An Interdisciplinary Elective Course to Build Computational Skills for Mathematical Modeling in Science and Engineering

Dr. Ashlee N. Ford Versypt, Oklahoma State University

Dr. Ashlee N. Ford Versypt is an assistant professor in the School of Chemical Engineering at Oklahoma State University. She earned her Ph.D. and M.S. degrees in ChE at the University of Illinois at Urbana-Champaign and her B.S. at the University of Oklahoma. She did postdoctoral research at the Massachusetts Institute of Technology. Her research focuses on developing computational models for multiscale tissue physiology and pharmacology. Her teaching interests focus on chemical reaction kinetics and computational science and engineering. She received an NSF CAREER Award in 2019.

An Interdisciplinary Elective Course to Build Computational Skills for Mathematical Modeling in Science and Engineering

Abstract

A cross-listed upper division and graduate elective course for students in science, technology, engineering, and mathematics (STEM) fields has been developed to build computational skills in mathematical modeling. The course aims to fill a gap in the practical training of students starting computational research projects across various STEM disciplines who have inconsistent previous experiences in computer programming and numerical methods. This is achieved by covering modern software tools for mathematical modeling in science and engineering and for reproducible research computing via an active, hands-on approach supplemented by reading materials. Rather than covering just the basics of programming or detailed algorithms for numerical methods, the course is geared towards implementing tools for solving realistic continuum scale science and engineering problems, managing open source code projects, and disseminating computational research results through scientific documentation and publications. The course is taught by a chemical engineering faculty member with research expertise in applied mathematics and computational science and engineering. MATLAB and Python are taught side-by-side throughout the course. The paper describes the course with the goal of enabling other educators to adapt and reuse the course content.

Introduction

Many STEM programs offer an introductory computer programming course to undergraduate students. This type of course typically focuses on details of a specific software or language, functional programming in a procedural or object-oriented paradigm, and the use of conditional statements and loops with basic data structures. A variety of languages are taught in such introductory courses as STEM educators have debated which is the "best" programming language to teach university STEM students[1-5]. Some undergraduate and graduate programs also include one or more courses on numerical methods, primarily aimed at teaching algorithms and error analysis techniques while surveying the appropriate methods to be used on categories of mathematical problems. Exposure to numerical methods is highly dependent on the university and the discipline[6]. Both the introductory programming and numerical methods courses clearly have their place in STEM fields; however, student training in these areas varies widely. This causes challenges for undergraduate and graduate students interested in research and industrial projects involving mathematical modeling using computational tools. Even with similarly titled prerequisite courses, students may have had drastically different experiences. Typically, students with gaps in their computational training are recommended or required by their research advisors or degree programs to take the time-intensive versions of the introductory undergraduate courses or comprehensive graduate courses, which are often not tailored to the practical computational skills that they need to engage in advanced mathematical and computational modeling in a relatively short time frame. By accelerating the training time to develop competency in implementing modern best practices, students are enabled to be productive at using computational tools for research early in their graduate studies, ideally allowing them to satisfy time-sensitive demands for generating research results.

Several chemical engineering educators have developed course materials, books, and software aimed at computational tools for solving problems commonly encountered in chemical engineering[7-11]. Often these resources are taught in courses for upper division undergraduate and first year graduate students in chemical engineering. Many other STEM fields do not have analogous resources. Additionally, modern computational topics such as version control, reproducible computing, and design of graphical user interfaces are generally excluded from chemical engineering modeling courses.

The Carpentries is a non-profit organization that has developed hands-on educational modules in software and data science aimed at rapidly teaching practical computational skills in short workshops to researchers with little relevant prior experience[12, 13]. Workshops on several topics are taught at many universities, and all the lessons are available online[14]. The idea for developing the elective course came from my experience in attending a Software Carpentry workshop on Python. The workshops include a significant amount of problem-based learning through active hands-on computational exercises during which the instructor answers questions around the room; this was preserved in the semester-long elective course. Additionally, both the workshops and the course aim to rapidly develop proficiency rather than provide a detailed coverage of all background and theoretical aspects, which are suitably covered by existing courses.

To build computational skills in students interested in mathematical modeling from all STEM backgrounds, I developed the interdisciplinary elective course titled Applied Numerical Computing for Scientists & Engineers at Oklahoma State University. The course fills a gap in the practical training of students interested computational research projects and problem solving across various disciplines. The course topics are focused on modern software tools for mathematical modeling in science and engineering: version control using Git, mathematical typesetting in LaTeX, graphical user interfaces, and high level programming languages with libraries of solvers and visualization tools (Python and MATLAB). Version control is a system for automating and documenting the management of changes in computer codes and software. LaTeX is the standard document preparation system for formatting technical and scientific communications involving mathematics. I discuss and emphasize best practices for computational problem solving and scientific computing research from examples (and counter examples) from my research and from the published literature[15-33]. Rather than covering just the basics of programming or detailed algorithms for numerical methods, the course is geared towards implementing tools for solving realistic continuum scale science and engineering problems, managing open source code projects, and disseminating computational research results through scientific documentation and publications.

Based on my personal experiences with MATLAB, Python, C, and FORTRAN, I opted to teach MATLAB and Python side-by-side in the course due to the widespread adoption of these two languages across STEM fields and the accessibility to novice programmers. MATLAB has an extensive collection of built-in functions, toolboxes, and visualization capabilities well-suited to STEM applications. Python is open source and is emerging as the standard language for open scientific computing. “The learning progression across two programming languages is critical to developing a student’s ability to generalize across various computational tools”[2]. Thus, we used

both MATLAB and Python for the course. With competency in both of these languages, students could extend their skills into other languages such as C, Java, or FORTRAN as needed.

This paper is written in a "steal this course" format to provide other instructors access to the course materials, which they can then adapt as desired.

Methods

Note that most of this Methods section is taken from the course syllabus with a few explanatory notes infused here for the readers.

Course Learning Objectives

Upon completion of this this course, students should be able to

- utilize Git for version control using common commands: status, add, commit, push, pull
- write scientific reports and similar documents in the LaTeX typesetting language with TeXMaker as the editor using an article template and include equations, figures, tables, document hierarchy, cross referencing, and citations (using BibTeX) in the documents
- use basic Unix commands to run programs, navigate and organize a file system, and access the university's supercomputing cluster
- use best practices for computational problem solving and research and scientific computing as described in publications provided as assigned readings
- develop graphical user interfaces for interactive model reuse
- program well-documented, readable code in the high-level languages of Python and MATLAB that uses libraries, built-in functions, and user-defined functions
    - to solve systems of linear and nonlinear equations,
    - to numerically integrate functions and data,
    - to solve systems of ordinary and partial differential equations,
    - to estimate parameters for mathematical models using optimization and data fitting tools,
    - to calculate statistics of data and stochastic processes, and
    - to create publication quality figures

General Course Structure

The class meets twice weekly in 75-minute periods. Prerequisites include an undergraduate differential equations course and a basic programming course or experience with any programming language (students should be familiar with functions, if statements, and for loops). Students are assigned to read relevant background materials before class. The first portion of each class period includes a 5 – 10 minute group discussion of the readings. The bulk of each class period is dedicated to working example problems or walking through instructor-led tutorials. Students bring their personal computers to class. Guest lectures from science and engineering faculty conducting computational research supplement the course. The course does not have exams. Student grades are distributed among ten reading assignments (2% each), six computational assignments (the first two worth 5% each, the third worth 10%, the fourth and

fifth worth 15% each, and the final worth 25%), and one video assignment (5%). These assignments are discussed in the following sections. Course topics are adjusted or reordered based on students' grasp of concepts in in-class exercises, office hours, and assignments and based on their feedback. For example, students in 2018 requested that tips for using LaTeX for manuscript and thesis/dissertation preparation be discussed at the midpoint of the course rather than in the last week as in previous years. Students are encouraged to work together on computational assignments and to seek assistance from the instructor.

Reading Assignments

In each reading assignment, students are required to type a brief summary (0.5 – 1 pages) for each of the documents assigned. The summary must be in their own words. Most of the readings are extracts from books and journal articles[15-33]. A few online manuals for software tools are also provided. All of the reading materials are posted on the course learning management system for students to access. Topics for the reading assignments include

1. Background and overview information on LaTeX for mathematical typesetting
2. Best practices for software engineering in scientific computing
3. Basics of Python programming
4. Using built-in functions in MATLAB and Python for solving common classes of problems in scientific computing with established numerical methods, focusing on nonlinear equations, numerical integration, and ordinary differential equations (ODEs)
5. Python modules NumPy and SciPy
6. Parameter estimation by linear and nonlinear least squares regression
7. Sensitivity analysis
8. Graphical user interfaces (GUIs) for scientific computing in MATLAB and Python
9. Verification and validation in scientific computing
10. Reproducible research computing and other tips for sharing figures, code, and documentation from computational projects

Computational Assignments

Computational assignments give students practice with programming well-documented, readable code in MATLAB and Python through use of libraries, built-in functions, and user-defined functions to solve systems of ODEs, to estimate parameters for mathematical models using optimization and data fitting tools, to create publication quality figures, and to design and implement GUIs for interacting with mathematical models. Students are also tasked with documenting their work in LaTeX, HTML, and/or Jupyter Notebook formats in certain assignments. The time allotted for each assignment is roughly proportional to the weight of the grade of that assignment.

The overviews for the assignments are as follows:

1. Version control in Git and document typesetting in LaTeX
    - Create a Git repository to track versions of assignment files (in this and subsequent assignments)

    - Produce a LaTeX document with several required components using research or major course work as the topic
2. Programming in MATLAB while developing best practices for scientific computing (version control, commenting, and documentation)
    - Write a function to define a system of ODEs
    - Provide well-documented code following specified standards
    - Generate an HTML output file from MATLAB documenting the code
3. Using built-in functions and library routines for numerical methods (specifically ODE solvers) in MATLAB and Python
    - Solve a system of ODEs using numerical solvers in MATLAB and Python
    - Plot the results
    - Generate an HTML file to document the code from MATLAB
    - Generate a Jupyter Notebook file and a LaTeX file to document the code from Python
4. Parameter estimation of dynamic models using MATLAB and Python
    - Solve a system of ODEs using numerical solvers in MATLAB and Python
    - Use an optimization routine to iterate the ODE model parameters to fit data
    - Plot the results
    - Generate an HTML file to document the code from MATLAB
    - Generate a Jupyter Notebook file and a LaTeX file to document the code from Python
5. Develop a GUI in MATLAB starting with an existing computational model
    - Create a GUI in MATLAB to take user inputs and display simulation results from a set of user-defined functions provided by the instructor
6. Design and construct a GUI in MATLAB, verify code implementation, and review content covered throughout the course
    - Develop a GUI for MATLAB that takes a user-specified number of ODEs and explicit equations as input, solves the system of ODEs using ode45 in MATLAB, returns and exports the solution vector, and plots the solution vector components against the independent variable
    - Verify that the GUI works for test cases from the systems of ODEs used in Computational Assignments 3 and 4

For computational assignment 5, the problem topic has varied between course offerings. The top student submissions have been reused in STEM outreach activities conducted by my lab. In 2016, the user-defined function for the topic was the numerical solution of a time-dependent 2D heat transfer problem (Figure 1). In 2017 and 2018, the user-defined function was an agent-based model for the motions of bees in response to a pesticide (Figure 2).

The final course assignment is treated as a cumulative course project in lieu of a final exam. Along with two students who took the course, I published a detailed description and sample submissions for the final assignment[34].

Differentiation of Instruction between the Undergraduate and Graduate Students and the Video Assignment

Only graduate students were allowed the first time the course was offered (Fall 2016). Based on student interest, senior undergraduate students were allowed in subsequent years (Fall 2017 and Fall 2018). During the 2017 course, the official university course designation and description was submitted for approval. At that time, a generic special topics course distinction was used. The instruction in 2017 was exactly the same for both cohorts of students, except that I asked that the graduate students present brief introductions to their research topics during a class period. This activity was not required for a grade. As all of the seniors were chemical engineering students who I had taught the previous semester, I was very familiar with their baseline of training in computational topics in our curriculum. I used a lot of examples from their earlier courses. Graduate students in disciplines outside of chemical engineering had a wide range of computational backgrounds, so I started the class with fundamentals that were approachable by the undergraduate and graduate students alike. To have the course approved for separate undergraduate and graduate course designations, the course requirements had to differ in some way. Thus, for the 2018 course, I included the video assignment for the graduate students to develop their skills in communicating to lay audiences and for them to think about how to implement course topics into their graduate research and education. All students, including the undergraduate students, were tasked with watching the videos to diversify their exposure to computational topics across the STEM disciplines represented among the students. To assess this, students were required to provide a substantive comment or question related to the video content for each video that they watched (brief comments such as "good job"—while positive and supportive—are not engaging nor do they provide evidence of watching, so students were instructed that these did not count towards the required number of comments).

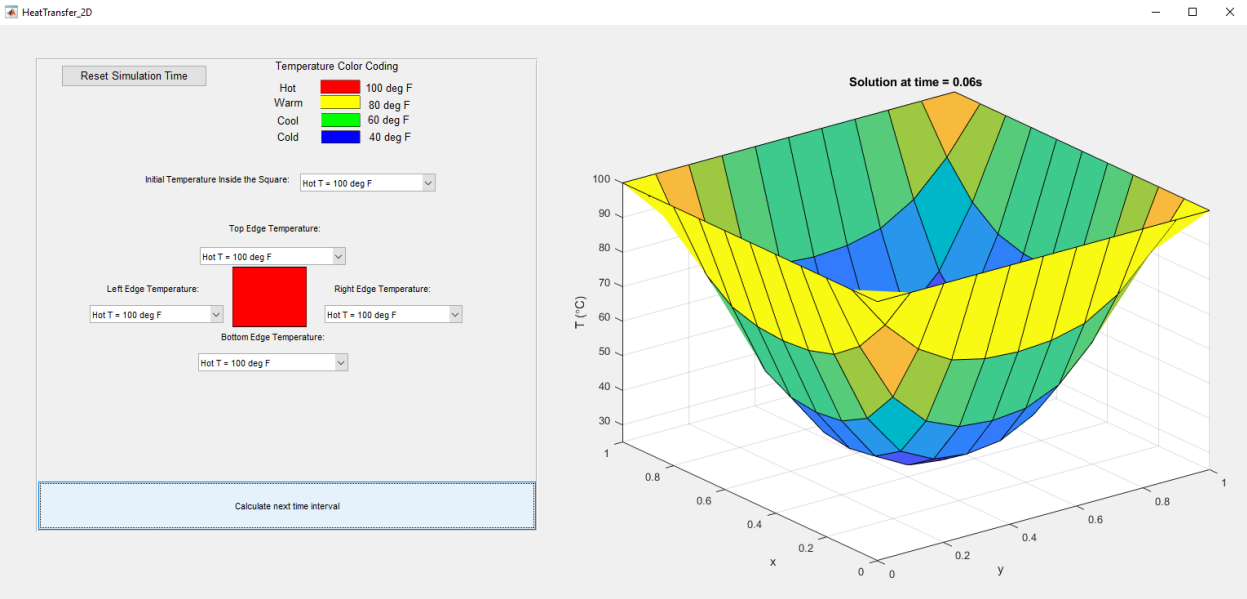

Figure 1: Sample student submission for Computational Assignment 5 in 2016.

Results

Student Performance

In the three offerings of the course thus far, 30 graduate students and 10 senior undergraduate students have taken the course. The student majors were chemical engineering, mechanical engineering, aerospace engineering, civil engineering, environmental engineering, chemistry, plant and soil sciences, and mathematics. Collectively, the final grades have averaged 3.73 on a 4.0 scale. The students who earned B's typically struggled with major concepts on two or three assignments and procrastinated on at least one assignment, where they submitted less than half of the required components. Each of these students discussed course performance with me one-on-one in office hours, learned from their mistakes on earlier assignments, and spent adequate time and effort on the final course assignment demonstrating that they had gained competency in course topics by the end of the course. Most of the more than 75% of students who earned A's in the course regularly attended class and asked me questions about homework assignments either after class or during office hours.

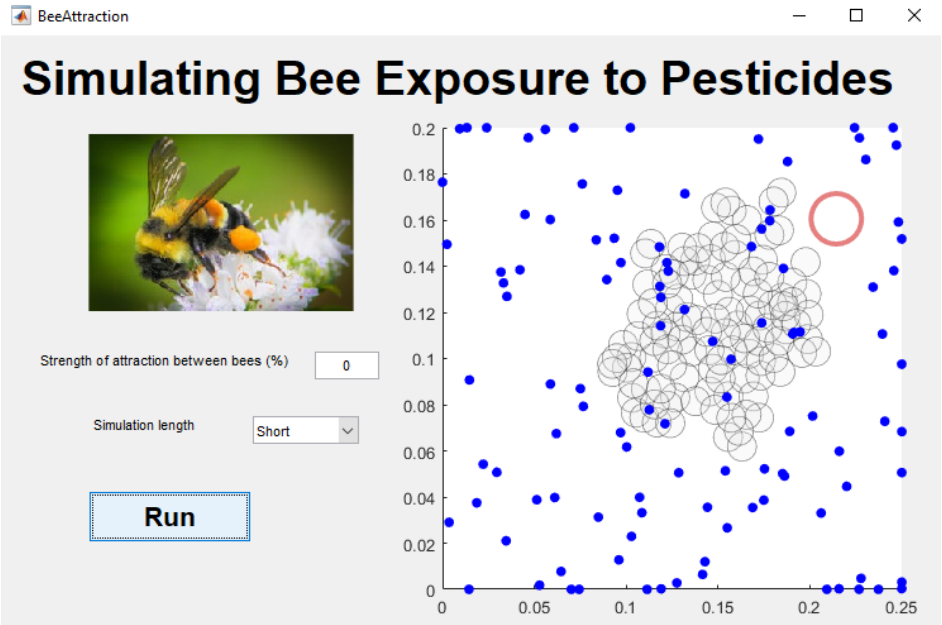

Figure 2: Sample student submission for Computational Assignment 5 in 2018.

Student Incorporation of Course Topics

In the 2018 course offering, one of the course assignments required each graduate student to create a 3 – 5 minute video describing how the course topics relate to their field of study or how they use the course topics in their research. The students evenly distributed themselves between these two alternatives. Of those that discussed how they used course topics, one graduate student from outside chemical engineering enthusiastically described how he was teaching his labmates and his research advisor how to use version control and LaTeX for scientific communication. Others echoed the adoption of version control and typesetting with LaTeX as key things they were translating into their research, no matter their field of study.

Graduate students from my research group and other computational research labs have used many of the skills developed in the course in their research endeavors. Several students showed me examples of GUIs in MATLAB and Python that they were developing for use on research or course projects outside of my class and asked for assistance in improving their GUIs. Almost all of the graduate students who have taken the course have utilized LaTeX for writing their theses or dissertations and some manuscripts.

The CHE seniors who took the fall 2017 course were competent MATLAB users by the end of the course. MATLAB training was not formally provided in any other course in the curriculum. In senior design in Spring 2018, at least two of the students told me how they had built a MATLAB simulation for their design team to use to complete the AIChE national design contest. All five of the students were able to help their peers with MATLAB when they used it in process control in Spring 2018 for a two-week course project involving using a hands-on temperature control lab[35].

Course Evaluations

The response of faculty to the course has been supportive as several faculty members (in several departments) are encouraging or even requiring all of their incoming graduate students to take the course. The university's high performance computing center distributes the course flyer to all users and labs with user accounts. The instructor advertises directly to the chemical engineering junior class each spring. These three factors and word of mouth account for much of the enrollment, which has been capped at 20 graduate students and 10 undergraduate students. The student responses to the course have been very positive, with evidence provided by university-administered anonymous online end-of-course evaluations. The university did not report evaluation results for the 2016 course or the 2018 undergraduate section of the course due to having fewer than three respondents for each. There were sufficient responses in 2017 for both student cohorts and for the graduate students in 2018. In aggregate over 29 responses in two semesters, the average rank for the instructor was 4.59/5 (4.40 ± 0.57, 4.57 ± 0.68, and 4.7 ± 0.53), and the average view of the course was 3.64/4 (3.71 ± 0.50, 3.63 ± 0.56, and 3.61 ± 0.51). In the free response section of the evaluations, students provided the following selected comments (copied without modification, except redaction of the instructor's name):

- "My computing skills (especially through Matlab and Python) I have gained throughout this course has been tremendously improved. The computational methods and MATLAB skills I have learned in this course have been not only beneficial in my research but also in other classes of mine, specifically Process Modeling course in the Fall, and Advanced Bioprocessing in the Spring. The course is highly recommended to be maintained and required for other science-related majors students (both graduates and undergraduates), especially those who are interested in computational research. Aside notes for math majored students or students who are taking numerical analysis course from the mathematics department; this course is highly suggested as a prerequisite, or taken simultaneously with numerical analysis."
- "The course 'Applied Numerical Computing' is very informative and is helping me a lot in my research. This course contains almost all the topics that I am wondering for and also much new topics that I am learning. For example, version control to track our

research progress, LaTeX to write our proposal or any future papers, Python and Matlab where I learned a lot about coding and it potential applicability and other numerous topics are covered. I must be very thankful for the course and Instructor."

- "I liked learning so many different computational topics in just one course."
- "Really enjoyed the diversity of the class in terms of lessons and computer languages taught."
- "Very interesting course for students to take where you can gain skills you may not normally gain in the classroom."
- "Very useful for applying in research."
- "Excellent and relevant topics for science-based students."
- "This course is highly related to my research work, and I really enjoyed this course. Students fought to take this course due to its reputation from last year."
- "Very good. Covered useful computational methods and coding practices. I really enjoyed the side-by-side coding in Python and Matlab on assignments. It helped me learn how to 'translate' my code into another language."
- "Really impressed, great course"
- "10/10 would recommend to anyone in science or engineering."
- "offered a lot reading materials during teaching, which helped me learn this course faster."
- "I think learning the version control and the latex software were the most beneficial content for me."
- "This course is highly recommended for graduates students/undergraduate research students who are interested in and conducting computational research."
- "Very useful course, have already suggested it for friends to take."
- "I found it very helpful that the course was taught in a way that was accessible to those without a strong background in computer science and programming."
- "It is a very good course to learn on Computational software."
- "Its a good course for the beginner in the field of numerical analysis."
- "Our professor prepares the materials lucidly and clearly that we can understand it, also does the coding on class in Python and Matlab the makes us understand each code. So, I learned many coding practices from basic level to advance."
- "Overall, this course is very good and all the graduates should take this course as a required graduate course. So that they can get basics in Latex, for their paper writing, computing skills in MatLab, python."
- "The course was great. I learned so much information yet I didn't feel overwhelmed. I was challenged but the course was still manageable. The material was all very useful."
- "Work Load: Almost too light."
- "Work Load: Can be pretty heavy at times. Reading assignments can be really time consuming (some are very long), especially when you also have a computational assignment to work on."
- "The course was really great but I wished we could focus more on open-source materials (Python) rather that MATLAB."
- "Work Load: The homeworks were rather long, but I believe she gave more than sufficient time to complete them. Additionally, there weren't that many of them, so overall, the homework to time ratio was perfect. However, I did not care for the reading

assignments. They were DEFINITELY useful and relevant, but just being given and graded on reading assignments in grad school seems a little weird and slightly annoying. Again, the reading assignments were useful and relevant, but it would have been nice to have them as supplemental instead of required."

Conclusions

An elective course suitable for upper division undergraduate and graduate students in STEM fields with limited prior experience in computer programming and numerical methods with diverse skillsets has been developed to train students to be competent users of Git, LaTeX, MATLAB, and Python for mathematical modeling and research computing. Although the topics were initially selected due to relevance to my research group, faculty from across several departments in engineering, science, and mathematics view the course as useful training for their incoming graduate students engaged in computational projects. Students from across the university have taken the course and have provided positive feedback on their experiences. Course reading and computational assignments were outlined here for brevity. I am willing to share any or all course files (syllabus, grading rubric spreadsheets, assignments, readings, etc.) electronically with educators upon being contacted by email.

References

1. Kavianpour A & Kavianpour S (2016) The first course of programming: Python, Matlab, or C? *Proceedings of the 2016 ASEE Annual Conference*.
2. Brophy SP & Lowe TA (2017) A learning trajectory for developing computational thinking and programming. *Proceedings of the 2017 ASEE Annual Conference*.
3. Rhudy M & Nathan R (2016) Integrated development of programming skills using MATLAB within an undergraduate dynamics course. *Proceedings of the 2016 ASEE Annual Conference*.
4. Kassim HO & Cadbury RG (1996) The place of the computer in chemical engineering education. *Computers & Chemical Engineering* 20:S1341-S1346.
5. Larsen KF, Hossain NMA, & Weiser MW (2016) Teaching an undergraduate introductory MATLAB course: Successful implementation for student learning. *Proceedings of the 2016 ASEE Annual Conference*.
6. Kaw A, Collier N, Keteltas M, Paul J, & Besterfield G (2003) Holistic but customized resources for a course in numerical methods. *Computer Applications in Engineering Education* 11:203-210.
7. Shacham M (2004) An introductory course of modeling and computation for chemical engineers. *Computer Applications in Engineering Education* 13:137-145.
8. Rodrigues AE & Minceva M (2005) Modelling and simulation in chemical engineering: Tools for process innovation. *Computers & Chemical Engineering* 29:1167-1183.
9. Yeo YK (2018) *Chemical Engineering Computation with MATLAB* (CRC Press, Boca Raton, FL).
10. Finlayson BA (2014) *Introduction to Chemical Engineering Computing* (Wiley, Hoboken, NJ) 2nd Ed.

11. Cutlip MB & Shacham M (2008) *Problem Solving in Chemical and Biochemical Engineering with POLYMATH, Excel, and MATLAB* (Prentice Hall, Upper Saddle River, NJ) 2nd Ed.
12. Jordan KL, Corvellec M, Wickes ED, Zimmerman NB, Duckles JM, & Teal TK (2018) Short-format workshops build skills and confidence for researchers to work with data. *Proceedings of the 2018 ASEE Annual Conference*.
13. Wilson G (2016) Software Carpentry: Lessons learned [version 2]. *F1000Research* 3:62.
14. The Carpentries (https://carpentries.org/).
15. Ekmekci B, McAnany CE, & Mura C (2016) An introduction to programming for bioscientists: A Python-based primer. *PLOS Computational Biology* 12:e1004867.
16. Hall T & Stacey JP (2009) Chapter 2: Designing Software. *Python 3 for Absolute Beginners*, (Springer, New York).
17. Hamby DM (1994) Review of techniques for parameter sensitivity analysis of environmental models. *Environmental Monitoring and Assessment* 32:135-154.
18. Hartmann AK (2015) *Big Practical Guide to Computer Simulations* (World Scientific, Hackensack, NJ) 2nd Ed.
19. Hemez FM & Kamm JR (2008) A brief overview of the state-of-the-practice and current challenges in solution verification. *Computational Methods in Transport: Verification and Validation*, ed Graziani F (Springer-Verlag, Berlin), pp 229-250.
20. Higham DJ & Higham NJ (2005) *MATLAB Guide* (Society for Industrial and Applied Mathematics, Philadelphia) 2nd Ed.
21. Kinder JM & Nelson P (2015) *A Student's Guide to Python for Physical Modeling* (Princeton University Press, Princeton, NJ).
22. Kiusalaas J (2005) *Numerical Methods in Engineering with Python* (Cambridge University Press, New York).
23. Lent CS (2013) *Learning to Program with MATLAB: Building GUI Tools* (Wiley, Hoboken, NJ).
24. Moler CB (2004) *Numerical Computing with MATLAB* (Society for Industrial and Applied Mathematics, Philadelphia).
25. Osborne JM*, et al.* (2014) Ten simple rules for effective computational research. *PLOS Computational Biology* 10:e1003506.
26. Prlic A & Procter JB (2012) Ten simple rules for the open development of scientific software. *PLOS Computational Biology* 8:e1002802.
27. Roache PJ (1998) Verification of codes and calculations. *AIAA Journal* 36:696-702.
28. Rother K, Potrzebowski W, Puton T, Rother M, Wywial E, & Bujnicki JM (2011) A toolbox for developing bioinformatics software. *Briefings in Bioinformatics* 13:244-257.
29. Stodden V & Miguez S (2014) Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. *Journal of Open Research Software* 2:e21.
30. Yale Law School Roundtable on Data and Code Sharing (2010) Reproducible research: Addressing the need for data and code sharing in computational science. *Computing in Science & Engineering* 12:8-12.
31. Piccolo SR & Frampton MB (2016) Tools and techniques for computational reproducibility. *GigaScience* 5:30.
32. Bangerth W & Heister T (2013) What makes computational open source software libraries successful? *Computational Science & Discovery* 6:015010.

33. Peng RD (2011) Reproducible research in computational science. *Science* 334:1226-1227.
34. Ruggiero SM, Zhao J, & Ford Versypt AN (2018) Building a MATLAB graphical user interface to solve ordinary differential equations as a final project for an interdisciplinary elective course on numerical computing. *J Comput Sci Ed* 9(1):19-28.
35. Hedengren J (2018) Temperature Control Lab. (http://apmonitor.com/pdc/index.php/Main/ArduinoTemperatureControl).