9-2016

Coding and Data Visualization in the Science Classroom

Philip C. Nelson
University of Pennsylvania, nelson@physics.upenn.edu

Coding and Data Visualization in the Science Classroom

Disciplines

Physical Sciences and Mathematics | Physics

Coding and Data Visualization in the Science Classroom

Philip Nelson

How it's gonna be: We hear a lot about the importance of training STEM students to work well in teams, and certainly that's true. But there is another skill set that will be critical for our students' future, one that's much less talked about but that jumped out at me as I read *Average is Over* by Tyler Cowen: In many areas, even beyond STEM, the successful professionals are going to be the ones who can *integrate with machines* to do things that neither humans nor computers can do by themselves. We may not want to hear this, but as artificial intelligence takes over more and more routine white-collar work there will be an ever-increasing premium on this skill set.

Once, we could say that learning to use a spreadsheet, say, was adequate computer literacy. Today, however, this is the barest minimum. My message here is instead that writing code *from scratch,* in a general-purpose programming language, is a skill that

- Many of even our best students have not yet acquired;
- Is central to most kinds of current scientific research;
- Represents an entirely new mode of mental activity distinct from the other things we teach students to do;
- Enables an instructor to assign much more interesting and real-world problems;
- Gets many students excited and gives them a toolkit that they can and do carry over into all their subsequent classes and beyond.

Once upon a time, the tools needed to do useful computer work fell into two classes: Some were powerful, but expensive and specialized. Others were free but tedious to use for everyday tasks. It seemed necessary for a whole department to standardize on one, and each had its fierce advocates. Today, in contrast, there are a number of free and open-source tools that give students skills with full-power, industry-standard programming languages (including implementations of the languages Python and R). Crucially, it is now easy enough to get started using these languages that my students can get going quickly, without needing to dedicate a big chunk of a semester to this preliminary material. Now yesterday's gridlock disappears—each instructor can use whatever language she likes.

Another general-purpose skill that students must master is extracting conceptual information from graphical representations of data and models. One way to do this is to become expert at *creating* such representations themselves *(see inset)*. Here, again, general-purpose computer tools are the key. Students find them frustrating at first, but immensely empowering once they have a few successes.

How we get there: Many of the students in my class (Physics 280) arrive with no coding background. They need a lot of daily experience before this unnatural activity begins to feel natural. Over a decade, I've arrived at a method that seems to work.

• On Day 1 of the class, I tell the students where to get a free download of the Python language, and I distribute installation instruction. I tell them they need to come to class on Day 2 with a laptop, with this system installed and running. I pass out and collect a questionnaire.

• The questionnaire asks students for their general experience level with any computer math system. I use the responses to make teams of two students each. I assign partners so that a student from the lower self-evaluation levels is paired with one from the higher. I send everybody an e-mail with their partner's contact, saying "even if you're an expert, you must come to support your partner."

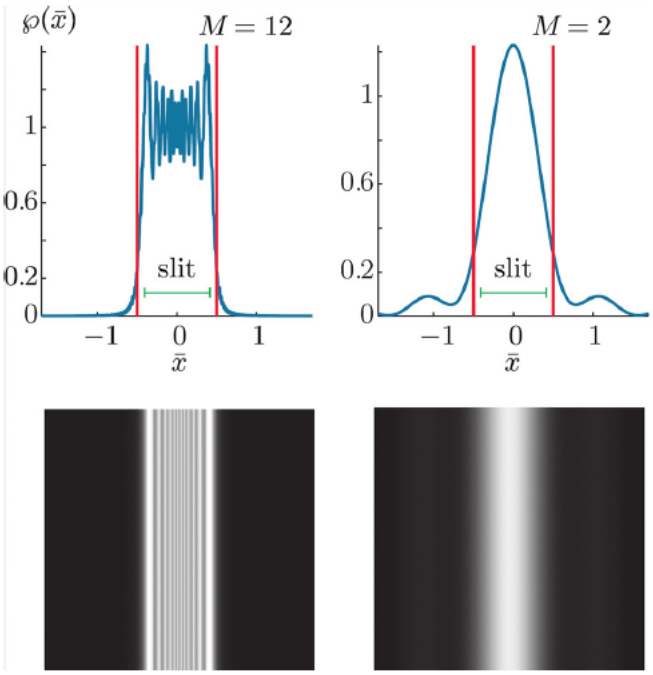
• On Day 2 of class I say, "Figure out if you or your partner has more computer experience; then that person should be advising, *not* touching the computer." I talk a little, show some things on the big screen, then stop and let the students try some things in the First Computer Lab section of the book. I walk around troubleshooting, along with one or two grad students. After maybe 10 minutes, I interrupt them and talk a little more, repeating till the class ends.

• Then we have some regular classes, followed by a second computer lab structured the same way but with different assigned partners, covering some new skills.

• After that I say, "If you liked either of your assigned partners, keep working together, but from now on work with whomever you like."

This is all the explicit programming instruction I do. From then on, I spend a little class time introducing some new syntax needed for that week's homework, and lots of instruction goes on in office hours.

The first lab session gets students started with tasks like pulling down an experimental data set from the Web, graphing it, and finding and displaying a mathematical function that resembles the data. The second lab session generally introduces simple ideas like loops, along with random number generation. This is enough to begin to see how the aggregate behavior of numbers that are individually random shows predictable behavior.



Learning to visualize the same data in different ways (top versus bottom figures) helps students to become better consumers of graphical representations, as well as empowering them to present their own work effectively. Both representations show predictions of diffraction patterns, which students compare to a real experiment to evaluate the theory embodied in the calculations.

How students respond: To do research, science students need skills including graphical presentation of data and model results, numerical math and handling of datasets. But few people enjoy studying a computer math package (or math itself) in an antiseptic, context-free way. My students get motivated when they have a concrete problem, perhaps one involved in obtaining a classic result, driving them to build up the skills to solve it.

I sent out a questionnaire to every alum of this course, some as much as seven years after they took it. I expected a low response rate, but in fact 80% of those I could reach responded. Overwhelmingly, when asked what if anything they were still using from this class, they replied "my first exposure to coding as a scientific tool." Yes, it's humbling to find that few are using the specific discipline knowledge that I thought I had imparted! But there it is: Over and over, they said that this one skill is what they use constantly today.

I'm proud of our students, of course, and of what they are doing with their Penn education. After all, the world has a lot of huge problems, which I'm counting on them to address. If they found this skill to be transformative, then I'm glad to have been the conduit.

Philip Nelson is professor of physics and the recipient of the 2001 Ira Abrams Award foar Teaching; he is the author with Jesse Kinder of A Student's Guide to Python for Physical Modeling *(Princeton Univ. Press).*

This essay continues the series that began in the fall of 1994 as the joint creation of the College of Arts and Sciences, the Center for Teaching and Learning and the Lindback Society for Distinguished Teaching. See www.upenn.edu/almanac/teach/teachall.html *for the previous essays.*