BOOK REVIEWS

Daniel V. Schroeder, *Editor*
Department of Physics, Weber State University, Ogden, Utah 84408; dschroeder@weber.edu

Computational Physics (second edition). Nicholas J. Giordano and Hisao Nakanishi. 544 pp. Prentice Hall, Upper Saddle River, NJ, 2005. Price: $100.40 ISBN 0-13-146990-8.

An Introduction to Computer Simulation Methods: Applications to Physical Systems (third edition). Harvey Gould, Jan Tobochnik, and Wolfgang Christian. 796 pp. Addison Wesley, San Francisco, 2006. Price: $74.20 (paper) ISBN 0-8053-7758-1. (Eric Ayars, Reviewer.)

Recently the authors of two well-known texts in the field of computational physics have released new editions. *Computational Physics*, by Nicholas Giordano, is now in its second edition with an additional author, Hisao Nakanishi. *An Introduction to Computer Simulation Methods* is now in its third edition, with Wolfgang Christian as an addition to the team of Harvey Gould and Jan Tobochnik. It is the intention of this review to provide a comparison of these two new editions that will be useful to anyone considering adopting one as a course textbook. For simplicity, I refer to the texts as *Giordano* and *Gould*.

SIMILARITIES

As is the case for any texts that cover the same material, there are significant similarities between the two books. Both begin with the process of solving relatively simple ordinary differential equations, even beginning with the same example: using Euler's method for modeling free-fall. From there, they cover orbital dynamics, oscillations, waves, chaos, diffusion, percolation, Monte Carlo techniques, E&M, Schrödinger's equation, Ising models, and so on—more material, really, than anyone could hope to cover in a typical semester-long computational physics course. The authors of both texts are fully aware of the time limitation problem, and have organized the books in such a way as to allow the instructor to pick and choose topics as desired after covering the material from the first few chapters.

Both texts are quite readable, written in a conversational tone that draws the reader into the material. Figures are clear, understandable, and appropriately chosen. Both have well-chosen problem sets interspersed throughout the chapters.

One nice feature of both texts is that they are written so that the physics drives the development of the computational tools, rather than the tools selecting the physics problems to solve. Each new computational tool is introduced via a specific physics problem, so the student has a clear goal in mind throughout the process. With either book as a text, there is a reasonable expectation that after completing the course students will be able to approach new problems with an attitude of "Here's a new problem, what tools can I use on it?" rather than "Here's a tool, what can I do with it?"

DIFFERENCES

The most significant difference between the two texts is their choice of computing language. The decision of what language to use in a computational physics course is not an easy one, given the number of very good options available today. The choices made here have enormous ramifications in the character of the resulting books. Interestingly, the previous editions of both texts used True Basic, and this has been changed in both of the current editions.

Giordano answers the question of what language to use with the unusual approach of being language agnostic. There are a few bits of example code early in the text, written in Fortran and in C, which give the reader a cursory look at general program structure without really teaching anything about either language. There are also links to extensive code examples in Fortran and in True Basic on the textbook's web site. The bulk of the text, however, uses non-specific pseudocode to display the algorithms, rather than by giving specific code examples.

Gould takes the diametrically opposite approach of choosing a specific language—Java—and using that language exclusively. Not only that, but Gould has been copublished with a companion book, *Open Source Physics* by W. Christian, which provides and documents an extensive library of open-source Java routines for computational work. (These libraries are also freely available on the web, but students may find this additional reference book helpful.) This choice necessitates dedicating more of the book to language-specific material, such as the object-oriented programming aspect of Java.

Another difference between the texts, which may not be immediately obvious, is Gould's approach to the problem sets. While Giordano takes a traditional problem-solution approach, Gould treats a significant number of the problems as laboratory exercises. Each of these "projects" has multiple parts, designed to guide the student stepwise through some challenging material. Some of the projects encapsulate significant amounts of material, which would have occupied a separate section of a chapter if presented in full. Given that most real computational physics problems are more multipart project than straightforward exercise, this is a welcome feature.

There are differences in coverage as well. Gould tends to be broader in his coverage. For example, in the coverage of methods of solving ODE's, Giordano discusses the Euler method, the Euler-Cromer method, Runge-Kutta methods in second and fourth order, and the Verlet method. When discussing the same material, Gould includes all of these and adds Euler-Richardson, Beeman, velocity Verlet, predictor-corrector, adaptive-stepsize Runge-Kutta, and even symplectic methods. There are some curious exceptions to this generalization, however. Most notably, nothing whatsoever is said in Gould about systems of linear equations and/or matrix methods, while Giordano dedicates an appendix to this important topic. Linear least-squares fitting is discussed in

Gould almost as an aside in the chapter on random walks, whereas Giordano has significant discussion of the linear case, and higher-order fits as well.

There are two chapters in Gould that have no equivalent coverage in Giordano: one on relativity and one on three-dimensional visualization.

STRENGTHS

The greatest advantage of Gould over Giordano is its choice of language. Having a specific language for the textbook allows the authors to put in enormous quantities of language-specific material: user-interface routines, 3D graphics, libraries of ODE solvers, and so on. The chapter in Gould on three-dimensional visualization is a good example of the advantages here. The basic material in this chapter—coordinate transformations, rotations, etc.—is not specific to any computer language, but the implementation of 3D graphics is so entangled with the specific details of the language and operating system used that it is nearly useless to discuss the subject without assuming a specific graphics library. The choice of Java as language gives the authors opportunity to provide that library and thus opens up the possibility of discussing this material in a meaningful way. The presence of these software libraries also opens up the course to much more challenging problems than would otherwise be reasonable to expect of undergraduate students.

The greatest advantage of Giordano over Gould is its *non*-choice of language. By not specifying a language, the authors allow themselves to focus purely on the computational physics aspect of their subject. Gould is so inextricably linked to Java that if you wish to teach a computational physics course in some language other than Java, this book should probably not be used as the primary text.

WEAKNESSES

The omission of systems of linear equations and matrix methods from Gould is a minor weakness of that text, but not a major flaw. Their goal of letting the physics guide the numerical topics, rather than the other way around, makes it easy to omit this material; and the ease of its omission may say something about the material's actual importance at this level.

Also, although it is the stated intent in Gould that the text can stand alone, it is not obvious that students would be able to use this book alone to become proficient in object-oriented programming of computational physics problems. Instructors of students with no previous object-oriented programming experience should probably plan on supplementing the textbook in this area.

The language independence of Giordano is one of that text's strengths, but it is in some ways a weakness as well. If your students do not have previous programming experience you should expect to bring to the course considerable resources from outside the text to fill this need. There are times also when one just wants to see a code example for a particular topic, and there aren't any in this text. The website for the text does provide code for the examples given in the text, though, and the minor bother of not having these actually in the text is far outweighed by the flexibility afforded by the independence of the text from any particular language.

The bibliography at the end of each chapter in Giordano is probably more than most students will ever use, but is still somewhat slim compared to the pages (typically) of equivalent material in Gould. Whether this is a weakness on the part of Giordano or overexuberance on the part of Gould is an open question.

CLOSING THOUGHTS

You can't go wrong with either of these texts. If you already have a preference for some language other than Java, Giordano is an excellent choice. Its language independence provides enormous flexibility if you wish to teach the course in C/C++, Basic, Python, or even Fortran.

If you are willing to teach the course in Java, though, Gould would be the better choice. It is a superb package of tightly integrated materials, powerful libraries, and broad coverage that will serve you and your students well.

Eric Ayars is an Assistant Professor of Physics at California State University, Chico, where he devotes much of his time to building up the experimental facilities for the physics students, and to teaching computational physics.

A First Course in Computational Physics and Object-Oriented Programming with C++. David Yevick. 403 pp. (plus CD-ROM). Cambridge University Press, New York, 2005. Price: $70.00. ISBN 0-521-82778-7.

A First Course in Scientific Computing: Symbolic, Graphic, and Numeric Modeling. Rubin H. Landau. 481 pp. (plus CD-ROM). Princeton University Press, Princeton, NJ, 2005. Price: $49.50 ISBN 0-691-12183-4.

Computation and Problem Solving in Undergraduate Physics. David M. Cook. Self-published; see http://www.lawrence.edu/dept/physics/ccli. (R. Torsten Clay, Reviewer).

Computational physics is often cited as a third type of physics research besides experimentation and theory. Computation shares features of both: first the construction of a theoretical model for a physical system, followed by analysis of the "experimental" data from numerical simulations. A successful computational physicist needs to draw from many areas of expertise, including theoretical physics, computer science, programming, parallel computing, and visualization and graphics techniques.

Although it is relatively easy to agree on what kind of research falls under the category of "computational physics," there are widely varying ideas on how best to teach science students to successfully use computers. While some physics departments may offer a degree specialization in computation and several courses in the area, in the majority of physics departments, computational physics is taught as a single-semester elective course. In a single semester one unfortunately cannot cover all of the elements mentioned above. Different departments may also see the course as fulfilling different requirements. For example, the course may be geared to provide basic computer literacy, such as familiarity with operating systems (often UNIX in the physics community), scientific writing packages such as LATEX, basic data analysis, and graphing, etc. In departments where computation is a major research focus, the course may include high-performance computing, parallel computing, programming, and so on. In most cases Computational Physics is a recent addition to university curricula, and hence the course content may be strongly affected by local committee requirements, which often forbid new courses from duplicating material found in other departments.

There are two main methodologies to organize the class, based on what computational scheme is used. The two schemes are programming using a low-level compiled language such as FORTRAN, C, or C++, or using a high-level symbolic environment such as MATHEMATICA or MAPLE. While the choice of programming scheme may seem a minor detail, it makes a large difference in how the class is organized. The choice usually boils down to the preference and experience of the instructor, as there are advantages and disadvantages to either approach. For a programming course, students typically write their own programs, learn a series of numerical methods such as integration, linear algebra, and Monte Carlo, and apply them to physics problems. This is an advantage for students planning to do research in computation, where being able to develop new codes is important. Being proficient in a programming language is also useful in the job market beyond physics. Further, once a student has learned one low-level language, it is usually easy to learn a different one later on. A major disadvantage is that a large amount of class time must be used to teach programming syntax that has little to do with physics, and students without prior programming experience may be at a large disadvantage. Graphical visualization beyond simple plots may be difficult.

Proficiency in a symbolic mathematics environment is very helpful for students in a wide variety of physics core courses. When using a high-level mathematical environment, students usually have less exposure to the underlying numerical methods used to solve the problems, since they may be hidden in "black box" computational routines. Students may not learn how to approach a numerical problem for which the appropriate black box is not available. High-level programming environments usually do not scale well to large problems, and hence are not typically used for large-scale computations. However, it is easy to cover a large number of interesting physics problems, and the graphical visualization tools are usually excellent and easy to use. Finally, while high-level environments have the advantage of being self-contained, they are often expensive and nonstandardized.

Some instructors may choose to cover both the symbolic and programming approaches. In a programming-based class, it is easy to throw in a week or two on an introduction to a symbolic mathematics package. Whether this is successful may depend on the preparation of the students, and whether programming experience is considered a prerequisite for the class. If students have never programmed before, learning a programming language proficiently in less than one semester is extremely difficult.

There are at least as many approaches to writing textbooks for computational physics. Here we review three recent books aimed at computational physics classes. These three books illustrate the basic choice outlined above: whether to teach via programming or in a high-level symbolic environment.

Yevick's book follows the straight programming-based format. The book is accompanied by a CD-ROM including example programs, a C++ programming environment, and graphics tools. The tools used can be obtained for free for Windows or Linux platforms. The best feature of the book is its good and concise description of the C++ language. More than half of the book could be considered just a programming text, followed by a short presentation of numerical derivatives, integration, root finding, differential equations, and linear algebra. This is followed by a large section detailing advanced object-oriented techniques, and finally short sections on Monte Carlo and partial differential equations.

The book would be a good fit for instructors who prefer to teach programming in detail. It is quite specific to C++, and could not easily be used with a different language. The organization of the book is to present the syntax and features of C++, with a variety of physics-based problems at the end of each section as exercises. One possible criticism of the book is that it focuses too much on programming and does not give enough of a survey of the type of physics problems that one can tackle. The final sections on Monte Carlo and partial differential equations seem to have been added as afterthoughts; probably the instructor would want to add more material if covering those topics. For the problems that are included, undergraduate-level mechanics and electromagnetism are sufficient background; there are no problems involving quantum mechanics. The advanced object-oriented sections might be considered by some to not fit in an introductory computational physics course, so one option might be to leave some of those sections out and instead fill in more physics examples.

This would be the ideal text for the student who is considering a career in scientific programming, and wants to learn C++ very proficiently. Besides being used as a textbook for a class, this book would be an excellent book for a practicing computational physicist who wants to learn object-oriented C++ programming.

Landau's book covers both the symbolic (MAPLE) and programming (JAVA) approaches to computational physics, roughly with equal weight. As with Yevick's book, an accompanying CD-ROM is provided containing the source code for program examples. Instructors must provide the compilers or environments needed to run the examples. The full title of this book is *A First Course in Scientific Computing: Symbolic, Graphic, and Numeric Modeling Using Maple, Java, Mathematica, and Fortran 90*. The title deserves some explanation, as the book appears to cover four different programming languages! In reality, this is deceptive advertising as the book is actually written to cover only MAPLE and JAVA. The CD-ROM provides MATHEMATICA and FORTRAN 90 translations of the programs (but not the book text). It would be difficult to use the text with either MATHEMATICA or FORTRAN 90. For example, in the MAPLE sections, the text is liberally filled with sample MAPLE program lines, and each one would have to be converted by the reader to MATHEMATICA syntax. Furthermore, certain elements of JAVA (graphics, object-oriented features, web applets) have no direct translation to FORTRAN 90.

The style of Landau's text (which he calls "computation by doing") is in some sense the reverse of Yevick's, who presents programming concepts, followed by example physics problems. The format will be familiar to users of the book *Computational Physics: Problem Solving with Computers* by Landau and Manuel J. Páez, which used FORTRAN and C. Each section starts with a physics problem that requires a numerical solution, and then presents the computational concepts needed to solve the problem. As in Yevick's book, undergraduate mechanics and electromagnetism are sufficient physics background, and Landau's book also does not require quantum mechanics. Students learn different computational elements as they work through the problems presented. This presentation may have an advantage in keeping students interested in the material without getting bogged down in learning programming syntax. However, one disadvantage is that similar material ends up scattered in different parts of the text, and can be consequently hard to track down. For example, root finding using MAPLE is first covered in Chapter 5; root finding using bisection (via MAPLE) is covered in Chapter 8.

The choice of JAVA deserves some discussion. A relatively new language, JAVA is primarily popular due to its inclusion of graphics capabilities and its use in Web-based applets. However, JAVA is not widely used by practicing computational physicists since it is slow and not widely supported by numerical libraries. For this reason some instructors might prefer C or FORTRAN. In the end, students who learn to program proficiently in one computer language rarely have trouble learning another.

The text covers the usual subjects of integration, differentiation, and matrices. Notably absent are random numbers and Monte Carlo methods. Finally, a section on the mathematical typesetting language LATEX is included. Overall, the book would be good for instructors teaching an introductory undergraduate class, and who want to include a little programming and a little symbolic mathematics.

Computation and Problem Solving in Undergraduate Physics was written and self-published by David Cook of Lawrence University. This text (or maybe more accurately, collection of *virtual* texts) addresses a common problem among computational physics texts: there is currently no one standardized programming language, environment, or hardware setup. Instructors may use a certain book because they like the presentation and problems, despite the fact that it uses a programming language that is not their first choice. Or they may be forced to use a book that they dislike, because of their university's choice of what software to site license. Therefore, Cook has constructed a text in which instructors may choose which languages they want. The appropriate sections are then assembled, printed, and bound together. In order to keep the various versions coherent, chapter numbers may not be consecutive, but an appropriate index for each version is included.

The programming languages/environments that can be included in the text are IDL, MATLAB, MACSYMA, MAPLE, MATHEMATICA, C, and FORTRAN. Cook further breaks down the "high-level environment" classification to include separate chapters on array-processing systems (IDL, MATLAB) and symbolic mathematics systems (MACSYMA, MAPLE, Mathematica). Subroutine libraries from *Numerical Recipes* are also discussed. Additional appendices are available covering LATEX, TGIF (a program for producing drawings), and LSODE (a Fortran library for solving differential equations). The sections describing the high-level languages (IDL, MATLAB, etc.) are all similar in organization, presenting systematically the capabilities of the language (including graphics). At the end of each section is a list of exercises. Sections on low-level programming are also included (FORTRAN, C). Although the introduction to programming is taught using a pseudocode rather than a specific programming language, the presentation is somewhat biased towards FORTRAN. For example, in the pseudocode, FORTRAN array notation and DIMENSION statements are used. The introduction is followed by separate sections explaining either C or FORTRAN syntax. One key feature of C, pointers, is not discussed. While it is possible to avoid pointers in simple C code, it is difficult to write code interfacing with many popular matrix libraries without some knowledge of pointers and C memory allocation. The introduction to programming is followed by sections on the *Numerical Recipes* libraries, differential equations, integration, and root finding. Random number generation and Monte Carlo are not covered.

Cook's text includes more than enough material for a course, both reading and classwork. The book has a good balance between being a reference on commands for the symbolic languages, including problems, and discussing physics applications. Of the three books reviewed, Cook's covers the largest amount of physics, and is the only one to include quantum mechanics (Schrödinger's equation in one dimension). The physics background assumed for some of the problems is more advanced than that of most undergraduates. However, the number of problems is large enough so that an instructor could pick appropriate problems for either a graduate or undergraduate class. The available customization options are unique, and could be adopted to a variety of courses, with possibly the exception of a course based solely on C programming.

R. Torsten Clay is an Assistant Professor in the department of Physics and Astronomy at Mississippi State University, and a member of the Center for Computational Sciences at the MSU High Performance Computing Collaboratory. His research is in the theory of strong correlated electron materials and computational methods for many-body systems. He also teaches the Computational Physics course at Mississippi State University.

INDEX TO ADVERTISERS

Physics Academic Software	Cover 2
Houghton Mifflin	561
Princeton University Press	563
WebAssign	564

MAKE YOUR ONLINE MANUSCRIPTS COME ALIVE

A picture is worth a thousand words. Film or animation can be worth much more. If you submit a manuscript which includes an experiment or computer simulation, why not make a film clip of the experiment or an animation of the simulation, and place it on EPAPS (Electronic Physics Auxiliary Publication Service). Your online manuscript will have a direct link to your EPAPS webpage.

See http://www.kzoo.edu/ajp/EPAPS.html for more information.