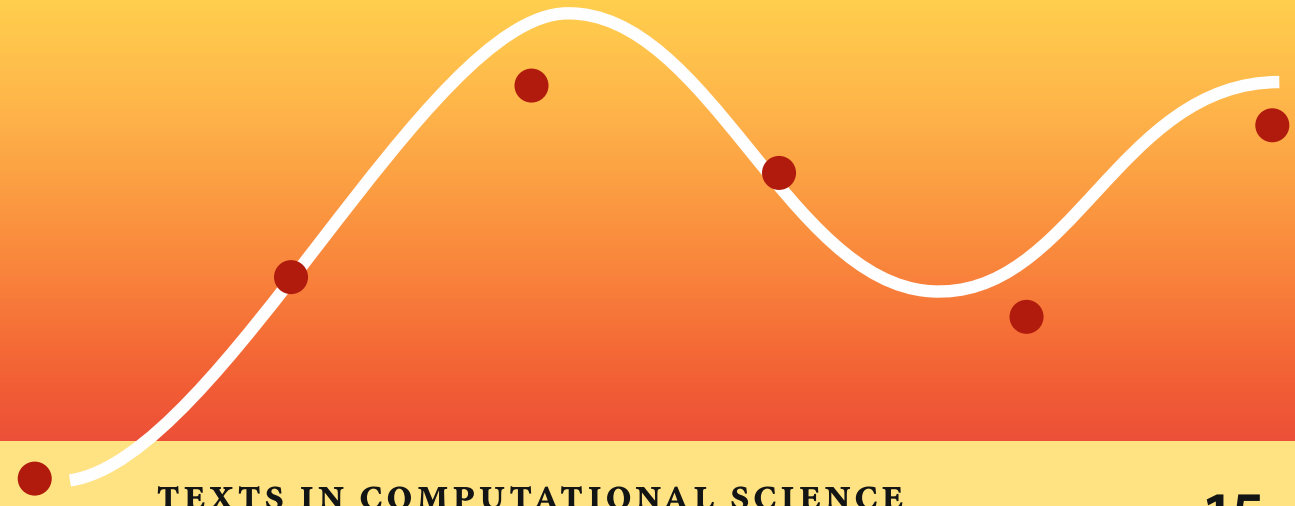

TEXTS IN COMPUTATIONAL SCIENCE AND ENGINEERING 15

Svein Linge · Hans Petter Langtangen

Programming for Computations – Python

Springer Open

Texts in Computational Science and Engineering

15

Editors

More information about this series at http://www.springer.com/series/5151

Svein Linge · Hans Petter Langtangen

Programming for Computations – Python

A Gentle Introduction to Numerical Simulations with Python

Springer Open

Svein Linge
Department of Process, Energy and
Environmental Technology
University College of Southeast Norway
Porsgrunn, Norway

Hans Petter Langtangen
Simula Research Laboratory
Lysaker, Norway

On leave from:

Department of Informatics
University of Oslo
Oslo, Norway

ISSN 1611-0994
Texts in Computational Science and Engineering
ISBN 978-3-319-32427-2 ISBN 978-3-319-32428-9 (eBook)
DOI 10.1007/978-3-319-32428-9
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2016945368

Mathematic Subject Classification (2010): 26-01, 34A05, 34A30, 34A34, 39-01, 40-01, 65D15, 65D25, 65D30, 68-01, 68N01, 68N19, 68N30, 70-01, 92D25, 97-04, 97U50

Printed on acid-free paper

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

Preface

Computing, in the sense of doing mathematical calculations, is a skill that mankind has developed over thousands of years. Programming, on the other hand, is in its infancy, with a history that spans a few decades only. Both topics are vastly comprehensive and usually taught as separate subjects in educational institutions around the world, especially at the undergraduate level. This book is about the *combination* of the two, because computing today becomes so much more powerful when combined with programming.

Most universities and colleges implicitly require students to specialize in computer science if they want to learn the craft of programming, since other student programs usually do not offer programming to an extent demanded for really mastering this craft. Common arguments claim that it is sufficient with a brief introduction, that there is not enough room for learning programming in addition to all other must-have subjects, and that there is so much software available that few really need to program themselves. A consequence is that engineering students often graduate with shallow knowledge about programming, unless they happened to choose the computer science direction.

We think this is an unfortunate situation. There is no doubt that practicing engineers and scientists need to know their pen and paper mathematics. They must also be able to run off-the-shelf software for important standard tasks and will certainly do that a lot. Nevertheless, the benefits of mastering programming are many.

Why learn programming?

1. Ready-made software is limited to handling certain standard problems. What do you do when the problem at hand is not covered by the software you bought? Fortunately, a lot of modern software systems are extensible via programming. In fact, many systems demand parts of the problem specification (e.g., material models) to be specified by computer code.
2. With programming skills, you may extend the flexibility of existing software packages by combining them. For example, you may integrate packages that do not speak to each other from the outset. This makes the work flow simpler, more efficient, and more reliable, and it puts you in position to attack new problems.

3. It is easy to use excellent ready-made software the wrong way. Insight in programming and the mathematics behind is fundamental for understanding complex software, avoiding pitfalls, and become a safe user.
4. Bugs (errors in computer code) are present in most larger computer programs (also in the ones from the shop!). What do you do when your ready-made software gives unexpected results? Is it a bug, is it wrong use, or is it the mathematically correct result? Experience with programming of mathematics gives you a good background for answering these questions. The one who can program, can also make tailored code for a simplified problem setting and use that to verify the computations done with off-the-shelf software.
5. Lots of skilled people around the world solve computational problems by writing their own code and offer their code for free on the Internet. To take advantage of this truly great source of software in a reliable way, one must normally be able to understand and possibly modify computer code offered by others.
6. It is recognized world wide that students struggle with mathematics and physics. Too many find such subjects difficult and boring. With programming, we can execute the good old subjects in a brand new way! According to the authors' own experience, students find it much more motivating and enlightening when programming is made an integrated part of mathematics and physical science courses. In particular, the problem being solved can be much more realistic than when the mathematics is restricted to what you can do with pen and paper.
7. Finally, we launch our most important argument for learning computer programming: the *algorithmic thinking* that comes with the process of writing a program for a computational problem enforces a thorough understanding of both the problem and solution method. We can simply quote the famous Norwegian computer scientist Kristen Nyggaard: "Programming is understanding".

In the authors' experience, programming is an excellent pedagogical tool for understanding mathematics: "You think you know when you can learn, are more sure when you can write, even more when you can teach, but certain when you can program" (Alan Perlis, computer scientist, 1922-1990). Consider, for example, integration. A numerical method for integration has a much stronger focus on what the integral actually is and means compared to analytical methods, where much time and effort must be devoted to integration by parts, integration by substitution, etc. Moreover, when programming the numerical integration formula, it becomes evident that it works for "all" mathematical functions and that the implementation should be in terms of a *general* function applicable to "all" integrals. In this way, students learn to recognize a special problem as belonging to a class of problems (e.g., integration, differential equations, root finding), for which we have general numerical methods implemented in widely applicable software. When they write this software, as we do in this book, they learn how to generalize and increase the abstraction level of the mathematical problem. When they use this software, they learn how a special case should be attacked by general methods and software for the class of problems that comprises the special case at hand. This is the power of mathematics in a nutshell, and it is paramount that students understand this way of thinking.

Target audience and background knowledge This book was written for students, teachers, engineers and scientists that know *nothing* about programming and numerical methods from before, but who seek a *minimum* of the fundamental skills required to get started with programming as a tool for solving scientific and engineering problems. Some knowledge of one- and multi-variable calculus is assumed. The basic programming concepts are presented in only 50 pages (Chapters 1 and 2), before practical applications of these concepts are demonstrated in important mathematical subjects addressed in the remaining parts of the book (Chapters 3-6). Each chapter is followed by a set of exercises that cover a wide range of application areas, e.g. biology, geology, statistics, physics and mathematics. The exercises were particularly designed to bring across important points from the text. The reader will realize that the modest content of the first 50 pages can in fact bring you quite far in powerful problem solving!

Learning the very basics of programming should not take long, but as with any other craft, mastering the skill requires continued and extensive practice. Some beginning practice is gained through Chapters 3-6, but the authors strongly emphasize that this is only a start. Students should continue to practice programming in subsequent courses, while those who exercise self-study, should keep up the learning process through continued application of the craft. The book is a good starting point when teaching computer programming as an integrated part of standard university courses in mathematics and physical sciences. In our experience, such an integration is doable and indeed rewarding.

Numerical methods An overall goal with this book is to motivate computer programming as a very powerful tool for doing mathematics. All examples are related to mathematics and its use in engineering and science. However, to solve mathematical problems through computer programming, we need numerical methods. Explaining basic numerical methods is therefore an integral part of the book. Our choice of topics is governed by what is most needed in science and engineering, as well as in the teaching of applied physical science courses. Mathematical models are then central, with differential equations constituting the most frequent type of models. Consequently, the numerical focus in this book is on differential equations. As a soft pedagogical starter for the programming of mathematics, we have chosen the topic of numerical integration. There is also a chapter on root finding, which is important for the numerical solution on nonlinear differential equations. We remark that the book is deliberately brief on numerical methods. This is because our focus is on *implementing* numerical algorithms, but to develop reliable, working programs, the programmer must be confident about the basic ideas of the numerical approximations involved.

The computer language: Python We have chosen to use the programming language Python, because this language gives very compact and readable code that closely resembles the mathematical recipe for solving the problem at hand. Python also has a gentle learning curve. There is a MATLAB/Octave companion of this book in case that language is preferred. Comparing these two versions of the book provides an excellent demonstration of how similar these languages are. Other computer languages, like Fortran, C, and C++, have a strong position in science and engineering. During the last two decades, however, there has been a significant

shift in popularity from these compiled languages to more high-level and easier-to-read languages like Matlab, Python, R, Maple, Mathematica, and IDL, for instance. This latter class of languages is computationally less efficient, but superior with respect to overall human problem solving efficiency. This book emphasizes *how to think like a programmer*, rather than focusing on technical language details. Thus, the book should put the reader in a good position for learning other programming languages later, including the classic ones: Fortran, C, and C++.

How this book is different There are numerous texts on computer programming and numerical methods, so how does the present one differ from the existing literature? Compared to books on numerical methods, our book has a much stronger emphasis on the craft of programming and on verification. We want to give students a thorough understanding of how one thinks about programming as a problem solving method and how one can be sure that programs are correct (well, you can never be completely sure, but we show how you can provide convincing evidence for correctness).

Even though there are lots of books on numerical methods where many algorithms have a corresponding computer implementation (see, e.g., [1, 3–6, 11, 16–18, 21, 24, 26–31] – the latter two are the only texts we know that apply Python), it is assumed that the reader "can program" beforehand. The present book teaches the craft of structured programming along with the fundamental ideas of numerical methods. Furthermore, we have so far not found any other numerical methods book that has a strong emphasis on verifying implementations. In this book, unit testing and corresponding test functions are introduced early on. We also put much emphasis on coding algorithms as *functions*, as opposed to "flat programs", which often dominate in the literature and among practitioners. Functions are reusable because they utilize the general formulation of a mathematical algorithm such that it becomes applicable to a large class of problems.

There are also numerous books on computer programming, but to our knowledge only one [13] that aims to teach how to *think* about programming in the context of numerical methods and scientific applications. That book [13] has its primary focus on teaching Python and is a very comprehensive introduction to Python as a language and the thinking about programming as a computer scientist. Sometimes one needs a text that does not go so deep into the language-specific details, but instead targets the shortest path to reliable mathematical problem solving through programming. With this attitude in mind, a lot of topics were left out of the present book, simply because they were not *strictly* needed in the mathematical problem solving process. Examples of such topics are object-oriented programming and Python dictionaries (of which the latter omission is possibly subject to more debate). If you find the present book too shallow, [13] might be the right choice for you. That source should also work nicely as a more in-depth successor of the present text.

Whenever the need for a *structured introduction to programming* arises in science and engineering courses, this book may be your option, either for self-study or for use in organized teaching. The thinking, habits, and practice covered in a couple of hundred pages will put readers in a firm position for utilizing and understanding the power of computers for problem solving in science and engineering.

Supplementary materials All program and data files referred to in this book are available from the book's primary web site: http://hplgit.github.io/prog4comp/.

Acknowledgments First of all, we want to thank all students who attended the courses FM1006 Modelling and simulation of dynamic systems, FM1115 Scientific Computing, FB1012 Mathematics I and FB2112 Physics at the University College of Southeast Norway over the last couple of years. They worked their way through early versions of this text and gave us constructive and positive feedback that helped us correct errors and improve the book in so many ways. Special acknowledgement goes to Guandong Kou and Edirisinghe V. P. J. Manjula for their careful reading of the manuscript and constructive suggestions for improvement. The careful proof reading by Yapi Donatien Achou is also highly appreciated. We thank all our good colleagues at the University College of Southeast Norway, University of Oslo, and Simula Research Laboratory for their continued support and interest, enlightening discussions, and for providing such an inspiring environment for teaching and science. In particular, Svein Linge is thankful to Marius Lysaker for their fruitful collaboration on introducing programming as an integral part of mathematics and physics bachelor courses at the University College of Southeast Norway. Finally, the authors must thank the Springer team with Dr. Martin Peters, Thanh-Ha Le Thi, and Yvonne Schlatter for the effective editorial and production process.

The text was written in the DocOnce[1] [12] markup language, which allowed us to work with a single text source for both the Python and the Matlab version of this book, and to produce various electronic versions of the book.

December 2015 Svein Linge and Hans Petter Langtangen

[1] https://github.com/hplgit/doconce