Modernizing Computational Nuclear Engineering Education in the Open

Kathryn Huff[1], Anthony M. Scopatz[2]

[1] *The University of California – Berkeley, Berkeley, CA 94709*
[2] *The University of South Carolina, Columbia, SC 29208*
huff@berkeley.edu, scopatz@gmail.com

INTRODUCTION

In the very early days of computing, nuclear engineering applications drove the majority of innovation in computers, computation, and scientific simulation [1]. Today's nuclear engineering students, however, are not poised to become leaders in computing, as was once the case. Modern scientific computing best practices have the potential to revitalize research computing in this domain and provide broader opportunities in industry for graduating students. But first, those practices must be taught to the next generation of nuclear engineers.

An example course syllabus is presented that will introduce university students to modern computational concepts and tools that encourage more reproducible, verifiable, and reusable scientific software. By incorporating recent insights from software development, this course equips students with skills essential to effective research in the data- and computing-intensive domain of nuclear engineering. To transform students into more effective researchers, this course emphasizes:

- **Accessibility**: with a contemporary, high-level, object-oriented, open-source programming language,
- **Power**: with sophisticated library design principles and powerful data structures,
- **Accuracy**: with workflows that emphasize testing and capture provenance,
- **Openness**: with projects and exercises emphasizing open science, code review, and collaboration.

In this way, the course follows the topics in the book, "Effective Computation in Physics: A Field Guide to Research in Python" [2] and uses that work as the course textbook. By emphasizing projects along the way, the students in this course will learn to create free and open source tools, verify the functioning of those tools, and publish them online. Additionally, the syllabus for this course is open-source licensed, allowing professors to use, re-mix, and collaboratively improve the syllabus for free.

This summary will first motivate the need for new university-level nuclear engineering computational curriculum from the perspective of scientific software development and data analysis best practices. Next, this summary will introduce the project-driven content and timeline for the course as well as the collaborative nature of lesson content itself. Finally, the success of similar training will be discussed along with a schedule of future pilots of the course.

MOTIVATION AND STRATEGY

Nuclear engineering once led the world in computing and has fallen behind, but not becaus computing challenges in the field have gotten simpler. Quite the contrary. Nuclear engineering applications continue to rely on datasets that are often enormous in many dimensions, traverse disparate scales, incorporate many physics, and demand precision.

For example, the driving equation for neutron behavior, the time-dependent Boltzmann equation, is solved in a 7-dimensional phase space, (3 in space, 2 in angle, and one each in energy and time). The scale of a nuclear reactor simulation is therefore inherently large, spanning five orders of magnitude in space and ten in neutron energy. Resolved discretization across those scales would require over 10^{17} degrees of freedom per timestep, well beyond the capabilities of even exascale computing. In this way, without sophisticated data and analysis methods, nuclear engieers would run out of computational resources before heat transport, fluid flow, or material performance in the reactor had even been addressed.

Faced with data and simulation scales such as these, it is imperative that nuclear engineering students are prepared to reclaim a leadership role in cutting edge scientific computation. Equipping the next generation of students with effective computational skills will certainly lead to breakthrough advancements in nuclear engineering research and development. Specifically, training them in best practices already commonplace in commercial software development could have an enormous impact on reproducibility in nuclear science and engineering. Thus equipped, their contributions to nuclear engineering may even have a revitalizing effect on the field.

Python as a First Language

One explanation for technological stagnation in nuclear engineering coursework might be the historic reliance on legacy code in the field. In nuclear engineering, the regulatory climate has has necessitated the use of battle-hardened software implemented in legacy programming languages. Accordingly, contemporary programming languages are absent from nuclear engineering curriculum, and so too are the best practices in scientific software development [3] that have evolved alongside them.

Specifically, due to reliance on legacy software written in Fortran, the undergraduate first course in computing at many Nuclear Engineering departments emphasize MATLAB or Fortran rather the contemporary equivalents, Python and C++.

Python, a high-level, dynamically-typed language, is now the most popular language for introductory Computer Science curriculum [4] and is very popular for education more broadly [5, 6, 7]. Its readable syntax [8], straightforward object-orientation, and open source packages give it an enormous educational advantage over competitors (see Fig. 1) whose licenses limit the flexibility and share-ability of software produced within those frameworks.

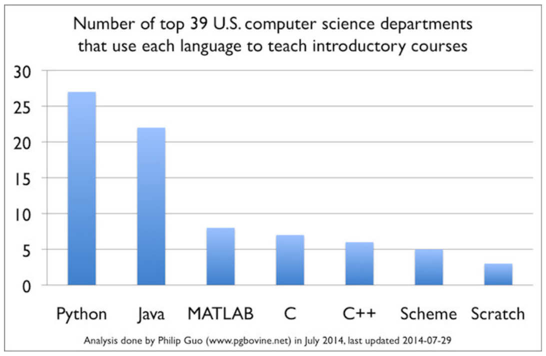

Fig. 1. Python popularity in introductory computer science curriculum. Figure reproduced from [4].

In addition to the above reasons, the authors recommend using Python because object-orientated languages are already becoming common in nuclear engineering software development. Neglecting object orientation at the high level (with MATLAB) or at the low level (with historical Fortran) leaves students unprepared to participate in the cutting edge software in nuclear engineering since new tools are increasingly written in those object oriented languages (e.g. Serpent[9], MOOSE[10], and PyNE[11]). It should be noted that modern Fortran specifications are object-oriented. However, these capabilities are often not covered in engineering courses in favor of a more long-standing procedural style from historical Fortran languages.

Efficient and Reproducible Workflows

Best practices for efficient and reproducible modeling and analysis, though common in commercial software development, are typically missing from nuclear engineering curriculum as well. A similar lack of such workflows across academic science at large and is at the root of an enormous credibility crisis in scientific computing [12, 13].

Tools available for reproducible science should be emphasized early in order to prepare students for the rigor of engineering work. Concepts such as verification and validation should be instilled in a computational course. Doing this through the implementation of unit test suites, version control, debugging skills, and other defensive programming techniques will create more accurate and reliable software.

Open Source and Open Science

Computational tools and data grow more robust with a broad base of users and developers [14, 15]. The physics community has used this fact to the great benefit of, for example, the Large Hadron Collider experiment. In that instance, faced with an enormous computing challenge and potential for human error, the experimenters opened both the software and the data to the whole world. This massive scale peer review is unknown in nuclear where a combination of export control issues and antiquated workflow paradigms mean that only a very small number of tools have been open sourced [10][11][16].

In addition for formulating the course around an open source language (Python), the projects in the course are focused on student contributions to actual open source projects in nuclear engineering. Potential code-bases to which students could propose contributions include MOOSE[10], PyNE[11], Cyclus[16], and even OpenMC[17] and OpenMOC[18].

The skills learned in contributing to open source software go far beyond best practices in computing. They extend to the skills needed for scientific collaboration including communication, peer review, and objectivity.

Example Syllabus

To train the workforce in a authors recommend a course covering the topics above in a high level language (Python). A twelve week course with two days per week would have the schedule listed in TABLE I.

The syllabus above, along with lesson plans, exercises, customizable homeworks, grading rubrics, and project assignments is to be released online by Fall 2016. The syllabus is also an open and collaborative resource, which can be used and modified freely.

EXPERIENCE

Similar approaches have produced positive outcomes in courses within other domains (e.g. biology [19, 20, 21, 22] and physics[23, 5]) and have succeeded as workshop-style interdisciplinary efforts [24, 25, 26]. The success of one example in the space of workshop-style courses is noted in the following section.

Software Carpentry

The material and concepts proposed here were heavily influence by the work done by the Software Carpentry Foundation, an organization that

Software Carpentry has been involved with teaching scientific computing best practices for over a decade. Students have come from many areas of science and usually no programming experience is necessary to attend a course. The courses are typically two days and cover skills for "Doing more with less pain."

Also, the testing, debugging, and collaboration sessions were taught as a part of a recent workshop at the University of California Berkeley. The students, in the course of four hours, were able to upload tests and example code to a version controlled repository and to deploy continuous integration for automated unit testing.

CONCLUSIONS

This course will equip students with contemporary concepts that will make them more effective researchers in the data and computing intensive domain of nuclear engineering. The course prepares students with fundamental skills such as

TABLE I. Lesson content per week.

Part	Lesson	Content	Project
Getting Started	1	Introduction to the Command Line	**Fuel Geometry Class**
	2	Programming Blast Off with Python	
	3	Essential Containers	
	4	Flow Control and Logic	
	5	Operating with Functions	
	6	Classes and Objects	
Getting It Done	7	Regular Expressions	**Database I/O and Visualization**
	8	NumPy: Thinking in Arrays	
	9	Storing Data: Files and HDF5	
	10	Important Data Structures	
	11	Analysis and Visualization	
	12	Performing in Parallel	
Getting It Right	13	Deploying Software	**Continuously Integrated Test Suite**
	14	Building Software Pipelines	
	15	Local Version Control	
	16	Remote Version Control	
	17	Debugging	
	18	Testing	
Getting It Out There	19	Automated Documentation	**Reproducible Publication**
	20	LaTeX	
	21	Collaboration Tools	
	22	Licenses, Ownership, and Copyright	

mobility in the UNIX shell, scripting, analysis, and visualization. Beyond that, it prepares them for more high performance computing tasks by introducing object orientation, databases, and parallelization. It also prepares them to incorporate reproducibility and accuracy into their scientific work through version control, testing, and automated workflow pipelines. Finally, by emphasizing the contributions to real open source projects along the way, the students in this course will learn to create and collaborate on free and open source tools, verify the functioning of those tools, and publish them online. Finally, the syllabus and course textbook are open and affordable, respectively.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the influence of Software Carpentry, The Hacker Within, the Berkeley Institute for Data Science, the African Institute for Mathematical Sciences, and especially Prof. Paul P.H. Wilson for their important contributions to the formation of both the book and the resulting course.

REFERENCES

1. R. RHODES, *The Making of the Atomic Bomb: 25th Anniversary Edition*, Simon & Schuster, New York, anv rep edition ed. (Jun. 2012).
2. A. SCOPATZ and K. D. HUFF, *Effective Computation in Physics*, O'Reilly Media, S.l., 1 edition ed. (May 2015).
3. G. WILSON, D. A. ARULIAH, C. T. BROWN, N. P. CHUE HONG, M. DAVIS, R. T. GUY, S. H. D. HADDOCK, K. D. HUFF, I. M. MITCHELL, M. D. PLUMBLEY, B. WAUGH, E. P. WHITE, and P. WILSON, "Best Practices for Scientific Computing," *PLoS Biol*, **12**, *1*, e1001745 (Jan. 2014).
4. P. GUO, "Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities," (2014), http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext.
5. C. R. MYERS and J. P. SETHNA, "Python for Education: Computational Methods for Nonlinear Systems," *Computing in Science & Engineering*, **9**, *3*, 75–79 (May 2007).
6. F. STAJANO, "Python in Education: Raising a Generation of Native Speakers," in "In Proceedings of the 8th International Python Conference, Washington DC," (2000), pp. 24–27.
7. A. BACKER, "Computational Physics Education with Python," *Computing in Science & Engineering*, **9**, *3*, 30–33 (May 2007).
8. A. STEFIK and S. SIEBERT, "An empirical investigation into programming language syntax," *ACM Transactions on Computing Education (TOCE)*, **13**, *4*, 19 (2013).
9. J. LEPPÃĎNEN, "SerpentâĂŞa continuous-energy Monte Carlo reactor physics burnup calculation code," *VTT Technical Research Centre of Finland*, **4** (2013).
10. D. GASTON, C. NEWMAN, G. HANSEN, and D. LEBRUN-GRANDIÃĽ, "MOOSE: A parallel computational framework for coupled systems of nonlinear equations," *Nuclear Engineering and Design*, **239**, *10*, 1768–1778 (Oct. 2009).

11. A. SCOPATZ, P. K. ROMANO, P. P. H. WILSON, and K. D. HUFF, “PyNE: Python for Nuclear Engineering,” in “Transactions of the American Nuclear Society,” American Nuclear Society, San Diego, CA, USA (Nov. 2012), vol. 107.
12. D. L. DONOHO, A. MALEKI, I. U. RAHMAN, M. SHAHRAM, and V. STODDEN, “Reproducible Research in Computational Harmonic Analysis,” *Computing in Science & Engineering*, **11**, *1*, 8–18 (Jan. 2009), 00110.
13. V. STODDEN, “The Scientific Method in Practice: Reproducibility in the Computational Sciences,” *SSRN Electronic Journal* (2010), 00037.
14. M. PETRE and G. WILSON, “Code Review For and By Scientists,” *arXiv:1407.5648 [cs]* (Jul. 2014), 00000 arXiv: 1407.5648.
15. J. M. WICHERTS, M. BAKKER, and D. MOLENAAR, “Willingness to Share Research Data Is Related to the Strength of the Evidence and the Quality of Reporting of Statistical Results,” *PLoS ONE*, **6**, *11*, e26828 (Nov. 2011).
16. R. W. CARLSEN, M. GIDDEN, K. HUFF, A. C. OPOTOWSKY, O. RAKHIMOV, A. M. SCOPATZ, Z. WELCH, and P. WILSON, “Cyclus v1.0.0,” *Figshare* (Jun. 2014), http://dx.doi.org/10.6084/m9.figshare.1041745.
17. P. K. ROMANO and B. FORGET, “The OpenMC Monte Carlo particle transport code,” *Annals of Nuclear Energy*, **51**, 274–281 (Jan. 2013).
18. W. BOYD, S. SHANER, L. LI, B. FORGET, and K. SMITH, “The OpenMOC method of characteristics neutral particle transport code,” *Annals of Nuclear Energy*, **68**, 43–52 (2014).
19. K. E. MATTHEWS, P. ADAMS, and M. GOOS, “Using the principles of BIO2010 to develop an introductory, interdisciplinary course for biology students,” *CBE-Life Sciences Education*, **9**, *3*, 290–297 (2010).
20. J. E. HONTS, “Evolving strategies for the incorporation of bioinformatics within the undergraduate cell biology curriculum,” *Cell Biology Education*, **2**, *4*, 233–247 (2003).
21. B. TJADEN, “A multidisciplinary course in computational biology,” *Journal of Computing Sciences in Colleges*, **22**, *6*, 80–87 (2007).
22. R. LIBESKIND-HADAS and E. BUSH, “A first course in computing with applications to biology,” *Briefings in bioinformatics*, **14**, *5*, 610–617 (2013).
23. G. W. BAXTER, “Scientific Computing with SciPy for Undergraduate Physics Majors,” in S. V. D. WALT and J. BERGSTRA, editors, “Proceedings of the 13th Python in Science Conference,” (2014), pp. 1 – 3.
24. K. D. HUFF, A. SCOPATZ, N. PRESTON, and P. WILSON, “Rapid Peer Education of a Computational Nuclear Engineering Skill Suite,” in “Transactions of the American Nuclear Society,” American Nuclear Society, La Grange Park, IL 60526, United States, Hollywood, FL, United States (Jun. 2011), *Training, Human Performance, and Work Force Development*, vol. 104, pp. 103–104.
25. G. WILSON, “Software Carpentry: Essential Software Skills for Research Scientists,” (2006).
26. G. WILSON, “Software Carpentry: lessons learned,” *F1000Research*, **3** (Feb. 2014).