 2, and Appendix C) is devoted to wave phenomena. Extended material on this topic is not so easy find in the literature, so the book should be a valuable contribution in this respect. Wave phenomena is also a good topic in general for choosing the finite difference method over other discretization methods since one quickly needs fine resolution over the entire mesh and uniform meshes are most natural.

Instead of introducing the finite difference method for diffusion problems, where one soon ends up with matrix systems, we do the introduction in a wave phenomena setting where explicit schemes are most relevant. This slows down the learning curve since we can introduce a lot of theory for differences and for software aspects in a context with simple, explicit stencils for updating the solution.

Independent chapters Most book authors are careful with avoiding repetitions of material. The chapters in this book, however, contain some overlap, because we want the chapters to appear meaningful on their own. Modern publishing technology makes it easy to take selected chapters from different books to make a new book tailored to a specific course. The more a chapter builds on details in other chapters, the more difficult it is to reuse chapters in new contexts. Also, most readers find it

convenient that important information is explicitly stated, even if it was already met in another chapter.

Supplementary materials All program and data files referred to in this book are available from the book's primary web site: URL: <http://github.com/hplgit/fdm-book/>.

Acknowledgments Professor Kent-Andre Mardal at the University of Oslo has kindly contributed to enlightening discussions on several issues. Many students have provided lots of useful feedback on the exposition and found many errors in the text. Special efforts in this regard were made by Imran Ali, Shirin Fallahi, Anders Hafreager, Daniel Alexander Mo Søreide Houshmand, Kristian Gregorius Hustad, Mathilde Nygaard Kamperud, and Fatemeh Miri. The collaboration with the Springer team, with Dr. Martin Peters, Thanh-Ha Le Thi, and their production staff has always been a great pleasure and a very efficient process.

Finally, want really appreciate the strong push from the COE of Simula Research Laboratory, Aslak Tveito, for publishing and financing books in open access format, including this one. We are grateful for the laboratory's financial contribution as well as to the financial contribution from the Department of Process, Energy and Environmental Technology at the University College of Southeast Norway.

Oslo, July 2016

Hans Petter Langtangen, Svein Linge



<http://www.springer.com/978-3-319-55455-6>

Finite Difference Computing with PDEs
A Modern Software Approach
Langtangen, H.P.; Linge, S.
2017, XXIII, 507 p. 150 illus., Hardcover
ISBN: 978-3-319-55455-6