



TWO APPROACHES TO TEACHING COMPUTATIONAL PHYSICS

By Charles H. Patterson

WE CAN EASILY ARGUE THAT ALL GRADUATES OF SCIENCE AND ENGINEERING DEGREE PROGRAMS SHOULD HAVE THE OPPORTUNITY TO DEVELOP GOOD COMPUTING

skills by the time they complete their studies. However, the depth and range of skills needed varies considerably—even in a single discipline such as physics. Moreover, the interests, backgrounds, and abilities of students taking physics courses vary widely, whereas the number of teachers with scientific computing skills is often limited. Providing appropriate courses for such diverse student groups is thus a challenge.

Computational physics provides exciting new teaching opportunities that can complement traditional methods of teaching in the lecture theater or experimental laboratory. Students usually realize that computing skills enhance their job prospects after graduation, and the hardware and software available for teaching such courses is relatively inexpensive and easy to maintain.

I have taught computational physics courses at Trinity College Dublin (TCD) for approximately 10 years, initially through a theoretical physics degree class in which final-year students must solve a problem in nine weeks by using computational methods. More recently, I have directed a degree course in computational physics that started in 1997. The computational parts of the syllabus are mainly taught together with a computational chemistry degree program. Two important factors led to the course's introduction: a skills initiative by the Irish Higher Education Authority and a need to increase the level of training of potential postgraduate students for work in computational science. (The IHEA aims to increase the number of graduates from Irish universities with skills in information technology and provides extra funding for two additional staff members and equipment.) This article describes two approaches to teaching computational physics to these and other student groups.

Computing and mathematics in physics

Two important uses of computing in an undergraduate physics syllabus are to teach students to combine their knowledge of mathematics with an understanding of its meaning in physical problems and to teach the emerging discipline known as computational physics. Numerical methods, programming, and the use of applications are obvious topics that should be part of any syllabus for a computational science or engineering degree course. However, not all student groups in any discipline need, or have the time, to take the full computational syllabus. Moreover, there is no single syllabus in computing suitable for all groups of physics students.

Physics students usually learn mathematics and physics topics requiring mathematics in different years. Generally, they learn mathematics first, but when they're taught courses requiring an understanding of that mathematics in physics in later classes, they frequently have difficulty connecting the two.

An appropriate course that instructors can teach in the computer laboratory could help students make the necessary connections. The type of problem that a laboratory class can tackle differs significantly from that which students can successfully tackle in the lecture theater. In the lecture theater, for example, it is sufficient and desirable for the teacher to solve a relatively simple problem analytically and explain that a fuller solution is achievable if the students apply numerical methods to the problem's solution. However, in many cases, it is more satisfactory to be able to solve a more complex problem numerically and to use the solution to this type of problem to reinforce the students' understanding of the problem's physics and mathematics as well as their understanding of how mathematics is applied in physics.

Programming

An early lesson I learned from teaching courses is that most students need a long time to develop basic proficiency in programming, including getting a program to compile successfully and testing it to make sure it gives the expected output. For example, learning how to tell what is causing a particular problem when the computer doesn't do what you

want it to do is an obstacle most people face when learning basic computing.

For students to gain confidence in operating a computer it is essential to begin teaching practical computing in the first year of the course; waiting until students are in their senior years and heading for more advanced topics is likely to cause them frustration if they lack basic computing skills and confidence. Fortunately, more students have access to computers at school and home before entering universities or colleges, so their basic levels of confidence and competence in the first year is growing, which should let teachers begin working with them at a higher level earlier. Regardless of where they learn it, however, a considerable amount of students' time must be invested in basic computing and programming if they are to write their own programs to solve physics problems.

We use Mathematica in computer laboratory classes for students who aren't in the computational physics degree course and laboratory classes based on short C programs for those who are. Students in the latter group receive whole programs or code fragments in laboratory classes. The physical problems addressed in either case are the same, however. For example, second-year students take three or four computational laboratory classes on topics that include root and minima finding, the pendulum, Fourier series, and coupled oscillators. Before symbolic mathematical packages such as Mathematica, Maple, and Matlab became available, laboratory classes on such topics would have required relatively lengthy programs, and graphical display of results would have been limited. However, with this type of package, solving interesting physical problems with codes only four or five lines long and getting excellent graphical output is now possible. (Several texts devote pages to the use of Mathematica in undergraduate physics courses—also, see www.mathsource.com.^{1,2}) Computational physics students complete their skeleton C programs and then use them to solve the problem set, whereas other student groups get Mathematica notebooks that only need input parameters to be modified to solve the problem set.

The computational physics degree

One of our aims in developing the syllabus for the computational physics degree was to retain all the parts of an undergraduate physics curriculum considered to be “core” physics. Thus, in the first two years of their course, students take the same lecture courses in chemistry, physics, and mathematics as natural sciences students studying these subjects do. At the end of the second year, they choose to study

either computational physics or computational chemistry for their final two years.

Computational physics students take the usual physics courses in electromagnetism, quantum theory, and so on, in addition to more specialized computational physics courses (see www.tcd.ie/Physics/Courses). This degree differs from a combined physics–computer science degree because physicists and chemists rather than computer scientists teach the computing component of these courses. It also emphasizes computer laboratory experience over lecture theater theory.

In the first two years, students spend up to one half of their physics laboratory classes (three hours per week) in the computer laboratory and attend tutorial classes devoted to scientific computing topics. In the third and fourth years, computational physics students take approximately 120 lectures in computing (languages and programming), numerical methods, and applications in computational science. They spend half of their laboratory class time (one day per week) in the computer laboratory in the third year and a full term (nine weeks) doing project work in the final year. More than half the lectures are taught to both computational chemists and physicists; staff members from both departments share the teaching responsibilities. This type of load sharing makes additional degree courses possible in relatively small departments.

Elementary numerical methods are introduced in third-year lectures. More advanced topics, including fast Fourier transforms, Monte Carlo methods for solving multidimensional integrals, and linear algebra libraries, are taught in the final year. C is taught as the main programming language with a short course in Fortran in the final year. The transition from Windows to Linux starts at the beginning of the third year, when laboratory classes are taught in the latter. (See the sidebar for details about hardware and software in the computer laboratory.) Applications courses in the third year are in partial differential equations and molecular dynamics. Final-year applications courses include partial differential equations, statistical physics, and molecular dynamics.

C-based computer laboratory classes are intended to let students learn to program and use numerical methods. Just like learning to drive a car, however, the student needs time to develop skills, experience, and confidence, so having a coherent set of computer laboratory classes over the course's four years is important. To give a clearer overview, we can summarize the aims of the four years as first year, basic computing skills; second year, illustration of physical concepts; third year, introduction to numerical methods and computer simulation; and fourth year, developing higher-level prob-

Hardware and Software in the Computer Laboratory

First- and second-year C laboratory classes at Trinity College Dublin use the Borland C compiler running on Windows 2000. This compiler is menu-driven, and a single click is all you need to initiate compilation. Code lines with errors are highlighted, and using the debugger is easy. No flags need be entered (as is the case for command-line compilers), which makes this a suitable environment for teaching C programming for the first time.

Instructors teach third- and fourth-year classes using Linux and the Gnu C compiler. All students at TCD have file storage space maintained by the Information System Service, which they can access using the Computational Physics Laboratory PCs running Windows 2000. When running under Linux, the student file systems are available on a local server.

lem-solving skills via project work.

In the first-year laboratory, students begin with laboratory classes that use Excel spreadsheets to compute projectile trajectories before beginning C programming. Later, they learn elementary C programming and receive code fragments to solve problems such as root finding. In the second-year laboratory, students tackle problems such as finding minima, dynamics of the pendulum, Fourier series, and dynamics of coupled oscillators. These are important elementary physics problems, and the computer laboratory classes provide a good means of enhancing students' understanding of the concepts they learn in lecture courses. In the Fourier series laboratory class, for example, students must write a program to calculate the Fourier series using Simpson's rule to perform necessary integrations:

$$f(t) = a_0 + \sum_{n=1}^{\infty} (a_n \cos(n\omega t) + b_n \sin(n\omega t)) \quad (1)$$

$$a_0 = \frac{1}{T} \int_0^T dt f(t) \quad (2)$$

$$a_k = \frac{2}{T} \int_0^T dt f(t) \cos(k\omega t) \quad k = 1, 2, \dots \quad (3)$$

$$b_k = \frac{2}{T} \int_0^T dt f(t) \sin(k\omega t) \quad k = 1, 2, \dots \quad (4)$$

$$\int_a^b dx f(x) \approx \frac{b}{6} \left(f(a) + f(b) + 2 \sum_{k=1}^{n-1} f(a+kb) + 4 \sum_{k=1}^n f\left(a + \left(k - \frac{1}{2}\right)b\right) \right) \quad (5)$$

This laboratory introduces students to numerical integration; specifically, it requires using nested loops (over the interval of the periodic functions and over the Fourier coefficient indices), functions, and writing data to a file. Integrating a simple function such as x^2 and calculating the Fourier series for combinations of sines and cosines tests the students' programs. Students calculate the Fourier series for functions such as square-wave and sawtooth functions and then back-transform them. The laboratory takes two three-hour laboratory sessions to complete (with extra time required for the laboratory write up).

Third-year laboratory classes address topics such as curve fitting, numerical integration, linear response of a pair of coupled linear oscillators, and molecular dynamics. Many people have published undergraduate computational physics texts recently. Two that we have adopted as general course texts are *Computational Physics: Problem Solving Using Computers* by Rubin Landau and his colleagues³ and *A First Course in Computational Physics* by Paul Devries.⁴ Instructors can glean ideas for laboratory class topics from these texts.

Final-year project work is an important aspect of all physics degree programs at TCD. Students spend the first nine-week term of their final year doing project work, and computational physics students spend approximately 30 hours per week on their computational projects. The main challenge the students find in writing their programs is making them work. In earlier years, the programming tasks they receive are structured, but the wide range of project topics offered is open ended. Examples of project topics include molecular dynamics of classical point charges, agent modeling, simulations of biological aging, and lattice gas models of foam evolution.

Mathematica laboratory classes

The Mathematica laboratory classes taught at TCD are intended to enable students (outside the computational physics program) to learn the same physical principles as

those illustrated in the C-based classes described earlier. The difference is that they require a minimal investment of time in learning a computer language. Students receive a Mathematica notebook containing an introduction to the problem, a set of exercises, and short sections of code embedded in the text (in separate cells) that they can run simply by pressing Shift-Enter. The students need little knowledge of Mathematica syntax to do the laboratory class, but they do receive a brief introduction to it and are shown how to use the online help. The online help and Mathematica book,⁵ which are both part of the package, make discovering what a particular command does easy.

When presented with a Mathematica notebook with the code already entered, students can focus more on the physics involved in the problem than when, for example, computational physics students are asked to program and solve physics problems simultaneously. However, it is the notebook writer's responsibility to pose suitable problems to develop and test students' understanding of physics problems when the code is already given. To illustrate this type of laboratory class, we use a familiar problem: the pendulum (the dynamics of which Gregory Baker and Jerry Gollub describe in their excellent undergraduate text⁶).

In this class, two exercises involve comparing the nonlinear and simple (linearized) pendulums' behavior and representing the dynamics of the damped, driven pendulum in a phase-space diagram. We can state the equation of motion for the damped, driven pendulum as three coupled, first-order ordinary differential equations (see Equation 6). We choose units such that the pendulum mass, length, and acceleration due to gravity are all unity; $\phi (= \Omega t)$ is the phase angle of the driving force, which has amplitude A :

$$\begin{aligned} \dot{\omega} &= -\sin\theta - \Gamma\omega + A \cos\Omega t \\ \dot{\theta} &= \omega \\ \dot{\phi} &= \Omega. \end{aligned} \quad (6)$$

One of the first exercises deals with the difference in behavior of the simple and the nonlinear pendulums. Students must explain why the maximum angle reached is the same for the simple and nonlinear pendulums, and why the period of the simple pendulum is independent of swing amplitude whereas that for the nonlinear pendulum is not. The answers are not always obvious to second-year students, but they can verify this fact by experimenting with various initial conditions. Figure 1 shows the output from the code

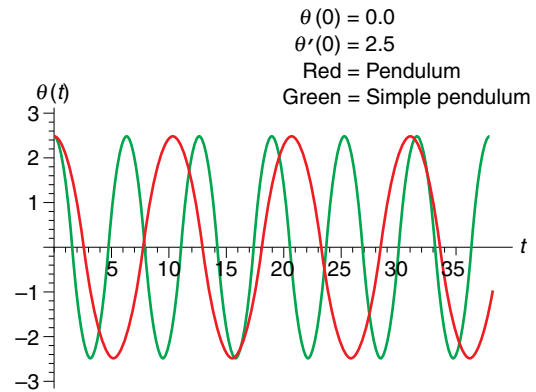


Figure 1. Comparison of $\theta(t)$ for the simple and nonlinear pendulums for a large amplitude swing.

```
solution = NDSolve[{θ'[t] == -Sin[θ[t]],
θ'[0] == 0.0, θ[0] == 2.5}, θ[t], {t,0,12π}]
solution1 = NDSolve[{θ1'[t] == -θ1[t], θ1'[0]
== 0.0, θ1[0] == 2.5}, θ1[t], {t,0,12π}]
Plot[{Evaluate[θ[t]/. solution], Evaluate[θ1[t]/.
solution1]}, {t, 0, 12π}, PlotRange → {-π, π},
PlotLabel → "\t\t θ'[0] = 0.0 θ[0] = 2.5
\n\t\t solid = pendulum \n\t\t dotted = simple
pendulum"]
```

which solves the equation of motion (Equation 6) for the simple and nonlinear pendulums and plots the pendulum angle versus time.

The example just given is straightforward and gets things started, but Mathematica can illustrate more advanced concepts including phase space and chaos in the pendulum. The following code solves the ordinary differential equations for the damped, driven, nonlinear pendulum, plots the angle and angular velocity as functions of time, and makes a phase-space plot (angular velocity versus angle) with parameter values $\Gamma = 0.5$, $\Omega = 0.6667$, and $A = 1.075$ (see Figure 2):

```
Solution = NDSolve[{ω'[t] == -Sin[θ[t]] - 0.5
ω[t] + 1.075 Sin[0.6667 t], θ'[t] == ω[t], θ[0]
== 0, ω[0] == 0}, {θ, ω}, {t, 0, 20π}, MaxSteps→
5000]
Plot[Evaluate [ω[t] /. Solution], {t, 0, 20π},
PlotRange → All, AxesLabel → {t, ω[t]},
PlotLabel → "θ'[0] = 0.0 θ[0] = 0.0 A = 1.075"
Plot[Evaluate [θ[t] /. Solution], {t, 0, 20π},
```

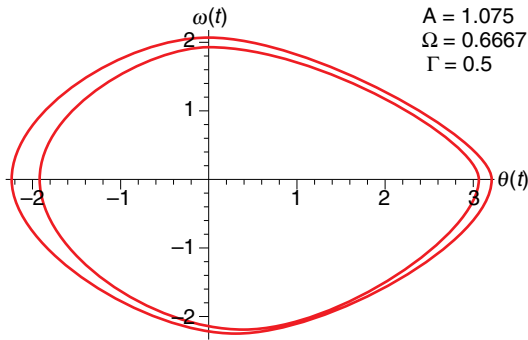



Figure 2. Phase-space diagram for period-doubled nonlinear pendulum.

```
PlotRange → All, AxesLabel → {t,  $\theta[t]$ },
PlotLabel → " $\theta'[0] = 0.0 \theta[0] = 0.0 A = 1.075$ "
ParametricPlot[Evaluate[{ $\theta[t]$ ,  $\omega[t]$ } /.
Solution], {t, 0,  $20\pi$ }, PlotRange → All,
AxesLabel → { $\theta[t]$ ,  $\omega[t]$ }, PlotLabel → " $\theta'[0] =
0.0 \theta[0] = 0.0 A = 1.075 \Gamma = 0.5$ "
```

This particular choice of parameters leads to a period-doubled pendulum. The period-doubling route to chaos in the pendulum is well known.⁶ We can omit the transient at the beginning of the dynamics from the plot by only plotting t values greater than 10π .

The TCD computational physics degree has run for five years now, so we're starting to see patterns emerge. High school science students are largely unaware of computational science, so we've had to advertise the course as widely as possible through open days, school careers guidance counselor newsletters, the *College Science Prospectus*, and so on. We're starting to find that several students transfer into the course from science or engineering courses once they have entered the College.

As I've mentioned, covering all aspects of many topics such as numerical analysis within a degree course such as this is impossible, but the course does give students the time and experience necessary to learn computational physics. Moreover, it will generate a pool of students with computational

skills for industry or academic research, and it offers students an excellent opportunity to acquire an additional set of skills during their undergraduate years. Employers frequently tell us that they want employees with a wide range of skills—physics plus communication skills, numeracy, literacy, and so on—and good computing skills and a knowledge of mathematical modeling are obviously a bonus. 

Acknowledgments

My colleagues Stefan Hutzler, Donail MacDonail, Graeme Watson, Sara McMurry, and Denis Weaire collaborated in establishing, developing, and teaching computational physics at Trinity.

References

1. R.L. Zimmerman and F.I. Olness, *Mathematica for Physics*, Addison-Wesley, Boston, 1995.
2. W. Kinzel et al., *Physics by Computer: Programming Physical Problems Using Mathematica and C*, Springer-Verlag, Berlin, 1997.
3. R.H. Landau et al., *Computational Physics: Problem Solving Using Computers*, John Wiley & Sons, New York, 1997.
4. P. Devries, *A First Course in Computational Physics*, John Wiley & Sons, New York, 1993.
5. S. Wolfram, *Mathematica*, 4th ed., Cambridge Univ. Press, Cambridge, UK, 1999.
6. G.L. Baker and J.P. Gollub, *Chaotic Dynamics: An Introduction*, 2nd ed., Cambridge Univ. Press, Cambridge, UK, 1996.

Charles H. Patterson is a lecturer in the Department of Physics at Trinity College Dublin, Ireland. His research interests include electronic structure theory, especially dielectric phenomena and strongly correlated electron systems. He received his BSc in chemistry from the University of Bristol and his PhD in chemistry from the University of Cambridge before switching to condensed matter physics as a postdoctoral fellow at the University of Pennsylvania. Contact him at the Dept. of Physics, Univ. of Dublin, Trinity College, Dublin 2, Ireland; charles.patterson@tcd.ie.