1
Introduction

Beginnings are hard.

Chaim Potok

1.1
Computational Physics and Computational Science

This book adopts the view that computational physics (CP) is a subfield of computational science. This means that CP is a multidisciplinary subject that combines aspects of physics, applied mathematics, and computer science (Fig. 1.1), with the aim of solving realistic physics problems. Other computational sciences replace the physics with biology, chemistry, engineering, etc. At present, grand challenge problems facing computational science include:

Climate Prediction	Materials Science	Structural Biology
Superconductivity	Semiconductor Design	Drug Design
Human Genome	Quantum Chromodynamics	Turbulence
Speech and Vision	Relativistic Astrophysics	Vehicle Dynamics
Nuclear Fusion	Combustion System	Oil and Gas Recovery
Ocean Science	Vehicle Signature (Mil.)	Undersea Surveillance

Although related, computational science is not computer science. Computer science studies computing for its own intrinsic interest and develops the hardware and software tools that computational scientists use. Likewise, applied mathematics develops and studies the mathematics of the algorithms that computational scientists use. Although we too find mathematics and computer science interesting for its own sake, our focus is on solving physical problems and in understanding the computing tools that we use in order to be sure that we have solved the problems correctly.

As CP has matured, we have come to realize that it is more than the overlap of physics, computer science, and mathematics (Fig. 1.1). It is also a bridge among them (the central region in the figure) containing core elements of its own, such as computational tools and methods. In our view, CP's common-

Computationyal Physics. Problem Solving with Computers (2nd edn).
Rubin H. Landau, Manuel José Páez, Cristian C. Bordeianu

ISBN: 978-3-527-40626-5

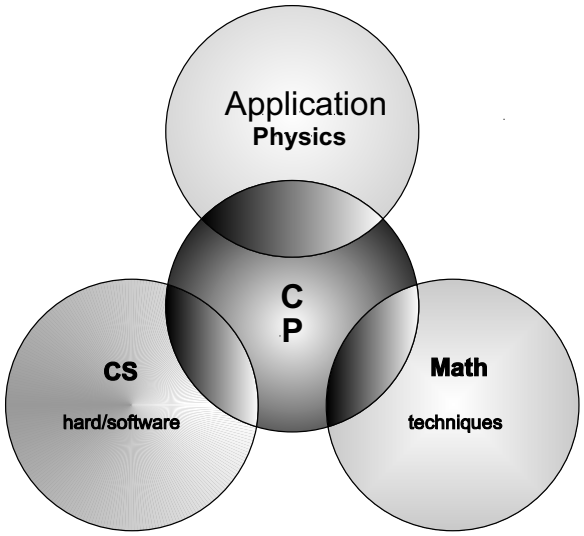

Fig. 1.1 A representation of the multidisciplinary nature of computational physics not only as the overlap of physics, applied mathematics, and computer science, but also as a bridge among the disciplines.

ality of tools and similar problem-solving mindset [1] draw it closer to the other computational sciences and away from the subspecialization found in so much of physics.

In order to emphasize our computational science focus, we present most of the subjects in this book in the form of a problem to solve, with the components that constitute the solution separated according to the scientific problem-solving paradigm (Fig. 1.2). Clearly, this paradigm is more appropriate to dealing with physical problems than to developing basic skills, yet even in the former case we often follow it for the purpose of uniformity and organization.

Traditionally, physics follows either an experimental or a theoretical approach to discover scientific truth. Being able to transform a theory into an algorithm requires significant theoretical insight, detailed physical and mathematical understanding, and mastery of the art of programming. The actual debugging, testing, and organization of scientific programs have analogies to experiment, with the numerical simulations of nature essentially virtual experiments. The synthesis of numbers into generalizations, predictions, and conclusions requires the insight and intuition common to both experimental and theoretical science. In fact, the use of computation and simulation has now become so prevalent and essential a part of the scientific process that many people have suggested that the scientific paradigm can be extended to include simulation as an additional dimension.

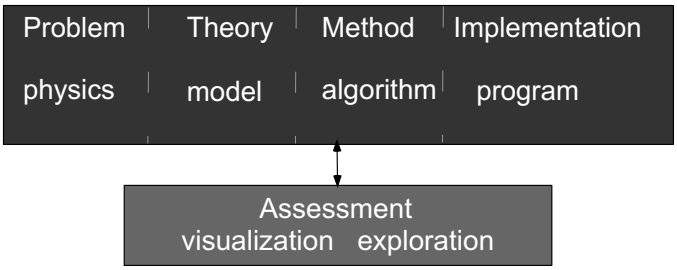

Fig. 1.2 The problem solving paradigm followed in this book.

We would like the reader of this book to learn how to use computing to help explore the physical world at depths greater than otherwise possible. This may be by solving problems whose difficulty and complexity place them beyond analytic solution or human endurance, or by using the computer as a virtual laboratory that simulates a physical system. Computing is not a substitute for analytic or mathematical skills, but rather a lever for those skills in order for the reader to do something new and different.

1.2 How to Use this Book

The operational aim of this book is to survey the multidisciplinary subjects that constitute what we think of as undergraduate computational physics. For this to be a learning experience, the reader must *do* the highlighted problem at the beginning of each chapter or unit. This means that the reader should study the text, write, debug and run a program, visualize the results, and then express in words what has been done and what can be concluded. In contrast to other books which may emphasize the theory behind CP, we want the reader to understand the algorithm and the program as well as the physics, and to consider the former two as much a part of the scientific process as the latter one.

Although it is possible to use a *problem-solving environment* such as *Maple, Mathematica*, or *Matlab* to solve many of these problems without having to write any programs, we recommend that the user "get their hands dirty" by writing programs in a compiled language. We believe it is an important part of the learning experience for you to get close to the hardware and software workings of the computer and to understand the algorithms being used. While indubitably valuable for the well-trained professional, we think that the problem-solving environments hide too much of the details from some-

one trying to learn CP.[1] In addition, while the analytic solutions provided by these packages are often perfect for textbook problems, they are hardly ever useful for realistic problems; or must be evaluated numerically to plot a graph.

While carrying through all the steps in our projects no doubt entails some work, it should be exciting and fun to see the math and physics come alive on the screen before your eyes. Often, as a consequence of having to interact with the materials from a number of viewpoints, and on your own, the materials will become part of your personal experiences and end up being more meaningful than possible via any other way of learning.

Programming is a valuable skill for all scientists, but it is also incredibly demanding and time consuming. In order to lighten the workload somewhat we provide "bare bones" programs in the text and on the CD. These can be used as models or helpers for the reader's own programs (the best learning experience if time permits), or tested and modified to solve the problem at hand. (More complete versions of the programs are available to instructors by contacting the authors or publisher.) The CD also includes animations and interactive applets based on the text materials, as well as sample data files and visualization scripts.

The programs in the text are given in the Java programming language. The CD contains corresponding versions in C and Fortran; regardless of the language, the algorithm, the math, the logic, and the physics remain the same. Our choice of Java may surprise some readers who view it mainly for Web computing. Actually, Java is quite good for CP education because it demands proper syntax, produces useful error messages, is essentially universal (the same on all computers), and has free program development environments available from Sun so that learners can work on any computer they want. Of particular importance for scientific computing is that Java is consistent and intelligent in its handling of precision, in contrast to C. Although Java is slower than C and Fortran (something we document in Chap. 13), we found this irrelevant for educational projects with the present-day fast computers. Because of the similarity in structure, it is easy and wise to transform a Java program into C for research projects. Finally, although Java is an object-oriented programming (OOP) language (something we discuss at length in Chap. 4), in order to keep the programs closer to the math and readable to a wider audience, we do not emphasize the object orientation in our sample programs, but do encourage you to use OOP techniques.

One of the most rewarding uses of computers is *visualizing* the results of calculations with 2D and 3D plots, with color, and with animation. This assists the debugging process, hastens the development of physical and mathemati-

[1] This is not meant to be a criticism of problem-solving environments; in fact our introductory scientific computing book [2] covers both Maple and Mathematica.

cal intuition, and increases the enjoyment of the work. It is essential that you learn to use visualization tools as soon as possible, and in [2] and [3] we describe a number of free visualization tools that we use and recommend. We include many figures showing visualizations in this edition (unfortunately just in grey scale), with additional animations, applets, and color images on the CD. Furthermore, there is an online version of a course using this book under development, and those interested should contact RHL.

The projects and problems incorporated into the text are an essential part of the learning experience. Some are exercises to ensure that you have your programs functioning properly, while others ask you to think creatively and do self-directed exploration and analysis. In working on these projects, we suggest that you try to develop an attitude of independence and organization, but also that you learn to communicate with others for assistance when you get stuck debugging.