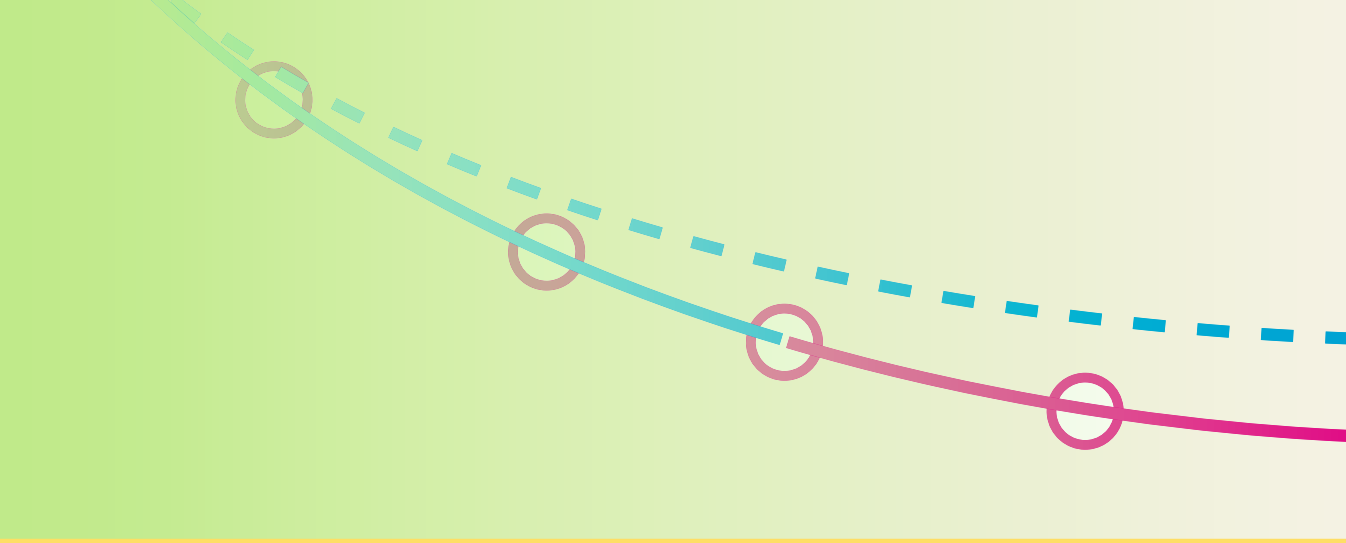

LECTURE NOTES IN COMPUTATIONAL SCIENCE AND ENGINEERING 110

Hans Petter Langtangen

Finite Difference Computing with Exponential Decay Models

Springer Open

Lecture Notes in Computational Science and Engineering **110**

Editors:

More information about this series at http://www.springer.com/series/3527

Hans Petter Langtangen

Finite Difference Computing with Exponential Decay Models

Springer Open

Hans Petter Langtangen
Simula Research Laboratory
Lysaker, Norway

On leave from:

Department of Informatics
University of Oslo
Oslo, Norway

ISSN 1439-7358 ISSN 2197-7100 (electronic)
Lecture Notes in Computational Science and Engineering
ISBN 978-3-319-29438-4 ISBN 978-3-319-29439-1 (eBook)
DOI 10.1007/978-3-319-29439-1
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2016932614

Mathematic Subject Classification (2010): 34, 65, 68

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

Preface

This book teaches the basic components in the scientific computing pipeline: modeling, differential equations, numerical algorithms, programming, plotting, and software testing. The pedagogical idea is to treat these topics in the context of a very simple mathematical model, the differential equation for exponential decay, $u'(t) = -au(t)$, where u is unknown and a is a given parameter. By keeping the mathematical problem simple, the text can go deep into all details about how one must combine mathematics and computer science to create well-tested, reliable, and flexible software for such a mathematical model.

The writing style is gentle and aims at a broad audience. I am much inspired by Nick Trefethen's praise of easy learning:

> 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. [13, p. 86]

Prerequisite knowledge for this book is basic one-dimensional calculus and preferably some experience with computer programming in Python or MATLAB. The material was initially written for self study and therefore features comprehensive and easy-to-understand explanations. For some readers it may act as an overview and refresher of traditional mathematical topics and likely a first introduction to many of the software topics. The text can also be used as a case-based and mathematically simple introduction to modern multi-disciplinary problem solving with computers, using the range of applications in Chap. 4 as motivation and then treating the details of the mathematical and computer science subjects from the other chapters. In particular, I have also had in mind the new groups of readers from bio- and geo-sciences who need to enter the world of computer-based differential equation modeling, but lack experience with (and perhaps also interest in) mathematics and programming.

The choice of topics in this book is motivated from what is needed in more advanced courses on finite difference methods for partial differential equations

(PDEs). It turns out that a range of concepts and tools needed for PDEs can be introduced and illustrated by very simple ordinary differential equation (ODE) examples. The goal of the text is therefore to lay a foundation for understanding numerical methods for PDEs by first meeting the fundamental ideas in a simpler ODE setting. Compared to other books, the present one has a much stronger focus on how to turn mathematics into working code. It also explains the mathematics and programming in more detail than what is common in the literature.

There is a more advanced companion book in the works, "Finite Difference Computing with Partial Differential Equations", which treats finite difference methods for PDEs using the same writing style and having the same focus on turning mathematical algorithms into reliable software.

Although the main example in the present book is $u' = -au$, we also address the more general model problem $u' = -a(t)u + b(t)$, and the completely general, nonlinear problem $u' = f(u,t)$, both for scalar and vector $u(t)$. The author believes in the principle *simplify, understand, and then generalize*. That is why we start out with the simple model $u' = -au$ and try to understand how methods are constructed, how they work, how they are implemented, and how they may fail for this problem, before we generalize what we have learned from $u' = -au$ to more complicated models.

The following list of topics will be elaborated on.

- How to think when constructing finite difference methods, with special focus on the Forward Euler, Backward Euler, and Crank–Nicolson (midpoint) schemes.
- How to formulate a computational algorithm and translate it into Python code.
- How to make curve plots of the solutions.
- How to compute numerical errors.
- How to compute convergence rates.
- How to test that an implementation is correct (verification) and how to automate tests through *test functions* and *unit testing*.
- How to work with Python concepts such as arrays, lists, dictionaries, lambda functions, and functions in functions (closures).
- How to perform array computing and understand the difference from scalar computing.
- How to uncover numerical artifacts in the computed solution.
- How to analyze the numerical schemes mathematically to understand why artifacts may occur.
- How to derive mathematical expressions for various measures of the error in numerical methods, frequently by using the `sympy` software for symbolic computations.
- How to understand concepts such as finite difference operators, mesh (grid), mesh functions, stability, truncation error, consistency, and convergence.
- How to solve the general nonlinear ODE $u' = f(u,t)$, which is either a scalar ODE or a system of ODEs (i.e., u and f can either be a function or a vector of functions).
- How to access professional packages for solving ODEs.
- How the model equation $u' = -au$ arises in a wide range of phenomena in physics, biology, chemistry, and finance.
- How to structure a code in terms of functions.

- How to make reusable modules.
- How to read input data flexibly from the command line.
- How to create graphical/web user interfaces.
- How to use test frameworks for automatic unit testing.
- How to refactor code in terms of classes (instead of functions).
- How to conduct and automate large-scale numerical experiments.
- How to write scientific reports in various formats (LaTeX, HTML).

The exposition in a nutshell

Everything we cover is put into a practical, hands-on context. All mathematics is translated into working computing codes, and all the mathematical theory of finite difference methods presented here is motivated from a strong need to understand why we occasionally obtain strange results from the programs. Two fundamental questions saturate the text:

- How do we solve a differential equation problem and produce numbers?
- How do we know that the numbers are correct?

Besides answering these two questions, one will learn a lot about mathematical modeling in general and the interplay between physics, mathematics, numerical methods, and computer science.

The book contains a set of exercises in most of the chapters. The exercises are divided into three categories: *exercises* refer to the text (usually variations or extensions of examples in the text), *problems* are stand-alone exercises without references to the text, and *projects* are larger problems. Exercises, problems, and projects share a common numbering to avoid confusion between, e.g., Exercise 4.3 and Problem 4.3 (it will be Exercise 4.3 and Problem 4.4 if they follow after each other).

All program and data files referred to in this book are available from the book's primary web site: http://hplgit.github.io/decay-book/doc/web/.

Acknowledgments Professor Svein Linge provided very detailed and constructive feedback on this text, and all his efforts are highly appreciated. Many students have also pointed out weaknesses and found errors. A special thank goes to Yapi Donatien Achou's proof reading. Many thanks also to Linda Falch-Koslung, Dr. Olav Dajani, and the rest of the OUS team for feeding me with FOLFIRINOX and thereby keeping me alive and in good enough shape to finish this book. As always, the Springer team ensured a smooth and rapid review process and production phase. This time special thanks go to all the efforts by Martin Peters, Thanh-Ha Le Thi, and Yvonne Schlatter.

Oslo, August 2015 Hans Petter Langtangen