COMPUTATIONAL NOTEBOOKS IN CHEMICAL ENGINEERING CURRICULA

Fani Boukouvala[1], Alexander Dowling[2], Jonathan Verrett[3], Zachary Ulissi[4], and Victor Zavala[5]
1. *Georgia Institute of Technology • Atlanta, GA 30332*
2. *University of Notre Dame • Notre Dame, IN 46556*
3. *University of British Columbia • Vancouver, BC V6T1Z3*
4. *Carnegie Mellon University • Pittsburgh, PA 15213*
5. *University of Wisconsin-Madison • Madison, WI 53706*

INTRODUCTION

Computational notebooks are documents that can be read similarly to a textbook section or journal paper, but can be run as computer code. The genesis of these notebooks has come from the continual evolution of computer programming methodology to make programs more comprehensible. This is by no means a new notion and is described by Knuth in his article on "literate programming" from 1984.[1] Traditional computing environments such as MATLAB® and Mathematica®, to name only a few, have evolved to offer interactivity, unique toolboxes, and well-documented help resources. These may be strong arguments to teach with such languages; however, studies contrasting teaching in Python® and MATLAB have generally found Python to be more effective for novice programmers, although there are advantages and disadvantages to each language.[2,3] Proprietary environments also have a significant downside for students due to their expensive licensing fees that may make it difficult for students to apply the codes they have developed after the course ends or to share code with colleagues who do not have a license.

Jupyter Notebook® is a tool introduced in 2015 that advances the notion of narrative in programming.[4] Jupyter's main aims were to create a system to ensure that computational tools can be used in a wide variety of contexts, be created in collaboration, and be easily understood and reproduced. Jupyter offers a web-based interactive computing platform. The web-based nature allows for relatively quick and easy use of these notebooks and programming capabilities; the user needs only a web browser to interact and code. However, all functionality can also be used offline if desired. Jupyter notebooks can combine live code, text, equations, interactive user interfaces, and other rich media. Jupyter supports a number of programming languages, including Python, Julia®, and R®.

Within this article we present five case studies for the use of Jupyter notebooks in the chemical engineering curriculum at a variety of institutions. In each case we describe how the notebooks are implemented and used in the classroom, as well as the impact on students and instructors.

GEORGIA INSTITUTE OF TECHNOLOGY: TEACHING DATA-DRIVEN DECISION-MAKING FOR CHEMICAL PROCESS ENGINEERING

Optimization problems can be found everywhere in engineering, in both industry and academia. However, most chemical engineering programs do not offer a course that discusses optimization specifically in the context of chemical engineering case studies. At the same time, over the past few years we have observed a tremendous increase in interest in data-analytics and machine learning across all areas of chemical engineering.[5,6] In academia, data-analytics is being used to enable or expedite scientific discovery in materials science, pharmaceuticals, process systems engineering, and more. Similarly, industry is entering an era of digitalization that has led to an explosion of chemical, process, and manufacturing and operations data. It is undoubtable that a large fraction of chemical engineering graduates will be faced with design, control, or operations optimization problems that will also involve handling of data sets. As a result, incorporation of such concepts in our curriculum will make our graduates competitive in today's market.

The above tasks (i.e. data-analytics for decision making) require proficiency in computer programming for data pro-

cessing, analysis, visualization, modeling, and optimization. In this case study, a new undergraduate elective course was developed to introduce chemical engineers to various techniques for data-driven decision-making in chemical process engineering, as well as Python programming. The course was taught primarily using Jupyter notebooks. The versatility and flexibility of Jupyter notebooks facilitated the instruction of optimization and machine-learning theory, but also enabled: (i) the students to learn and practice their programming skills in the classroom; (ii) the creation of a very interactive, hands-on lecture where the data and results were visualized; and finally (iii) the incorporation of active-learning group exercises and discussion in the lecture.

The main learning outcomes of this course are (i) understanding of basic theory of linear, nonlinear and mixed-integer optimization; (ii) introduction to sampling, regression, validation and data-reduction techniques; (iii) understanding of how to use data-driven techniques for optimization; and (iv) comprehension of the dangers and ethics of the use of data for decision making. Overall, the course maintained a balance between theory, tools, and applications such that the students obtain key knowledge on how to efficiently formulate their optimization problem, know its mathematical characteristics, select the appropriate software/tool and, if needed, create a customizable tool to solve the given problem at hand. This balance was enabled through the use of open source Python tools for data processing, machine-learning, visualization and optimization and most importantly, Jupyter notebooks.

Each lecture started with the review of the background, theory, and mathematical representation of a specific type of optimization or machine-learning algorithm. This was followed by the loading of a data-set and the formulation of the computer program that employed the theory to the specific data set, concluding with the visualization and discussion of the results. Students followed the lecture through projection of the instructor's screen and their personal laptops, which allowed them to keep notes on their own version of Jupyter Notebook, as well as participate in active learning exercises. In 80% of the lectures, an active-learning exercise was included, during which the students were asked to complete a short assignment in small groups of 2-3 members, followed by an in-class open discussion of the solution.[7] These active-learning exercises ranged from purely conceptual questions to purely computer programming assignments. For example, the students were asked to generate a computer program from scratch that would optimize a function, or, most frequently, they were asked to make a modification to a provided code to obtain the result (e.g. correct an error, fill in a few missing coding lines, change some parameters and observe the sensitivity of the outcome, etc.)

The student feedback on the usefulness and effectiveness of Jupyter notebooks has been very positive. Students found that hands-on programming in class and visualization of the results were crucial towards understanding hard concepts, such as the use of derivatives in searching for an optimum, the representation of data in reduced-dimensions, and the effects of lack of data, noise, and outliers on data-driven modeling and optimization.

At the beginning of the course, almost 70% of the students did not have any prior Python programming experience, and almost 90% of the students had not used Jupyter notebooks before. All of the students had previous programming experience in MATLAB or other programming languages. Three lectures at the beginning of the course were dedicated to tutorials in Python and Jupyter Notebook. One of the challenges pointed out by students with minimal prior Python programming expertise was the steep learning curve of the course, which included learning optimization theory, machine learning, and statistics concepts at the same time as learning to program in a new language. However, none of the students found that it was challenging to learn how to use, read, and modify Jupyter notebooks. In fact, several students recognized that the use of Jupyter notebooks is by itself a skill that can be beneficial in other aspects of their careers. For example, students now use Jupyter notebooks for research projects and research communication within their lab, with industry, and as supplementary material to accompany their publications. Despite the challenge of learning a new programming language, all students passed the course and were able to complete challenging projects using programming in Python.

Overall, through this experience, it became clear that a course on topics that involve modern data analytics, machine learning, realistic data sets, and decision-making was effective mainly because of this new medium that enables the combination of theory, images, programming, and plotting within a single lecture. The instructor plans to make the lectures available as stand-alone teaching modules that can be shared with the community, and this points to another advantage of the use of this flexible teaching medium.

UNIVERSITY OF NOTRE DAME: NUMERICAL METHODS, APPLIED STATISTICS, AND MORE

Over the past few years, Jupyter notebooks have become an integral part of at least eight chemical engineering courses at Notre Dame taught by three faculty:[8]

1. CBE 20255 Introduction to Chemical Engineering Analysis (sophomore, required)[9]
2. CBE 20258 Numerical and Statistical Analysis (sophomore, required) (discussed in this article)
3. CHE 30324 Physical Chemistry for Chemical Engineers (junior, required)[10]

4. CBE 30338 Chemical Process Control (junior, required)[11]
5. CBE 40455 Process Operations (senior, elective)[12]
6. CBE 60499 Nonlinear and Stochastic Optimization (graduate, elective) (discussed in this article)
7. CBE 60547 Computational Chemistry (graduate, elective)[13]
8. CBE 60553 Advanced Chemical Engineering Thermodynamics (graduate, required)[14]

This case study highlights pedagogical advantages and shortcomings of Jupyter notebooks in the context of teaching numerical methods and applied statistics in CBE 20258 and CBE 60499 with Python.

Jupyter notebooks facilitate active learning by co-locating code and diverse outputs (text, error messages, and graphics) into a single, living document. Previously, when teaching CBE 20258 with MATLAB, it was difficult to engage students with examples since code (script), text output (console), and graphics were all on separate windows. Moreover, text (markdown) cells in Jupyter Notebook allow descriptive text, equations (through LaTeX), and embedded images to be easily interspersed between computer code and output. Though the equivalent can also be done in MATLAB using the live editor feature, it is not as intuitive as in Jupyter.

The latest iteration of CBE 20258 is organized with one notebook for each class meeting. Before each class, students are required to (i) read the notebook and any assigned book sections, and (ii) complete brief home activities. The notebook also includes several more complex, often multipart, class activities. During class session, we work through the class activities individually, in partners, or as a large group. While scrolling between class activities in the notebook, we stop to answer questions on the reading and home activities. As needed, 5 to 10-minute mini-lectures are given on important and difficult concepts. Jupyter notebooks are a key enabling technology for this class structure because examples, activities (with detailed instructions), and reading material are integrated into a single living document.

Cloud-based hosting platforms for Jupyter notebooks can eliminate barriers for access, simplify class administration, and further facilitate active learning. CBE 60499 was last taught using Jupyter locally installed on each student's personal laptop. Although Anaconda®, a Python distribution, simplifies the installation process for students, this local installation works best with small classes. During the last two iterations of CBE 20258, we have experimented with two cloud-based Jupyter systems: Google Colaboratory™ and Vocareum™. Both systems allow students to complete all class assignments through a web-browser on any internet connected computer.

Google Colaboratory ("Colab" for short) is a free computing environment built for research. Colab is best described as "Google Docs™ but for code," with similar comments and editing from multiple users. At Notre Dame we shared all class notebooks with students in a read-only Google Drive™ folder. Colab would then prompt students to save a copy of the notebook in their own Google Drive. This facilitates easy sharing, interactive editing, and automated back-ups with revision history. To submit assignments, students would create a shareable URL, add the URL to the top of their notebook, and save the notebook as a PDF.

Vocareum is a hosted pay-for-use Jupyter server built for education. A key advantage of Vocareum is that it integrates directly with learning management systems (LMS) including Sakai. Most importantly, Vocareum supports auto-grading scripts, including the open-source nbgrader platform for Jupyter notebooks. Auto-grading has two main advantages: (i) it gives students instant feedback on their work, and (ii) dramatically reduces the time required to grade the assignment. However, auto-grading also has significant drawbacks in that it can be time-consuming to set up and cannot give qualitative feedback (for example, in assessing figures or code for readability). For CBE 20258, the auto-grader allows for accountability with the home activities before each class, which is not feasible otherwise due to time constraints. Although Vocareum allows for manual grading of Jupyter notebooks, the interface is cumbersome. Instead, students are required to submit both (i) a notebook to Vocareum for auto-grading sections, and (ii) a PDF printout to Gradescope™ to assess coding style (comments, etc.), figures, and pseudocode or model derivations. In summary, neither Colab or Vocareum is perfect. Colab offers more flexible sharing and is free, but Vocareum has education-focused features, including LMS integration and auto-grading that can greatly streamline classes.

UNIVERSITY OF BRITISH COLUMBIA: INTERACTIVE NOTEBOOKS IN REACTOR DESIGN AND PROCESS CONTROL

At the University of British Columbia (UBC), Reactor Design (CHBE 355) and Process Control (CHBE 356) are junior-level chemical engineering courses taken in the same term that require significant mathematical and computational backgrounds. The reactor design course focuses on homogeneous chemical reactor design and modelling. The process control course focuses on modelling chemical processes in order to design effective control strategies. Engineering undergraduates with limited mathematical knowledge and programming skills have the most difficulty with these courses. Engineers working in industry often need to solve poorly-defined problems that require proficiency with advanced computational and design tools. Engineering graduates who lack design and programming experience can face difficulty in the job market

as they have little exposure to these types of problems.

Guzman et al. highlighted the use of interactive tools to encourage active participation and facilitate student-focused learning in engineering education.[15] These tools allow students to understand difficult mathematical concepts through manipulatable figures. Students can develop a comprehensive understanding of course materials and obtain design experience through interactive tools and integrated projects that requires content from multiple courses to solve. In order to develop students' design, analysis, and problem-solving skills, computational tools for these two courses using Python were introduced through Jupyter notebooks. Figure 1 shows a screenshot from a Jupyter notebook from the process control course.

Syzygy is an online Jupyter Notebook hosting service offered by the Pacific Institute of Mathematical Sciences, Compute Canada, and Cybera to researchers and educators across Canada. This tool is free for instructors, students, and researchers and allows notebooks to be easily run and shared. During tutorial sessions, the teaching assistant leading the tutorial would work through a notebook with their screen mirrored on a projector as students worked in their own copy of a Jupyter notebook. The communication tool Slack™ was used as a discussion board to facilitate student question during the tutorial period.

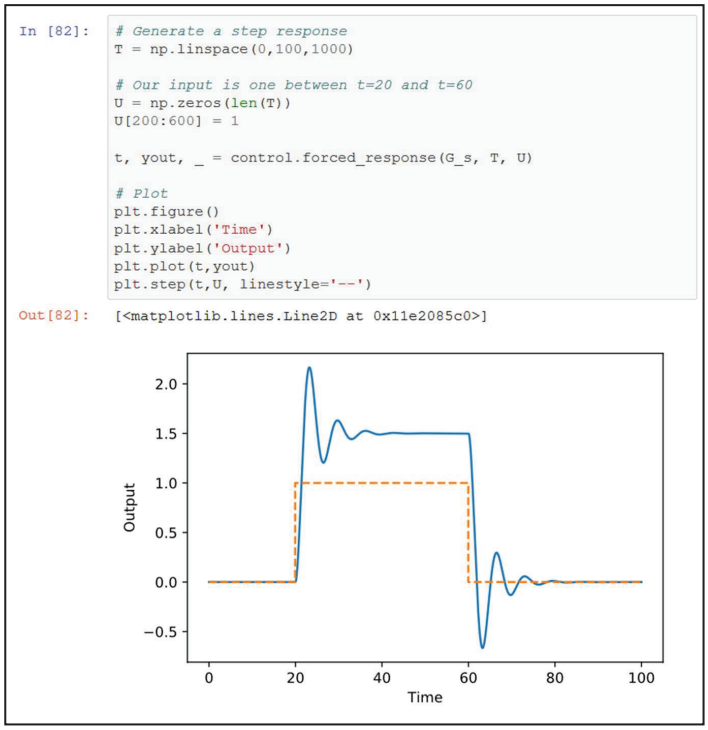

Figure 1. *A screenshot from a Jupyter notebook from the process control course showcasing coding input that can be manipulated and corresponding graphical output for a step change in a control process.*

Students in these courses had prior experience with MATLAB, but were assumed to have no prior experience with Python. In order to introduce students to Python and its application in solving a variety of mathematical problems, six tutorial notebooks were created to introduce students to relevant functions for solving a variety of mathematical systems. These were used in tutorials in both courses during the first third of the course. The following six tutorials in each course then focused on the use of computational tools to solve relevant problems. In addition to notebooks used in the tutorials, students were given take home assignments (two in reactor design and one in process control) and one final project in each course. The final projects applied the skills students had learned in previous tutorials to address a complex problem in an industrial context. All of these tutorial notebooks can be accessed online through a GitHub™ repository.[16]

A survey was run at the end of the courses focusing on student experience in the final projects. Responses, found in Table 1, were received from 19 students out of the roughly 120 unique students in both classes (note that about 95% of students are common between the two courses). This low turnout was perhaps due to having no incentive for survey completion as well as the survey being at the end of the term

TABLE 1
Student survey results from reactor design and process control Jupyter-based terms projects.

Survey Question	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
The project enhanced my learning of the course material. I can see the relevance of the project to the course material.	4	7	4	4	0
The project content was interesting.	3	6	8	2	0
I can see the benefit of learning programming and Python to my future career.	11	5	3	0	0

when students were busy with other projects and studying for final exams. Given the small number of responses, no overall conclusion can be reached, but the students who did respond indicated they saw the benefit to their future careers of learning programming. Open comments from students at the end of the survey point to the range of background programming abilities of students. Many comments focused on the pace of the tutorials and programming content, with some saying the pace was too fast while others said the opposite.

CARNEGIE MELLON UNIVERSITY: INTEGRATING JUPYTER INTO THE MASTER OF SCIENCE PROGRAM

Jupyter and related methods have been used extensively in the Master of Science (MS) program in the Department of Chemical Engineering at Carnegie Mellon University (CMU). The MS program was designed to give chemical engineering students specialization in numerical methods and computational skills to tackle more complicated and realistic problems in industrial settings. The students come from diverse educational backgrounds. Most of these students have ChE undergraduate degrees, but background varies and includes large US research universities, small liberal arts colleges, and foreign universities including China, India, and Japan (among others). Most undergraduate programs integrate a small amount of programming exercises into the core curriculum, typically as MATLAB assignments and labs. Informal polls of this population of students indicate that about 10% have no programming experience at all from their undergraduate curriculum, and less than half have been introduced to Python. This diversity of backgrounds presents a challenge to teaching a course completely in Python, but by the end of their first semester with two Python-intensive courses all students are able to solve technical numerical programming problems.

Although chemical engineering instruction with Jupyter is quite recent, similar approaches have been used for some time at CMU. Two of the first-semester core classes in the MS program are taught entirely with computational methods, including a numerical methods course (06-623) and a reactor engineering course (06-625). The format for both courses was pioneered and developed by Prof. John Kitchin, who has described extensively how Emacs' org-mode can be used for education.[17] Emacs is a powerful open source text editor, and org-mode is a package that can be run within Emacs. Org-mode allows a number of features, including a simple document markup syntax, the capability to embed interactive code and data in a document, and the capability to export a document into another format such as PDF or LaTeX. From 2015 to 2017, 06-625 was taught with all lecture notes, homework, and exams using the org-mode structure. In 2017, Prof. Zack Ulissi transitioned to Jupyter notebooks based on their increasing popularity and relevance to other fields. In 2018 and 2019, both Fall core classes (06-623 and 06-625) were taught in the Jupyter format. Lecture notes, assignments, and exams are available for 06-623.[18] This approach was also adopted for one semester for the undergraduate reaction engineering course, 06-634. Email surveys of students one semester after taking these courses indicate that 80% of the students are still using Jupyter in their research/classes, suggesting that the skills are transferable to broader chemical engineering challenges.

Course Format

06-623 and 06-625 at CMU are relatively unique among numerical methods courses in that nearly 100% of in-class lectures, exercises, homework, and exams are in the form of Jupyter notebooks. Lectures are given by projecting the instructor's laptop screen with the interactive notebook. Lecture notes are distributed before class, and students are expected to bring their own laptops and follow along. This format is interactive as the instructor can develop Python solutions to engineering problems as a live demonstration, can adjust and interact with existing solutions (e.g. change initial conditions or tolerances and observe the effect on solutions), and answer student questions about the methods or ideas with additional examples. All homework and exams are distributed in the form of Jupyter notebooks, and students complete the notebooks and submit them as the notebook file and a PDF. This significantly simplifies the evaluation process since the exams/homeworks are in exactly the same format as the daily lectures.

Intro to Python On-Ramp

In the context of a one-semester course for students using Python and Jupyter for the first time, the on-ramp process is extremely important. Various strategies that have been used in teaching 06-623 and 06-625 include: (i) introduction through the first two weeks of class (with in-class demonstrations and exercises); (ii) use of tailored on-line Python materials as out-of-class exercises in week 1, and (iii) a short half-day Python-intensive workshop before the first week of class. Option (ii) was implemented using the Open Learning Inititative (OLI) system at CMU with help from the Eberly Center and a Wimmer teaching fellowship for Prof. Ulissi. Option (iii) was implemented as a single 5-hour session using the established Software Carpentry "Introduction to Programming" module the week before classes started.[19] In all cases, nearly all students were able to complete the courses, and most students were comfortable with the format by the end of the third week of lectures. Self-reported homework and out-of-class study times in the first two weeks were considerably lower with options (ii)/(iii) instead of (i), suggesting that these methods helped with the on-ramp experience.

Grading

Grading of Jupyter notebooks is still imperfect. Distributing and collecting notebooks can be performed with some packages such as nbgrader, but this may require specific code structure depending on how it is implemented. LMS integration of nbgrader is dependent on the Jupyter platform and LMS being used. Furthermore, setting up standard/repeated feedback (e.g. different students making the same mistakes) in nbgrader can be time consuming. The choice to use nbgrader will likely depend on class size and whether assignments can be re-used over multiple years. Prof. Ulissi currently uses Gradescope to grade the notebooks submitted as PDFs. Students simply print their notebook to PDF and upload it to Gradescope. Note that the default Jupyter to PDF conversion, that works via pdflatex, is less favorable as equation errors in the markdown cells, which often display correctly within Jupyter, frequently break the pdflatex conversion. Teaching assistants (TAs) then use the Gradescope interface to quickly grade the assignment as they would a normal engineering homework. Students are also expected to submit a copy of the Juptyer notebook in case the TAs wish to manually check the solution/code or for plagiarism detection (detailed below).

Plagiarism

An obvious concern in programming-heavy courses is the possibility of plagiarism. These challenges are not unique to chemical engineering or Jupyter notebooks. Since students submit copies of their notebooks, standard methods such as Measure Of Software Similarity (MOSS) can be used.[20] Collected Jupyter notebooks are batch-converted to Python scripts, which are then submitted to MOSS for analysis. This process is not perfect but quickly flags the most flagrant violations. A secondary benefit of this approach is that this process also identifies students who copied and modified particularly relevant examples from the lecture notes. While not academically dishonest if disclosed, this method of solution is less desirable and often correlated with students who struggle to creatively solve engineering problems in formats they have not seen before. Finally, with an exam format of every student working on a laptop and submitting their own notebook, it is very difficult to guarantee students are not messaging each other with solutions. Standard approaches (random seating, TAs roaming the room, etc.) can partially address this challenge.

Installation/Usage

Most students do not have experience installing and using Python and Jupyter notebooks. We have used pre-installed anaconda installations, docker images, and on-line hosted instances such as Google Colab or Microsoft Azure Notebooks™. All were sufficient for the needs of the classes, but the hosted solutions have had the fewest technical difficulties. Prof. Kichin has developed a flask web app interface (Techela) to Jupyter notebook distribution and submission that largely hides many of these difficulties if a local Python installation is used.[21]

UNIVERSITY OF WISCONSIN-MADISON: TEACHING ADVANCED OPTIMIZATION TECHNIQUES TO UNDERGRADUATE STUDENTS USING JULIA AND JUPYTER NOTEBOOKS

Optimization has traditionally been a difficult topic to teach to undergraduate students. The mathematics behind it (and associated software tools) are rather abstract and sophisticated. Commercial optimization tools (e.g. AMPL®, AIMMS®, GAMS®) are computationally powerful, but they are targeted towards graduate-level students and industrial practitioners. These tools also have proprietary syntax and stand-alone implementations that complicate integration with other tools (e.g. visualization). On the other hand, MATLAB (the most popular scientific computing package for undergraduate education) provides a flexible environment, but its optimization tools are computationally inefficient and difficult to use (e.g. translating an optimization problem into matrix-vector form and/or computing derivatives is difficult). As a result, instructors who teach optimization using such tools can spend significant amounts of time (weeks to months) teaching students how to use software tools, as opposed to teaching students how to properly formulate the problem at hand and analyze the results. Moreover, students tend to lose motivation if they feel that the tools learned cannot solve industrially-relevant problems.

Recent developments in open-source modeling languages such as Pyomo (in Python) and JuMP (in Julia) have dramatically changed the landscape of optimization software. In particular, these tools are much more accessible to students.[22,23] Pyomo and JuMP also largely benefit from the fact that they reside in a flexible computing environment (as with MATLAB), which facilitates the use of supporting tools (e.g. linear algebra, data, statistics, visualization) in the modeling and analysis process. Such tools are also under active development by the open-source Julia and Python communities.

Jupyter notebooks are a premier example of how open-source software benefits undergraduate education. At UW-Madison we have been using JuMP and Jupyter notebooks to teach optimization to undergraduate students. In the Chemical and Biological Engineering Department, Jupyter notebooks are being used to teach the senior Process Design course (CBE 450), while in the Electrical and Computer Engineering Department, Jupyter notebooks are being used to teach the Introduction to Optimization course (CS/ECE/ISyE 524). In CBE 450 the instructor spends a total of 4 hours teaching students the basics of optimization modeling

and implementations in JuMP (the students have no previous background in optimization). This is possible because the syntax of JuMP is compact and intuitive (the software implementation of a model looks similar to that on paper). For instance, a simple but powerful feature of JuMP is the ability to express variables as unicode symbols, which allows students to define variables using letters from the Greek alphabet. JuMP also incorporates set notation and allows indexes to take names (e.g. chemical formulas to identify components). Jupyter notebooks provided to the students are executed in the public portal https://www.juliabox.com (students do not have to install new software in their computers).

Learning JuMP allows students to tackle complex optimization problems rather easily. For instance, we teach them how to optimize a flowsheet, how to design water and heat recovery networks, how to design compressor trains, and how to conduct equilibrium calculations. These examples are non-trivial (e.g. involve difficult sets of nonlinear equations) and are sufficient for students to start comprehending the concepts of objective functions, constraints, trade-offs, and nonlinearity as well as to understand the power of modern optimization tools. Specifically, JuMP is not an educational package but is in fact actively used in academic and industrial research (JuMP can tackle problems with millions of variables).[24] The objective is thus to help students appreciate that the tools learned in their undergraduate education do solve real problems.

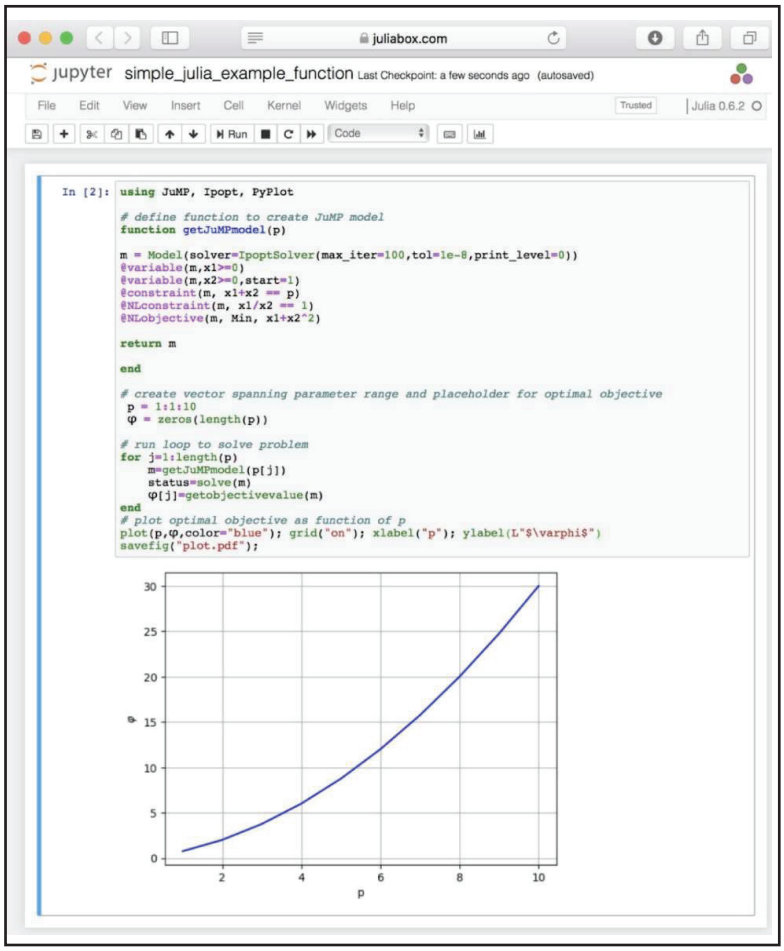

Figure 2. *Snapshot of Jupyter notebook implementing sensitivity analysis for a simple optimization problem.*

It is important to emphasize that all of these benefits are largely enabled by Jupyter notebooks. Implementing JuMP models as plain Julia scripts, while doable, does not provide an interactive experience and can be frustrating to students who are not familiar with the language. Specifically, Jupyter allows students to experiment with their code and quickly see the outcomes (i.e. execute pieces of code on-the-fly). This feature is particularly important when building and debugging complex optimization models. Moreover, students are asked to submit their homework assignments as Jupyter notebooks, and this teaches them how to properly document and share code. Jupyter notebooks also enable students to learn and appreciate the benefits of LaTeX (an advanced type-setting system). The use of Jupyter notebooks also help students understand the computational workflow behind their code (an insight that gets lost when executing a script all-at-once). The use of embedded visualization tools in notebooks has also proven to be of high value, as most students prefer to visualize data and like to perform sensitivity analyses (see Figure 2). Notebooks thus provide a natural segue to the use of data analysis, uncertainty quantification, and visualization techniques.

CONCLUSIONS

Computational notebooks are powerful and versatile tools that can be used at a variety of instructional levels. They have become relatively easy to implement and can be leveraged to help students understand physical, chemical, and biological phenomena that chemical engineers deal with on a daily basis. Furthermore, their use helps promote data and programming literacy, which is becoming increasingly important for all engineering disciplines.

ACKNOWLEDGMENTS

The authors would like to acknowledge the financial support provided by Computer Aids in Chemical Engineering (CACHE) for faculty attending the CACHE 50th Anniversary meeting.

REFERENCES

1. Knuth DE (1984) Literate programming. *The Computer Journal* 27(2): 97–111. https://doi.org/10.1093/comjnl/27.2.97
2. Fangohr H (2004) A Comparison of C, MATLAB, and Python as teaching languages in engineering. *International Conference on Computational Science* 1210–1217.
3. Colliau T, Rogers G, Hughes Z and Ozgur C (2017) MatLab vs. Python vs. R. *Journal of Data Science* 15(3): 355-372.
4. Perez F and Granger BE (2015) Project Jupyter: Computational narratives as the engine of collaborative data science.
5. Chiang L, Lu B and Castillo I (2017) Big data analytics in chemical engineering. *Annual Review of Chemical and Biomolecular Engineering* 8: 63–85. https://doi.org/10.1146/annurev-chembioeng-060816-101555
6. Qin SJ (2014) Process data analytics in the era of big data. *AIChE J*. 60(9): 3092–3100. https://doi.org/10.1002/aic.14523
7. Felder RM and Brent R (2009) Active learning: An introduction. *ASQ Higher Education Brief* 2(4): 1-5.
8. Kantor J (2019) Jupyter Notebooks for ChE education. *CAChE Summer 2019 Newsletter* available at https://cache.org/summer-2019-newsletter
9. Kantor J (2019) CBE20255: Introduction to chemical engineering analysis. https://github.com/jckantor/CBE20255 accessed September 14, 2019.
10. Schneider WF (2019) CHE 30324: Physical chemistry for chemical engineers. https://github.com/wmfschneider/CHE30324 accessed September 14, 2019.
11. Kantor J (2019) CBE 30338: Chemical process control. http://jckantor.github.io/CBE30338/ accessed April 18,2020.
12. Kantor J (2019) CBE 40455: Process operations. https://github.com/jckantor/CBE40455 accessed September 14, 2019.
13. Schneider WF (2019) CBE 60547: Computational chemistry. https://github.com/wmfschneider/CBE60547 accessed September 14, 2019.
14. Schneider WF (2017) CBE 60553: Advanced chemical engineering thermodynamics https://github.com/wmfschneider/CBE60553 accessed September 14, 2019.
15. Guzmán JL, Åström KJ, Dormindo S, Hägglund T and Piguet Y (2008) Interactive learning modules for PID control. *IEEE Control Syst Mag* 28(5): 118-134. https://doi.org/10.1109/MCS.2008.927332
16. Triandafilidi V, Tsai Y, Lim S, Lo NT, Kritharis A, Ioannidis N, Shen EQ and Verrett J (2019) CHBE 356 & CHBE 355. https://github.com/OpenChemE accessed September 14, 2019.
17. Kitchin JR (2015) Examples of effective data sharing in scientific publishing. *ACS Catal*. 5(6): 3894-3899. https://doi.org/10.1021/acscatal.5b00538
18. Kitchin JR (2019) f19-06623. https://github.com/jkitchin/f19-06623 accessed September 14, 2019.
19. The Carpentries (2019) Programming with python. https://swcarpentry.github.io/python-novice-inflammation/ accessed September 14, 2019.
20. Schleimer S, Wilkerson DS and Aiken A (2003) Winnowing: Local algorithms for document fingerprinting. *Proceedings 2003 ACM SIGMOD international conference on management of data*, 76-85. https://doi.org/10.1145/872757.872770
21. Kitchin JR (2019) techela. https://github.com/jkitchin/techela accessed September 14, 2019.
22. Dunning I, Huchette J and Lubin M (2017) JuMP: A modeling language for mathematical optimization. *SIAM Rev*. 59(2): 295-320. https://doi.org/10.1137/15M1020575
23. Bezanson J, Edelman A, Karpinski S, and Shah VB (2017) Julia: A fresh approach to numerical computing. *SIAM Rev*. 59(1): 65-98. https://doi.org/10.1137/141000671
24. Jalving J, Cao Y and Zavala VM (2019) Graph-based modeling and simulation of complex systems. *Comput. Chem. Eng*. 125: 134-154. https://doi.org/10.1016/j.compchemeng.2019.03.009 ❐