WIP: Integrating Computation within an Engineering Physics Course

Darren K Maczka, University of Tennessee at Knoxville

Darren Maczka is a Lecturer and Research Assistant Professor in the Engineering Fundamentals program at the University of Tennessee, Knoxville. He received his PhD in Engineering Education from Virginia Tech. His research interests include broadening participation in Engineering, computing education, and sociotechnical aspects of teaching and learning.

Erin J. McCave, University of Tennessee at Knoxville

Erin is a Research Assistant Professor and Lecturer in the Engineering Fundamentals Program at the University of Tennessee. She completed a postdoctoral/ lecturer position split between the General Engineering program and the Engineering & Science Education Department and a Ph.D. in Bioengineering from Clemson University. Erin's research interests include preparing students for their sophomore year, minority student engineering identity development, and providing mentoring relationships to help foster student growth and success.

WIP: Integrating Computation within an Engineering Physics Course

Introduction

There has been much interest in supporting the development of computational thinking skills in engineering students. Computational thinking (CT) supports both general problem solving as well as computer programming. This work-in-progress paper describes efforts to develop a new two-course sequence that combines an introduction to engineering physics with computation and modeling. These courses were developed to support students who entered not calculus ready in their first semester. Retention rates for these students were significantly lower than calculus ready students, with 40% of these students never reaching their first engineering course. Evidence that integrated curricula lead to strengthened learning outcomes was a significant motivator in the development of this course sequence.

As expectations of computational literacy in the engineering workplace continue to grow, there is increasing interest in effective methods to help engineering students gain proficiency in computer programming and computational thinking. While the practices associated with computational thinking are not restricted to computer programming [1], an introduction to computer programming is a common element of first-year engineering programs [2] and in many cases may be the only context in which these skills are explicitly taught. Teaching programming, even to CS students who ostensibly are motivated to learn the skills involved, is a well documented challenge [3, 4]. Introducing core computing skills to general engineering students faces the additional challenge of engaging an audience that is not intrinsically motivated to learn the skills, or worse who chose a non-CS major *because* of negative perceptions of computer programming. One strategy that may be effective at motivating non-computing majors to learn computing skills is situating the skills within a disciplinary context that they *are* motivated to learn [5].

While computer programming is often the tacit skill that students as well as educators associate with computing skills, there has been an increasing emphasis on the related but more general skills encompassing ”computational thinking”. Wing [1] defines computational thinking as “the skills needed to reason about a problem solution in a way that could be implemented on a computer”, and while that definition still couples the idea with a literal computer there is recognition that “computational thinking” can be more broadly applied to all types of problem solving. Thus, computational thinking provides a framework to help bridge concepts and content within an integrated engineering problem solving and computer problem solving course [6]. By integrating computational thinking instruction within an engineering physics class, we hope that students will be able to demonstrate greater understanding of problem solving, physics concepts, and computing concepts and skills [7].

Course Context

The primary motivation for developing a new course sequence was to increase retention within the college of non-calculus ready students. The hypothesis was that by creating a course sequence that allowed non-calculus ready students to take engineering courses and interact with

engineering professors their first year, they would be more likely to stay with engineering. This context is important for two reasons: 1) it situates the goal of structuring the course around a CT framework as emergent from the challenges intrinsic to creating a unified experience for learners, and 2) the pre-calculus population suggests learners that may have lower math self-efficacy than their peers which is significant as many of the skills required for math proficiency are also needed for computational thinking [8].

The first iteration of the course aimed to remix the learning objectives and content of a 3 credit engineering physics courses, a 1 credit computer-based skills course, and a 1 credit introduction to the college of engineering course into a sequence of two 3-credit hour courses over two semesters. During the initial pilot we discovered that this class-contact time was not sufficient to bring this population to the desired level of mastery across all learning objectives. In particular, the extra time and support students needed to develop and practice math skills in the context of becoming familiar with the required physics concepts made it impossible for students to reach a level of proficiency across the prerequisite physics concepts and problem-solving skills needed to succeed in later courses. This motivated a redesign to a two 4-credit hour sequence with a pilot during the 2022-2023 academic year.

Organizing Framework

Curzon and colleagues [9] organize a framework for course design around computational thinking defined as "...the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent". They divide CT into the following:

- algorithmic thinking: determining a sequence of steps that can be taken to solve a given problem
- evaluation: determining whether a particular solution is viable, or particular algorithm does what it is expected to do
- abstraction: reducing complexity by eliminating information that is not directly relevant
- decomposition: breaking a large problem into smaller pieces that are individually easier to solve than the whole

We found that learning activities that had been developed in the precursors to this new course could map skills to parts of this framework such as algorithmic thinking, evaluation, while links to abstraction, and decomposition could be made more explicit. There is also the opportunity to frame core-skills, such as algebraic manipulation in a CT context, e.g. by prompting learners to list a sequence of steps that can be followed to symbolically solve a class of equations.

Assessment

Current practices for assessing computational thinking vary widely, from using course work grades, computational thinking instruments, self efficacy instruments, to observations [10]. Currently, computational thinking is not an explicit learning outcome of our course and thus existing assessments serve only as a proxy to these skills. We are currently evaluating whether we should assess computational thinking more directly as this would allow us to answer questions about possible correlations between CT skills and conceptual physics skills.

Challenges

Casting certain physics problems to a suitable form for a computer solution may not always be feasible. For example, since most of the physics problems are meant to be worked by hand with eventual assessment on a written exam, algorithmic structures such as iteration and conditional branching do not arise naturally. Certainly these algorithmic components are part of a learner's process as part of problem solving, e.g. "First I need to calculate the push force and force of impending motion, if the push force is greater than the force of impending motion, use the kinetic friction equation, otherwise the force of friction comes from the free body and kinetic diagram"

We also found that trying to situate computational activities within a physics problem-solving context necessarily coupled physics problem-solving ability with computational skills. Finding an appropriate level of scaffolding for programming assignments was a challenge, particularly due to a high variance in comfort with physics concepts and math skills across the student population. The level of scaffolding varied across programming assignments from very high: providing the derived equations and flowchart describing an implementation, to minimal: providing a problem statement describing a physics problem. The students who struggled with algebra and working with physics concepts tended to struggled with all levels of scaffolding: For minimally scaffolded assignments they understandably struggled with even setting up the problem, while tending to view highly scaffolded assignments as busywork or having little value since they did not see any connection to the physics they were learning. Further, students that struggled with conceptually or algebraically understanding the problem and solution structure tended not to recognize this as their bottleneck and would attempt a MATLAB implementation for a solution they didn't understand. This unsurprisingly lead to a jumble of error-riddled code that was as difficult for graders to decipher as it was for the authors to describe.

Future Work

The pilot and first revision of this course focused on identifying and assembling a reasonable sequence of content, activities, and assessment. In the next revision we plan to make the links to computational thinking more explicit and build more synergy between existing physics concepts and data analysis through complementary lab activities. We hope this provides a balance that can help reduce the tension between need for abstraction and motivation that comes more naturally from concrete application. To assess these changes we plan to administer a survey instrument adapting existing self-efficacy and motivation measures to the CT and engineering problem solving context. We will share a draft edition of this survey and solicit feedback.

References

[1] Jeannette Wing. Computational thinking. 24(6):6–7.

[2] Kenneth Reid and David Reeping. A classification scheme for "introduction to engineering" courses: Defining first-year courses based on descriptions, outcomes, and assessment. In *American Society for Engineering Education Annual Conference & Exposition. Indianapolis, IN (1-11). Washington DC: American Society for Engineering Education*.

[3] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. A Multi-national, Multi-institutional Study of Assessment of Programming Skills of First-year CS Students. 33(4):125–180. ISSN 0097-8418. doi: 10.1145/572139.572181. URL `http://doi.acm.org/10.1145/572139.572181`.

[4] Chin Soon Cheah. Factors contributing to the difficulties in teaching and learning of computer programming: A literature review. 12(2):ep272.

[5] Andrea Forte and Mark Guzdial. Motivation and nonmajors in computer science: Identifying discrete audiences for introductory courses. 48(2):248–253. URL `http://ieeexplore.ieee.org/xpls/abs`$_{a}ll.jsp?arnumber = 1427874$.

[6] David Weintrop, Elham Beheshti, Michael Horn, Kai Orton, Kemi Jona, Laura Trouille, and Uri Wilensky. Defining computational thinking for mathematics and science classrooms. 25:127–147.

[7] Alejandra J. Magana, Michael L. Falk, Camilo Vieira, and Michael J. Reese. A case study of undergraduate engineering students' computational literacy and self-beliefs about computing in the context of authentic practices. 61:427–442.

[8] Kathryn M. Rich, Elizabet Spaepen, Carla Strickland, and Cheryl Moran. Synergies and differences in mathematical and computational thinking: Implications for integrated instruction. 28(3):272–283.

[9] Paul Curzon, Mark Dorling, Thomas Ng, Cynthia Selby, and John Woollard. Developing computational thinking in the classroom: A framework.

[10] Chang Lu, Rob Macdonald, Bryce Odell, Vasyl Kokhan, Carrie Demmans Epp, and Maria Cutumisu. A scoping review of computational thinking assessments in higher education. 34(2):416–461.