

Preface

Scientists and engineers have always been among the leading innovators in computation: the designers of the first electronic computers, the developers of the first high-level computer language (FORTRAN), and the creators of the Internet and the World Wide Web. Scientists have also been innovators in computer education, perhaps motivated by the importance of computers in modern science and the ongoing need to train young scientists and engineers to become proficient in developing the new computational methods and ideas of their disciplines.

This book continues the tradition of innovation in the computer education of scientists and engineers by providing an integrated approach. It addresses not only the topic of programming but also the important methods and techniques of scientific computation (graphics, the organization of data, data acquisition, numerical methods, etc.) as well as the design and organization of software.

In almost all scientific programming texts, the emphasis is on numeric data. A major theme of this book is that numbers, despite their central role in scientific computation, are not the only important objects of computation. Scientists have to represent on the computer the real-world or theoretical entities that are the subject of their scientific work. This goes well beyond the built-in data types of computer languages (floating point numbers, character strings, etc.) to include complex entities like census databases, bridge designs, voice recordings, and x-ray photographs. Today's students have grown up using computers to perform sophisticated searches of distributed databases, compose and play musical sounds, and edit photographs

and videos. A modern text on scientific computation must be based on examples that have a similar level of realism and richness.

Another way in which this book attempts to be innovative involves finding the appropriate middle path between two extreme approaches used in teaching about computers: *teach-a-package* and *teach-a-language*. The *teach-a-package* approach emphasizes the computations that can be performed with a particular piece of software such as EXCEL, MINITAB, or SPSS. This approach is appropriate in teaching subjects such as introductory statistics where there is a limited set of computations that need to be performed (e.g., the *t*-test, analysis of variance [ANOVA], and regression). Unfortunately, it fails to give students the skills needed to express their own ideas or to work outside of a limited domain.

The *teach-a-language* perspective is that students should work with a general-purpose system (e.g., C++ or Java) that is sufficiently flexible to build anything that needs to be built. This provides training in programming but is so time consuming that there is little opportunity left to learn about computation. For example, one of the most important operations for scientists is the production of scientific graphics; however, using C++ or Java few people are able to generate scientific graphics with less than a couple of months' experience.

The approach taken here is to teach general-purpose language skills and concepts and to take advantage of existing software so that realistic computations can be performed after just a few days or weeks of class time. Examples in the book include text processing, database searching, the synthesis of false-color images and the building of user interfaces. Given the right software and the right language skills, impressive feats of computation can be accomplished by beginners.

The book grows out of my own experiences over two decades working with scientists and students of science—chemists, economists, electrical and biomedical engineers, mathematicians, molecular biologists, neuroscientists, physicists, physicians, statisticians, and others—both in research and in formal classroom settings. My goal is to provide readers with the skills and concepts needed to be able to use the computer expressively in scientific work.

Most students using this book will take one and only one course on computation and programming. It's necessary, but not sufficient, to cover the basics of programming: how to express algorithms, and how to use programming-language constructs such as conditionals and loops, and ways to organize, save, and retrieve information. But today's students will be working in a web of software of startling complexity. An important part of their training must be in the principles of software development that help them to create programs that exploit and work with a large body of existing software. If modern scientists are to see further, they must, like Newton, stand on the shoulders of giants. In the case of software, this often means using existing software rather than building from the ground up.

Studying computation is similar in many ways to learning a foreign language. It's not enough to read about the principles of communication and grammar; one has to practice and use them actively. This is why beginning language courses are always about a particular language (e.g., English, Chinese or French). There is little point in studying languages abstractly except at an advanced level.

This book uses the computer language MATLAB. The use of a particular language allows us to be specific and to illustrate abstract concepts with working examples. MATLAB has many important advantages as a first language for scientists: It has a relatively simple grammar and is interpreted rather than compiled; it provides an integrated development environment that includes computation, graphics, user interface design and debugging; and it is powerful enough to handle realistic computations. Many operators of importance to scientists are built-in. Beyond its uses in education, MATLAB is a powerful professional-level programming environment that is widely used in academic, commercial, and government centers.

MATLAB is flexible sufficiently so that it can be used for general-purpose projects, and MATLAB's package-like aspects make it highly efficient for scientific computation. Almost all of the concepts that one learns using MATLAB are transferable to languages such as C++ and Java. Inevitably, though, there are important operations, such as the detailed manipulation of graphics and graphical user interfaces, where learning how to perform the operation in MATLAB provides little in the way of transferable skills. I have tried to minimize the use of such operations and, when they are needed, to emphasize general principles over language-specific issues. For example, in discussing user interfaces, the general concept of a “callback” function is introduced; the means of creating them in MATLAB is touched on only lightly.

I use this book in a one-semester course at Macalester College called “Introduction to Scientific Programming.” The course attracts students from all scientific disciplines and levels, including many first-year students, most of whom have no previous programming experience and a mathematical background consisting of only a single semester of calculus. The examples used in the book are designed to be accessible to such students, while remaining compelling to more advanced students.

The basic material on programming is contained in Chapters 1 through 8, which can be covered in six to nine weeks depending on the intensity of the course and the enthusiasm of the students. Depending on the interests and orientations of the students and the instructor, the remaining chapters can be covered in a variety of ways. For example, a course with an emphasis on programming might supplement the first eight chapters with Chapters 9 through 12: scope, events, data organization, and recursion.

I find that students are motivated strongly by graphical user interfaces. Despite my warnings that they are among the most difficult types of programs to write, students want to learn this skill as soon as possible.

Chapters 9 and 10 provide an introduction to simple but useful graphical user interfaces. The goal is not to reproduce the highly refined user interfaces of mass-market programs but to enable scientists to collect those sorts of input most effectively provided with a mouse or with keystrokes.

General science students are well served by the material in Chapters 13 and 14. Such students are keen to perform computations on real-world objects; sounds and images provide an excellent motivation and illustrate general principles. Chapters 15 and 16 introduce basic concepts and methods of what is traditionally called “scientific computing.” The material in that chapter—solving equations, optimization, interpolation—is accessible to most students because it is limited to one dimension. Concepts can be illustrated simply and concretely with a graph.

A course oriented to mathematicians might cover the first eight chapters, then Chapters 12 (Recursion), 15, 16, and 17. Chapter 17 extends the concepts and methods of Chapter 16 to multiple dimensions. This is suitable for students with a comparatively advanced background including linear algebra and serves as a transition to a follow-up numerical methods course.

Each chapter includes a number of short exercises that provide a means to learn and reinforce the material in that chapter. In later chapters, there is a shift of focus to short projects of a more open-ended nature. These projects integrate the various skills needed for scientific computation. For example, scientific graphics is only partly about drawing lines and points and the operators needed to do this; it’s important to be able to store and read in data and therefore to manipulate files and their contents, translating them into physical units as appropriate. Project 9.10 involves all of these facets in the task of plotting an electrocardiogram according to the clinically accepted format. Other projects introduce diverse areas of science and technology, but all are self-contained, requiring no previous exposure to the relevant area of science.

I would like to acknowledge my colleagues G. Michael Schneider, Tom Halverson, Libby Shoop, Stan Wagon, Lenny Smith, Leon Glass, Sarah Little, and Steve Panizza who contributed to this book in various ways, and to thank Priya Arora, Nelson Coates, Issidor Iliev, Nazim Osmanicik, and the other students who worked with successive drafts of this book over several years in CS 20 (now CS 121) at Macalester College. Special thanks are due my wife, Maya Hanna, and daughters Tamar, Liat, and Netta who provided support and encouragement for this work.