Computational Modeling in Introductory Physics Courses and Across the Curriculum

Dr. Todd Zimmerman, University of Wisconsin - Stout

Todd Zimmerman is an associate professor of physics in the Department of Chemistry and Physics at the University of Wisconsin - Stout. He is also the editor-in-chief of the Partnership for Integrating Computation Into the Undergraduate Curriculum.

Computational Modeling in Introductory Physics Courses and Across the Curriculum

Todd A. Zimmerman
Department of Chemistry and Physics
University of Wisconsin - Stout, Menomonie WI, 54751

Abstract
Modern research and design are carried out using the three-pillars of theory, experiment, and computational modeling but many courses are still taught focusing on just theory and experiment. To rectify this mismatch, we have introduced a series of 8 computational modeling activities in a second semester introductory course on electricity and magnetism. A major hurdle to adding a computational component to introductory courses is that students enter with a wide variety of computational experience. Methods for dealing with differing skill levels will be discussed. Glowscript, an implementation of Python that runs in a browser window with no installation, is used to create 3D visualizations of electric and magnetic fields and to animate the motion of particles in the associated fields. Emphasis is placed on applying problem solving strategies to creating computational models and evaluating the output of the models, with a focus on developing computational thinking skills. It is important to create activities that focus on creating a computer model and involve minimal computer programming – the goals of the activity should highlight the model and not the programming syntax. In order for students to maintain the skills they learn as well as to understand the importance of computational thinking, modeling activities must be incorporated across a variety of courses. Our efforts to infuse computational modeling across the physics and engineering curriculum will be covered. Issues incorporating computational modeling across the curriculum will also be discussed.

1 Introduction

Modern scientific research relies on three equally important tools; theory, experiment, and computational modeling[1]. Despite the importance of computational modeling, and while the number of departments incorporating computation into homework and projects has grown, few show consistent use of computation in active engagement in the classroom or exams[2].

The American Physics Society, in conjunction with the American Association of Physics Teachers released a report titled ”Phys21- Preparing Physics Students for 21st Century Careers”[3], looking at what skills our students need to succeed. Although this report focused on physics graduates, it is relevant to engineers since a large fraction (43%) of the students in physics end up in engineering-related fields or non-physics STEM fields. The J-TUPP report found that physics graduates and their employers felt a need for a wider and deeper understanding of computational analysis tools.

Recent research on the effectiveness of traditional physics labs in reinforcing mastery of physics concepts has shown labs do not aid student understanding [4] [5]. Instead, the researchers suggest focusing on scientific skills and ways of thinking that the lab time is uniquely suited for [6]. We decided to focus on creation and interpretation of computational models, collecting and analyzing data, scientific communication, and use of typical equipment such as soldering irons, digital multimeters, and oscilloscopes. Since understanding multimeters and oscilloscopes comes after the introduction of electric potential, this leaves time early in the semester that can focus on computational modeling. Discussion periods can be spent on aspects of computational modeling as well.

University Physics II is a 5-credit introductory electricity and magnetism course with two semesters of calculus as a prerequisite. The students are primarily engineering students who have taken a semester of Statics and a semester of Dynamics, although a smaller fraction of students have taken an introductory classical mechanics course instead. The course meets face-to-face with three hours of lecture each week and a two hour lab and two hour discussion. Since labs and discussions are held in a lab room, there is flexibility as to the types of activities that can be done during lab or discussion times. Several of our courses operate as flipped classrooms, where first exposure to the material occurs outside the class and lecture time is spent with students answering questions and solving problems. This allows more flexibility for computational modeling during some discussion periods.

The University of Wisconsin - Stout provides laptop computers to all students. This provides us with a great deal of flexibility since we don't need to schedule computational activities in computer labs and can use the physics labs. Sections are 24 students and they are organized into groups of 3 for lab, discussion, and computational modeling activities.

This paper will discuss how computer modeling was adding into this course. We discuss an important distinction between programming and modeling, and how we tie traditional homework problems to show the similarity to modeling. We describe how we avoided dealing with installing VPython on student computers and how we deal with the different levels of computer programing incoming students have. Grading is an important part of motivating students to take modeling seriously and this is discussed. Examples of some computational activities are outlined. Finally, some issues related to integrating computation across the curriculum is discussed.

2 Computational Modeling in the Physics Classroom

Modeling vs. Programming

A common complaint from students is they didn't sign up for a programming class. To this end it is important that the course does not turn into a programming class. Minimal working code is provided to students for all activities. A minimally working program is one that will run as provided but must have a few lines added or changed to provide the modeled behavior desired. For example, in an activity on drawing the electric field vectors of a point charge, code drawing a ball representing a sphere and a series of arrows with fixed directions at several locations are provided. Students must add code to calculate the electric field at the location of the arrows.

When students are confronted with a programming-related challenge, such as syntax errors or other misunderstandings that are related to confusion about coding and not about physics content, students are provided with the answer immediately. Rather than using Socratic questioning for

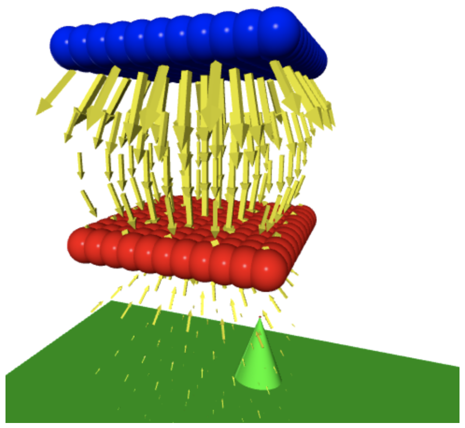

Figure 1: Example VPython Output

programming issues, they are given the answers. This reduces their level of frustration with modeling.

When solving standard pen-and-paper homework problems in class, students are encouraged to use a problem solving format of sketching the problem, drawing and labeling vectors, then listing their known and unknown quantities along with any equations they might need. This same format is used with the computational modeling activity. With a traditional homework problem students would go on to manipulate equations to get the correct result, then evaluate the validity of their answer. With computational modeling, the only difference is during the manipulation of equations the student should let the computer do most of the work. Students write out the first part of the problem before they start modifying code. Students are encouraged to transition from a mathematical description to a code model by rewriting the equations in terms of variable names as pseudo-code before typing anything into the computer. The questions that make up the activity then ask students to reflect on their results, which takes the place of the evaluation step when working homework problems.

Great pains are taken to refer to these activities as computational modeling activities. Additionally, it is made clear to students that computer programming is focused on how to structure the components of a program while the computational modeling involves making changes to working code and drawing conclusions from the output of the model.

Questions provided with each activity guide students through the process of making sense of the graphical output from the model. They are also asked to compare what they are seeing on the screen with what they expect based on the equations and what they learned in lecture.

Connections are made between numerical integration techniques and analytical results. For example, even though students are not expected to be able to code a for-loop by themselves, they are expected to understand how a for-loop can be used to calculate the electric field of a charge distribution. By breaking a distribution into a set of discrete charged pieces, finding the field of each piece, then adding up the resulting fields, students see that this sort of computational approach can be used in many situations throughout the course.

Logistical Considerations
Python was chosen as the language because of the author's familiarity with the language, the fact that the textbook used for the course uses Python, and the fact that several faculty in the engineering and mathematics departments make use of Python in their courses. A module called VPython was developed by the textbook authors to aid in visualizing and animating objects in motion[7]. Glowscript is an implementation of VPython that can run in the browser. The terms Glowscript and VPython are frequently treated as synonyms.

All of our students are provided with laptops as part of their tuition so we looked for something that was easy to run with minimal installation. Glowscript.org and trinket.io are two websites that allow VPython to run in browsers with no installation. We have opted for Trinket because the paid version allows me to create web pages to write up instructions along side the code that can be run on the page.

Although Trinket has an option to collect assignments and give feedback, we have students submit a link to their code through our learning management system (LMS), along with answers to the questions in the activity.

In the past I have put students into two-person groups for computational modeling activity. Pair programming is fairly common in computer science courses and even in the software development industry[8]. In the class, one student is responsible for typing while the other student provides guidance and feedback. Every thirty minutes or so the students are asked to switch roles, to make sure one student does not dominate the activity. This proved to be problematic for a couple of reasons. Students were placed in three-person groups for lab and discussion activities, while switching to two-person groups for programming, leading to some confusion about which groups to get into and who was in what group. Additionally, if a single student is absent it significantly hampers the progress of the other student.

An ideal size for lab groups and group problem solving is three to four students [9] so we have groups of three for non-computational activities. This has proved to be confusing to students and, when a single student is missing from a group, results in lone students trying to work through the computational activities. In the future I will use the same three-person groups for lab, discussion, and computational modeling.

Differing Levels of Computer Programming
Students coming into the class have a wide array of coding skills. Some have had three or more semesters of computer science courses, while some struggle with navigating the directory structure of their computer. The first semester introductory physics course includes VPython activities, but engineers do not take this course. Instead they take a semester of Statics and a second semester of Dynamics. Computer modeling activities have not been widely implemented in the Statics and Dynamics course at this time.

A number of steps are taken to lower the entry barrier for students with little background. The first is to run VPython through the browser. Having to download, install, and configure, and troubleshoot a VPython installation can be a surprisingly large hurdle, especially with larger classes. Coding in the browser has grown with the proliferation of on-line computer programming courses. Glowscript.org and Trinket.io allow running VPython in browsers.

Next, students are given minimally working code, which can be run without making any changes but doesn't yet correctly model the desired behavior. The main coding task for the student is converting a mathematical equation into something the computer can understand. A ”Getting Started” page is included with all activities, which gives background on any new commands that are used, as well as places to practice modifying code. All algorithms are included in pseudo-code, which is easier for students without significant coding to understand.

Students work in groups of three, which increases the likelihood that one or more students will have some programming. In smaller classes it is even possible to poll students for their programming background and form groups based on that information. This has been used in upper level physics courses but not in the introductory courses.

An important step was to give students direct answers to problems related to coding and syntax and not try to guide them to the answer. Rather than use a Socratic method to help students fix their errors, giving clear solutions and sometimes stepping in and typing in corrections reduces student frustration with the coding. The goal of the activities is not that they get the syntax correct, merely that they can understand what is going on.

The ”Getting Started” pages frequently include common error messages that past students have encountered. Trinket code boxes are included with incorrect code and students are encouraged to correct the errors.

Grading Activities
Since the point of these activities is to develop modeling skills and not programming skills, the grade focuses on evaluating the output of the model. Frequently this is a 3D image or animation involving moving charges and vector arrows to indicate various quantities. Part of the grade is based on whether the code runs or not. This is primarily for the instructor's benefit - to cut down on time spent tracking down bugs. This makes up 50% of the first activity to incentivize students to make sure things work before turning it in, but the fraction decreases over the semester to only 10% of the activity grade by the second half of the semester. Grading is also greatly simplified if the instructor does not have to spend time debugging the activity.

Roughly 30% of the grade is based on correctness of the computational model. Although this can be done by running the code, the easiest means of assessing this is to ask students to include screen-shots of the VPython output. Not only does this speed up grading, it also insures you are looking at the same results that the students drew conclusions from. With students modifying code and changing physical parameters, there is no insurance that the code they turn in will replicate the results they had earlier in the activity.

The remaining portion of the grade is based on answering a series of questions correctly based on their model output, as well as drawing connections between their model and what they had learned in class. For example, one of the first activities has them modeling a storm cloud as a negative point charge and drawing the electric field vector at various points around a landscape. Students are asked to use Coulomb's law to explain why the direction and magnitude of the electric field at different locations always points towards the cloud and decreases with distance from the cloud.

Questions relating to the computational modeling are included on exams. The questions typically fall into one of two categories. The first category is interpreting the output of the model. These questions present an image similar to one of the activities in class and ask students to describe what is happening in the image or to explain why the image is wrong. The second category of question related to reading code and tying parts of the code back to some physics idea or equation. For example, the student might be asked to explain how a block of code breaks a distribution of charge up into small parts and adds up the fields of each part, or to identify whether a particular equation in the code is correctly used or not. By including questions on the exam it gives the computational activities more importance in the class and gives students a stronger incentive to study the activities.

To further incentivize use of VPython, students are allowed to use VPython as their calculator on exams. Students can use VPython inside a lockdown browser during the exam. A quiz is created in the LMS, with an embedded trinket-code box on the page. Students submit their code through the quiz in case questions arise during grading about how they got their answer. The lockdown browser limits them to one page on the Trinket website and prevents them from opening other applications on their computer during the exam. They are also encouraged to use VPython to complete homework problems.

A friend of yours has just completed the first discussion activity where a storm cloud is modeled as a negative point charge with q = 200 C a height of 1000 m directly over your position. They show you their computer screen and you notice something can't be right with their model. Describe in detail how you know their results are wrong. Include an explanation of what you would expect to see instead. Feel free to include a sketch.

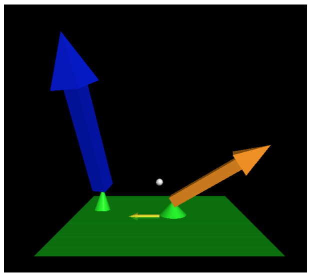

Figure 2: Example Exam Question

3 Examples of Computational Modeling Activities

The computational modeling activities are clustered towards the first half of the semester, prior to the introduction of potential difference. Once potential difference is introduced there are several lab activities that can take place. These activities focus on helping students visualize the electric and magnetic fields and the impact they have on particle trajectories.

Examples of the activities can be found on the Trinket website[10]. All activities are designed to be completed in a 2-hour period. Each activity includes a "Getting Started" page that covers the background of any new VPython commands that are introduced in the activity. Students are not expected to use most of the commands, but the background is provided for more ambitious students.

Each activity has two or three separate code boxes on each page, with students progressively adding more features as they progress through the activity. Each code box either includes a minimally working program which needs to be modified or a blank space to copy and paste previous code for further modification. A series of questions are provided that students answer in a separate Word document and turn in via an LMS. Five of the eight activities are context-rich problems involving roll-to-roll manufacturing, storm clouds, velocity selectors, and mass spectrometers. The activities I am currently using are:

- Computation as a Calculator
- Roll-to-Roll Manufacturing
- Electric Field of a Dipole
- Cloudy Line of Charge
- A Better Cloud Model
- Numerical Integration of Potential Differences
- Mass Spectrometer
- Velocity Selectors

The first activity students complete is using VPython as a calculator. They are assigned a couple of problems that involve vectors through the on-line homework system. This gets them familiar with the interface and basic syntax while also demonstrating how easy solving vector problems can be using VPython. As previously mentioned, students are allowed to use VPython on exams.

The next four activity involves calculating and drawing electric field vectors near different shapes of charge distributions. Drawing arrows using VPython is as simple as specifying where an arrow is and which way it points. The primary goal of these activities is to get students to correctly calculate the distance vectors needed to find the electric field of a point charge, as well as to see how the electric field of a distributions of charges can be found by summing the contribution from small parts of the distribution.

The activity on numerical integration of potential differences has students calculating the electric field from a distribution of charges and adding up contributions to potential difference as a result of the changing fields. It is meant to help students see the connection between fields and potential differences.

The last two activities deal with particles moving in a magnetic field. Many students are surprised to see charged particles following spiral trajectories even though they can readily state that the magnetic force is perpendicular to the velocity and magnetic field. The final activity shows how crossed electric and magnetic fields can be used as a velocity selector for charged particles.

4 Integration Across the Curriculum

To insure students continue to develop their modeling skills, and to show the usefulness of modeling, Python modeling has been incorporated into a number of courses. The other faculty teaching the upper level courses all see the value of computational modeling and fortunately all have been willing to use Python. VPython is integrated into first semester introductory physics, although engineering students take a Statics course and a Dynamics course in place of the first semester physics.

Several upper level mathematics courses make use of Python, such as courses in numerical analysis, cryptography, and machine learning. Some of the engineering courses, such as Introduction to Numerical Methods in Engineering, do use Python as well.

To further integrate computer modeling in general and Python in particular across the curriculum, a Python users group was formed with faculty from mathematics, engineering, physics, and biology. Meeting roughly once a semester, faculty share techniques and tips about incorporating Python into courses. Topics include the use of Jupyter notebooks, setting up a Jupyter Hub for a class, and how to grade modeling activities. These meetings are also a time for faculty to discuss how they incorporate computer modeling into classes so that instructors of later courses know what skills they can expect their incoming students to have.

Since students often do not take courses in the same sequence, it is hard to count on students having seen a particular modeling topic. This does limit the integration of modeling. Instead of relying on them having specific skills, instructors can count on a significant fraction of students having familiarity with Python.

Instructor turn-over is another hurdle. New instructors may not be interested in including computation or may have their hands full just preparing for the normal content. Faculty use of computation in their research with students and faculty belief in the benefits of computational modeling are the highest predictor of whether faculty will incorporate computation into their classes[11] while departmental norms are not. This indicates that seeking to hire new faculty that value computation is a good way to infuse it throughout the curriculum. Sharing activities, especially ”Getting started”-type activities can ease new faculty into integrating modeling.

5 Discussion

Asking students to learn how to program is not a requirement of most of our courses. To that end, the focus of any computer-related assignments should be on creation and interpretation of the models. Giving students skeleton programs that they can modify lowers the barrier for students. Additionally, use of browser-based programming environments like those at Glowscript.org or Trinket.io means not having to install software which aids students in getting started.

Students come into the course with different levels of programming skills. The activities are designed to focus more on making modifications to working code and scaffolding is provided to deal with common syntax or other coding-related errors. Having students connect the visual outputs from the model to what has been discussed in class helps them find errors in their code as well.

Grading of activities focuses on the code running without errors, use of the correct equation when modeling a system, and correctly answering questions related to tying the output of the model to what has been learned in class. By including modeling questions on the exam, it demonstrates that the modeling is an important part of the class.

We have managed to implement Python activities in a number of courses across the math, engineering, and physics curriculum. An on-campus users group meets regularly to share tips and to coordinate what we are doing in various classes.

References

[1] M. Caballero, “Integrating computation into core physics courses: A perspective from a research-intensive university,” *Bulletin of the American Physical Society*, vol. 64, 2019.

[2] M. D. Caballero and L. Merner, “Prevalence and nature of computational instruction in undergraduate physics programs across the united states,” *Physical Review Physics Education Research*, vol. 14, no. 2, p. 020129, 2018.

[3] D. MacIsaac, “APS-AAPT Joint Task Force on Undergraduate Physics Programs (J-TUPP) report released,” *The Physics Teacher*, vol. 55, no. 3, pp. 190–190, 2017.

[4] C. Wieman and N. Holmes, “Measuring the impact of an instructional laboratory on the learning of introductory physics,” *American Journal of Physics*, vol. 83, no. 11, pp. 972–978, 2015.

[5] N. Holmes, J. Olsen, J. L. Thomas, and C. E. Wieman, “Value added or misattributed? A multi-institution study on the educational benefit of labs for reinforcing physics content,” *Physical Review Physics Education Research*, vol. 13, no. 1, p. 010129, 2017.

[6] N. G. Holmes and C. E. Wieman, “Introductory physics labs: We can do better,” *Physics Today*, vol. 71, pp. 38–38, 2018.

[7] D. Scherer, P. Dubois, and B. Sherwood, “Vpython: 3d interactive scientific graphics for students,” *Computing in Science & Engineering*, vol. 2, no. 5, pp. 56–62, 2000.

[8] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, “Strengthening the case for pair programming,” *IEEE software*, vol. 17, no. 4, pp. 19–25, 2000.

[9] P. Heller and M. Hollabaugh, “Teaching problem solving through cooperative grouping. part 2: Designing problems and structuring groups,” *American journal of Physics*, vol. 60, no. 7, pp. 637–644, 1992.

[10] Available at https://trinket.io/todd_zimmerman_phd_gmail_com/courses/second-semester-introductory-physics-electricity-and-magnetism#/getting-started-with-vpython/why-vpython.

[11] N. T. Young, G. Allen, J. M. Aiken, R. Henderson, and M. D. Caballero, “Identifying features predictive of faculty integrating computation into physics courses,” *Physical Review Physics Education Research*, vol. 15, no. 1, p. 010114, 2019.