

# Computational Physics with Python

Rubin H. Landau<sup>1</sup>, Cristian C. Bordeianu<sup>2\*</sup>, Manuel J. Paez<sup>3</sup>

Oregon State University, Physics Department, Corvallis, OR 97331, USA  
University of Bucharest, Faculty of Physics, Bucharest-Măgurele, P.O. Box MG 11,  
077125, Romania

University of Antioquia, Medellin, Colombia

\*E-mail: cristian.bordeianu@brahms.fizica.unibuc.ro

## Abstract

*A coherent set of material for upper-division university education in computational physics/science has been developed at Oregon State University, USA. It contains an introductory course in scientific computing, a course in Computational Physics, and a coordinated collection of multimedia interactive animations which enhance the book and the courses. Computational Physics programs using Python programming language are presented and displayed. It is proposed that presentation using Python is a more effective and efficient way to teach physics than the traditional one.*

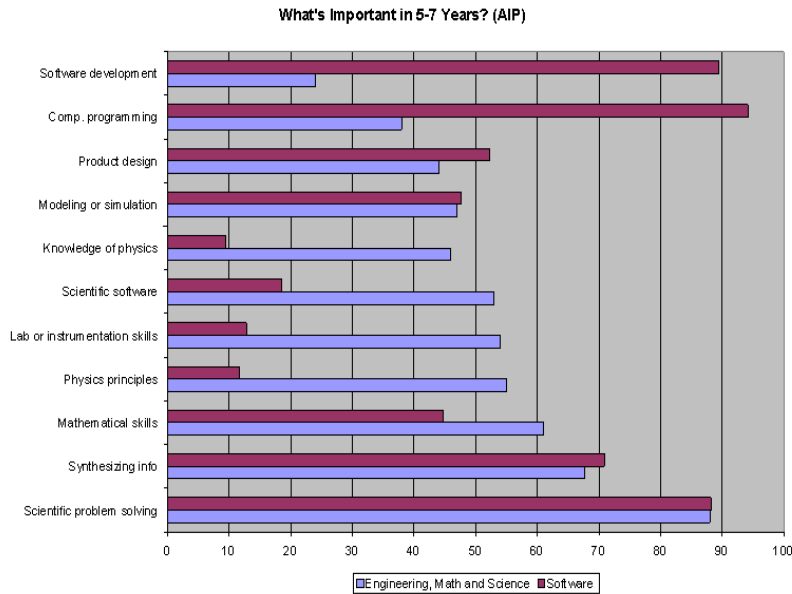
## 1 The Need for Computational Education

We start by looking at the results of a survey of physics bachelors conducted by the American Institute of Physics that determined which aspects of their education are most valuable in their current employment five years after graduation (AIP, 1995). The results, shown in Figure 1, indicate that for graduates whose primary field of employment is engineering, mathematics and science, the three most important skills are scientific problem solving, synthesizing information, and mathematical skills. These skills are also highly important for graduates who find employment related to software. While it is to be expected that knowledge of software and programming are most important for graduates in software development, notice how, otherwise, synthesizing information is the most important skill for both groups, and that knowledge of physics is essentially the least important.

## 2 Framework for Teaching Physics with Computation

Figure 2 illustrates the scientific problem - solving paradigm that is at the core of computational research. Although such diagrams have been shown often enough to become visual clichés, they remain relevant to the focus of this paper since they provide the general structure for computational education. In fact, we believe that the commonality of tools across the computational sciences combined with the common problem-solving mindset is a truly liberating and attractive aspect of computational science because it permits its practitioners to understand and participate in a much wider set of problems than occurs otherwise in the sub specialization of science.

---



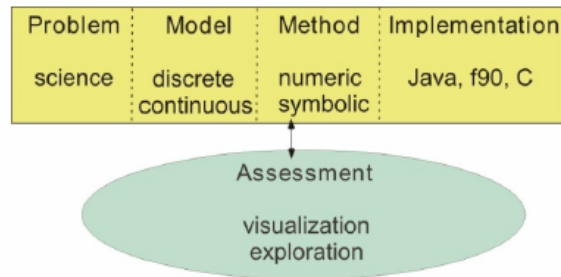
*Figure 1. Importance of knowledge areas for physics bachelors 5 - 7 years after graduation.*

In general, we recommend that computational educational materials be structured around the scientific problem-solving paradigm in Figure 2. This demonstrates where the multiples disciplines are relevant, provides concrete examples that assist in understanding the abstract concepts, and stresses the importance of assessment of the various components through visualization. From a pedagogical perspective, we believe that a Computational Physics education following the problem-solving paradigm is a more efficient approach to undergraduate education than a pure Physics education. Although students may take fewer Physics classes, they tend to learn Physics, Computer Science, and math better when placed in context, and thus get more out of their courses. So even if the number of Physics courses needs to be reduced to make room for teaching computation, this is compensated for by the increased efficiency of the pedagogy. Furthermore, this approach has been shown to be appealing to a more diverse group than those presently attracted to computer science or physics.

The materials and classes we've built along the way reflect our own rules of education, which are personal observations gleaned from decades of teaching:

- Most of education is learning what the words mean; the concepts are usually simple if only you can understand what is being said.
- Confusion is the first step to understanding.
- Traumatic experiences tend to be educational.
- Scholarly and pedagogical presentations are often designed to either impress the audience with the presenter's brilliance and depth or to make the materials appear simple and logical (we opt for the latter).

Figure 2. The scientific problem-solving paradigm. A problem is set, the tools from multiple disciplines are employed within context, and the continual assessment aides debugging and steering.



A key component of many computational programs is having students get actively engaged with projects as if each were an original scientific investigation, and having projects in a large number of areas. In this way students experience the excitement of their personal research, get familiar with a large number of approaches, acquire confidence in making a complex system work for them, and continually build upon their accomplishments. We have found the project approach to be flexible and to encourage students to take pride in their work and their creativity. It also works well for independent study or distant learning. In order to teach a Projects-based course, we employ a combination of lectures and "over the shoulder" labs. The students work on and discuss their projects with an instructor, and then write them up as an "executive summary" containing sections for

• Problem • Algorithm • Visualization • Equations employed • Code • Discussion & Critique

The emphasis is professional, much like reporting to manager in a workplace. Visualizations are important for all the classes, and we teach the use of *Maple/Mathematica*, *PtPlot*, *gnuplot*, *AceGr*, and *OpenDX* (Figure 3) for 2D, 3D, and animated plots. Taken together, this approach produces significant learning, even though we may be "teaching with our mouths shut". Also we teach modern digital signal processing techniques as wavelet analyses (Bordeianu, 2009).

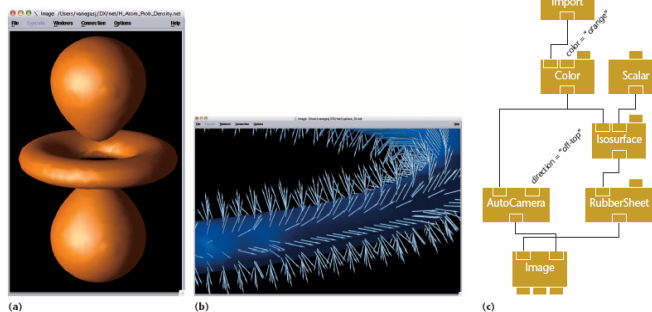


Figure 3. Example visualizations produced with *OpenDX*. (a) The 3D state of hydrogen, (b) an equipotential surface for a toroidal capacitor with the resulting electric field, and (c) the visual program that produced the hydrogen visualization.

### 3 What to teach

In Figure 4 we present a concept map for our *Computational Physics* course and text (Landau et al, 2008). After two years in administrative processing, in October 2001 the Oregon State Board of Higher Education approved a Bachelor degree in Computational Physics (Landau, 2004). The first students entered in fall 2002, the first graduate left in June 2003, and 3-5 students typically graduate each year. Although these numbers are small, the classes are well attended by physics majors, graduate students, and engineering students. A sample of the Computational Physics curriculum is given in Table 1. It is an example of how a complete package of computation classes can be fit into a four-year curriculum that is still strong in its mother discipline.

Year	Fall	Winter	Spring
<b>Fresh</b>	Differential Calculus (MTH 251, 4) Writing I, 3 Gen Chemistry (CH 221, 5) Perspective, 3	<b>I. Scientific Computing 1</b> (Ph/MTH 265, 3) Integral Calculus (MTH 252, 4) Perspective, 3 Gen Chemistry (CH 222, 5)	<b>CP Seminar</b> (PH 405, 1) Vector Calculus I (MTH 254, 4) Gen Physics + Calc (Ph 211, 5 ) Gen Chemistry (CH 222, 5)
<b>Soph</b>	<b>II. Scientific Computing 2</b> (PH 464, 3) Vector Calculus II (MTH 255, 4) Gen Physics + Calc (PH 212, 5) Perspective, 3	Infinite Series & Sequences (MTH 253, 4) Gen Physics + Calc (PH 213, 5) Perspective, 3 Writing II, 3	Applied Differential Eqs (MTH 256, 4) Intro Modern Phys (PH 314, 4) Writing III, 3 Fitness, 3 Linear Algebra (MTH 341, 3)
<b>Jr</b>	<b>CS Elective, 3</b> Harmonic Oscillations (PH 321, 2) Static Vector Fields (PH 322, 2) Energy & Entropy (PH 323, 2) Biology, 4 Elective, 3	Perspective, 3 Waves in 1D (PH 424, 2) Quant Measurements (PH 425, 2) Central Forces (PH 426, 2) Synthesis, 3 Elective, 3	<b>Computer Science Elective,</b> 3 <b>CP Seminar</b> (PH 405, 1) Periodic Systems (PH 427, 2) Classical Mechanics (PH 435, 3) Electives, 6
<b>Sr</b>	<b>III. CP 1</b> (PH 465, 3) Electromagnetism (PH 431, 3) Quantum Mechanics (PH 451, 3) Math Methods (PH 461, 3) Electives, 3	<b>IV. CP 2</b> (PH 466, 3) Physical Optics (PH 481, 4) <b>Computer Science</b> <b>Elective, 3</b> Elective, 3	<b>V. Adv Computational Lab Thesis</b> (PH 467, 401; 3, 3) CP Seminar (PH 405, 1) Synthesis, 3 Electives, 4 <b>Interactive Multimedia</b> (CS 395, 4)

Table 1. A sample *Computational Physics for Undergraduates (CPUG)* curriculum

This curriculum has been built up course by course since 1989 as we proposed, developed, taught, and modified new courses. The computer classes (**bold**) are seen to be

distributed throughout all years of study. In total, the curriculum is a mix of existing applied math and CS classes, with the new computation classes acting as the glue that holds it together.

There is another way to answer the questions “What to teach?” and “How to teach it?” That way is to provide computation-based textbooks that help define which topics constitute proper computational education, and provide a coherent presentation of the subject. The OSU Computational Physics Education group has been trying to do that for the last 15 years. Lists of more than 50 texts and other resources are to be found in a recent resource letter (Landau, 2008). Although most of those resources and most of this paper focus on more specialized computational topics, there is still very much an open question on what and how to teach computation to beginning college science students, and who should be doing the teaching. Our attempt takes the form of an *Introductory Scientific Computing* course designed to provide first and second year students with the computational tools needed throughout their undergraduate careers, and its associated text, *A First Course in Scientific Computing* (Landau, 2005). In recognition of the widespread disagreement over which computing tools lower division college students should learn, the paper text covers Maple and Java, while the accompanying CD contained essentially identical texts in Mathematica and Fortran90, as well as the associated notebooks, worksheets, programs, and data sets. The combination of *A First Course in Scientific Computing* and *A Survey of Computational Physics* (Landau et al, 2008) pave a continuous computational path throughout the undergraduate curriculum.

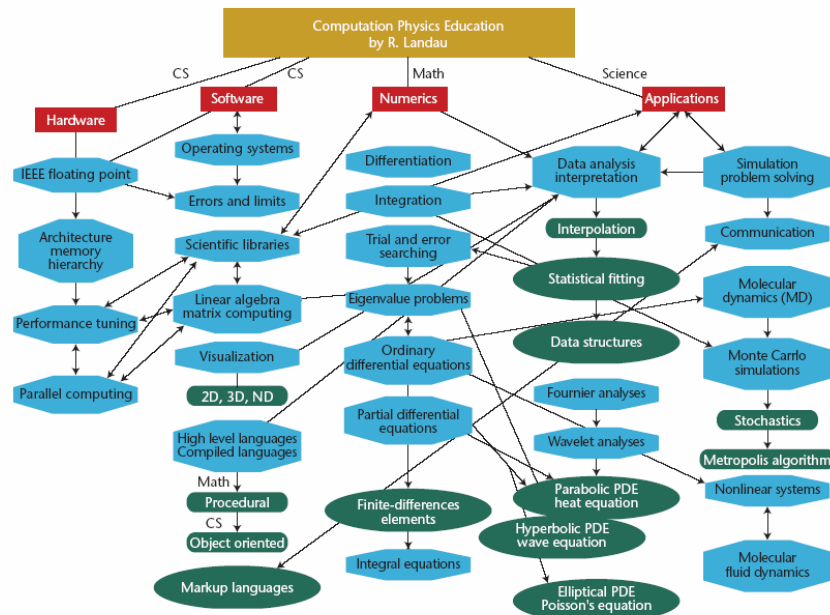


Figure 4. Concept map. Shows hardware and software components from computer science, applied mathematics algorithms, and physics applications.

#### 4 Online Courses and Digital Books

In addition to publishing text books, another way of encouraging the inclusion of more computation into curricula is to make at least the basic elements of computation courses available to faculty. As part of a demonstration project for establishing a national repository of computational science courses (EPIC), we have produced video based modules for our *Introductory Computational Science* course (Video, 2008). We already used them with good results in our teaching, while we are told that faculty and students at other schools are also finding them useful. In light of the previously documented large overlap among different computational classes, the plan is to have modules cover individual topics which then can be assembled and used in a variety of classes and in a variety of schools. (A full course would require problem sets, quizzes, assessment exercises, and possibly supplementary materials in a specific discipline.) Although we do not view the web as a good teaching medium for general education, or for students with weak self discipline or limited motivation, it is appropriate for computational science where the best way to learn it is while sitting at a computer in a trial and error mode. Actually, the web is essentially ideal for computational science (it was invented for particle physics analysis): projects are always in a centralized place for students and faculty to observe, codes and data are there to run or modify, and interactive visualizations can be striking with 3D, color, sound, and animation.

#### 5 Using Python

Python is a popular programming language used for both standalone programs and scripting applications in a wide variety of domains. It is free, portable, powerful, and remarkably easy to use.

One of the reasons why we decided to migrate to Python in our CP books and courses is that it provides a really nice balance between the practical and the conceptual (Downey et al, 2008). Since Python is interpreted, beginners can pick up the language and start doing neat things almost immediately without getting lost in the problems of compilation and linking. Furthermore, Python comes with a large library of modules that can be used to do all sorts of tasks ranging from web-programming to graphics. Having such a practical focus is a great way to engage students and it allows them to complete significant projects. However, Python can also serve as an excellent foundation for introducing important computer science concepts. Since Python fully supports procedures and classes, students can be gradually introduced to topics such as procedural abstraction, data structures, and object-oriented programming.

Another reason is the fact that Python is freely available for download. Versions are available for almost every operating system, including UNIX, Windows, Macintosh, and Java. In addition, the Python website includes links to documentation, how-to guides, and a wide assortment of third-party software.

The tools we have used in preparing the visualizations are:

**Matplotlib:** Matplotlib is a very powerful library of plotting functions callable from within Python that is capable of producing publication quality figures in a number of output formats. It is, by design, similar to the plotting packages with Matlab, and is made

more powerful by its use of the *numpy* package for numerical work. In addition to 2-D plots, Matplotlib can also create interactive, 3-D visualizations of data.

**Visual (VPython):** The programming language “Python” is so often employed with the *Visual* graphics module and the IDLE interface that the combination is often referred to as *Vpython*. Much of the use of the Visual extension has been to create 3-D demonstrations and animations for education, which are surprisingly easy to make and useful.

**Tkinter:** Python also contains a graphical user interface (GUI) programming module called *Tkinter* or *Tk*.

## 6 Conclusions

We think that only time will judge the viability of computational physics programs such as ours. However they do appear to attract new students and to provide them with broad preparation for future career choices. Also the use of Python programming language seems to be a good choice judging by the feedback of the students.

## REFERENCES

- AIP (1995): Skills Used Frequently by Physics Bachelors in Selected Employment Sectors. Technical report: American Institute of Physics Education and Employment Statistics Division.
- BORDEIANU, C. C., LANDAU, R. H. and PAEZ, M. J. (2009): Wavelet analyses and applications. *European Journal of Physics* 30, 1049-1062.
- DOWNEY, A., ELKNER, J and MEYERS, C. (2008): *How to Think Like a Computer Scientist. Learning with Python*. Green Tea Press.  
<http://www.greenteapress.com/thinkpython/thinkCSpy/thinkCSpy.pdf>
- EPIC: Engaging People in Cyberinfrastructure,  
[www.eotepic.org](http://www.eotepic.org)
- LANDAU, R. H. (2004): Computational Physics for Undergraduates, the CPUG Degree Program at Oregon State University. *Computing in Science and Engineering*. 6.
- LANDAU, R.H. (2005): *A First Course in Scientific Computing*. Princeton University Press, [www.physics.oregonstate.edu/~rubin/IntroBook/](http://www.physics.oregonstate.edu/~rubin/IntroBook/).
- LANDAU, R. H., PAEZ, M. J. and BORDEIANU, C. C. (2007): *Computational Physics. Problem Solving with Computers*, 2nd, Wiley VCH.
- LANDAU, R. H., PAEZ, M. J. and BORDEIANU, C. C. (2008): *A Survey of Computational Physics. Introductory Computational Science*. Princeton University Press.
- LANDAU, R. H. (2008), Resource Letter CP-2: Computational Physics, *American Journal of Physics* 76, 296-306.
- VIDEO (2008): Video Lectures in Intro Computational Science,  
[www.physics.oregonstate.edu/~rubin/COURSES/VideoLecs/](http://www.physics.oregonstate.edu/~rubin/COURSES/VideoLecs/)