AC 2010-401: A LEARNING-BY-DOING APPROACH TO TEACHING COMPUTATIONAL PHYSICS

Radian Belu, Drexel University

Alexandru Belu, Case Western Research University

A Learning-by-Doing Approach to Teaching Computational Physics

Abstract

Scientific research is becoming unthinkable without computing. The ubiquity of computerized instrumentation and detailed simulations generates scientific data in volumes that no longer can be understood without computation. Computational physics is a rapidly growing subfield of physics and computational science in large part because computers can solve previously intractable problems or simulate natural processes that do not have analytic solutions. One can easily argue that all graduates of science or engineering 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 instructors with scientific computing skills has been rather limited. Providing appropriate courses and instruction in computational physics for such diverse student population is a challenge. On the other hand, computational physics provides exciting teaching opportunities that can complement traditional methods of teaching in the lecture or the laboratory. We use a laboratory project-based approach, where the students are learning by doing. The course is divided into two sections, lecture and laboratory session. During the laboratory session, the students work at mid-term and final projects, while the lecture the programming, numerical and computational techniques and methods are discussed. The usefulness of this approach is evaluated by surveys conducted every semester, and feedback from other educators is highly appreciated.

I. Introduction

Computational physics is an independent way of doing physics, and an essential tool of the physics research. Numerical computations are essential to further understanding of physics problems, and computers and computing play a central role in much of modern scientific research. Almost all analytical theories require the help of a computer to complete the calculations. On the experimental side, computers are essential for the control of experiments and the collection and analysis of data. However, computational physics also includes a fundamentally different way of doing physics that goes beyond using the computer as a specific tool. We have in mind the part of computational physics, called *computer simulations*[1, 8-11], in contrast to many of the tasks listed above which we classify as *numerical analysis*. Using a computer to model physical systems is at its best more art than science. The successful computation al physicist exploits the numerical power of the computer through a mix of numerical analysis, analytical models, and programming to solve otherwise intractable problems. It is a skill that can be acquired and refined - knowing how to set up the simulation, what numerical methods to employ, how to implement them efficiently, when to trust the accuracy of the results. In the last two decades, however, computational physics has largely been neglected in the standard university physics curriculum[1-5]. In part, this is because it requires balanced integration of three commonly disjoint disciplines: physics, numerical analysis, and computer programming (Figure 1). The lack of computing hardware suitable for teaching situations

involved in computational physics courses, together with the lack of programming language licenses available for students have also been other factors. Students usually acquire what computing skills they do have by working on specific thesis or senior project problems and as a result, their exposure is often far from complete.

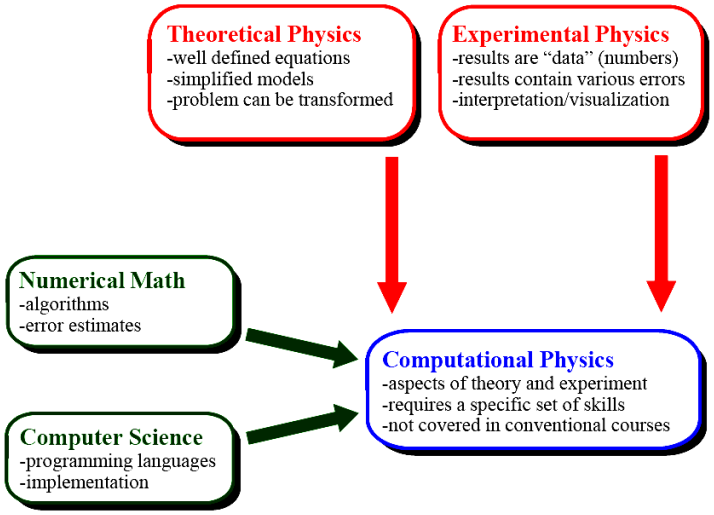

Figure 1: Place of computational physics in the physics and science fields

Physics, like other structured disciplines, has a very full curriculum because students need to learn challenging material dating back at least two centuries ago. Much of the older ideas, methods or techniques are still important and cannot simply be replaced by the most recent advances. However, computers and computing are indispensable tools for theorists and experimentalists alike for calculations, data analysis, acquisition and reduction, visualization, modeling, simulations and so on. Computers are now much more than tools for analysis. They provide us with innumerable intellectual challenges, from algorithm development to the exploration novel concepts such as self-organized criticality. Almost all physics theories require the help of a computer to complete the calculation. Despite of the importance of computational physics, about half of the physics departments in the United States offer them in only one course on numerical methods and computational physics[1, 6,7]. However, the education of undergraduate physics students would be greatly enhanced by learning computational and numerical methods[4-9].

Teaching physics is not an easy task, and teach computational physics makes no exception. Our approach in teaching computational physics consists in splitting the standard credit hours allotted to this course in two sections: one-hour lecture, where the computational and numerical methods are introduced to the students; and the rest for practical section reserved for exercises and projects. The students are required to write self-contained programs in a high-level scientific language--i.e., either FORTRAN or C++ and/or by using a computer algebra system, such as Maple or Maxima. Students are involved in the creation of code programming and algorithms in order to increase their level of understanding as well as critical and creative thinking. Finally, we introduce a

scientific solving problem paradigm based on the above typology and we present software applications in various thematic units. Our goal is to provide students with direct experience in modeling non-trivial physical systems and to impart to them the minimal set of techniques for dealing with the most common problems encountered in such work. The computer was to be viewed neither as a "black box" nor as an end in itself but rather as a tool for gaining a deeper understanding of physics.

II. Brief Course Description

The objectives of this course are: a) to use computers as an aid to understanding real physical systems; and b) learn efficient methods for the analysis of these systems. The goal of this course is to provide students with the ability of solving difficult problems by using computational methods and to learn and use programming languages. The course is intended to provide students with the computational tools, techniques and methods used in contemporary physics. During the laboratory sessions, a significant fraction of the students' time, the students are required to write, compile, and run programs, together with the analysis of the results, and present their results as scientific reports. In this teaching approach, the students ***learn by doing***[8-11]. It is a problem-solving course, that is, the measure of a student's progress is demonstrated by the ability to solve numerical problems in physics. Upon completion of this course, the student will possess the basic computing knowledge that may be required for graduate school or in industry.

Course structure: Different people learn in different ways. In this course, we try to build a learning environment with a diversified set of options that can be tailored to the each student individual learning style. There are class meetings, homework assignments, projects and practice. There are also many ways in which the students get assistance with the material in this course: the office hours, appointments, e-mail, phone and via online through the course management system.

Curse description: This course gives a modern introduction to the basic methods in computational physics and an overview of the recent progress in scientific computing. Many examples from recent research in physics and related areas are given with the program listing. Basic computational tools and routines, including the ones for differential equations, spectral analysis, and matrix operations, are dealt with through relevant examples, and more advanced topics are treated. The broad categories of computational physics studied are simulation, visualization and modeling, numerical methods, algorithms and data analysis. Simulation and modeling are taught by stressing numerical techniques and programming language(s) techniques employed. Besides learning how to solve numerical problems with a computer, the student also will gain experience writing manuscripts in a scientific journal style.

Projects: There are midterm computational projects and one final project. The projects aim to solve specific physics and/or electrical and physics engineering problems. The students are required to submit a report for each project. All reports should have the following sections: title, problem description, equations and computational model, testing, example of the input parameters, results (diagrams, figure, tables, and analysis), program

code, and conclusion. The report and the computer programs are required to be submitted electronically by the due date and time. This enables the assessor to run and check the program(s) if necessary. The project policy: "***no submitted program = no credit for the assignment***". Each assignment was graded for completeness and correctness

Exams: Since this is a project-based course, there are no traditional exams but midterm projects, and final project. The final exam consists of an oral presentation of the final project and an accompanying written report (in the format of a research article). The final examination is mandatory and will be given only at the scheduled time.

***Course Outline*:**

1. **Basics:** The nature of computational physics; Tolls of computational physics; Machine representation; precision and errors in computations.
2. **Introduction to Computer Languages:** Computer languages for scientific computations and available compilers (C++, FORTRAN 90);
3. **Introduction to Scientific Software:** Computer algebra systems (Maple, MathCad, Mathematica).
4. **Basic Tools of Numerical Analysis in Science and Engineering**: Nonlinear equations; Interpolation and curve fitting; Numerical differentiation and integration; Available numerical libraries of C++ and FRTRAN.
5. **Matrix Calculations and Applications in Physics**: Linear systems; Eigenvalue problem.
6. **Ordinary Differential Equations and Applications in Physics and Engineering:** Initial value problem; Boundary value problem.
7. **Spectral Analysis and Physics and Engineering Applications:** Fourier series and transforms.
8. **Nonlinear Systems, Chaos and Applications: Random Processes**
9. **Data Analysis and Modeling with Applications in Physics.**
10. **Introduction to Scientific Visualization and Animation Techniques**

III. Laboratory-based Computational Physics Course

The formal prerequisites for this course have been college level physics and calculus level courses, but I have often been willing to waive some of the prerequisites if a student has had solid programming experience. Student taking the course are expected to have at least some minimal knowledge, although the programming is not a formal prerequisite. Over the years, there have been a significant number of students who have taken the course without prior programming knowledge. Finally, they developed good programming skills along the way, even though this lack entailed a somewhat steep learning curve at the beginning of the course. An immediate question that I faced when I teach such course was what programming language I should adopt. One constrain was the available programming language licenses at that institution. Usually I opted for FORTRAN, and a computer algebra system if available. My decision was mostly motivated by its intrinsic array, its mathematical library and the available free source codes which turned out to very useful for the algorithms I had planned to illustrate and

the for the projects included in this course. I also use C/C++ for selected projects, helping to show the interoperability of two programming languages.

A distinctive aspect of our approach is that the physics drives the course rather than programming. There are many topics that can be discussed, and they may vary from year-to-year, depending on our interests and those of the students. Particular programming constructs and numerical algorithms are introduced as needed. In addition, we are stressing visualization and animation more and more as a way of understanding the behavior of a system. An important goal is to instill good programming habits. Thus, we encourage modular programs with reusable components. We also encourage students to avoid tricky programming constructs that might lead to faster but less readable code.

The project-based approach provides much flexibility in the choice of topics. Students work at their own pace and do not all work on the same assignments at the same time. Those with stronger programming skills are encouraged to explore the physics deeper while novices of programming might spend most of their time working on coding. The mix of abilities and levels encourages students to teach each other. In addition to the instructor, we typically have a teaching assistant to help during the laboratory times. First, we discuss problems that can be solved using deterministic algorithms. These might involve the numerical solution of differential equations, used to solve classical problems. Thus, they provide a natural means of showing students that there is more to physics than what is discussed in introductory physics. We have students compare different algorithms and determine the level of accuracy one can expect from such algorithms. Many of these topics require very little formal preparation, and are rarely discussed in other physics courses. Topics may include examples of systems traditionally considered by other scientists such as polymers or earthquakes. Thus, the student can see how their skills can be used in a much broader context. Second part deals with projects, focusing on more advanced topics. The project based structure of the course leads to a number of important differences between our course and the traditional ones. For example, the instructors do not know all the answers and hence their role is different from in a traditional lecture course. Because some students know certain aspects about computer languages and programming better than us, these student have an opportunity to contribute to the course, and thus become more committed to it. Occasionally, students discover behavior that was not expected, and can tell the class about their results. This possibility exists because most of the assignments have some open ended aspects in addition to direct questions. The direct questions help students see what the relevant issues are, and the open-ended suggestions allow students to explore a topic in more depth.

A key component of our course is teaching students how to communicate scientific information. Students write laboratory reports, which are the basis for their grade in the course in addition to class participation and initiative. These reports include a summary of the system being modeled, a discussion of the method used to simulate the system, verification of the program, presentation, analysis and interpretation of the data, and a critique discussing the key physical concepts and techniques learned by doing the project. In addition, students are expected to keep a computer-based laboratory notebook that includes a log of the time spent on their work. The reports are an excellent vehicle for

teaching writing skills. Each project begins with a discussion of a physical situation, followed by a discussion what we can be learned about the system using analytical methods. During the lectures, students are received programs that they can use as a starting point for their own program or as pseudo-code if they are using a different language. They run the program and collect data that they can use to test its validity and their understanding of what the program is doing. The final project is written up as laboratory report and presented as a poster. The above process follows the typical method of doing research in science. Students learn the importance of testing and how easy it is to make mistakes. They learn how difficult it is to distinguish between interesting behavior and various errors. They also learn the importance of having some analytical results and physical insight to guide their investigations. They also learn to share information. *All of these lessons are important to the researcher and rarely incorporated in a meaningful way in traditional courses*.

The instructor and a teaching assistant are available during the laboratory sessions to answer and ask questions. Specific programming questions are discussed in the laboratory. Students are asked to work a total of 8–10 hours per week on the course, a total that is often more than some of their other courses. There are many obstacles that must be overcome to achieve an effective course. One example is that students frequently have difficulty analyzing their data or the use the proper visualization programming tools. The mix of undergraduate and graduate students in the same class, when this is possible, has turned out to be good for both types of students. Although the course usually begins with traditional topics that are well-known to the graduate students, they enjoy improving their programming skills in a traditional context. It also is not a good assumption that the graduate students will do better than the undergraduates. The progress of graduate students in the computer simulation course proves to be a much better predictor of how students will do in research than their grades in the traditional graduate courses. Some students do better in open-ended environments than others. Grades in the course are determined by students' progress since they started the course, rather than on their absolute knowledge. This procedure can cause some confusion and uncertainty among students. However, students generally appreciate the fact that their grade is usually proportional to their effort within the time constraint.

III.1 Teaching Goals

An important question that arises when teaching computational physics is: **What do we want to achieve?** During the course, students learn to solve a physics problems using computer simulation. This includes diverse tasks: a) formulating the physics problem in a way suitable for computer simulations; b) choosing an efficient computational algorithm; c) writing and testing computer code; d) running the computer simulations and collecting data; e) analyzing, interpreting and visualizing the data obtained; and, f) extracting the solution of the physics problem. A distinctive aspect of our approach is that the physics drives the course rather than programming. There are many topics that can be discussed, and from year to year, they may vary depending on our interests and those of the students. Particular programming constructs and numerical algorithms are introduced as needed. Students with stronger programming skills are encouraged to explore the physics

deeper, while novices in programming might spend most of their time working on coding. In our project-based, typically the students work about two short projects and they do a four-week final project of their choice at the end. The above process follows the typical method of doing research in science.

III.2 Project Format

In our view the team must clearly define a physical problem that they want to solve. There are three recommendations for the project are: a) Don't choose anything too ambitious or challenging; b) Write your own numerical program to solve it; and c) While doing this, discuss with others, including myself, the best method to use and the smartest way to implement it. I am encouraging the students to do groups of two. Once the numerical results from the program were computed, the students are required to test that the program is working. While writing the program is often perceived as the hardest part, it is actually the analysis of the results, which make numerical projects worthwhile. Therefore, I required students to spend considerable time on thinking of everything you can do and test with their programs. The report does not have to be long, but it should include: (i) a description of the problem to be solved, (ii) a description of the method used (with the program attached), (iii) tests of the code to convince everybody that it is working properly, and (iv) the presentation and analysis of results.

Time Schedule for the Projects:

1. Second week of the semester, the mid-term projects start.
2. The seventh week of the semester, the e-files of the midterm projects, programming codes and written report are due.
3. Eighth week of the semester, the final projects start,
4. The last week of the semester the carbon copy of the final projects are due.

In the exam week, the e-files of the written reports of the final projects are due. During this week, the oral presentations of the final projects are scheduled.

Suggested Projects; the students are encouraged to propose good projects related to their research if any. They can also pick one of the projects listed by the instructor.

III.3 Project Sample - Radiative Transfer in a Planetary Atmosphere

In this project, you will construct a Monte Carlo calculation of radiative transfer in a planetary atmosphere. The radiative transfer will be simulated using a random walk of photons through the atmosphere. In this calculation, photons are emitted by the planet's surface and random walk their way through absorption and reemission in the atmosphere until they eventually escape to space or are reabsorbed by the ground.

Project Description and Program Development: It is a good idea to begin by developing the script for a single random walk through the atmosphere. The problem implies that the layers are nearly transparent. Thus, each time you encounter a layer, there

is only a small chance that the photon will be absorbed and reemitted. The program input is the *total* optical depth of the atmosphere. Then the atmosphere is divided into a large enough number of layers so that, for each layer, the chance of absorption is small. Obviously, for a given total optical depth, the approximation that the chance of absorption is small will be better for larger and larger numbers of layers. But, also obviously the program will take longer to run with larger numbers of layers. So the artistry here is to pick an optical thickness for the layers that is small enough for an accurate calculation, but big enough so that the calculation does not take forever. Here you will want to keep track of some of the critical data in the calculation:

- Number of photons which escape: P_E.
- Number of photons which are absorbed by the ground: P_A
- For each layer, *i* in the atmosphere, the number of photons that are absorbed in that layer: $P_L(i)$.

All quantities to the emission from the top of the atmosphere, which we know should be equivalent to the rate of emission from a blackbody at temperature T_{eff} are normalized. For a given effective temperature, input by the user, and for a given optical depth, input by the user, your program should produce:

- An estimate of the ground temperature of the system.
- An estimate of the temperature of the top layer of the atmosphere.
- A plot of the temperature of the atmosphere versus optical depth in the atmosphere, where the optical depth defined to be 0 at the top of the atmosphere and increasing with depth into the atmosphere.

You will need to decide how many trial photons are required to get accurate answers. In this case, we will define "accurate" to mean that the ground temperature is derived to an accuracy of 0.2K. *In your report, you should be sure to tell the number of trial photons used and justify your selection.*

Questions

1. We claim that the above model is a good simulation of the Earth's atmosphere. Compare the ground temperature result to the Earth's typical ground temperature. How good is the agreement? What would have to be done in order to improve the agreement of your model?
2. Compare the behavior of the temperature of the top of the nominal model atmosphere to the temperatures high in the Earth's atmosphere. (Here the most relevant region is the base of the Earth's *stratosphere*.) Is there good agreement?
3. Using the optical depth and effective temperature values given for the nominal calculation, test the sensitivity of the ground temperature to a change in optical depth. How big a change in optical depth is required to increase the surface temperature by 5 degrees?

4. A change in the planet's reflectivity can change its effective temperature. For a one percent increase in reflectivity (that is 0.34 rather than 0.33 for the albedo), how much does the ground temperature decrease with NO change in optical depth.
5. Venus has an effective temperature of 240K, but its surface temperature is 700K! Find the atmospheric optical depth that is required to accomplish this.

Extra Credit Calculation: To make our calculation easy, we have divided the atmosphere into layers of equal optical depth. In the real atmosphere, these layers would have different thickness since the optical depth depends on the mass of the layer and the density of air decreases with altitude. The density of air in the Earth's atmosphere, d, follows the exponential law:

$$d(h) = d(0) \exp(-z(km)/8(km))$$

where z is the altitude and 8 km is the e-folding scale of the exponential law (known as the scale height in atmospheric physics.) Each of your optical depth layers must have the same amount of material in it, so we can use this relationship for density versus height to derive the height corresponding to each layer, assuming that the opacity only depends on the amount of material. Make a graph of the temperature derived from the model versus the actual height in the atmosphere. How does this compare to the temperature profile of the Earth's atmosphere?

IV. **Student Assessment**

To have an estimate of the effectiveness of our teaching approach of computational physics at the end of the course, using the project-based teaching approach all students are requested to answer (with a five point scale: 1-very poor, 2-poor, 3-satisactory, 4-good and 5-very good) an anonymous questionnaire as shown in Table 1.

Table 1 Questionnaire for course evaluation

Q1	Are the course challenging and interesting?
Q2	Have you learn more than expected with the course?
Q3	Are the projects useful to you?
Q4	Are the lectures of high quality and easy to understand?
Q5	What was the level the laboratory exercises?
Q6	Please, provide an overall evaluation of the course

According to the results, the project-based approach received ratings above 3.6, comparing with an average rating of 3.4/5.0 for the all courses at program (the average of the last five years). The results from the students' feedback have been extremely positive with the regard to the projects and the exercises provided during the laboratory sessions. The majority of students felt that such projects enhanced their understanding of the theoretical materials and made the course more interesting, either they consider that the work required, during the course is higher than in a traditional one. Similar surveys are planned to be conducted at the end of each term when the curse is offered.

V. Conclusions

Computational physics is taught as project-based laboratory course, instead of the lecture course traditional approach. We find that this format allows us to introduce various computational methods and gives a broader overview of computational physics. It also helps the students to become more proficient at programming. The students use the symbolic computational skills they learn in this course to do their home-works and projects. After completing the course the students may use programming, symbolic and numerical skills acquired during the course in their research, gaining a set of skills that will help them succeed after they graduate. In our interactive approach of teaching computational, we aim to: a) focus on mainly the science aspects of computational physics, not on programming; b) give students hands-on experience in designing and running computer programs and simulations; and c) play a role similar to that of laboratory courses for experiment. The project-based computational physics course is aimed at upper-level undergraduates and beginning graduate students, and is intended not only for physics students but also students from the other sciences and engineering.

References

1. Binder K. in *Thermodynamics and Statistical Physics: Teaching Modern Physics*, p. 45–66, Proc. 4th IUPAP Teaching Modern Physics Conf.,
2. Velarde M. G. and F. Cuadros, eds., World Scientific, Singapore (1995).
3. Frenkel D. and B. Smit, *Understanding Molecular Simulation*, Academic Press (1996) and http://molsim.chem.uva.nl/frenkel smit/.
4. H. Gould, "*Computational physics and the undergraduate curriculum*," Comput. Phys. Comm. 127 (1), 6 (2000).
5. H. Gould, J. Tobochnik, and W. Christian, *An Introduction to Computer Simulation: Applications to Physical Systems*, third edition, Addison-Wesley (2006).
6. J. Tobochnik and H. Gould, "*Teaching computational physics to undergraduates*," in Annual Reviews of Computational Physics IX, edited by D. Stau_er, World-Scienti_c (2001), p. 275.
7. H. Gould and J. Tobochnik, "*Integrating computational science into the physics curriculum*," in Lecture Notes in Computer Science Vol. 2074, Part I, 1031, Springer (2001).
8. H. Gould and J. Tobochnik, "*Using simulations to teach statistical physics*," in Computer Simulation Studies in Condensed Matter Physics XVI, edited by D. Landau, Steven P Lewis, and Heinz-Bernd Schuttler, Springer (2004).
9. Johnson, A. I. Mel'cuk, H. Gould, W. Klein, and R. D Mountain, Phys. Rev. E **57**, 5707 (1998).R.L. Spencer, *Teaching computational physics as a laboratory sequence,* Am. J. Phys., Vol. 73(2), February 2005, pp. 151-152
10. *R. H. Landau, Resource Letter CP-2: Computational Physics,* Am. J. Phys., Vol. 76(4&5), April/May 2008, pp. 296-306.
11. S. Koonin, *Teaching Computational Physics,* Engineering Sciences, March 1987, pp. 17-20.