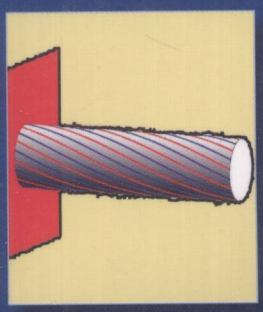
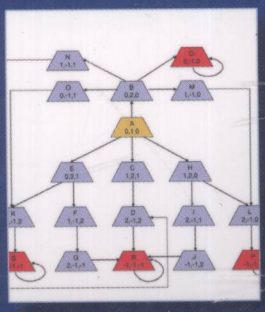
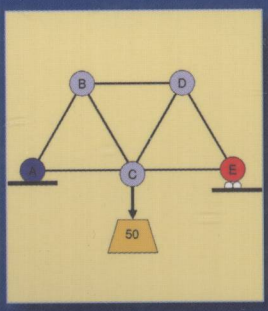
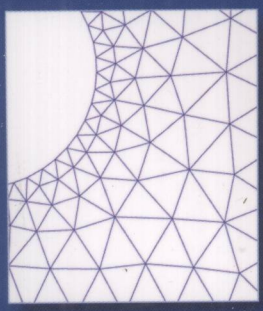
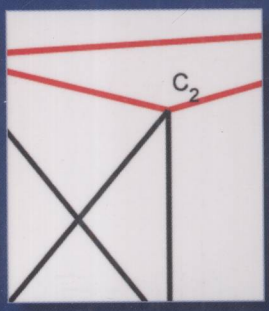
Scientific Computing

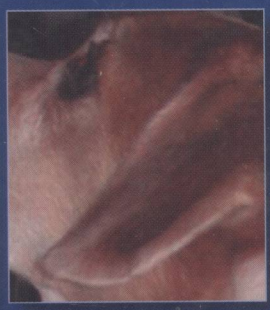
WITH CASE STUDIES









Dianne P. O'Leary

Scientific Computing

WITH CASE STUDIES

Dianne P. O'Leary

University of Maryland
College Park, Maryland

siam
Society for Industrial and Applied Mathematics
Philadelphia

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data

O'Leary, Dianne P.
Scientific computing with case studies / Dianne P. O'Leary.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-898716-66-5
1. Mathematical models--Data processing--Case studies. I. Title.
QA401.O44 2008
510.285--dc22

2008031493

To Gene H. Golub, my first research mentor.
To my parents, Raymond and Anne Prost.
To my husband, Timothy.
To my children, Theresa, Thomas, and Brendan.
With love.

Preface

A master carpenter does not need to know how her hammer was designed or what Newton's laws say about the force that the hammer applies. But she does need to know how to use the hammer, when to use a ball-peen hammer instead, and what to do when things go wrong, for example, when a nail bends as it is driven.

We take the same viewpoint in this book. Although there are fascinating stories to tell in the details of how basic numerical algorithms are designed and how they operate, we view them as tools in our virtual toolbox, discussing the innards just enough to be able to master their uses. Instead we focus on how to choose the most appropriate algorithm, how to make use of it, how to evaluate the results, and what to do when things go wrong.

This viewpoint frees us to explore many diverse applications of our tools, and through such case studies we practice the analysis and experimentation that are the mainstays of computational science.

The reader should have background knowledge equivalent to a first course in scientific computing or numerical analysis. Excellent textbooks for learning this information include those by Michael Heath [71], Cleve Moler [108], and Charles Van Loan [148].

Examples and illustrations use the MATLAB® programming language. Standard MATLAB functions provide us with our basic numerical algorithms, and the graphics interface is quite useful. For some problems, we make use of some of the MATLAB toolboxes, in particular, the Optimization Toolbox. If you do not have access to MATLAB, the basic numerical algorithms can also be obtained from NETLIB and other sources noted in the text. Sample programs for each case study are available at the website

`www.cs.umd.edu/users/oleary/SCCS/`

No single book can give a computational scientist all of the background needed for a career. In fact, computational science is primarily a collaborative enterprise, since it is rare that a single individual has all of the computational and scientific background necessary to complete a project. My hope is that this particular slice of knowledge will prove useful in your work and will lead you to further study, exciting applications, and productive collaborations.

I'm grateful to my many mentors, collaborators, and students, who through their probing questions forced me to seek deeper understanding and clearer explanations. May you too be blessed with good colleagues.

Notes to Students

This book is written as a textbook for a second course in scientific computing, so it assumes that you have had a semester (or equivalent) of background using a standard textbook such as that by Heath [71], Moler [108], Van Loan [148], or equivalent. The Basics box at the beginning of each unit tells you what part of this material you might want to review in preparation for the unit. The Mastery box is a checklist of points to master in working through the unit.

The basic premise behind this book is that people learn by doing. Therefore, the book is best read with a pencil, paper, and MATLAB window close at hand. Challenges are sprinkled throughout the text, and they are meant be worked as they are encountered, or at least before the end of the chapter. Answers are provided for most challenges at

`www.cs.umd.edu/users/oleary/SCCS/`

There you can see examples of how someone else worked through the challenges. Mastery will be best if the answers are used to verify and refine your own approach to the problem. Merely reading the answer, though tempting, is (unfortunately) no substitute for trying to work the challenge on your own.

Pointers give important information and references to additional literature and software. I hope the content of this book leads you to want to learn more about scientific computing.

Notes to Instructors

The material in this book has been used for a semester and a half in a graduate level course in the applied mathematics program at the University of Maryland.

- I lecture from the introductory material in each unit, with material from the Case Studies used to occasionally provide extra information and motivation. Students can become quite passionate about some of the Case Studies, especially the more visual ones such as the image deblurring problem (Chapter 6), the data clustering problem (Chapter 11), and the epidemiology models (Chapter 19 and 21).
- For quizzes and exams, I derive problems from the Mastery points at the beginning of each unit.
- If possible, I like to allow "laboratory time" in class for students to work on some of the Challenges. The opportunity to see how other people solve problems is helpful even to the best students. This is especially true if, as at the University of Maryland, the students in this course come from backgrounds in mathematics, computer science, and engineering. This provides a remarkably diverse set of viewpoints on the material and enriches the dialog.
- Many of the **Case Studies** were originally homeworks.
- For a term project, I often ask students to develop a Case Study, using the tools presented in the course to solve a problem in their application area. Such projects can then be adapted for use in later terms. My students Nargess Memarsadeghi, David A. Schug, and Yalin E. Sagduyu developed particularly interesting case studies, and adapted versions of them are included here.

- There are not many unsolved exercises in this book. In the age of the Internet, there are very few textbook problems for which solutions cannot be found somewhere, and providing solutions here at least puts all students on equal footing. Some unsolved exercises and Case Studies are available on the book's website, and I would be grateful for your contribution of additional ones to post there.

There is a great deal of flexibility in choice and ordering of units, except that the optimization unit should be covered before nonlinear equations, and dense matrix computations should be discussed before optimization. The first six units form the syllabus for a one semester course at Maryland, while the final one is combined with a textbook in numerical solution of partial differential equations for the second semester.

Acknowledgments

I am grateful for the help of many, including the following:

- *Computing in Science and Engineering*, published by the American Institute of Physics and the IEEE Computer Society, for permission to include chapters derived from the case studies published there: Chapters
  1 (Vol. 8, No. 5, 2006, pp. 86–90),
  3 (Vol. 8, No. 3, 2006, pp. 86–89),
  4 (Vol. 7, No. 6, 2006, pp. 78–80),
  6 (Vol. 5, No. 3, 2003, pp. 82–85),
  7 (Vol. 8, No. 2, 2006, pp. 66–70),
  8 (Vol. 5, No. 6, 2003, pp. 60–63),
  11 (Vol. 5, No. 5, 2003, pp. 54–57),
  12 (Vol. 6, No. 5, 2004; pp. 60–62),
  13 (Vol. 6, No. 3, 2004, pp. 66–69),
  14 (Vol. 7, No. 1, 2005, pp. 56–59),
  15 (Vol. 7, No. 2, 2005, pp. 60–62),
  17 (Vol. 9, No. 1, 2007, pp. 72–76),
  18 (Vol. 6, No. 6, 2004; pp. 58–62),
  19 (Vol. 6, No. 1, 2004, pp. 68–70),
  21 (Vol. 6, No. 2, 2004, pp. 50–53),
  22 (Vol. 5, No. 4, 2003, pp. 68–71),
  23 (Vol. 7, No. 3, 2005, pp. 20–23),
  26 (Vol. 9, No. 2, 2007, pp. 96–99),
  27 (Vol. 7, No. 5, 2005, pp. 62–67),
  28 (Vol. 8, No. 4, 2006, pp. 74–78),
  29 (Vol. 6, No. 4, 2004, pp. 74–76),
  30 (Vol. 7, No. 6, 2005, pp. 74–77),
  31 (Vol. 7, No. 4, 2005, pp. 68–70),
  32 (Vol. 8, No. 5, 2006, pp. 86–90).

- Jennifer Stout, Lead Editor of *Computing in Science and Engineering*, who patiently edited the case studies.

- Mei Huang, for her work on Chapter 18.

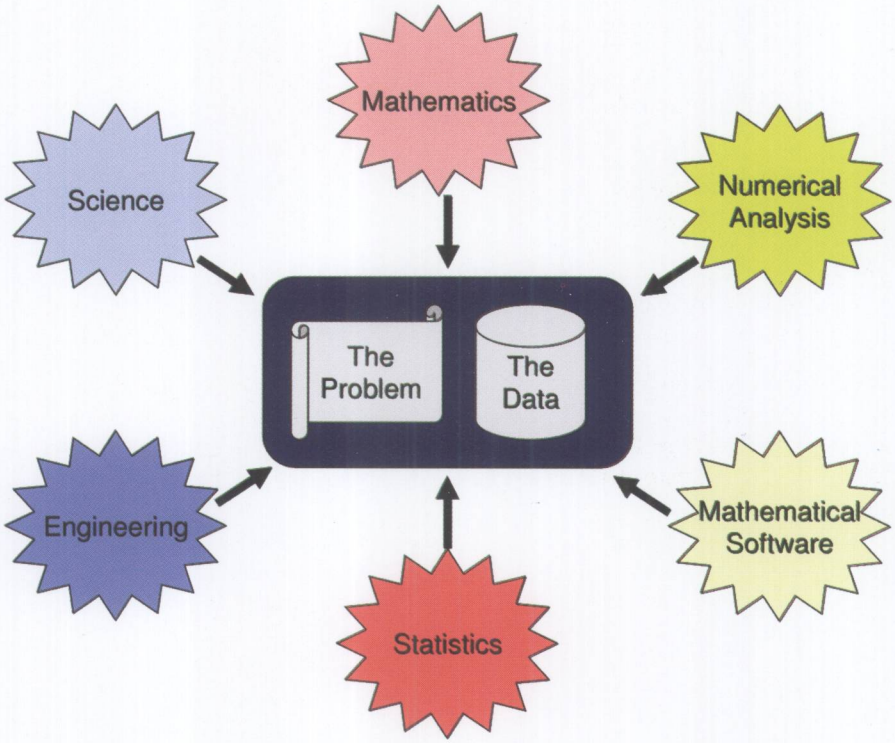
- Jin Hyuk Jung, who as a teaching assistant wrote supplementary lecture notes from which some of the figures were taken, particularly those in Chapters 5, 9, and 24.
- Nargess Memarsadeghi, David Schug, and Yalin Sagduyu, whose term projects were so interesting that they led to case studies included here.
- Staff in the Technical Support Department at The MathWorks, for discussions about the sources of overhead in MATLAB interpreted and compiled instructions.
- James G. Nagy, a master teacher, who inspired the case studies and coauthored the first one.
- The National Science Foundation and the National Institute of Standards and Technology, for supporting my research into many of the problems discussed in the case studies.
- Timothy O'Leary for the photo of Charlie in Chapter 11.
- Students in the University of Maryland courses Scientific Computing I and II: (especially Samuel Lamphier) for their patience and debugging as the notes were developed.
- G. W. Stewart, for his example of clearly written textbooks and for the privilege of being his colleague at Maryland.
- Howard Elman, David Gilsinn, Vadim Kavalerov, Tamara Kolda, Samuel Lamphier, K.J.R. Liu, Brendan O'Leary, Bert Rust, Simon P. Schurr, Elisa Sotelino, G. W. Stewart, and Layne T. Watson for helpful comments.

The images in Figure 1.1 were taken from `http://nightglow.gsfc.nasa.gov/eric_journal_files/sydney_bridge.jpg` and `http://www.cpsc.gov/cpscpub/prerel/prhtml07/07267a.jpg`, and that in Figure 26.1 (`http://www.myrmecos.net/insects/Tribolium1.html`) is owned by Alex Wild.

Contents

Unit I

Preliminaries: Mathematical Modeling, Errors, Hardware and Software



The topic of this book is efficient and accurate computation with mathematical models. In this unit, we discuss the basic facts that we need to know about error, software, and computers.

We begin our study with some basics. First in Chapter 1 we consider how errors are introduced in scientific computing and how to measure them. We apply these principles in Chapter 2, studying how small changes in our data can affect our answers. In Chapter 3, we see how computer memory is organized and how that impacts the efficiency of our algorithms. Then in Chapter 4, we study the principles behind writing and documenting our algorithms.

BASICS: To understand this unit, the following background is helpful:

- MATLAB programming [78].
- Gauss elimination; see a linear algebra textbook or a beginning book on numerical analysis or scientific computing [148].

MASTERY: After you have worked through this unit, you should be able to do the following:

- Identify the sources of error in scientific computing.
- Represent an integer in a fixed-point number system and a real number in a floating-point number system.
- Use the parameter ϵ_m (machine epsilon) to determine the error introduced in floating-point representation.
- Measure relative and absolute errors and determine how they are magnified during computation.
- Write algorithms that compute values such as the sum of a series, avoiding unnecessary inaccuracies.
- Determine ways to avoid catastrophic cancellation in designing algorithms.
- Use forward and backward error analysis to assess the quality of a computed solution to a problem.
- Determine whether a problem is well-conditioned or ill-conditioned.
- Discuss the importance of stability in an algorithm.
- Measure the sensitivity of a problem using derivatives, condition numbers, Monte Carlo experiments, and confidence intervals.
- Distinguish between a row-oriented matrix algorithm and a column-oriented matrix algorithm, and be able to write them for simple tasks.
- Explain how matrices are stored in main memory and moved to cache, and perform counts of page moves.
- Count the number of multiplications in a given MATLAB algorithm.
- Explain what the BLAS are and why they are useful.
- Document computer programs effectively.
- Understand the principles of modular design.
- Write a program to validate a function that you have written.