1
Introduction

Beginnings are hard.
—Chaim Potok

Nothing is more expensive than a start.
— Friedreich Nietzsche

We start this book with a description of how computational physics (CP) fits into the broader field of computational science, and how CP fits into physics. We describe the subjects we cover, the coordinated video lectures, and how the book may be used in a CP course. Finally, we get down to business by discussing the Python language and its many packages, some of which we'll use. In Chapter 2 we give an introduction to Python programming, and in Chapter 7 we examine Python's treatment of matrices.

1.1
Computational Physics & Science

As illustrated in Figure 1.1, we view computational physics as a bridge that connects physics, computer science (CS), and applied mathematics. Whereas CS studies computing for its own intrinsic interest and develops the hardware and software tools that computational scientists use, and while applied mathematics develops and studies

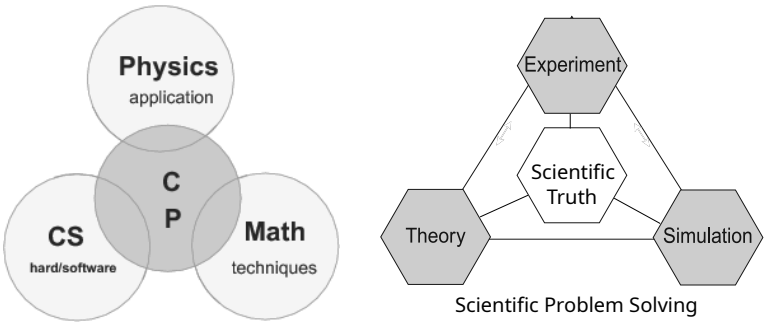

Figure 1.1 On the left a view of Computational Physics as a discipline encompassing physics, applied mathematics, and computer science. On the right is a broader view of Computational Physics fitting into various components of scientific problem solving.

the algorithms that computational scientists use, CP focuses on using all of that to do better and new physics. Furthermore, just as an experimentalist must understand many aspects of an experiment to ensure that her measurements are accurate and believable, so should every physicist undertaking a computation understand the CS and math well enough ensure that her computations are accurate and precise.

As CP has matured, we see it not only as a bridge among disciplines, but also as a specialty containing core elements of it own, such as data-mining tools, computational methods, and a problem-solving mindset. To us, CP's commonality of tools and viewpoint with other computational sciences makes it a good training ground for students, and a welcome change from the overspecialization found in so much of physics.

As part of this book's emphasis on problem solving, we strive to present the subjects within a problem-solving paradigm, as illustrated on the right of Figure 1.1. Our's is a hands-on, inquiry-based approach in which there are problems to solve, a theory or an appropriate model to apply, an appropriate algorithm to use, and an assessment of the results. This approach can be traced back to the post World War II research techniques developed at US national laboratories. They deserve the credit for extending the traditional experimental and theoretical approaches of physics to also include simulation. Recent developments have also introduced powerful data mining tools, such as neural networks, artificial intelligence, and quantum computing.

1.2 This Book's Subjects

We do not intend this book to be a scholarly exposition into the foundations of CP. Instead, we employ a learn-by-doing approach with many exercises, problems, and ready-to-run codes. We survey many of the subjects that constitute CP at a level appropriate for undergraduate education, except maybe for the latter parts of some chapters. Our experience is that many graduate students and professionals may also benefit from this survey approach in which a basic understanding of a broad range of topics facilitates further in-depth study.

The early chapters covers basic numerics, ordinary differential equations with (many) applications, matrix computing using well-developed linear algebra libraries, and Monte-Carlo methods. Some powerful data mining tools such as discrete Fourier transforms, wavelet analysis, principal component analysis, and neural networks are covered in the middle of the book.

A traditional way to view the materials in this text is in terms of its use in courses. For a one-quarter class we used approximately the first third of the text, with its emphasis on computing tool familiarity with a compiled language [CPUG, 09]. The latter two-thirds of the text, with its greater emphasis on physics, has typically been used for a two-quarter (20-week) course. What with many of the topics taken from research, these materials can easily be used for a full year's course, and for supplementary research projects.

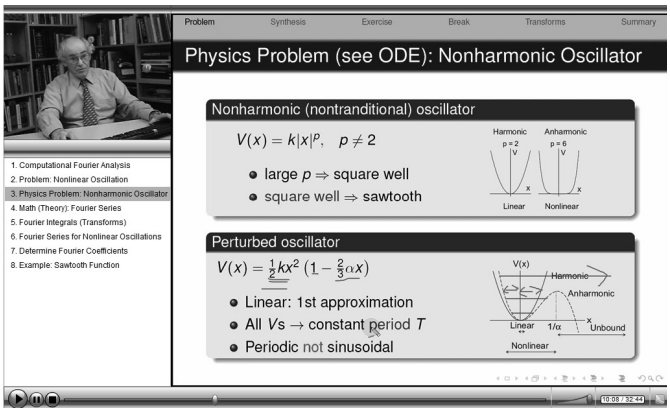

Figure 1.2 A screenshot from a lecture module showing a dynamic table of contents, a talking head, video controls, a slide with live scribbling, and some old man. (Originally in Flash, now as mpegs.)

1.3 Video Lecture Supplements

As as an extension of the concept of a "text", we provide some 60 video lecture modules that cover most every topic in the book. The modules were originally a mix of Flash, Java, HTML and mpeg, but with Flash no longer supported, we provide them as mp4 videos and pdf slides. They are available on our Website:

sites.science.oregonstate.edu/∼landaur/Books/CPbook/eBook/Lectures,

as well as on our YouTube channel under *Landau Computational Physics Course*:

www.youtube.com/playlist?list=PLnWQ_pnPVzmJnp794rQXIcwJIjwy7Nb2U.

The video lectures can be used to preview or review materials, as part of an online course, or in a blended course in which they replace some lectures, thereby freeing up time for lab work with the instructor.

1.4 This Book's Codes and Problems

Separate from the problems and exercises throughout the text, most every chapter starts off with a keynote "**Problem**" that lead into the various steps in computational problem solving (Figure 1.1). The additional problems and exercises distributed throughout the chapters are essential ingredients for learning, and are meant to be worked through. This entails studying the text, writing, debugging and running programs, visualizing the results, and expressing in words what has been performed, and what can be concluded. We asked our students to write up mini lab reports containing

Equations solved Numerical method Code listing

Visualization Discussion Critique

Although we recognize that programming is a valuable skill for scientists, we also know that it is incredibly exacting and time-consuming. In order to lighten the workload, *we provide programs for most of the problems in the text*, both at the end of each chapter and online at:

sites.science.oregonstate.edu/∼landaur/Books/CPbook/Codes/.

A complete list is given in the Appendix. We recommend that these codes be used as guides for the reader when writing their own programs, or, at the least, tested and extended to solve the problem at hand. We have been told that learning how to use someone else's code is a valuable workplace skill to develop; as with programs encountered in a workplace, they should be understood before use!

1.5 Our Language: The Python Ecosystem

The codes in this edition of *Computational Physics* employ the computer language *Python*. Previous editions have employed Java, Fortran and C, and used post-computation tools for visualization.[1)] Python's combination of language plus packages now make it the standard for the explorative and interactive computing that typifies present-day scientific research.

Although valuable for research, we have also found Python to be the best language yet for teaching and learning CP. It is free, robust (programs don't crash), portable (programs run without modifications on various devices), universal (available for most every computer system), has a clean syntax that permits rapid learning, has dynamic typing (changes data types automatically as needed), high-level, built-in data types (such as complex numbers), and built-in visualization. Furthermore, because Python is interpreted, students can learn the language by executing and analyzing individual statements within an interactive shell, or within a notebook environment, or by running an entire program in one fell swoop. Finally, it is easy to use the myriad of free Python packages supporting numerical algorithms, state-of the art visualizations, as well as specialized toolkits that rival those in Matlab and Mathematica/Maple. And did we mention, all of this is free?

Although we do not expect the readers to be programming experts, it is essential to be able to run and modify the sample codes in this book. For learning Python we recommend the online tutorials [PyTut, 23; Pguide, 23; Plearn, 23], the book [Langtangen, 16], and the many books in the "Python for Scientists and Engineers" genre. For general numerical methods, [Press *et al.*, 07] is the standard, and fun to read. The NITS Digital Library of Mathematical Functions [NIST, 22] is a convenient reference for mathematical functions and numerical methods.

Python has developed rapidly since its first implementation in December 1989 [History, 22]. The rapid developments of Python have led to a succession of new versions, and the inevitable incompatibilities. The codes presented in the book are

1) All of our codes, even the old ones, are available online.

in the present standard, Python 3. The major difference from Python 2 is the print statement:

```
>>> print 'Hello, World!'      # Python 2
>>> print('Hello, World!')     # Python 3
```

1.6 The Easy Way: Python Distributions

The Python language plus its family of packages comprise a veritable ecosystem for computing. A *package*, or library, or module, is a collection of related methods, or classes of methods, that are assembled and designed to work together. Inclusion of the appropriate packages extend the language to meet the specialized needs of various science and engineering disciplines [CiSE, 15]. The Python Package Index [PyPi, 23], a repository of free Python packages, currently contains 425,320 projects and 7,313,641 files. In this book we use:

Jupyter Notebooks: A web-based, interactive Python computing environment combining live code, type-set equations, narrative text, visualizations, and whatever. Some of our programs (`.ipynb` suffix) were developed in Jupyter, and our programs using Vpython work only within Jupyter. There is a previous edition of this text in notebook form at
sites.science.oregonstate.edu/∼landaur/Books/CPbook/eBook/.
The interactive Python shell, *IPython* can also be used within Jupyter.

Numpy (Numerical Python): A comprehensive library of mathematical functions, random number generators, linear algebra routines, Fourier transforms, and most everything else. Permits the use of fast, high-level multidimensional arrays (explained in Chapter 7). The successor to both *Numeric* and *NumArray*, NumPy is used by Visual and Matplotlib.

Matplotlib (Mathematics Plotting Library): A 2D and 3D graphics library that uses NumPy, that produces publication-quality figures in a variety of hard copy formats, and that permits interactive graphics. Similar to Matlab's plotting (except Matplotlib is free and doesn't need its license renewed yearly).

Pandas (Python Data Analysis Library): A collection of high-performance, user-friendly data structures and data analysis tools (used in Chapter 11).

SymPy (Symbolic Python): A system for symbolic mathematics using pure Python (no external libraries) that provide a simple computer algebra system including calculus, differential equations, etc.. Similar to Maple or Mathematica, with the *Sage* package being even more complete. Examples in §2.3.6.

Visual (Vpython): The Python language plus the no-longer-supported *Visual* graphics module (superseded by GlowScript). Particularly easy for creating educational 3D demonstrations and animations. Still useful as `Web Vpython` and within Jupyter notebooks.

Although most Python packages are free, there is true value for both users and vendors to distribute a collection of packages that have been engineered and tuned to work well together, and that can be installed in one fell swoop. (This is similar to what Red Hat and Debian distributions for Linux.) These distributions can be thought of as complete, Python ecosystems and are highly recommended. In particular, all you really need to do to get started with Python computing for this book is to load:

AnaConda: a free Python distribution including more than 8000 packages for science, mathematics, engineering, machine learning, and data analysis. Anaconda installs in its own directory and so runs independently from other Python installations on your computer. Go to
www.anaconda.com/products/distribution
to download Anaconda. Once you install *Anaconda*, the *Navigator* should open, and it will let you choose all that you will need.

Spyder IDE: The Scientific PYthon Development EnviRonment. An Integrated Development Environment (IDE) with advanced editing, interactive testing of code, debugging, and more.

Jupyter Notebook: The Web-based interactive computing notebook environment used for editing and running type-set-like documents, while also running Python code within the documents. As we have already said, a notebook (`.ipyn`) version of an earlier edition of this text is at
sites.science.oregonstate.edu/∼landaur/Books/CPbook/eBook/.

Powershell Prompt: A powerful terminal that run *conda* commands under the Windows shell environments `cmd.exe` (Command Prompt) and `powershell.exe`. Apple has a *Terminal* app where you will find a command prompt.

Conda: A package management and environment system included in Anaconda that finds, installs, and updates packages and their dependencies for you.

In Chapter 11 we describe how to load and run Google's *TensorFlow* package for machine learning, and in Chapter 12 we describe how to load and run the *Quantum Computing* packages, *Cirq, IBM Quantum*, and *Qiskit*.