# Coding Competency and Confidence to Prepare for Opportunity

**Dr. Jonathan Weaver-Rosen, Texas A&M University**

Jonathan Weaver-Rosen is an Instructional Assistant Professor in the Department of Mechanical Engineering at Texas A&M University. His research has focused largely on design automation and methodologies – specifically parametric optimization and the design of morphing or otherwise adaptive systems. His passion for teaching lies in preparing future design engineers to properly utilize analysis tools and work effectively as part of a team.

**Dr. Arkasama Bandyopadhyay, Texas A&M University**

Dr. Arkasama Bandyopadhyay is an Instructional Assistant Professor in the Department of Mechanical Engineering at Texas A & M University (TAMU). She previously earned a B.S. in Mechanical Engineering with a minor in Mathematics from Oklahoma State University and a Ph.D. in Mechanical Engineering from the University of Texas at Austin. Arkasama is interested in engineering education and is currently working on a project introducing responsible Generative Artificial Intelligence (AI) usage in undergraduate mechanical engineering courses to improve student learning outcomes. Additionally, she is collaborating on a project exploring the effect of implementation of an autograder for an open-ended collaborative term project in a junior-level Numerical Methods class. Previously, Arkasama has worked on projects involving Bloom's taxonomy-based assessments, distributed energy resources, residential demand response, and building energy systems. In addition to academic research and teaching, she enjoys mentoring students in engineering disciplines within and beyond TAMU.

# Coding Competency and Confidence to Prepare for Opportunity

This is a Work in Progress paper.


## Abstract

This paper explores the effect of autograders on student coding abilities, coding confidence, and overall learning experience in a junior-level mechanical engineering class on numerical methods with individual assignments and an open-ended collaborative term project. In coding-intensive large enrollment courses where human grading might take several weeks, autograders can provide immediate and consistent feedback. The real-time feedback allows students to learn from their mistakes and rectify those mistakes, thereby improving understanding and coding confidence. Autograded assignments can also reduce the grading time and effort for graders and teaching assistants in addition to lowering the burden for instructors to provide quick feedback on intermediate attempts of student codes during office hours. However, existing studies highlight potential concerns such as fostering students' overreliance on the autograder instead of encouraging independent debugging efforts. Additionally instructors using autograders must provide additional instruction with the assignment so that students are able to effectively cater to the autograder requirements. In this study, we aim to analyze whether the availability of an autograder implemented with Gradescope and developed for mechanical engineering students with limited formal coding exposure can improve student coding confidence, perceived coding ability, and student engagement. Additionally, we investigate whether the autograder reduces debugging time on assignments, promotes over-reliance on the instant feedback, and whether hidden tests for some tasks could mitigate any overdependence. Preliminary results from an end-of-semester survey administered to students suggest that the autograder increased student coding confidence and abilities, helped them remain interested in project goals and deliverables, and reduced debugging time before submission. Additionally, students reported that the autograder on individual assignments helped them learn the basics of Python programming and fundamentals of mathematical concepts taught in the course. Interestingly, students reported that the hidden autograder tests did not help them attain independent debugging skills, but student confidence was largely unaffected by removing the autograder in subsequent assignments.


## Introduction

Coding and the implementation of computational methods to model, analyze, and ultimately design real-world systems is a skill taught in most accredited undergraduate mechanical engineering curriculums across the world. The primary reason for this trend is that lacking general proficiency in at least one programming language would largely hinder employment opportunities in today's digital age. Although the nature and number of coding languages taught can vary from one institution to another, multiple coding-heavy formative and summative assessments are usually administered to students to gauge their competence. In coding-intensive large enrollment classes, manual grading and providing detailed feedback by teaching assistants can often take several days or weeks for each assignment. This lag in the grading turnaround time is detrimental to student learning (often students do not remember what they submitted a few weeks ago), does not allow students to continually improve their work by implementing

feedback, and logistically limits the number and length of assignments that can be administered in a given semester. Further, inconsistency in grading across multi-section courses can be introduced due to human bias, even if detailed grading rubrics are provided by instructors. In such scenarios, autograders can provide immediate, consistent, and meaningful feedback to students, thereby allowing them to refine their work and deepen their understanding. Autograded assignments can also reduce the grading burden for teaching assistants in addition to lowering instructor effort to provide quick feedback on intermediate attempts of student codes during office hours. Thus, instructors are able to utilize their time and effort to update lecture content, develop novel assessments, and devise active learning strategies to make the classroom more engaging.

## Literature Review

The idea of automatic grading itself is not new. Publications from the 1960s discuss the use of automatic grading for programming assignments to manage growing class sizes [1-2]. Since then, many automatic grading tools have been introduced for various purposes including, but not limited to, programming assignments [3-7]. Autograders have the potential to increase student motivation [8-9], enhance teaching and tutoring sessions [9-10], and improve student perception of the course [9]. However, developing autograders can be challenging since students may find correct solutions not counted by the autograder [11]. Students may also abuse the autograder or become over-reliant on the autograder to complete assignments [12-14]. Nevertheless, we seek to employ autograders in this study to take advantage of its benefits while trying to mitigate the disadvantages.

In this study, we investigate the effect of implementation of autograders for individual homework assignments and an open-ended collaborative term project on student coding abilities, coding confidence, and overall learning experience in a junior-level mechanical engineering numerical methods class. The novel contributions of this paper are as follows:

1. While autograders have been in use in computer science/engineering (and other computing heavy) disciplines for several decades, we put the autograders into practice in a class of mechanical engineering students with limited formal coding exposure.
2. The autograders for this course are created with simple Python scripts and implemented via Gradescope [15-16]. Thus, other engineering educators (even with limited coding experience) can easily automate grading of coding assignments by adopting the methodology and tools described in this paper.
3. Instead of analyzing the effect of this intervention on student grades, we explore the consequences on ‘unconventional’ metrics like students' perceived coding abilities, coding confidence, comprehension of fundamental concepts, interest in project goals, self-reported debugging time, etc.
4. We implement the autograders for the first two phases of the open-ended team design project (and select tasks on individual assignments) and analyze the effect of unavailability of the autograder for the latter phases (and portions of individual homework) on student coding confidence.

## Autograder Implementation

The junior-level numerical methods course covers root finding methods, integration and differentiation techniques, ordinary differential equations (initial value and boundary value) problems, linear algebra, linear regression, interpolation, and optimization. The students are expected to work on individual assignments - approximately biweekly - including coding problems and hand-calculations covering each of these topics. Additionally, the course involves a small-group multi-phase project where students combine their mathematical acumen with Python coding skills to design a code-based Martian rover.

The autograders – implemented via Gradescope [16] in this work – are designed to check for proper behavior of student-defined Python functions which can be easily imported and used within the autograder alongside correct versions of each function. For example, in an early assignment students are expected to write a function to perform bisection root finding on a user-defined equation. In the assignment, instructions are given to the proper calling syntax for said function detailing the inputs and outputs. The autograder then performs several tests: i) do the inputs and outputs behave as expected, ii) can the student code correctly find the root within some tolerance to problems with a known solution, and iii) does the student code provide meaningful error messages to a user when something goes wrong? For each test (and sub-test), the instructor can assign point values and toggle its visibility to the student. Additionally, Gradescope generates a similarity report to detect how similar student code is to other students and other sources. See the appendix for a brief example of an autograder function and corresponding assignment.

To mitigate some reliance on the autograder, some tests are not visible until after the final submission. Students are informed of the existence of such tests and encouraged to perform their own debugging and testing based on the assignment instructions. As the course progresses, the hidden portion of the autograder grows until the autograder is ultimately removed entirely by the end of the course. See Figures 1-2 for a brief example of an autograder response for the same test when passed or failed.

## Surveys

Two end-of-semester surveys, one geared towards the individual homework assignments and the other for the collaborative course project, are generated using Google forms and made available to students through the Canvas learning management system. Each survey consists of seven to eight multiple choice questions and one open-ended question. The purpose of the two surveys is to gauge students' perceived coding abilities, coding confidence, comprehension of fundamental concepts, interest in project goals, self-reported debugging time, and independent debugging skills. The motivation behind the research study and goals of the project are introduced to students in lecture. Additionally, the voluntary and anonymous nature of student participation is emphasized. Despite multiple reminders given in lecture as well as an additional reminder via a Canvas announcement after the final exam, the response rate is extremely low. Across two course sections in Fall 2024 (with the same instructor) of 177 students total, there are six responses for the project survey and ten responses for the individual assignment survey. Thus, while we discuss our observations below, the results cannot be generalized. More data needs to be collected from different student groups in future semesters to unearth any underlying trends.

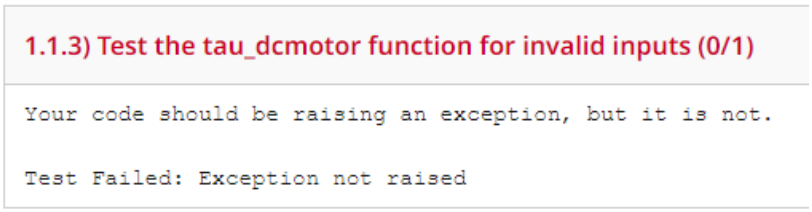


***Figure 1**: Example of the autograder output to a failed test*

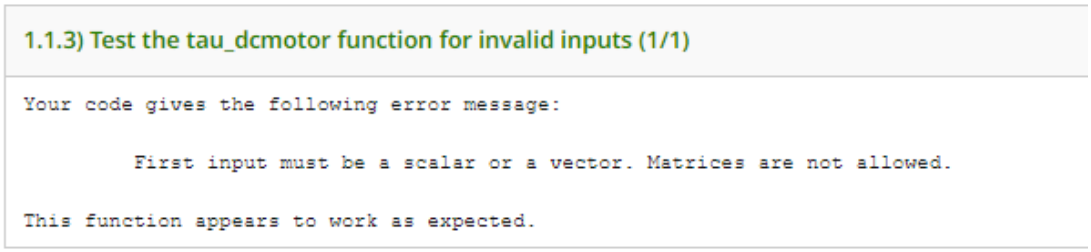


***Figure 2**: Example of the autograder output to a passed test*

*Project Survey*

As shown in Figure 3, five out of six respondents agree (strongly or somewhat) that the availability of the autograder for the first two project phases improved their coding abilities by allowing them to rectify their mistakes immediately and resubmit. Further, four out of six students agree that the autograders increased their coding confidence. The positive feedback regarding coding confidence and self-perceived coding abilities is promising since mechanical engineering students generally have limited exposure to programming languages across the undergraduate curriculum. However, regardless of their plans after graduation (graduate school or corporate sector) and nature of work (computational vs experimental), most engineers can utilize programming proficiency to maximize their professional worth. Four out of six participants agree that the instant feedback available from the autograders helped them remain interested in the project goals and deliverables. Additionally, five out of six respondents agree that the autograders reduced the time needed to debug their code. While this result can be deemed to be positive from the point of view of student time invested and student frustration/mindset, the authors believe that spending sufficient time identifying and resolving programming errors should be an essential part of the student learning experience in any coding-intensive course. Four out of six respondents agree that hidden tests helped them attain independent testing/debugging skills and not become over-reliant on the real-time autograder feedback. However, it is interesting to observe from Figure 3 that a majority of participants are of the opinion that the unavailability of the autograder for the open-ended later project phases did not reduce their coding confidence. The increase in student coding abilities and confidence levels due to the early autograders might have helped the students succeed in the later assignments without missing the autograder. This is analogous to learning to ride a bicycle with training wheels to gain enough experience and confidence to safely remove the training wheels. Finally, four out of six students are neutral regarding whether the unavailability of an autograder for Project Phase 3 (mostly open-ended), helped them attain independent testing/debugging skills and not become over-reliant on instant feedback.

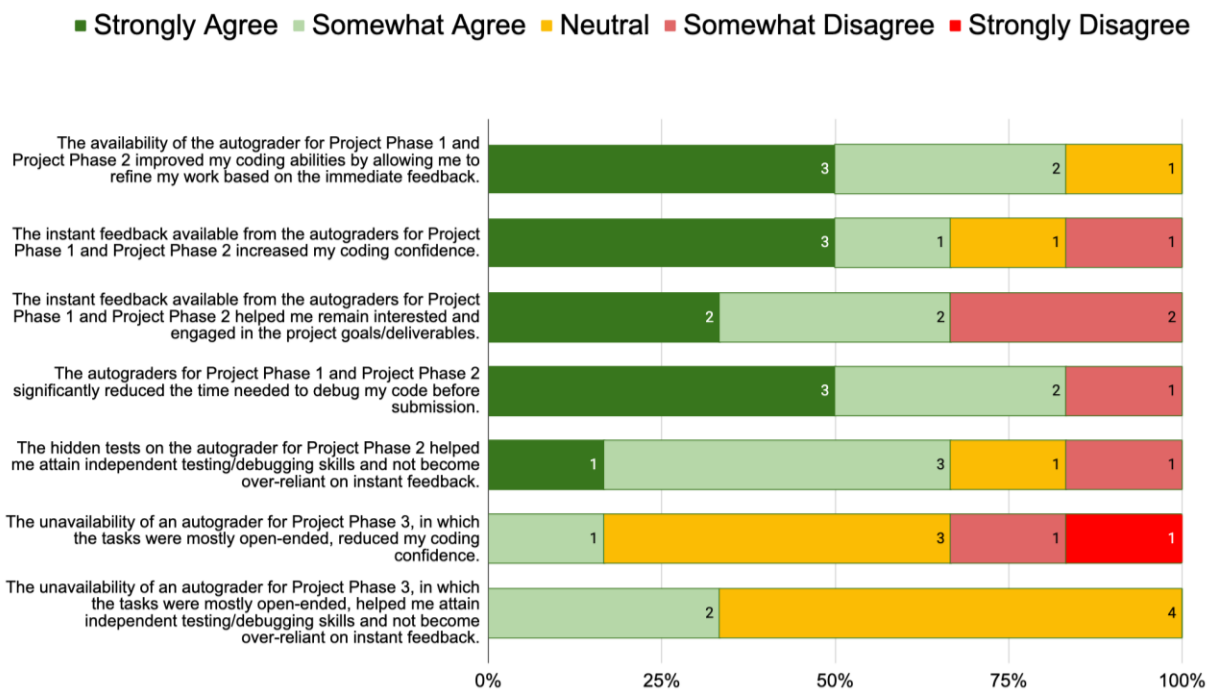


***Figure 3**: Survey responses from the project autograder survey (n=6 out of 177 students)*

Only two students responded to the open-ended question on the survey asking if there was anything else they would like to share about the autograders in general for the project phases. One student commented that '*The autograders were incredibly helpfu*l' while another mentioned that '*I think the auto graders should give hints as to what is required otherwise even debugging multiple times ends up giving the same errors*.' The authors are of the opinion that in a junior-level mechanical engineering class, an autograder is not meant to *fix* student code. Instead, we want students to exercise their critical thinking to build their own debugging skills given minimal direction. In the root-finding example, students are not told which equation(s) the autograder is using to determine their function's accuracy. However, students are given a list of equations to perform their own tests. It is our expectation that if the autograder says "*Your bisection code is not finding the correct root*", then the students should investigate their bisection code on the given equations to determine where the error lies. While this message does not give any specifics, students should be able to see the entire (visible) autograder output to determine that the function is executing correctly and the input/output structure is correct (for example). Therefore, something must be wrong in the method or data handling. If students continue to struggle, we hope that students will come to instructional office hours for further assistance. The autograder has the potential to make these interactions more efficient since instructors can see the student code alongside the backend of the autograder to see exactly where students are having trouble. Instructors can also verbalize their own debugging thoughts (similar to above) to teach students how to think critically about the information they have for future debugging.

*Individual Assignment Survey*

Figure 4 shows the results from the individual homework assignment survey. Similar to the results from the project survey, student feedback was positive. Six out of ten students agree that the autograders helped them learn the basics of Python programming and fundamentals of the mathematical strategies coded in the respective assignments. Students overwhelmingly agree that the autograders improved their coding abilities (nine out of ten) and coding confidence (eight out of ten). This is the desired outcome since all engineers can benefit from being proficient in at least one programming language. Eight out of ten respondents agree that the autograders significantly reduced the time needed to debug their code before submission. Interestingly, six out of ten students disagree when asked whether it was challenging to restructure their code to meet the specifications of the autograders. This result reveals that the autograder in its current form is not over-particular and is accepting of various student coding styles. Only two out of ten survey participants agree that the autograder made them over-reliant on instant feedback to debug their code. While this result cannot be generalized due to limited data, it is still encouraging for instructors since students must ultimately be trained to independently design, model, analyze, and evaluate real-world engineering systems using programming languages without the help of an autograder. As observed from Figure 4, student feedback about the hidden tests on the autograders is more diverse. Finally, six out of ten students do not agree that the unavailability of the autograder for some assignments reduced their coding confidence. Based on limited responses on both project and individual assignment surveys, it can be observed that while the autograders are able to increase coding abilities and confidence, the unavailability does not hamper student confidence levels.

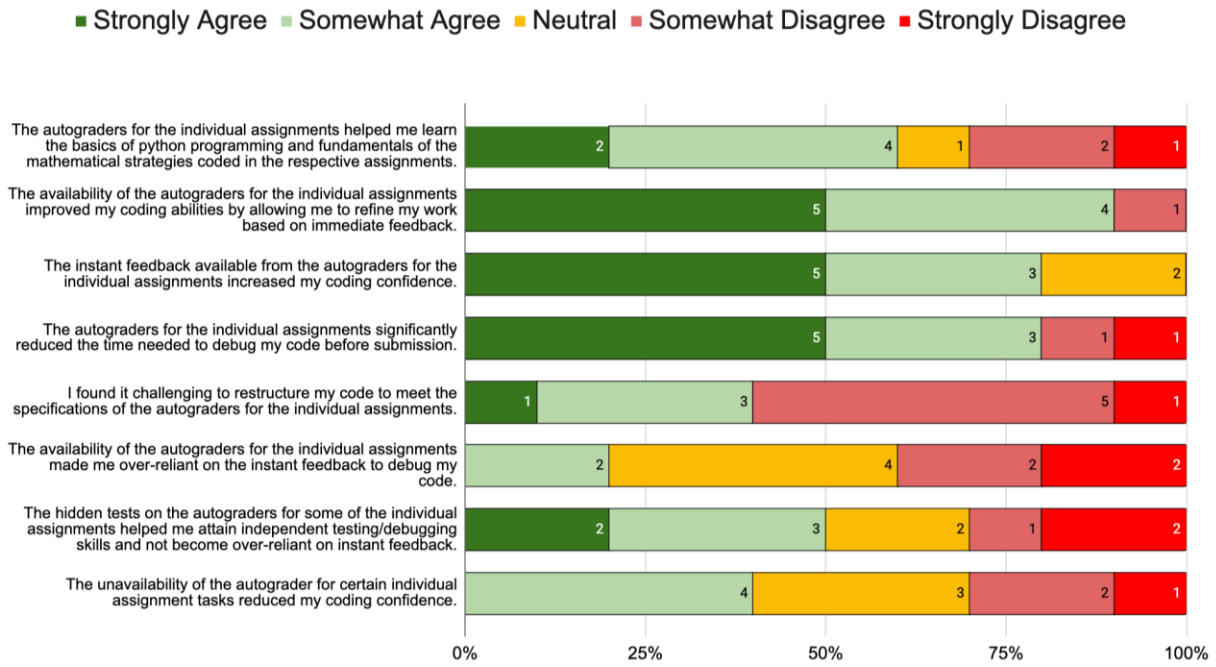


***Figure 4**: Survey responses from the individual assignment autograder survey (n=10 out of 177 students)*

Figure 5 shows the responses from the open-ended question on the survey asking students if there was anything else they would like to share about the autograders for the individual assignments. Seven out of ten survey participants provided feedback on this question. Multiple students (Students 2, 4, and 6) expressed frustration regarding the purposefully vague feedback from the autograder on incorrect code. As discussed previously, the authors' teaching philosophies are not aligned with telling students exactly where they went wrong and how to rectify their code to achieve full points on an assignment. The authors believe that this strategy does not help students develop their critical thinking and independent analytical skills. However, the constructive feedback about the autograder not accepting correct answers due to methodology differences (Student 3) and small rounding issues (Student 5) will be used to improve the autograder (and consequently student learning experiences) in future semesters.

Student 1: *Having specific output requirements for the autograders were extremely confusing and have led to hours of useless debugging that could have been solved had we been provided the desired output format*

Student 2: *I understand making the feedback vague to encourage independent debugging, but there were several instances where it basically says 'it isn't right' and i've spent hours trying to figure out what wasn't right, only to realize it was a very little thing like output formatting or something along those lines*

Student 3: *The autograders were incredibly helpful, but sometimes didn't accept correct answers due to methodology differences.*

Student 4 : *Overall they are helpful, however they can be detrimental to the work on my code if the auto-grader doesn't give a helpful reason why the code is wrong.*

Student 5: *The autograder for some assignments wouldn't take into account slight rounding. It would be great if the autograder could have a margin of rounding error taken into account.*

Student 6: *The auto grader should give a hint as to what is required because even debugging multiple times outputs the same error, which actually means that the auto grader makes debugging harder. It would be better if a human tested the output.*

Student 7: *hidden autograder isnt effective, all it does is cause confusion as to whether the code is fully right or not*

***Figure 5****: Survey responses from the individual assignment autograder survey open-ended question*

## Limitations

Our empirical teaching intervention study is subject to certain limitations. First, despite multiple reminders given to students, the survey response rates are very low (n=6 and n=10 out of 177 students). This low participation rate leads to the inability to make generalized claims. One potential reason could be that no extra credit or other incentive was offered for attempting the surveys. Additionally, the surveys were introduced towards the end of the semester when students have several project and homework deadlines and are also encouraged to complete other course evaluation surveys. In the future, students will be exposed to these surveys earlier in the

semester and will be allotted in-class time to respond, if they wish to. Secondly, the number of attempts that each student had to resubmit their assignment based on autograder feedback was not restricted. Thus, it is unclear whether some students critically reflected on the autograder feedback and modified their submissions or kept using trial and error till they got full points. Finally, the current autograder is unable to accept slight deviations in numerical responses or usage of novel Python packages. This often led to frustration for students. In those scenarios, when the concerned student raised the issue with the instructor, graduate teaching assistants were instructed to manually grade the submissions. We plan to continually update the autograder in future semesters based on student feedback to make it more inclusive.

## Conclusions

This work details an inaugural step towards effectively using programming autograders in a junior-level mechanical engineering course. Although future work needs to prioritize and further encourage student response rate, we observe worthwhile preliminary results with the proposed method. The autograder allows us to more quickly assess a large class and encourages student learning by giving opportunities to fix their mistakes for credit. To encourage students' independent debugging skills, the proposed work includes several hidden tests that students cannot see until after grades are returned. This framework allows students to identify big picture mistakes and gain confidence in their work by using the visible autograder while simultaneously encouraging them to validate the entirety of their code to satisfy the hidden autograder. Additionally, the autograder is used more heavily in the beginning of the course when assignments are more procedural and deterministic, and the autograder is taken out entirely by the end of the course when students are asked to solve realistic open-ended problems. Therefore, the autograders are analogous to training wheels that are gradually removed such that students are forced to take full ownership of their coding abilities by the end of the course. The student responses are encouraging, albeit inconclusive due to the sample size. Overall, the survey respondents found the autograders helpful and identified some issues that can be resolved in future course offerings. Although there was some frustration with the vague feedback provided by the autograder, this is a desired feature of the autograder (to promote students' critical thinking skills) that we will better explain to students in the future. We will also stress that the autograder interface is not the *only* way students can receive feedback on their assignments but it is the fastest. Students will be encouraged to use the autograders first and then seek human help through an instructor or teaching assistant when struggling. Finally, in future semesters, we plan to use additional tools like student assignment grades or exit interviews to further investigate the effectiveness of the autograders.

## Acknowledgement

The National Science Foundation Grant EEC-2022275 supported the initiation of this project. The authors would like to acknowledge the contributions of Devesh Bhasin during the ideation stage of the autograder. Finally, the authors are grateful to the other course instructors who provided feedback on initial implementations of the autograder.

## References


[1] J. Hollingsworth, “Automatic Graders for Programming Classes,” *Communications of the ACM*, vol. 3, pp. 528-529, Oct. 1960.

[2] G. E. Forsythe and N. Wirth, “Automatic Grading Programs,” *Communications of the ACM*, vol. 8, pp. 275-278, May 1965.

[3] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppala, “Review of recent systems for automatic assessment of programming assignments,” in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, ACM, 2010. pp. 86–93.

[4] J. C. Caiza and J. M. Del Alamo, “PROGRAMMING ASSIGNMENTS AUTOMATIC GRADING: REVIEW OF TOOLS AND IMPLEMENTATIONS,” in *Proceedings of the 7th International Technology, Education and Development Conference, INTED2013*, Valencia, Spain, March 4-5, 2013, pp. 5691-5700.

[5] S. Combéfis, “Automated Code Assessment for Education: Review, Classification and Perspectives on Techniques and Tools,” *Software*, vol. 1, pp. 3-30, Feb. 2022.

[6] M. Messer, N. C. C. Brown, M. Kolling, and M. Shi, “Automated Grading and Feedback Tools for Programming Education: A Systematic Review,” *ACM Trans. Comput. Educ.*, vol. 24, no. 1, Mar. 2024. [Online]. Available: https://doi.org/10.1145/3636515/.

[7] R. Gao, H. E. Merzdorf, S. Anwar, M. C. Hipwell, and A. R. Srinivasa, “Automatic assessment of text-based responses in post-secondary education: A systematic review,” *Computers and Education: Artificial Intelligence*, vol. 6, pp. 100206, Jan. 2024.

[8] H. Keuning, J. Jeuring, and B. Heeren, “A systematic literature review of automated feedback generation for programming exercises,” *ACM Transactions on Computing Education (TOCE)*, vol. 19, pp. 1–43, Sept. 2018.

[9] G. Hagerer, L. Lahesoo, M. Anschütz, S. Krusche and G. Groh, "An Analysis of Programming Course Evaluations Before and After the Introduction of an Autograder," in *Proceedings of the 19th International Conference on Information Technology Based Higher Education and Training (ITHET)*, Sydney, Australia, 2021, pp. 01-09.

[10] R. Landa and Y. Martinez-Treviño, “Relevance of immediate feedback in an introduction to programming course,” *2019 ASEE Annual Conference & Exposition*, Tampa, Florida, USA, June 2019.

[11] X. Liu, S. Wang, P. Wang and D. Wu, "Automatic Grading of Programming Assignments: An Approach Based on Formal Semantics," *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, Montreal, QC, Canada, 2019, pp. 126-137.

[12] V. Karavirta, A. Korhonen, and L. Malmi, “On the use of resubmissions in automatic assessment systems,” *Computer Science Education*, vol. 16, pp. 229-240, 2006.

[13] K. M. Ala-Mutka, “A Survey of Automated Assessment Approaches for Programming Assignments,” *Computer Science Education*, vol. 15, pp. 83-102, 2005.

[14] E. Baniassad, L. Zamprogno, B. Hall, and R. Holmes, “STOP THE (AUTOGRADER) INSANITY: Regression Penalties to Deter Autograder Overreliance,” In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21)*, Association for Computing Machinery, New York, NY, USA, pp. 1062–1068.

[15] A. Singh, S. Karayev, K. Gutowski, and P. Abbeel, “Gradescope: A Fast, Flexible, and Fair System for Scalable Assessment of Handwritten Work,” In *Proceedings of the*

*Fourth (2017) ACM Conference on Learning @ Scale (L@S '17)*, Apr. 2017, Association for Computing Machinery, New York, NY, USA, pp. 81–88.

[16] "Gradescope Autograder Documentation," *Readthedocs.io*, 2024. https://gradescope-autograders.readthedocs.io/en/latest/

## Appendix

Write Python functions for the Euler, Midpoint, and Classic RK-4 methods of solving IVPs. Save all functions to a file called **A4.py**.

**Valid call:** `T, Y = euler_integrate(fun, t0, y0, tStop, h)`

**Inputs:**

- `fun` : (callable) ODE function handle
- `t0` : (float) Initial time step
- `y0` : (float or 1D numpy array) Initial value of y corresponding to t0
- `tStop` : (float) Final time step
- `h` : (float) Step size

**Outputs:**

- `T` : (1D numpy array) integrated t-values
- `Y` : (1D or 2D numpy array) integrated y-values

**Assumptions:**

- The ODE function is called by: `dydt = fun(t,y)`
- h will be provided such that the solver will not step past tStop

***Figure A1**: Example of assignment instructions given to students.*

```
class Test_Task1(unittest.TestCase):
    def setUp(self):
        pass

    visibility_setting = 'visible'

    @weight(3)
    @number("1.1")
    @visibility(visibility_setting)
    def test_euler(self):
        """Check euler_integrate"""
        # randomize the ODE to solve
        a = random.random()*15
        b = random.random()*15
        c = random.random()*5
        ode_test = lambda t,y: ODE1(t, y, a, b, c)

        # solve with the solution code
        sol1 = euler_integrate_sol(ode_test, 0, 1, 2, 0.4)
        sol2 = A4.euler_integrate(ode_test, 0, 1, 2, 0.4)

        check1 = np.allclose(sol1[0], sol2[0]) # check t-values
        self.assertTrue(check1, 'Your t-values are not lining up for the test problem.')

        check2 = np.allclose(sol1[1], sol2[1]) # check y-values
        self.assertTrue(check2, 'Your y-values are not lining up for the test problem.')
        print('Your function appears to work as expected.')
```

Assign point value

Set visibility to students

Set the test title (visible to students based on the setting above)

Create a stochastic ODE (to deter "cheating" the autograder)

Solve the IVP with solution code

Solve the IVP with student code

Write messages displayed on test failure

If all assertions are passed, display the success message.

***Figure A2**: Example of autograder backend (not visible to students).*

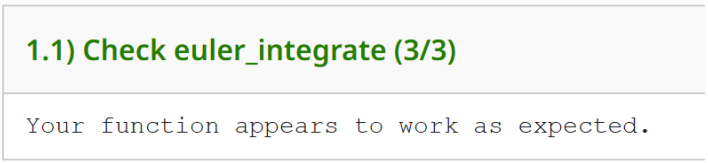


***Figure A3**: Example of autograder frontend (visible to students based on settings) when the test is passed.*

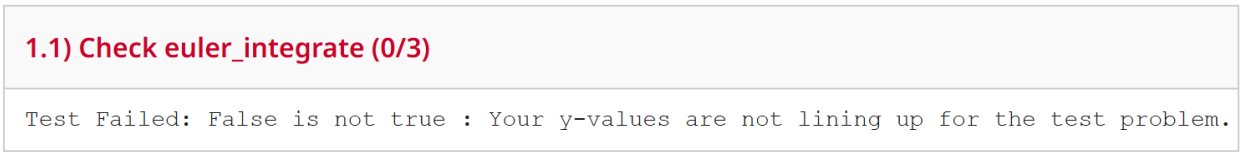


***Figure A4**: Example of autograder frontend (visible to students based on settings) when the test is failed.*