2021 ASEE ANNUAL CONFERENCE
Virtual Meeting | July 26–29, 2021 | Pacific Daylight Time
Paper ID #34021

Advancing Computational Knowledge and Skill Through Computing Projects in Sophomore-level Mechanics Courses

Prof. Keith D. Hjelmstad, Arizona State University

Keith D. Hjelmstad is President's Professor of Civil Engineering in the School of Sustainable Engineering and the Built Environment at Arizona State University.

Dr. Amie Baisley, University of Florida

I have a M.S. in structural engineering from Arizona State University and a Ph.D. in engineering education from Utah State University. My teaching and research interests are centered around the sophomore level courses that engineering students take and how changes in those courses can impact student learning and retention.

©American Society for Engineering Education, 2021

Advancing computational knowledge and skill through computing projects in sophomore-level mechanics courses

Abstract

The desire to graduate students with more advanced computational knowledge has become a hot topic in curriculum design. One route to do that is through integration of computing in the foundational mechanics courses (statics, dynamics, and solid mechanics). The implementation of computing projects in these sophomore-level courses has resulted in computing becoming an integral part of those courses at our institutions. This shift has changed the mindset in both students and faculty, greatly expanding the range of problems that students can explore at the sophomore level. Computing projects offer the ability to introduce more open-ended problems in the mechanics courses where students can think about certain concepts more deeply. It also provides the opportunity to introduce important ideas of numerical analysis in a way that makes those techniques immediately relevant. The projects also encourage students to get more creative, in courses often viewed as skill development, by seeking means to verify their codes and then use those codes to explore the target problem without the encumbrance of tedious hand calculations. Requiring that each student write a full technical report for each project pushes the students to make connections between theory and results and it gives the instructor a window into the depth of their learning. In our implementation, each of the three courses requires four to six computing projects during a semester that involve problem formulation, coding in MATLAB, exploration using the code, and writing a final report. The level of inquiry and documentation for each project goes far beyond what we typically see in the standard handwritten problem solutions that have long been the stock and trade of these courses. Each course has its own unique set of projects that focus on various mechanics concepts. The paper will discuss the types of projects implemented and observed student outcomes.

Introduction

Computing is central to almost all engineering activity today. It is increasingly difficult to imagine a future for engineers that does not include substantial computer literacy, including the ability to program. However, most mechanics-based engineering curricula (e.g., civil, mechanical, and aerospace engineering) have not yet fully embraced computation as an integral part of the engineering pedagogy. While learning concepts like mechanics with pencil-and-paper calculations remains an important part of pedagogy, long gone are the days where such approaches are at the center of professional engineering analysis and design.

Several decades ago, most mechanics-based engineering curricula introduced a required computing course, usually to be taken in the freshman year. In most curricula, computing is then systematically ignored for the remainder of the program, resulting in graduates who are weak in computational science. In many institutions, ours included, the freshman computer science course fell victim to cuts caused by pressure to reduce the number of total hours in the curriculum. These outcomes stand at odds with the feedback from industry that students need *more* computational knowledge upon graduation [1].

There is room for debate as to which computational environment is best for students to learn and what should be the nature of computational knowledge taught. The answers vary across disciplines and universities, yet faculty generally agree that students need more education in how to use computers as a tool to solve engineering problems. Any attempt to confine the teaching of computation to a single course only makes these debates more difficult to resolve. In addition, a single course offered in the freshman

year is disconnected from applications that connect to the engineering concepts associated with the major not only because the instruction is usually carried out by the computer science faculty, but also the students at that level do not yet have much knowledge of their chosen discipline.

The curricular shortcomings associated with computing parallel the problem of instilling solid communication skills in engineering students, wherein the most common strategy has been to insert one or two required writing courses early in the curriculum. A sounder pedagogical approach is to include communication throughout the curriculum so that students get the repetition and feedback needed to improve and grow. Our hypothesis is that the same is true for computing. If you want to graduate engineers who are prepared for a world that runs on computers, the education in computing needs to occur throughout the curriculum and it needs to connect with the ideas associated with the field.

Regular exposure to computing introduces students to new concepts, new ways of thinking about solving problems, and even tools that may not otherwise show up in courses [2, 3]. It gives students the opportunity to understand how commonly used engineering software packages work instead of relegating them to use black boxes that generate results from inputs. There are a few examples of incorporating computing into mechanics courses [4–7], but the depth and range of the projects vary from using a canned program as a black box to projects that involve the full creation of a working program. In one case, the introduction of regular programming assignments in an aerospace department proved beneficial for students, but they concluded that the students lacked the needed foundation in programming for the project to be successful [8]. Further, integration of MATLAB as a regular exercise in a dynamics course was shown to have an impact on the students learning of programming concepts [9] as well as exposure to more complex problems and the concept of uncertainty [10]. Through incorporating these types of projects, the students are forced to understand the theories presented in the courses at a deeper level in order to implement those ideas in a computer program.

Almost a decade ago we embarked on a complete redesign of the sophomore-level mechanics courses: statics, dynamics, and deformable solids at a large R1 university in the southwest (Institution A). We changed the structure, environment, and feedback mechanisms in a redesign that affected every aspect of the courses, an effort we call *The Mechanics Project* [11]. One of the major changes was to introduce computing into all three courses. This paper will focus on how computing projects were integrated across the three mechanics courses and the impact that had on the students skills and culture. Our students did not have a dedicated computer science course before starting the mechanics courses. Therefore, we assume that the students entering this process have little or no experience with programming.

Implementing computing in undergraduate mechanics courses

Our extensive revision of the sophomore-level mechanics courses provided the opportunity to consider where and how to introduce computing in these three foundational courses. Since the courses occur in the second year of study, implementation held the promise of moving toward a curriculum in which computing is ubiquitous as instructors of upper-division courses would then be able to rely on their students having a foundation in computing. Taking on three courses also allowed us to coordinate across the courses to create a developmental path, to reinforce important concepts, and to see the benefits of students adapting to a learning paradigm that was not confined to a single course.

The main learning goals for our students were to:

- Develop skills in computation as a tool for solving engineering problems.
- Learn about numerical methods needed to complete the projects.

- Understand the concept of verification and its role in engineering.
- Deeply investigate and explore mechanics concepts.

Dynamics was the first course to be transformed and was arguably the one that most naturally lent itself to the need for computing. Almost every problem in rigid-body dynamics is governed by a nonlinear differential equation of motion. Traditional textbook problems avoid this difficult reality by framing problems as "snapshots" at certain times. Questions often contain wording like, "Find the acceleration at time zero." In essence, the subject of dynamics has been reduced to something that is more like advanced statics. In this context, students do not experience dynamics as the study of systems that evolve with time. Numerical computations can easily remedy this shortcoming. In fact, once set up, almost all dynamics problems yield to the same computational strategy. Additionally, this approach allows the introduction of several important numerical methods techniques that are often taught later in the curriculum outside of the context of mechanics (e.g., numerical integration of differential equations and Newton's method for solving nonlinear algebraic equations).

We decided to use MATLAB because it was freely available to all students in our school. This platform has become a popular tool in engineering courses that students can access after a short introduction into the layout and structure due to its straightforward programming environment and syntax [10]. Once MATLAB was chosen as the computing environment, the design of the computing projects (CPs) for each course began.

Our first challenge was the fact that our students had little previous exposure to computing and had very little background in numerical methods. Even though MATLAB had been briefly introduced in other courses, most students still claimed that they knew almost nothing about MATLAB. Knowing that adding a computer science course to the curriculum would be harder than chiseling stone, we opted to provide the instruction needed on a just-in-time basis as part of the mechanics courses themselves. Furthermore, once statics was transformed, we could conceptualize a developmental path for the students to learn to program. In the subsequent courses (dynamics and deformable solids) the students could start at a higher level and advance their programming skills.

The projects in statics were designed to be more tutorial and focused more on making small modifications to existing code and teaching the students how to use MATLAB. The CPs were designed to open up a route to exploration using the codes and framed around topics at the heart of the course.

In dynamics, the students start with a code that solves the projectile motion problem. This code provides a template for their first computing project (and for many of the others, too). Instructional materials evolved to help the students to understand the code. In its simplest form, the code **`projectile.m`** is only about forty lines long. Through this code, students learn about stepping through time with numerical integration. The first project adds an elastic tether to the particle and the task is to formulate the equations, make the necessary modifications to the code, and to add plotting to visualize the results. Subsequent projects build on this initial foundation and explore other aspects of dynamics like contact, impact, stability, and vibrations.

Traditional courses on dynamics focus on the derivation of equations of motion, but do not advance the solution. The computing projects provide a perfect complement to the traditional environment in which students still learn to derive equations by hand, but then have an opportunity to see what those equations produce. As a result, we did not need to sacrifice any of the traditional values while we gained the ability to launch much deeper investigations of the problems than the traditional environment supports.

Since dynamics problems are nonlinear and require integration over time there was also the need to introduce numerical method techniques to advance the solutions. This afforded an opportunity to

introduce two simple ideas—generalized trapezoidal rule and Newton's method—that up to that time were held hostage in an upper division numerical methods course. With a brief lecture and set of course notes, the students got instruction on the techniques and programming logic that would be used in all of the projects. Repetition was key to getting the students to competent understanding of the methods.

Statics was implemented a year after dynamics. This ordering was fortuitous because we knew better what the students needed. The projects were created based on the concepts that we wanted our students to dive deeper into. For statics we included topics like load placement, truss analysis, and shear force and bending moment diagrams. Once the projects were put in place for statics, we noticed a substantial improvement in the ability of the students to do projects in dynamics. The projects in statics introduced the students to the environment and syntax, acquainted them with the use of arrays and indexing, and showed them how to use some of the basic I/O tools in MATLAB.

A year after statics, we transformed the course on deformable solids. The introduction of computing projects in deformable solids was quite different from our experiences with statics and dynamics. Most students take dynamics in the same semester as deformable solids, so the capabilities are built on the same programming foundation as dynamics (i.e., what they get in statics). It was more natural to develop projects in deformable solids that did *not* conform to the repeated use of a template code as was the case in dynamics. For deformable solids, this included topics like axial bar deformation, beam deformation, and multiaxial states of stress and strain in a deformable body.

The projects in statics and deformable solids also offered insight into mathematical techniques that students typically learned outside of the engineering context. Heavy use of linear algebra techniques to solve the truss and beam codes in statics supplemented the knowledge gained in the required linear algebra course. Students in deformable solids are introduced to additional numerical analysis techniques like Simpson's rule for numerical integration. In each case, a brief lecture and set of notes allows the students to see how the techniques are used and implemented, but the coding ultimately provides the framework and motivation for their learning.

What did we give up?

Learning mechanics is a significant challenge for sophomores in engineering. These courses represent a key gateway to concepts in upper division courses. A commonly used learning strategy is to have students solve numerous small problems that exercise the concepts. The list of topics to be covered is fairly standard, and most instructors would characterize the courses as "full," with little opportunity to add new topics or assignments.

It is reasonable, then, to ask what topics we gave up in order to implement the computing projects. The short answer to that question is none. We cover all of the standard topics found in these courses as traditionally taught. We open up opportunity by changing the way we teach some of the topics and changing the nature of the assignments we ask students to carry out. For example, the concept of conservation of momentum as applied to impact problems can be taught through computing projects just as well, if not better, than through the traditional small snapshot problems. Developing a code to do static analysis of trusses opens a different avenue to discuss static determinacy and stability. A code that integrates the beam equations numerically is just a different way of thinking about organizing the calculations for those problems—one with a lot less tedious algebra.

If you look carefully at what we traditionally teach when we teach mechanics to undergraduates, you often find an emphasis on recipes for organizing calculations shoring up the core concepts. Few of those recipes are sacrosanct; some don't even align with modern practices in engineering. To open up room for

computing projects we shifted some of the emphasis on hand-calculation strategies and invested those in computationally based strategies keeping the same topical coverage.

Verification, exploration, and communication

While learning to code and learning certain numerical methods provides a powerful way to introduce these ideas into the curriculum without adding courses, the real value of the computing projects emanates from their use, first to verify the codes, then to use the codes to explore the topic at hand, and finally to communicate their findings through a written report.

Code Verification

At the start of *The Mechanics Project*, our students did not know what code verification was, let alone why it is important. And yet, the act of formal code verification is probably the most intrinsically "engineering" thing that they will learn through computing. Engineers are constantly faced with the task of deciding if a result is correct and all engineers develop strategies to inform those decisions. Verification of a computer code simplifies the context and allows the development of this skill.

The students come in with the notion that computer programs work perfectly and frequently have unearned trust in programs. Requiring the students to verify their program before embarking on exploration forces them to confront this bias. The students are required to verify each program through hand calculations, by locating reliable references, by identifying special solvable cases, and by applying scientific reasoning when looking at computed results. More often than not, they realize that programs seldom turn out right on first implementation and it takes a deep dive into the organization, theory, and derivation to successfully debug a program.

Code verification is, perhaps, the best argument for implementing computing projects in any course. Students struggle mightily with this task as each problem presents a new verification challenge. From our experience, it is evident that students learn more from the code verification process than any other single aspect of the computing projects.

Open-ended exploration

Once the code is working, the students have a tool for exploration. At the sophomore level, most students have little or no experience with open-ended exploration. Learning how to form hypotheses and test them is an essential skill. The students can easily study different physical parameters, initial conditions, boundary conditions, loadings, and other features that simply require changes in the inputs. Because results are almost immediate, the environment fosters a trial and error approach to studying the target problem. It is no longer a daunting task to determine the effects of changing, say, the initial velocity for a dynamics problem. Student are encouraged to enhance the graphical outputs available to them in order to better see what the system is doing. For dynamics we provide code segments that they can drop into their codes to quickly get animations of the system they are studying.

Communication of results

We initially debated the merits of including formal reports as part of the CP process. Teaching mechanics is usually conceptualized as getting students to grind through as many problems as possible with the hope that the exposure and repetition will permanently lodge the ideas. On the other side of the ledger is the unfinished task of teaching students how to communicate better. We ultimately embraced the notion of

communication across the curriculum as the nobler goal. Students solve fewer problems in these courses than in the traditional environment, but they explore them in much greater depth and through writing they leverage different cognitive strategies in their learning.

For each CP, the students are required to write a report to document their exploration. The report is the primary document used for assessing their work on the project. The requirements of the report are provided for the students in a document entitled *Guidelines for Doing Computing Projects* that describes the necessary parts of a CP report. They are required to include an introduction, theory section, code verification, study and results, and conclusions. The report and its sections must focus on the mechanics ideas of the CP and not what was learned about MATLAB from a particular project. It is a professional document that involves text, equations, tables, and figures.

The grading rubric is available to students (and the grader) in a document entitled *Evaluation of Computing Projects* that outlines specific attributes that their reports should have to increase their value. Students also engage in peer review of projects done by other students (randomly assigned and anonymous until complete). The task is informed by a document entitled *Peer Review of Computing Projects* that describes what to look for and how to create a peer review that has value to the original report writer. All of these documents are available on the course website and all provide insight not only into how to prepare their final reports, but more broadly how to go about the process of inquiry.

The reports are open-ended for length and direction. We found that when left completely open, which we did at the start of *The Mechanics Project*, the students floundered. We added a small set of compulsory explorations to the assignment document that were designed to get them started with the process of conceiving and executing their study. Students are then encouraged to dig deeper around the compulsory ideas or to branch out into other aspects of the problem. Each student chooses the direction and scope for their study and each student produces a unique report. Reports typically vary from five to twenty pages in length. Students can be creative in their exploration path and with their report organization. While the rest of the course follows a structured solution approach for hand calculations, these projects give students freedom to pursue their interest.

The computing projects

The development of specific computing projects is a challenge because the projects need to align with the course learning objectives in the module in which they appear, they need to tackle an important problem, and it needs to be apparent that the problem would be difficult, if not impossible, to solve without the computer. In general, we seek to have projects that solve a broad class of problems, not only to make the case for computation as a vital aspect of engineering, but also to ensure that the exploration part of the project will be fertile.

Over time we have created new projects, retired others, and adjusted projects to be better. The projects currently in use in the three courses are given in Table 1. In this section the projects are briefly described, and one is described in more detail.

Statics projects

In statics there are only four computing projects since that is the students' first introduction into CPs and it is a challenge for students to adapt to all the other unique features of the learning environment of *The Mechanics Project* [11]. The first project is designed as an introduction to MATLAB. The statics problem it is derived from is simple and looks at placing a load at different points along a beam then plotting the

reactions as a result of the load placement. Students are introduced to a *for* loop, how to type equations into MATLAB, and basic plotting functions.

Table 1. *Computing Projects in Mechanics*. This table gives the short title for each of the projects in the three sophomore mechanics courses

Statics	Dynamics	Deformable Solids
1. Load Placement	1. Particle	1. Axial Bar
2. Distributed Load	2. Contact	2. Multiaxial States
3. Truss Analysis	3. Pendulum	3. Properties of Areas
4. SF and BM Diagrams	4. Impact	4. Planar Beam
	5. Stability	5. Multispan Beam
	6. Vibration	6. Design

The second and fourth projects are designed to advance the students understanding of two key topics in statics with a program designed to analyze a general class of problems. In CP 2, the students implement the logic to compute a triangular loading function along any portion of a beam and determine the resulting support reactions. In CP 4, students implement the boundary conditions for statically determinate beams to compute shear force and bending moment diagrams for different transverse load functions. Together, CP 1, CP 2, and CP 4, provide programs for deeper exploration of point loads vs. distributed loads and the effect of the load location and support type.

The third project in statics, CP 3, involves the development of program to analyze statically determinate space trusses. The students are challenged with the need to create a logical labeling system for the joints and members, identify a binary code for the support reactions, and understand how to create and reduce an entire truss system of equations to solve for all the unknowns in a truss. Concepts of linear algebra are needed but many students do not yet have a strong linear algebra background. So, we provide resources on how to create a system of equations for a truss and how the system can be solved.

For this project, the students are given an “almost working” planar truss code where they are asked to first code in the calculation to compute the unit vector direction of each member in a truss. Once the program works for planar truss analysis the students verify the code with their hand calculations. The next step requires the students to modify the program to perform space truss analysis. This requires understanding the degrees of freedom in a truss system and how the system of equations is assembled to correctly get the program running. This modification involves adding only four segments of code, but until the student grapples with how to logically add in the degrees of freedom required to go from a planar truss to a space truss it seems challenging. Once that connection is made, the code modifications take only a matter of minutes for most students. Then they must verify the space truss code with hand calculations exercising method of joints for a 3D problem, and finally, the students write their report on space trusses. While exploring the space truss code a discussion of statical determinacy often comes up, and students must think about why certain problems do not work in the statics truss program and how to interpret the residual error that is computed as part of the program.

Dynamics projects

In dynamics, the students complete six computing projects. The first project (CP 1, projectile) is designed to connect with their prior knowledge of particle mechanics and involves a particle attached to a point with an elastic tether. The tether can go slack if the particle moves to a location where the distance from

the origin to the particle is smaller than the unstretched length of the tether. The tether will break if the tension exceeds an allowable value.

The second (CP 2, contact) and fourth (CP 4, impact) computing projects introduce the students to contact mechanics, first for a particle bouncing off of a curved surface and then for two rigid bodies impacting each other in motion. Through these projects the students learn about conservation of momentum, the coefficient of restitution, and how contact is handled in a computational environment. Specifically, they learn how to detect and resolve the contact in a time-stepping environment. Impulse-momentum principles are the most difficult concepts for students in dynamics, and these projects offer a different view of the most important application of conservation of momentum.

The third project (CP 3, pendulum) is the first one that involves a rigid body. Here they encounter problems where the variable being tracked is not the position itself but rather a time-dependent variable (the angle describing the orientation of the pendulum) from which the positions of all of the particles in the system can be determined. The fifth project (CP 5, stability) extends the third project to multiple degrees of freedom. This project also introduces the concept of stability of the system as the system is an inverted double pendulum with rotational springs providing the restoring forces. The force of gravity provides the destabilizing force. Students learn about stability in a dynamic context where stability means that a small perturbation of the system leads to a small and proportionate response. Also, they get to see the chaotic behavior of the double pendulum without springs.

The sixth project (CP 6, vibration) is a simple four-degree-of-freedom system of masses and springs subjected to either initial conditions or forced sinusoidal vibrations. Students learn about natural modes of vibration, natural frequencies, resonance, and other important dynamic phenomena. This project allows the introduction of the eigenvalue problem without the onerous task of learning how to do the computations by hand.

These six projects together provide both a developmental sequence for the students to progress from particle mechanics to rigid body mechanics and to encounter important dynamics concepts that are difficult to appreciate through hand calculations. All of these problems are impossible or impractical to solve outside of the computational environment.

Deformable solids projects

In deformable solids, the students complete six projects. The first (CP 1, axial bar), fourth (CP 4, planar beam), fifth (CP 5, multispan beam) and sixth (CP 6, design) projects comprise a set of problems that get increasingly complex but share a similar code structure. We start the course with the axial bar problem as an introduction to deformable solids in the simplest possible context. The axial bar project formulates the problem for all possible boundary conditions and any arbitrary distributed load form. As such, the code solves a general class of problems and students are asked to explore things like the point load as a limiting case of a distributed load, the effects of boundary conditions, and how the internal stresses from different load forms come about.

The code for the first projects serves as a template for the fourth project in which we develop a code to solve the prismatic beam problem for all possible boundary conditions and arbitrary distributed load forms. The additional complexity of beam bending is offset by the familiarity that the students already have for the organization of the code from the axial bar project. The project on the multispan beam extends the beam code to include intermediate supports and the sixth project uses the code from the fifth project in support of the design of a multispan bridge to meet design criteria of strength and deflection. For most students the last project is their first encounter with engineering design concepts, and they learn

that the analysis that they spend so much of their time on in these courses is a step in the design process, not an end in itself.

The second (CP 2, multiaxial states) and third (CP 3, properties of areas) projects are quite different. The study of multiaxial states of stress involves finding principal values, the principal angle, drawing Mohr's circle, implementing Hooke's law, and finding the maximum shear stress and strain. These calculations are vulnerable to plug-and-chug approaches, in part because the calculations are somewhat tedious and in part because there is a fairly strict limit on how far we can go with multi-axial problems in the undergraduate course. In the second project, students write a code that does all of these calculations and renders the associated graphical representations. The code cements their understanding of stress and strain as tensor objects and provides a tool to examine many important states of stress and strain.

For the various theories typically covered in deformable solids, the properties of cross-sectional areas play a key role. It is easy for this course to fall into the pedagogical trap where the main calculations that is not plug-and-chug is the computation of the cross-sectional properties. And yet, from a physical standpoint, that calculation is only a small piece of the deformable solids puzzle. The third project creates a code that computes the area, centroid, and moments of inertia of any cross-section, open or closed, that can be represented as a polygon. As such, it creates a powerful tool that the students can use to support their hand calculations as they learn how to do the computations. With the tool they can explore the efficiency of cross-sectional shapes knowing how the cross-sectional properties appear in formulas for stress and deflection.

A more detailed example

To get a sense of the nature of a computing project, consider CP 2 in *Dynamics* in a little more detail. The assignment document is accompanied by a set of notes that outlines the key features of doing numerical computations associated with a particle contacting a curved surface. The document provides a strategy for detecting contact in a time-stepping scheme, a strategy for interpolating the state to resolve the exact time of contact, and derives the equations for resolving contact using conservation of momentum in conjunction with a model based upon the coefficient of restitution.

The students start with a code that carries out the contact algorithm for a particle contacting a planar surface. The first task is to exercise and understand that code. The computing task for this CP is to modify the code to have the particle contact a parabolic, elliptical (see Fig. 1), or corrugated surface. They are also asked to create their own surface (which tests their understanding of the requirements of the contact detection algorithm). In the study part of the project, students are asked to investigate the nature of energy loss in contact and the roles angle of impact and of the coefficient of restitution in that energy loss. They are then prompted with several open-ended questions about the nature of contact to lure them into further exploration.

The *code verification* task is difficult for this project as there aren't any analytical results available for contact on curved surfaces. We often give them suggestions when they ask. For example, they could verify that energy is conserved if the coefficient of restitution is equal to one. They can also verify that a particle that starts at a focal point of an ellipse will always pass through the two focal points in its trajectory (you have to turn off gravity for that case).

Even though the coding challenges for this CP are modest, there is no route around understanding the starter code to make the modifications required. Further, much of the code is similar to the one they developed for CP 1, so the repetition helps to cement concepts that came up in the first project.

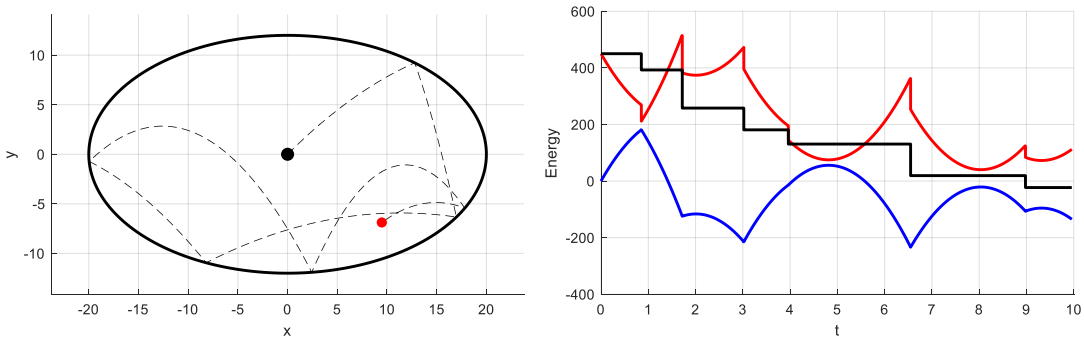

Figure 1. A typical result from CP 2 in *Dynamics* showing the path of a particle bouncing around inside of an elliptical container. The black dot is the initial position and the red dot is the final position (after 10 seconds). The black line in the energy plot on the right is total energy, the blue line is potential energy, and the red line is kinetic energy.

Evaluation of computing project reports

In any assigned work, the assessment strategy drives students to achieve desired outcomes. In the computing projects we want students to be successful in accomplishing the technical goals, but we also want them to advance in their understanding of inquiry and in their ability to communicate.

The grading rubric for the computing projects has three main areas, each with four subobjectives, giving a total of 12 items. Each of the 12 items receives a score of 0-4 (0 = missing or non-responsive, 1 = below expectations, 2 = satisfactory, 3 = above expectations, and 4 = well above expectations). The three main areas of evaluation are (1) technical execution, (2) exploration, and (3) communication. Each area is equally weighted in the grade and all are designed to foster the creation of a high-quality product. The list of rubric items is given in Table 2.

Table 2. *Evaluation Rubric for CPs*. This table lists the different areas that are evaluated for each report.

Main Objectives Area	Subobjectives
Technical execution	1. Theory summary
	2. Code quality
	3. Code verification
	4. Code usage
Exploration	5. Study roadmap
	6. Depth of study
	7. New observations
	8. Scientific competence
Communication	9. Writing quality
	10. Completeness
	11. Quality of conclusions
	12. Presentation style

The students are provided with this rubric and a detailed description of each of the 12 areas at the start of the semester. The document *Evaluation of Computing Projects* includes a more detailed description of what each of the rubric levels looks like for each of the subobjectives. For example, a 4-level performance on item 6 (depth of study) says:

> "The study *explores significant aspects of the problem at hand*. The required elements are all done well. Parameters are varied and trends are noted. The results show good synthesis of results obtained from the computations. The study may go beyond the required elements to explore some additional aspect of the problem. Synthesis is clearly evident."

Students receive feedback for each project with their score in each of the areas along with written comments. The comments provided are both general to common issues amongst students and specific comments for their project. The feedback is provided to each student before the next project submission to help foster the opportunity to keep improving with each project.

Students also participate in a peer review process, which gives them fast turnaround feedback on their projects. Each student is randomly assigned three to five reports done by their peers to review. The students are provided with instruction on how to execute peer review through a document entitled *Peer Review of Computing Projects*, which gives guidance that a good peer review contains actionable comments that the original report author could use to improve the report. Students do not give grades but are encouraged to use the grading rubric as guidance to understand how to articulate the qualities we seek in the reports. Using peer review also helps students to embrace the rubric which ultimately helps them to improve their own projects.

Having four to six projects in each course provides a feedback path that fosters improvement in their performance. In the earliest versions of the peer review process, students had an opportunity to revise and resubmit their reports after getting feedback. However, some students abused the process, so we eliminated the revision round. Even though they do not have the option to revise a project based upon the feedback on that project, they can use the feedback to improve subsequent projects. Since subsequent projects are similar, the feedback has value for those future efforts.

Student outcomes

The addition of computing in mechanics has provided several outcomes, some measurable quantitative results and some qualitative. The students have been graded in the same 12 rubric objectives for all courses for all semesters that the projects have been offered providing a large amount of information on how students perform on the computing projects.

Since starting *The Mechanics Project*, the CPs have now been offered at two universities. For the entire duration of the project the CPs have been required in all three mechanics courses at the large R1 university in the southwest (Institution A) and for the past two years the CPs have been offered in statics at a large R1 university in the southeast (Institution B). The results that are provided are from statics for the past two years and for dynamics and deformable solids for the past four years. The population breakdown for each course is given in Table 3.

Table 3. *Student population*. Breakdown by course for the number of students that have completed the CPs, the number of semesters included in the sample, when instruction took place, and where instruction took place.

Course	Number of students	Number of semesters	Semesters included	Institution
Statics	279	3	FA19 – FA20	B
Dynamics	540	8	SP17 – FA20	A
Deformable Solids	628	8	SP17 – FA20	A

The average scores given for the 12 objectives during the included semesters for each course has been tracked for multiple semesters and the trends noted. For example, Fig. 2 shows the trends for dynamics over the 8 semesters starting with the SP17 semester through the FA20 semester. The cohort in Table 3 includes students who either withdrew from the course or failed to achieve a final grade of C or better (which is required to move forward in the curriculum), the figures include only students with grades of C or better. Note that many of the students in Institution A are common to both dynamics and deformable solids (and most took the courses in the same semester).

The progress of students through a semester is evident in Fig. 2. Almost all of the 12 rubric areas show improvement over the course of the semester, some showing more improvement than others. Observe that the ordinates are different for the different areas and it is evident that students are better at some things from the start than they are at others. Observe that some of the lowest initial scores are in code verification, in part because students are least familiar with it at the beginning of the semester. While students struggle with the open-ended exploration, they tend to do well in in the area of 'code quality' with average values between 3 and 3.5. Students generally spend the most time getting their codes to work and often allocate the least time for the exploration and communication parts of the project.

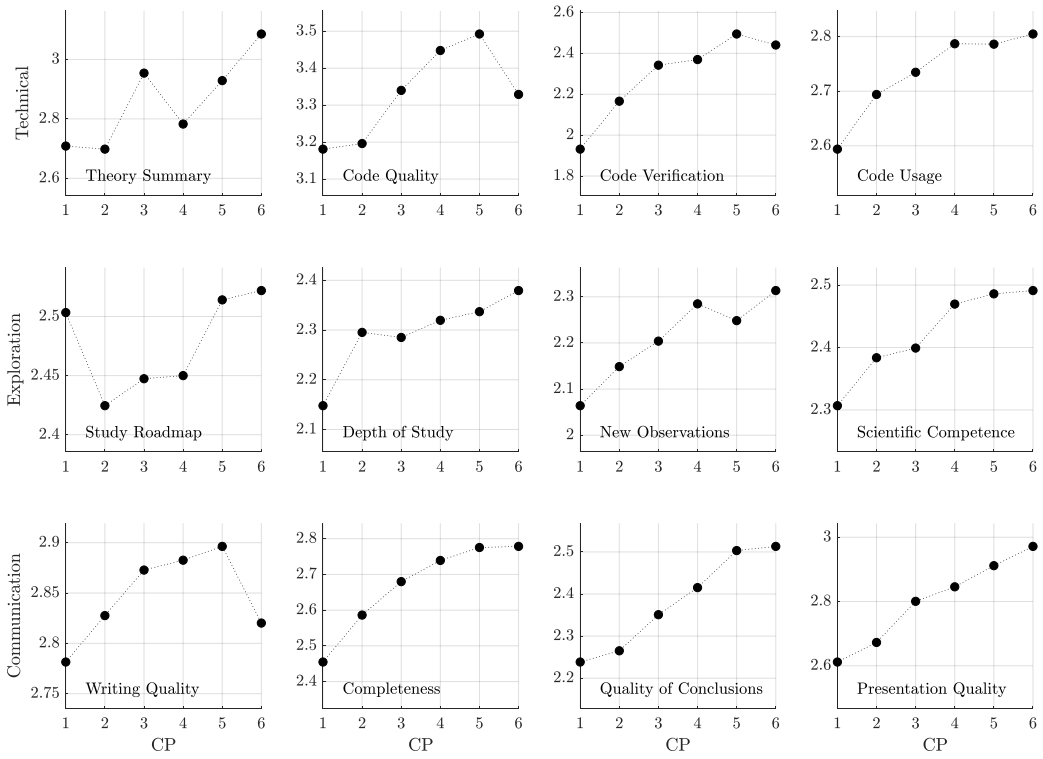

Figure 2. Progression of average scores in the twelve rubric areas across the six CPs in Dynamics. The average is over 476 students who completed the course with a grade of C or better over eight semesters between SP17 and FA20.

There are a few anomalies in the data. For example, it appears that students score well in the study roadmap in the first project but do worse in the next one. It also appears that the writing quality drops for the last project (they might be running out of gas at that point). Some of the variation (e.g., code quality on CP 6) can be explained by the nature of the project or the associated code.

The results for deformable solids are shown in Fig. 3. These results do not show the same trends in continuous improvement that the results for dynamics do. However, a close inspection shows that CP 6 is

responsible for much of the impression that students are not improving through the semester. In fact, without CP 6 students show steady improvement in all aspects of communication. Most of the other areas fluctuate a bit. That may, in part, be due to the nature of the target problems in the projects.

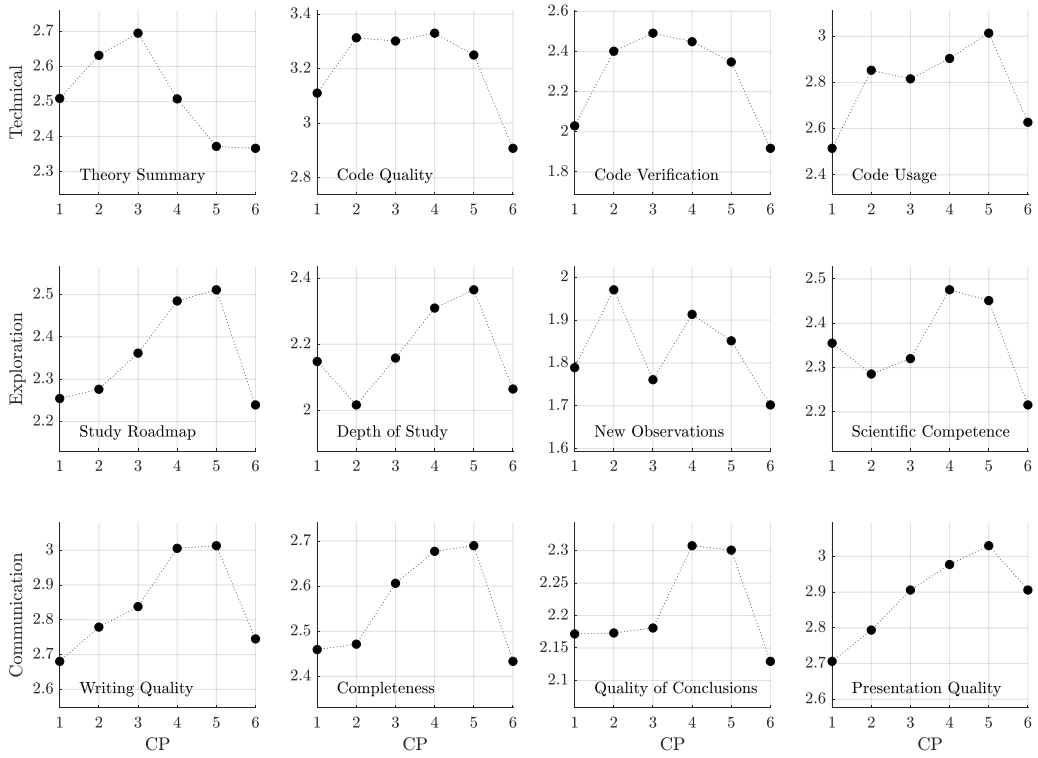

Figure 3. Progression of average scores in the twelve rubric areas across the six CPs in Deformable Solids. The average is over 549 students who completed the course with a grade of C or better over eight semesters between SP17 and FA20.

The results for statics are shown in Fig. 4. In statics, the scores drop for the third computing project but are otherwise consistent in most of the rubric areas. The notable large increases happen for code quality and code verification after the first computing project. Since statics is the first course that introduces students to projects of this nature, the first project is a new learning experience in coding, verification, and exploration. The students quickly realize how to verify and ensure they have a correct code for the future projects after receiving their feedback from CP 1.

The decrease in scores for CP 3 can be attributed to the type of project it is compared to the other projects in the course. It is the truss analysis program that introduces some linear algebra concepts that students have not had much exposure to, and the code theory does not follow the same method as hand calculations for trusses. Also, the route for verification and exploration is more open-ended than the other projects (at least that is how students view it) and this is reflected in their overall scores. The value of this project, however, can be built upon in future courses for more complex system analysis and often students reflect on this project as they get into those upper-division classes.

There are several sources of information that provide observations about student success through the computing projects. For example, written comments are provided on each project report. These written comments are a source of qualitative information on how student performance evolves through the semester. It is also common for students to go through all three courses at the same institution, which provides an opportunity to see how they progress over two semesters. Finally, some of the students go on to graduate studies at the same institution and the instructors who teach both the undergraduate and

graduate mechanics courses observe that the students who have had the experience of doing computing projects at the undergraduate level perform much better on computing projects at the graduate level than do students who have not had the same experience. While these sources are less formal, they do confirm that the computing projects improve students' ability to execute open-ended inquiry and to communicate the results in written reports.

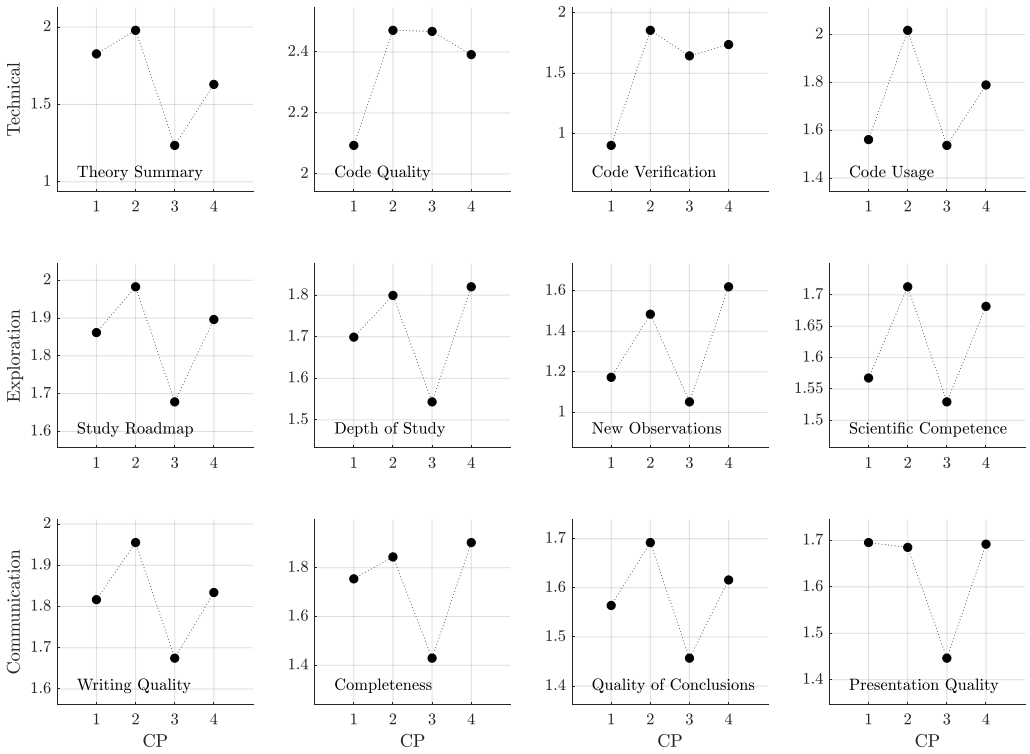

Figure 4. Progression of average scores in the twelve rubric areas across the four CPs in Statics. The average is over 260 students who completed the course with a grade of C or better over three semesters between FA19 and FA20.

At Institution A, students in the program had no exposure to computing before we started *The Mechanics Project* in 2012. Instructors who taught upper division courses claimed that students would not use a computing tool like MATLAB even if it was the best choice for solving the problem at hand and even if the instructor strongly encouraged them to do so. Now, many upper-division instructors include computing projects in their courses because the students are prepared for it (and the instructors do not have to provide any additional instruction). This trend indicates not only an elevation of the abilities of the students in computation, but also a change in the learning culture around the use of computational tools.

Course survey results

At the end of each semester we administer a survey in each course that is aimed specifically at asking students about the specific course features. We get extensive written responses on each course feature and the response rates are typically between 90 and 100 percent. The information we gather provides deep insights into how the students respond to specific course features.

The statistical information that we gather shows that students value the computing projects (with an average rating of 2 on a Likert scale of 1=strongly positive to 5=strongly negative. In reading the comments, it is clear that most of the negative sentiment relates to workload issues and not value. The

specific comments give some insight about how students value the computing projects. Three students in *Deformable Solids* (FA20) wrote:

> "Computing projects offered insight on problem solving."
>
> "The computing projects allowed me to understand the material better. It gave a different perspective."
>
> "The [computing] projects are neat applications of the course material as it takes the beam problems off the page in a way, and often the CPs can model more than one type of problem. I think it's also practical to work with writing codes. Even if we never use MATLAB again after undergrad, there will be a need for writing programs like these later in our careers. At least I hope so."

Two students in *Dynamics* (FA19) wrote:

> "I liked the computing projects because they are an excellent way of applying MATLAB coding and technical writing to the concepts taught it class. I believe by telling what the code to do and then explaining it in a technical paper, I gained a better understanding of the mechanics and theory of the concepts. I also liked having the opportunity to improve my technical writing skills."
>
> "[Computing projects were] a good way for me to show how much I know on each subject. I am not super interested in solving one individual problem algebraically, but when I have a chance to code up a general solution to the problem which could later even be added onto it feels more meaningful."

These comments, while anecdotal, speak to the value of learning mechanics through the computing projects. They show that students see the computing projects as a mechanism that has value beyond simply learning a set of computational skills. Indeed, as a source of *insight*. Students practice communication and by doing so see mechanics in different ways. The last comment reminds us that some of the tasks traditionally associated with learning mechanics (i.e., lots of algebra) actually demotivate some students and possibly contribute to loss of interest in engineering.

Since all of our students experience the new learning environment, we have no control group to assess change in performance relative to traditional approaches that do not include computing projects. However, course exams indicate that the addition of computing projects has not degraded student performance in executing hand calculations for exam-sized mechanics problems. The fundamentals of engineering (FE) exam results for Institution A confirm that performance levels have held steady through this instructional transition. The computing projects may not increase student performance on traditional mechanics tasks, but they significantly increase student abilities in areas not previously included in the course (e.g., computing and writing).

Conclusion

After the first year of CPs in dynamics at Institution A, the success and benefits of the projects were clear and the investment of effort in them appeared to be justified, despite considerable pushback from the students and the faculty colleagues not associated with the project. Student resistance dissipated fairly quickly as expectations around the courses, as communicated from one class to the next, stabilized. Students continue to view the computing projects as difficult and time-consuming and they continue to struggle with learning to operate effectively in the MATLAB environment. But students show:

- Increased ability to learn programming, in part because a peer network exists to support it and in part because the teaching has adapted to help shore up weaknesses.
- Increased ability to execute open-ended inquiry.
- An understanding and appreciation for the code verification.
- Improvement in their written communication skills.

Introducing computing in the foundational mechanics courses has been accomplished without dilution of topical coverage and has enabled instruction in computation without adding a course to the curriculum. This model demonstrates the power of problem-based learning by adding tools to existing courses that open up new learning modalities and opportunities to practice skills that should be present in all courses.

References

[1] A. Craig *et al.*, "Computing Across Curricula," in *American Society for Engineering Education*, 2008.

[2] J. M. Wing, "Computational Thinking," *Commun. ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[3] J. Zhao, "Computer application in solving statically determinate truss problems using FEA and MATLAB," *Comput. Appl. Eng. Educ.*, vol. 17, no. 4, pp. 363–371, 2009.

[4] A. Mobasher, A. R. Jalloh, E. Rojas-Oviedo, Z. T. Deng, and C. Qian, "Incorporating MATLAB in the Mechanical Engineering Courses at Alabama A&M University," in *American Society for Engineering Education Annual Conference & Exposition*, 2002.

[5] A. Nagchaudhuri, "Dynamic modeling and analysis of a crank slider mechanism," in *ASEE Annual Conference Proceedings*, 2000.

[6] R. G. Jacquot and B. R. Dewey, "Solution of static and dynamic beam bending and static buckling problems using finite differences and MATLAB," in *ASEE Annual Conference Proceedings*, 2001.

[7] M. Sathyamoorthy, "Integrating MATLAB in mechanics and structural analysis courses," in *ASEE Annual Conference Proceedings*, 2003, pp. 11099–11109.

[8] K. A. Wingate, A. W. Johnson, L. R. Ruane, and D. Akos, "Assessment of programming pre-requisites and interventions for student success in an aerospace curriculum," in *ASEE's Virtual Conference*, 2020.

[9] M. Rhudy and R. Nathan, "Integrated development of programming skills using MATLAB within an undergraduate dynamics course," in *ASEE Annual Conference and Exposition, Conference Proceedings*, 2016.

[10] J. B. Dabney and F. H. Ghorbel, "Enhancing an advanced engineering mechanics course using MATLAB and Simulink," *Int. J. Eng. Educ.*, vol. 21, no. 5, pp. 885–895, 2005.

[11] K. D. Hjelmstad and A. Baisley, "The mechanics project: A pedagogy of engagement for undergraduate mechanics courses," *ASEE Annu. Conf. Expo. Conf. Proc.*, June 2020.