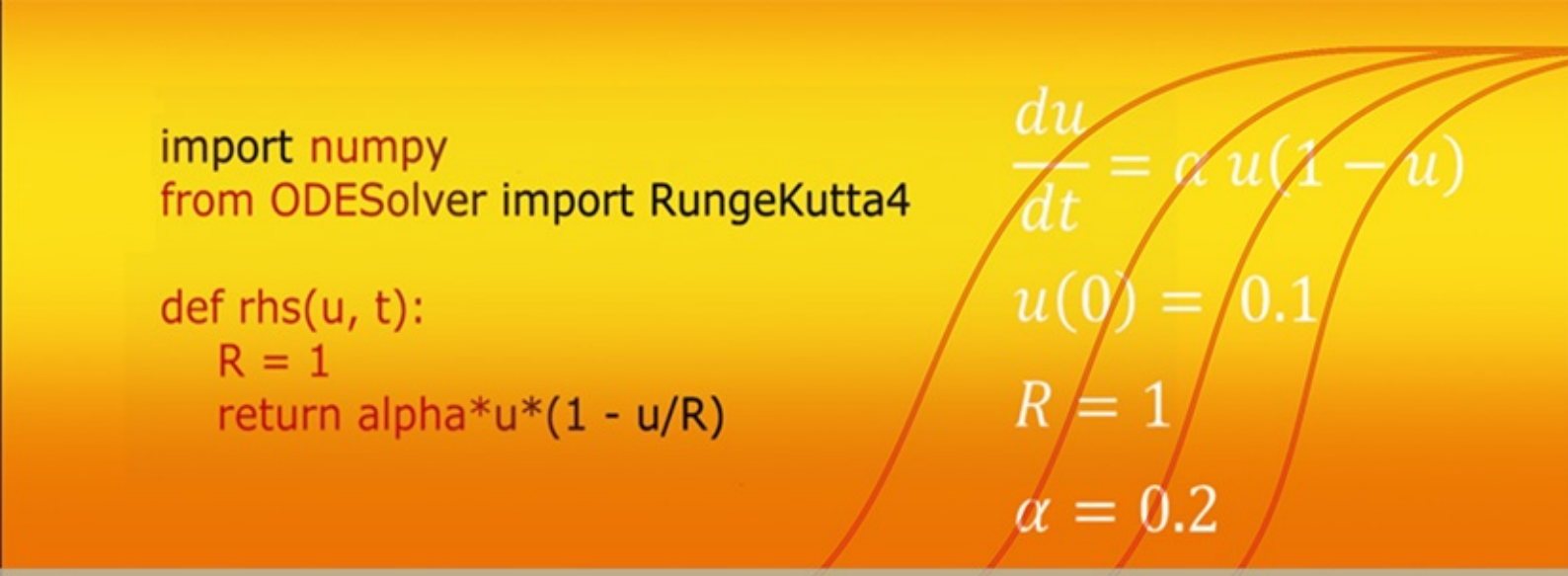

TEXTS IN COMPUTATIONAL SCIENCE AND ENGINEERING 6

Hans Petter Langtangen

A Primer on Scientific Programming with Python

Fourth Edition

Springer

Volume 6
Texts in Computational Science and Engineering

Series Editors

Timothy J. Barth, Michael Griebel, David E. Keyes, Risto M. Nieminen, Dirk Roose and Tamar Schlick

This series contains graduate and undergraduate textbooks on topics described by the term "computational science and engineering". This includes theoretical aspects of scientific computing such as mathematical modeling, optimization methods, discretization techniques, multiscale approaches, fast solution algorithms, parallelization, and visualization methods as well as the application of these approaches throughout the disciplines of biology, chemistry, physics, engineering, earth sciences, and economics.

Hans Petter Langtangen

A Primer on Scientific Programming with Python

Hans Petter Langtangen
Simula Research Laboratory, Lysaker, Fornebu, Norway

ISSN 1611-0994

ISBN 978-3-642-54958-8 e-ISBN 978-3-642-54959-5
DOI 10.1007/978-3-642-54959-5

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2014945465

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The aim of this book is to teach computer programming using examples from mathematics and the natural sciences. We have chosen to use the Python programming language because it combines remarkable expressive power with very clean, simple, and compact syntax. Python is easy to learn and very well suited for an introduction to computer programming. Python is also quite similar to MATLAB and a good language for doing mathematical computing. It is easy to combine Python with compiled languages, like Fortran, C, and C++, which are widely used languages for scientific computations.

The examples in this book integrate programming with applications to mathematics, physics, biology, and finance. The reader is expected to have knowledge of basic one-variable calculus as taught in mathematics-intensive programs in high schools. It is certainly an advantage to take a university calculus course in parallel, preferably containing both classical and numerical aspects of calculus. Although not strictly required, a background in high school physics makes many of the examples more meaningful.

Many introductory programming books are quite compact and focus on listing functionality of a programming language. However, learning to program is learning how to think as a programmer. This book has its main focus on the thinking process, or equivalently: programming as a problem solving technique. That is why most of the pages are devoted to case studies in programming, where we define a problem and explain how to create the corresponding program. New constructions and programming styles (what we could call theory) is also usually introduced via examples. Particular attention is paid to verification of programs and to finding errors. These topics are very demanding for mathematical software, because the unavoidable numerical approximation errors are possibly mixed with programming mistakes.

By studying the many examples in the book, I hope readers will learn how to think right and thereby write programs in a quicker and more reliable way. Remember, nobody can learn programming by just reading - one has to solve a large amount of exercises hands on. The book is therefore full of exercises of various types: modifications of existing examples, completely new problems, or debugging of given programs.

To work with this book, I recommend using Python version 2.7. For Chapters 5 - 9 and Appendices A.1-E.1 you need the NumPy and Matplotlib packages, preferably also the IPython and SciTools packages, and for Appendix G.1 Cython is required. Other packages used occasionally in the text are nose and `sympy` . Section H.1 has more information on how you can get access to Python and the mentioned packages.

There is a web page associated with this book, http://hplgit.github.com/scipro-primer , containing all the example programs from the book as well as information on installation of the software on various platforms.

Python version 2 or 3?

A common problem among Python programmers is to choose between version 2 or 3,

which at the time of this writing means choosing between version 2.7 and 3.4. The general recommendation is to go for Python 3, because this is the version that will be developed in the future. However, there is still a problem that much useful mathematical software in Python has not yet been ported to Python 3. Therefore, scientific computing with Python still goes mostly with version 2. A widely used strategy for software developers who want to write Python code that works with both versions, is to develop for version 2.7, which is very close to what is found version 3.4, and then use the translation tool 2to3 to automatically translate from Python 2 to Python 3.

When using v2.7, you should employ the newest syntax and modules that make the differences between Python 2 and 3 very small. This strategy is adopted in the present book. Only two differences between versions 2 and 3 are expected to be significant for the programs in the book: `a/b` for integers `a` and `b` implies float division Python 3 and integer division in Python 2. Moreover, `print 'Hello'` in Python 2 must be turned into a function call `print('Hello')` in Python 3. None of these differences should lead to any annoying problems when future readers study the book's v2.7 examples, but program in Python 3. Anyway, running `2to3` on the example files generates the corresponding Python 3 code.

Contents.

Chapter 1 introduces variables, objects, modules, and text formatting through examples concerning evaluation of mathematical formulas. Chapter 2 presents programming with `while` and `for` loops as well as lists, including nested lists. The next chapter deals with two other fundamental concepts in programming: functions and `if-else` tests. Successful further reading of the book demands that Chapters 1 - 3 are digested.

How to read data into programs and deal with errors in input are the subjects of Chapter 4 . Chapter 5 introduces arrays and array computing (including vectorization) and how this is used for plotting $y = f(x)$ curves and making animation of curves. Many of the examples in the first five chapters are strongly related. Typically, formulas from the first chapter are used to produce tables of numbers in the second chapter. Then the formulas are encapsulated in functions in the third chapter. In the next chapter, the input to the functions are fetched from the command line, or from a question-answer dialog with the user, and validity checks of the input are added. The formulas are then shown as graphs in Chapter 5 . After having studied Chapters 1 - 5 , the reader should have enough knowledge of programming to solve mathematical problems by what many refer to as "MATLAB-style" programming.

Chapter 6 explains how to work dictionaries and strings, especially for interpreting text data in files and storing the extracted information in flexible data structures. Class programming, including user-defined types for mathematical computations (with overloaded operators), is introduced in Chapter 7 . Chapter 8 deals with random numbers and statistical computing with applications to games and random walks. Object-oriented programming, in the meaning of class hierarchies and inheritance, is the subject of Chapter 9 . The key examples here deal with building

toolkits for numerical differentiation and integration as well as graphics.

Appendix A.1 introduces mathematical modeling, using sequences and difference equations. Only programming concepts from Chapters 1 - 5 are used in this appendix, the aim being to consolidate basic programming knowledge and apply it to mathematical problems. Some important mathematical topics are introduced via difference equations in a simple way: Newton's method, Taylor series, inverse functions, and dynamical systems.

Appendix B.1 deals with functions on a mesh, numerical differentiation, and numerical integration. A simple introduction to ordinary differential equations and their numerical treatment is provided in Appendix C.1. Appendix D.1 shows how a complete project in physics can be solved by mathematical modeling, numerical methods, and programming elements from Chapters 1 - 5 . This project is a good example on problem solving in computational science, where it is necessary to integrate physics, mathematics, numerics, and computer science.

How to create software for solving ordinary differential equations, using both function-based and object-oriented programming, is the subject of Appendix E.1. The material in this appendix brings together many parts of the book in the context of physical applications.

Appendix F.1 is devoted to the art of debugging, and in fact problem solving in general. Speeding up numerical computations in Python by migrating code to C via Cython is exemplified in Appendix G.1,. Finally, Appendix H.1 deals with various more advanced technical topics.

Most of the examples and exercises in this book are quite short. However, many of the exercises are related, and together they form larger projects, for example on Fourier Series (3.15 , 4.20 , 4.21 , 5.39 , 5.40), numerical integration (3.6 , 3.7 , , 5.48 , A.12), Taylor series (3.31 , 5.30 , 5.37 , A.14, A.15, 7.23), piecewise constant functions (3.23 - 3.27 , 5.32 , 5.45 , 5.46 , 7.19 - 7.21), inverse functions (E.17-E.20), falling objects (E.8, E.9, E.38, E.39), oscillatory population growth (A.19, A.21, A.22, A.23), epidemic disease modeling (E.41-E.48), optimization and finance (A.24, 8.39 , 8.40), statistics and probability (4.23 , 4.24 , 8.21 , 8.22), hazard games (8.8 - 8.13), random walk and statistical physics (8.30 - 8.37), noisy data analysis (8.41 - 8.43), numerical methods (5.23 - 5.25 , 7.8 , 7.9 , A.9, 7.22 , 9.15 - 9.17 , E.30-E.37), building a calculus calculator (7.33 , 9.18 , 9.19), and creating a toolkit for simulating vibrating engineering systems (E.50-E.55).

Chapters 1 - 9 together with Appendices A.1 and E.1 have from 2007 formed the core of an introductory first semester bachelor course on scientific programming at the University of Oslo (INF1100, 10 ECTS credits).

Changes from the third to the fourth edition.

A large number of the exercises have been reformulated and reorganized. Typically, longer exercises are divided into subpoints a), b), c), etc., various type of help is factored out in separate paragraphs marked with **Hint** , and information that puts the exercise into a broader context is placed at the end under the heading **Remarks** . Quite some related exercises have been merged.

Another major change is the enforced focus on testing and verification. Already as

soon as functions are introduced in Chapter 3 , we start verifying the implementations through test functions written according to the conventions in the nose testing framework. This is continued throughout the book and especially incorporated in the reformulated exercises. Testing is challenging in programs containing unknown approximation errors, so strategies for finding appropriate test problems have also become an integral part of the fourth edition.

Many chapters now refer to the Online Python Tutor for visualizing the program flow. This is a splendid tool for learning what happens with the variables and execution of statements in small programs. The `sympy` package for symbolic computing is a powerful tool in scientific programming and introduced already in Chapter 1 . The sections in Chapter 4 have been reorganized, and the basic information on file reading and writing was moved from Chapter 6 to Chapter 4 . The fourth edition clearly features three distinct parts: basic programming concepts in Chapters 1 - 5 , more advanced programming concepts in Chapters 6 - 9 , and programming for solving science problems in Appendix A.1-E.1.

Sections 4.9 and 4.10.2 have been rewritten to emphasize the importance of test functions. The information on how to make animations and videos in Sections 5.3.4 and 5.3.5 has undergone a substantial revision. Section 6.1.7 has been completely rewritten to better reflect how to work with data associated with dates.

Appendix E.1 has been reworked so that function-based programming and object-oriented programming appear in separate sections. This allows reading the appendix and solving ODEs without knowledge of classes and inheritance. Much of the text in Appendix E.1 has been rewritten and extended, the exercises are substantially revised, and several new exercises have been added.

Section H.1 is new and describes the various options for getting access to Python and its packages for scientific computations. This topic includes, e.g., installing software on personal laptops and writing notebooks in cloud services.

In addition to the mentioned changes, a large number of smaller updates, improved explanations, and correction of typos have been incorporated in the new edition. I am very thankful to all the readers, instructors, and students who have sent emails with corrections or suggestions for improvements.

The perhaps biggest change for me was to move the whole manuscript from LaTeX to Doconce [1] . This move enables a much more flexible composition of topics for various purposes, and support for output in different formats such as LaTeX, HTML, Sphinx, Markdown, IPython notebooks, and MediaWiki. The chapters have been made more independent by repeating key knowledge, which opens up for meaningful reading of only parts of the book, even the most advanced parts.

Acknowledgments.

This book was born out of stimulating discussions with my close colleague Aslak Tveito, and he started writing what is now Appendix B.1 and C.1. The whole book project and the associated university course were critically dependent on Aslak's enthusiastic role back in 2007. The continuous support from Aslak regarding my book projects is much appreciated and contributes greatly to my strong motivation. Another

key contributor in the early days was Ilmar Wilbers. He made extensive efforts with assisting the book project and establishing the university course INF1100. I feel that without Ilmar and his solutions to numerous technical problems the first edition of the book would never have been completed. Johannes H. Ring also deserves special acknowledgment for the development of the Easyviz graphics tool and for his careful maintenance and support of software associated with the book over the years.

Professor Loyce Adams studied the entire book, solved all the exercises, found numerous errors, and suggested many improvements. Her contributions are so much appreciated. More recently, Helmut Büch worked extremely carefully through all details in Chapters 1 - 6 , tested the software, found many typos, and asked critical questions that led to lots of significant improvements. I am so thankful for all his efforts and for his enthusiasm during the preparations of the fourth edition.

Special thanks go to Geir Kjetil Sandve for being the primary author of the computational bioinformatics examples in Sections 3.3 , 6.5 , 8.3.4 , and 9.5 , with contributions from Sveinung Gundersen, Ksenia Khelik, Halfdan Rydbeck, and Kai Trengereid.

Several people have contributed with suggestions for improvements of the text, the exercises, and the associated software. I will in particular mention Ingrid Eide, Ståle Zerener Haugnæss, Kristian Hiorth, Arve Knudsen, Tobias Vidarssønn Langhoff, Martin Vonheim Larsen, Kine Veronica Lund, Solveig Masvie, Håkon Møller, Rebekka Mørken, Mathias Nedrebø, Marit Sandstad, Helene Norheim Semmerud, Lars Storjord, Fredrik Heffer Valdmanis, and Torkil Vederhus. Hakon Adler is greatly acknowledged for his careful reading of early various versions of the manuscript. Many thanks go to the professors Fred Espen Bent, Ørnulf Borgan, Geir Dahl, Knut Mørken, and Geir Pedersen for formulating several exciting exercises from various application fields. I also appreciate the cover image made by my good friend Jan Olav Langseth.

This book and the associated course are parts of a comprehensive and successful reform at the University of Oslo, called Computing in Science Education . The goal of the reform is to integrate computer programming and simulation in all bachelor courses in natural science where mathematical models are used. The present book lays the foundation for the modern computerized problem solving technique to be applied in later courses. It has been extremely inspiring to work closely with the driving forces behind this reform, especially the professors Morten Hjorth-Jensen, Anders Malthe-Sørenssen, Knut Mørken, and Arnt Inge Vistnes.

The excellent assistance from the Springer system, in particular Martin Peters, Thanh-Ha Le Thi, Ruth Allewelt, Peggy Glauch-Ruge, Nadja Kroke, Thomas Schmidt, Patrick Waltemate, and Donatas Akmanavicius, is highly appreciated, and ensured a smooth and rapid production of all editions of this book.

Oslo, March 2014 Hans Petter Langtangen

Contents