Scientific Computation in Jupyter Notebooks using Python

Mark Matlin
Bryn Mawr College
mmatlin@brynmawr.edu

ABSTRACT

Computation is a significant part of the work done by many practicing scientists, yet it is not universally taught from a scientific perspective in undergraduate science departments. In response to the need to provide training in scientific computation to our students, we developed a suite of self-paced "modules" in the form of Jupyter notebooks using Python. These modules introduce the basics of Python programming and present a wide variety of scientific applications of computing, ranging from numerical integration and differentiation to Fourier analysis, Monte Carlo methods, parallel processing, and machine learning. [1] The modules contain multiple features to promote learning, including "Breakpoint Questions," recaps of key information, self-reflection prompts, and exercises.

KEYWORDS

Scientific computation, Python, Jupyter notebooks

1 INTRODUCTION

The modules were developed under a grant from the TIDES (Teaching to Increase Diversity and Equity in STEM) project of the Association of American Colleges & Universities. [2] It is intended that the modules will provide a largely independent learning experience in algorithmic thinking and scientific programming for science students, but they also could serve as the core content of or a supplement to a formal course.

Our strategy in the first year of the grant was to design a Computational Methods pilot course, primarily for physics majors but also other physical science students, that would serve as a temporary vehicle to develop, pilot, and assess modular learning units in a blended learning format, and to gain experience with teaching practices that support diverse classrooms. That course, PHYS 350, was offered at Bryn Mawr in the spring of 2015. It was run mainly by the author as a computational lab: students were asked to do readings in the modules ahead of class time, and they worked on the embedded exercises during class. Lecturing was kept to a bare minimum. Help was available from the author, a colleague, and a TA, but the students usually looked to their classmates for help, and that was encouraged.

[1]Much of the scientific computation content was based on the outstanding textbook Computational Physics, Mark Newman, CreateSpace Independent Publishing Platform, 2012.

[2]Former colleagues Elizabeth McCormack (now at Bowdoin College) and Doug Blank (now at Comet ML) were members of the Bryn Mawr TIDES team.

© 2024 Journal of Computational Science Education
https://doi.org/10.22369/issn.2153-4136/15/2/5

A secondary goal of the set of modules beyond teaching computation (and the focus of the TIDES project) is to attract to computing and computational science students from groups typically underrepresented in those fields. One approach to that goal is to give students an appreciation of the wide range of individuals who have contributed to the development of computational science, or who have used it in interesting or important ways. This approach is implemented in nearly every module by the inclusion of a brief scientist profile, most of which were developed by the students in PHYS 350. A second approach, which was employed at regular intervals in that course, was to ask students a "reflection question" meant to encourage them to think about how science and computation was relevant to them and could serve their personal goals. Multiple reflection questions are now presented in the modules.

Since the first offering of PHYS 350, the modules have been used more than a half dozen times in labs offered at both the first- and second-year levels, by the author and separately by two colleagues who were not previously familiar with the modules. As a result of these experiences, involving roughly 100 students, we now offer the lab in the second semester of the sophomore year, when we believe our students can get the most from the course while also acquiring computational skills in time for them to be used in upper-level courses.

2 THE MODULES

At the core of our methodology for developing the computational materials was an open-source technology called a "computational notebook." These integrate text, formatted equations in LaTeX, computations, visualizations, and other media such as sound and video, and render it in a web browser window. Computational notebooks easily allow students to author, reproduce, and adapt code, and insert text and hyperlinks. We believed the notebook format of computation would allow us to implement our blended approach to create modules that could be easily distributed to other institutions. Most importantly, the notebook platform allowed us to integrate culturally responsive material and activities directly into the computational instruction modules. In addition, computational notebooks could be used by the students to express their own narratives and personal connections in the form of "reflections" directly alongside their coding work.

Specifically, for our project we used the Jupyter open-source notebook framework (https://jupyter.org/), which supports over 40 computing languages. A significant advantage to utilizing an open-source platform is that students will have access to it free of charge, even after they have left Bryn Mawr College. (An open-source distribution of Python, which includes the Jupyter platform as well as many useful Python packages, is Anaconda, https://www.anaconda.com/.)

Our 14 initial modules (one with multiple parts) all use the Python programming language. (A zeroth module introducing computation and providing lists of resources is offered as a PDF file; two additional modules on advanced topics have been added more recently.) Python is especially well-suited for computing education and is used by researchers in a variety of fields, including biology, physics, chemistry, linguistics, and computer science. Using authentic tools on authentic problems has been shown to be effective in education [1].

Module 0 introduces computing in general and, over three parts, Module 1 introduces the basics of programming and Python, as well as some software library packages that extend the power of Python for scientific computing. Module 2 discusses the concepts of numerical error and computing time. Module 3 presents the first application of computing to problems relevant in physics: the Euler-Cromer method is used to solve one- and two-dimensional motion problems, without and with drag. (The exercises for this module include ones modeling projectile motion in the distance-dependent gravitational field of Earth, the flight of a rocket accounting for mass loss, and a solar-sail- powered spacecraft.) Module 4 introduces the forward, backward, and central derivatives, as well as interpolation. Our typical second-year student can get through the preceding modules and at least part of Module 4 in a one-semester course.

Subsequent modules cover important topics in scientific computation that a student might find useful in their upper-level courses or in research: numerical integration, linear equations, eigenvalue equations, data analysis /visualization, Fourier techniques, nonlinear equations, ordinary differential equations, partial differential equations, Monte Carlo methods, symbolic computation, object-oriented programming, parallel computing, and machine learning. A detailed table of contents of all the modules is provided in an appendix to this article.

An Instructor's Guide to the modules, including a flowchart of module dependencies, is provided along with the modules (see the Conclusion section). It also includes an overview of all the modules with some instructions for their use, the prompt we present to students for developing scientist profiles, and several reflection prompts.

As noted, the computational modules were designed with several features intended to help meet the goals of the TIDES project to enhance diverse and inclusive learning. Some of these features were formulated with the principles of Universal Design for Learning in mind. (The UDL philosophy is to provide learning materials and modalities that will assist all students, including those from underrepresented groups in STEM.) The features we incorporated in the modules are:

(1) *Context information*: The start of each module lists the prior modules with which students must be familiar in order to undertake the current module; a time estimate for reading and working through the module, including the exercises; and the learning goals for the module. Some modules also include a brief list of some interesting applications of the mathematical tools presented in the module. (Module 0 omits the last two elements, as well as features 2-5 below.)

(2) *Scientist profile*: Each module contains a personal profile of a scientist connected with computing that describes their work and some interesting facts about their life; most of these profiles were generated by students. In constructing the profiles, the students were asked to focus on more-or-less contemporary individuals whose lives or work they found inspiring in some way. We plan to continue to ask students working with the modules to generate such profiles. A couple of comments received from the students in the PHYS 350 course regarding one of the profiles were "To me, the most inspiring thing about this story was that Dean grew up in a very difficult time for Black Americans, yet still succeeded academically and professionally to a huge degree. It reminds me that no matter where you come from or what disadvantages you might start out with, if you are dedicated, work hard, and love what you do, nothing can stop you from achieving your goals. I think I can apply this to my academics and career choice to help me decide what I spend my life doing based on my true interests." And "His [Dean's] story was very inspirational considering he was an African American growing up in the 60's and 70's and was able to achieve so much despite any adversity he faced. He was able to work on many large projects, not just one, which is something I aspire to do."

(3) *Breakpoints*: These are quick questions provided to prompt students to stop and think about what they have just read—along with answers to those questions—so students can confirm their understanding immediately. (Student feedback from the pilot course motivated inclusion of the breakpoint answers in the modules themselves rather than in separate solution sets.) A couple of example breakpoint questions from the module on numerical differentiation are "Prove this claim that b is the derivative of the quadratic curve $y = ax^2 + bx + c$ at $x = 0$" and "Write out the expression for the quartic (degree 4) approximation to the derivative."

(4) *Exercises*: These ask students to put into practice the concepts they learn in the modules. Those in early modules are relatively straightforward to complete; the exercises increase in difficulty as students progress through the sequence of modules and build their skills and confidence. Many of the exercises involve standard applications or computations that students will encounter in their physics courses. (We hope to add application problems from other scientific disciplines where appropriate in the future.)

(5) *Reflection prompts*: During the Computational Methods pilot course—which was run as an interactive session, sometimes including a short lecture but with the bulk of the time used for students working independently and in small groups—we periodically posed questions designed to improve the students' metacognitive skills. Some of those questions and others have been incorporated into the module notebooks themselves so that students will encounter them no matter the context in which they engage with the modules (i.e., whether in a course or during independent study). Examples of reflection questions posed in the modules are "Which components of this module did you find more difficult to work through, and why do you think they were challenging?" and "When you got stuck, what did you do to get unstuck? Could this or similar actions be helpful if you get stuck in future

work?" The Instructor's Guide includes additional prompts that might prove useful.

(6) *Recaps*: Bullet-point lists near the end of each module summarize the key takeaway ideas.

Other pedagogical elements connected with the modules that can enhance their effectiveness include:

Term project: As part of the pilot course, students completed a term project applying computation to some problem of personal interest to themselves. Most students chose project topics related to other courses they had taken or were taking, but one student chose a topic connected to her personal interest in music. Assignments that invite connections with a student's personal goals and interests can enhance motivation and persistence, two elements of learning found to be key in the retention of students from underrepresented groups in STEM [2, 3].

E-portfolio: To encourage students to see the connections between the modules and to recognize the full scope of their work, we asked each student in the pilot course to compile all of their module-related activities into an electronic portfolio ("e-portfolio") which they can use as evidence of their computational skills when applying to jobs or graduate schools.

3 IMPLEMENTATION

The pilot course produced extensive feedback on the individual modules from the 17 students in the course (nine women from Bryn Mawr College, one woman and seven men from nearby Haverford College), which was used to improve their clarity and sequencing. The feedback resulted in the refinement of breakpoints and self-reflection prompts, as well as the integration of the scientist profiles directly into the modules. The first few of the updated and expanded modules were next taught as part of the laboratory accompanying our second-semester, calculus-based mechanics course for physical science majors. There, we found that attempting to teach the basic material in a few weeks' worth of two-hour lab meetings to primarily first-year students was an overreach. Most of them had no programming experience, and the challenge of learning that skill on top of the physics they were learning in the course, not to mention the other demands of being a first-year student in college, was a bit too much to handle for many of them.

After an extensive departmental discussion, we decided to rearrange our sophomore-level labs so that one of them could be converted to an exclusively computational lab. By placing the basic modules in a second-semester sophomore lab, we intended to leverage the students' greater mathematical maturity while still introducing them to computing early enough in their college careers that they would have many opportunities to reinforce and further extend those abilities in their junior and senior years. The first offering of this computational lab occurred in Spring 2017 and was taught successfully by a new faculty member who had not been involved in the development of the modules. The results were promising: the students were able to get through the first five modules, apparently without significant difficulty, and the faculty member was able to learn and adapt the modules into the course pedagogy. Additionally, the author has now taught this lab five times since the pilot course and has updated the modules each time in response to student difficulties. We will continue to present the computational learning program in this way each year. More-advanced modules will be assigned in upper-level physics courses, and in our capstone senior seminar we will ask students to complete and present their e-portfolios with a final reflection on their computational learning.

4 CONCLUSION

With the twin goals of providing training in scientific computation to our majors and recruiting and retaining more students from U.S. underrepresented groups in STEM fields, we developed, piloted, and have made publicly available curricular materials on scientific computing.

We intentionally embedded and integrated learning-promoting features such as personal profiles of noteworthy individuals and reflection prompts into the instructional modules in order to send a strong signal to both students and other faculty members that inclusivity and metacognition are valued in our classrooms, and to make it less likely that these features will be overlooked or dropped from courses utilizing the modules due to time constraints. The modules designed in this way may also serve as a model for other faculty members adapting these materials or designing their own.

Our curricular materials can be incorporated into curricula at other institutions. Outcomes will be highly dependent not only on the details of those curricula, but also on the prior knowledge and experiences of students, as well as their expectations.

The computational modules are available as open educational resources under a Creative Commons license, and others are invited to use and adapt them as they wish. The modules and supplementary materials are provided at https://github.com/BrynMawrCollege/TIDES. They're also posted in the Faculty Commons section of the PICUP website at https://www.compadre.org/PICUP/ under the title "Scientific Computation with Python in Jupyter Notebooks." Supplementary materials include notes for a mini lecture on Python, which the author has begun presenting in the first two class meetings to get students started on learning Python. Also included is a Python Quick Reference Guide showing code for common basic course tasks. Exercise solutions are available for instructors upon request. Readers interested in learning more about inclusive teaching practices will find materials from a workshop we offered in the "BMC TIDES Diversity and Equity Workshop" folder on the github site. Those materials, which include handouts for facilitators and participants, as well as feedback forms and additional suggested readings, should be straightforward to adapt to other institutions wanting to run their own workshops.[3] An extended discussion of our project (and the other TIDES projects) can be found in [4].

REFERENCES

[1] Andrea Forte and Mark Guzdial. 2005. Motivation and nonmajors in computer science: Identifying discrete audiences for introductory courses. *IEEE Transactions on Education* 48, 2 (2005), 248–253. https://doi.org/10.1109/TE.2004.842924

[2] Judith M. Harackiewicz and Chris S. Hulleman. 2010. The Importance of interest: The role of achievement goals and task values in promoting the development of interest. *Social and Personality Psychology Compass* 4, 1 (2010), 42–52. https://doi.org/10.1111/j.1751-9004.2009.00207.x

[3] Judith M. Harackiewicz, Yoi Tibbetts, Elizabeth Canning, and Janet S. Hydea. 2014. Harnessing values to promote motivation in education. In *Advances in Motivation and Achievement*, Vol. 18. https://doi.org/10.1108%2FS0749-742320140000018002

[3]Jennifer Spohrer contributed to these materials.

[4] Kelly M. Mack, Kate Winter, and Melissa Soto (eds.). 2019. *Culturally Responsive Strategies for Reforming STEM Higher Education: Turning the TIDES on Inequity.* Emerald Publishing.

APPENDIX: MODULE TABLE OF CONTENTS

Module 12: Monte Carlo Methods.

- Random Numbers in Numerical Computation
- Monte Carlo Integration
  - The Mean Value Method
  - Importance Sampling
  - The Transformation Method
- Monte Carlo Simulations
  - The Ising Model
  - Simulated Annealing

Module 13: Symbolic Computing in Python.

- Starting with sympy
- Evaluation
- Derivatives
- Integrals
- Limits
- Power Series Expansions
- Equation Roots
- Simultaneous Equations
- Differential Equations
- Matrix Operations

Module 14: A Brief Introduction to Object-Oriented Programming.

- Why Use Object-Oriented Programming?
- The Idea Behind Object-Oriented Programming
- A Simple Example
- A More Sophisticated Example
  - Two Balls Connected by One Spring
  - Three Balls Connected by Two Springs
    - An Atomic Example
    - A Simplification

Module 15: Parallel Computing.

- Introduction
- The `multiprocessing (multiprocess)` package
- The `concurrent.futures` package
- The `joblib package`
- The `ipyparallel package`
- The `dask package`
- `cuda` and `numba`

Module 16: Machine Learning.

- Basics of Neural Networks
  - What is a Neural Network?
  - Neuron Inputs and Outputs
  - The Learning Process
- `PyTorch`
- `TensorFlow`
- `scikit-learn`