Engineering Educators Bringing the World Together
2025 ASEE Annual Conference & Exposition
Palais des congrès de Montréal, Montréal, QC • June 22–25, 2025 ASEE
Paper ID #47990

Iterative Driven Competency-Based Assessment in a First-Year Engineering Computation Module

Dr. James Bittner, Michigan Technological University

James Bittner is an Assistant Teaching Professor in the Engineering Fundamentals Department at Michigan Technological University. His recent courses focus on foundational engineering subjects, including statics, design practices, and computational problem-solving, emphasizing active learning methodologies in his classroom. He has research experience in explorative active learning practices, nondestructive testing of civil infrastructure materials and nonlinear wave theory. Prior to his academic career, he worked as an engineer in the maritime construction industry, specializing in hydraulic sediment transport and geotechnical analysis.

Dr. Matt Barron, Michigan Technological University

Dr. Barron's teaching interests include solid mechanics, engineering fundamentals, and transitional mathematics. His research interests include educational methods, non-cognitive factors, and bone tissue engineering. Prior to MTU, Dr. Barron worked for Bay de Noc Community College for eleven years and he also has several years of experience working for Kimberly-Clark Corporation in Research and Development.

Dr. AJ Hamlin, Michigan Technological University

AJ Hamlin is a Principle Lecturer in the Department of Engineering Fundamentals at Michigan Technological University, where she teaches first-year engineering courses. Her research interests include engineering ethics, spatial visualization, and educatio

Complete Evidence-Based Practice: Iterative Driven Competency-Based Assessment in a First-Year Engineering Computation Module

Introduction

In our connected world, engineers must possess a strong foundation in applied computation. Daily engineering decisions rely on data analysis, which necessitates the use of computational tools. This work investigates the transition from manual grading to a competency-based automated grading system for introductory computation engineering problems.

Introductory knowledge of programming and problem solving is often core components to a broad first-year engineering curriculum. While traditional computer science curriculum often emphasizes scientific theories, practicing engineers focus on the societal impact and practical applications of their work. As a result, teaching introductory programming concepts to young future engineers can be a challenging task.

One common tool to aid in teaching beginning programming theory is to program an automated assessment. Automated assessment tools have long been deployed successfully in teaching computer science curriculum [1]. The benefits of an automated assessment tool are documented to include more timely feedback, and building up the student's confidence [2].

Despite adoption within computer science curriculum, automated graders have had limited deployment within engineering programming curriculums [3]. Several common limitations to deployment are the upfront costs of developing and running a system, creating sample problems that are engaging for engineers and assessment checks that are relevant. Recent available commercial software, MATLAB Grader, has provided an accessible and unified framework for automated assessment of engineering programing assignment [4].

The Grader software package allows the instructor to build personalized a problem statement of their choosing. Then the instructor creates a reference solution of an answer in MATLAB code and designs a series of tests to check the students' submissions against the reference solution. Points can be awarded based upon the number of tests passed or only when all tests successfully pass. In the original design the comparison against the reference solution was a static behavior comparing numerical equivalency. Learners have unlimited attempts to submit and test their problem solutions, emphasizing application and utility, a key focus for engineers.

In addition to the software documentation, Bartolini initiated efforts to share a series of engineering focused introductory programming tasks [5]. Other efforts towards more robust testing of code submissions have created two independent external libraries GraderPlus [6], and Malab-Grader-Utils [7]. Each of these libraries provides a series of capabilities that can be leveraged by an instructor to create dynamic and enriching problem sets. GraderPlus was created with a focus on engineering evaluation and Matlab-Grader-Utils focused primarily on the mathematical processes behind a solution.

In the automated grading system concept there is a notable loss of the numerous 'human-in-the-loop' checks that a trained human grader would provide [8]. In submission of assignments validating integrity of the solution is critical to maintaining rigor within the learning environment.

In this paper we document our process of taking a rubric graded summative assessment and converting each rubric item into an automatically assessed MATLAB Grader test. Additionally, we will explore results of a small formative programming assignment that was assigned as a manual graded assignment and as an automatically assessed assignment in two sequential offering of our course.

Experimental Methods/Project Approach

Part 1 Automatic Assessment Tests
The first-year engineering program under study commences with a Fall semester focusing on problem solving and data analysis. This semester is divided into three modules: Mechanics and Energy, Computation and Sensing, Simulation and Integration. The first summative assessment in the Computation and Sensing module of our course focuses on variable assignment, conditional statements, plotting and basic loops.

The complete detailed example assessment problem statement and configuration are detailed in Appendix A. One constant challenge in designing problem sets is to keep the learning environment dynamic and having no single answer valid for all students. To reduce the chance or effectiveness of plagiarism and diversify the pool of problems a dynamic question and solution are needed. In most learning management systems, a common form of quiz is to form a quiz bank of possible permutations of any given question.

One way to implement a dynamic question behavior is to generate permutations of the input data provided to the student. In this example problem the student enters their student identification number (M-Number) into a variable to seed the generation of their specific problem data. This technique allows students to use a set of various predetermined inputs that remain constant during their individual testing. In our example a function, `get_temperature` in Appendix B, was created to generate permutations of input vectors for different sets of students.

The final challenge of creating a dynamic question behavior is to sync the input data between the reference solution and the student solution. The external library Matlab-Grader-Utils provides a function `get_str_value_from_learner` in the `RandomParameters.m` file. This function will open the student submission and extract the values of a specific variable. In our example problem, we extract the student ID variable `MNumber` and use that identical seed to populate the input vectors of the reference solution. The code to perform this synchronization can be summarized at the start of the reference solution as:

```
%% Reference Solution
MNumber = 'M12345678';
MNumber = RandomParameters.get_str_value_from_learner('MNumber').char()
[minutes, temp] = get_temperatures(MNumber);
```

The combination of the created `get_temperatures` function and the sync of the `MNumber` variable allow the problem inputs to become dynamic between students. Nevertheless, the problem values remain constant within each student's execution of the code, enabling efficient troubleshooting.

During previous years the evaluation of student submissions was carried out with an instructional team running each code sample and scoring the submission manually. The process of loading a student file, running the file and debugging which of the competencies were demonstrated took approximately 4 minutes per submission. The manual rubric used is provided in Table 1. In this sub-section we will demonstrate the assessment test code used to evaluate competency for each of the rubric items.

Table 1: Manual Grading Rubric for Summative Assessment of Computational Skills

Rubric Item	Description
Runs without errors	Submission executes completely without errors
Comments are present	Useful header comments and comments throughout code are present
Create or Access Vectors	Vectors are created and referenced to clearly show
Correct Data Plotted on Each Axis	Data plotted correctly (used correct x & y variables) and plotted with markers/symbols only
Threshold Plotted	Data plotted correctly (used correct x & y variables) and plotted as a line only (no markers)
Output Generated	Present and correct equation
Axis Labels	x- and y-axis labels contain category, symbol/variable name, and units

Rubric Item #1 Runs without Errors
The first rubric item is to provide assessment that the submitted solution runs without errors. This feature is built into the basic functionality of the Grader tool, once an error is reached the assessment stops. The assessment tests used to evaluate this is to check for a value of the provided input variable. The MATLAB code to accomplish this test in the provided example problem is:

```
assert(~strcmp(MNumber, 'M12345678'))
```

This line uses the assert function to test if the variable `MNumber` is set to the template student ID. If the student code successfully completes execution and the first variable defined remains present, then we can reasonably assume no error crashed the execution. Additionally, this forces the student to update the ID number with their own unique identifier.

Rubric Item #2 Comments are Present
The second rubric item is to assess if the student submission contains comments. For this check we will use a brief two lines of code to open and inspect the raw student solution file for any '%' characters. The MATLAB code to accomplish this test is:

```
linesWcomments = sum(contains(readlines('solution.m'),'%'))
assert(linesWcomments>7)
```

This test code opens the standard script solution file and counts the number of lines that contain the target character. The number of lines is then tested to be above a threshold of 7 lines. The student learner template in the example already includes 7 comments, so this would require the student adds additional comments to their added lines of code. This method can be easily modified if a specific number of comments are required to be in the submission.

Rubric Item #3 Create or Access Vectors

The third rubric item is to assess if the student submission creates or accesses vectors in the solution. This test code will use the `mg_SolutionContainsExplicit` function from the GraderPlus library. The code to accomplish this test is:

```
assert(mg_solutionContainsExplicit(…
    ".*temp(.*", 'Random', 'get'))
```

This code will search for any direct referencing an element within the `temp` array using a regular expression as the search parameter. The final two arguments of the `mg_solutionContainsExplicit` function specify partial filenames to be excluded during the search for all available ".m" files. This functionality is crucial because our example incorporates files from external libraries (two in this case) and our custom `get_temperature` function. By excluding these files, we ensure accurate and meaningful comparisons with just the learner solution code.

Rubric Item #4 Correct Data Plotted on Each Axis

The fourth rubric item evaluates if the correct data is plotted on the correct axis. Depending on the design of the question multiple data vectors need to be checked, and each check should be a separate unit test with specific points. Following standard textbook conventions, temperature should be plotted on the vertical axis as discrete points (without a connecting line) in the example problem [9]. This test code will use the `mg_isCurveInPlot` function from the GraderPlus library to evaluate all plot series. The code to accomplish this test is:

```
assert(mg_isCurveInPlot('YData', …
    referenceVariables.temp', 'LineStyle', 'none'));
```

The function `mg_isCurveInPlot` will cycle through all the plot series present in the current figure and look for a match to the parameters supplied as arguments. In the case of experimental temperature measurements, we need to identify that the correct vector values are plotted and that the style is only markers with no line. The correct vector values based upon the student's input ID are be obtained from the reference solution by the structure `referenceVariables.temp`.

Rubric Item #5 Threshold Plotted

The fifth rubric item is to evaluate if a threshold line is plotted. One challenge in creating test is that there are several ways to create a threshold line on a plot in MATLAB. The first is to create a data series of multiple points with only a line and no markers. The second is to use the built-in function `yline`.

To check for both cases we will combine our use of `mg_CurveInPlot` and the `mg_solutionContainsExplicit` functions as follows:

```
assert( or( mg_isCurveInPlot('Marker', 'none'), …
    mg_solutionContainsExplicit(…
        ".*yline(.*",'Random', 'get')…
    ));
```

In this test we are searching for the just the creation of a solid line in the plot, and we are not explicitly checking the contents of the data series. An additional check can be constructed to identify the value of the data series; however, the creation of the line is the basic competency we are aiming to assess in this test.

Rubric Item #6 Output Generated

The sixth item in the rubric is to evaluate the created program output statement. The most direct way of evaluating this is to require students to populate a formatted `OutputStatus` string variable with the results. The MATLAB code to test for successful completion is:

```
assert(…
  all(…
    [1,…
    contains(lower(UserStatus), 'quenching occurred for'),…
    contains(lower(UserStatus), compose("%2.0f", …
     referenceVariables.QuenchCount)),…
    contains(lower(UserStatus), 'minimum')…
    ])…
)
```

In this test code we assert that all the string comparisons are valid. The first comparison is that the desired format of the program output is used. The second comparison checks the required value and requested precision of the primary numeric answer. The third check verifies that the secondary answer is addressed in the output. This test requires that students use sprintf function rather than the more widely introduced fprintf function, since the output needs to be captured into a variable. Capturing output within a Grader test is not a simply supported feature. Fortunately, the arguments and formatting are identical between the two functions.

Rubric Item #7 Axis Labels

The seventh item in the rubric is to evaluate the contents of the axis labels on the plot. The evaluation compares the actual labels to the expected units or labels. This test can be done using the `mg_getPlotInfo` function from the GraderPlus library. The MATLAB code to test for the content of an axis label is:

```
a = mg_getPlotInfo()
assert(contains(lower(a.yLabel), 'temp'))
```

In this code the function `mg_getPlotInfo` returns a structured array of common plot properties, including the y-axis label. The code then compares the lowercased yLabel property to the target value that needs to be present in the axis label. Alternatively instead of the library function, this code could address the axis label string directly through the code: `gcf().CurrentAxes.XLabel.String`. The library allows ease of use and access to other

elements of the plot information. The test for each axis label needs to be flexible enough to accommodate all reasonable correct answers. Good practice has been found to check for just the units in the independent variable and the name in the dependent variable.

Additional tests and checks that can be tested through built-in functionality within the Grader product have been omitted from this example methods. These items include validating that a specific function call is present within the student code or comparing a student variable value against a reference solution value [10].

Part 2 Comparing Automatic vs Manual Assessment

The Grader tool was first implemented in the first-year engineering courses in the Fall Semester of 2019. The positive effects of quick feedback and multiple quick retakes attempts were observed in students' interactions with the tool. One concern that emerged through use was the introduction of an additional interface in the learning process. As a result of these concerns the design of curriculum has adjusted the amount of guided versus unguided programming time on task. To explore the isolated impact of the assessment environment we chose one formative assignment to study the implementation as an automatically graded assignment in the Grader environment. Then following year, we transitioned the assignment back to a manually grading instead to try and maximize unguided time on task. While this specific assignment was switched between automated and manual grading, the Grader tool continued to be used for other short, formative assignments in both years. Critically, all problem-solving assignments beyond plotting and looping were consistently graded manually.

The problem selected focused on using a `for` loop to calculate the temperature change of an object subjected to a vector of applied heat values. Identical problem statements were used, and a starting template was provided for both the automated and manual assignments. Since the automated version of the assignment provided a method to check answers, the manual version similarly included an answer for the 2nd element of the output vector. The assignment was provided as an in-class activity on the same day in the semester after the students have been introduced to looping methods.

The submissions were evaluated regarding submission performance, time to complete the assignment, and uniqueness. Submission performance was documented as an average recorded marks and the count of number of missing submissions.

Completion of the assignment and time to return graded feedback was calculated by using the data logging features of the Canvas Learning Management System. The time from when the assignment page was initially accessed to the time that the assignment was submitted was used as the duration of the student's work. This was filtered to exclude assignments durations longer than 2 hours (the duration of the course period) after initially accessing the assignment.

The uniqueness of submission was calculated by comparing each submission against the pool of other submissions. Files are first stripped of capitalization, white spaces, and lines that start with a comment. Then each line is compared against all other lines in the pool, the number of

matching lines for every submission pairing is recorded as a similarity score. This method of comparison was preferred by the authors over more rigorous algorithmic systems due to the introductory nature of this course and the simplicity of the assigned algorithmic problem.

Results and Discussion

Comparing an Automatic vs Manual Assessment

The tabulated results of the automated versus manual assessment groups of a formative `for` loop assignment are presented in Table 2. While the two assignment types had approximately the same number of total enrollments, the manually graded assignment type had a higher number of missing assignments. The automated assignment allowed for resubmissions; therefore it has been expanded into a first submission and a multiple submission scoring summary. The average submission score of the submitted work was logically observed to be higher for the multiple submission automated group than the single submission manual group. The lowest average score and largest standard deviation was recorded for the first submission of the automated group. The average automated multiple submission score enabled students to reengage with the assignment and to resubmit before the due date. One of the limitations of the manual grading is that the grading only occurs after the assignment is due.

Table 2: Summary of student performance in different assessment types

Assignment Type	# Students	Missing Submissions	Average Score	Std. Dev. Score
Manual	103	13	94%	1.4
Automated – First Submission	116	5	48%	38.7
Automated – Multiple Submissions	116	5	100%	0

The comparison of time spent to complete the formative `for` loop assignment is presented in Figure 1. The average for each category was 0.56 hours for the manual assignment and 0.25 hours for the automated assignment. The submissions in the automated environment were submitted faster on average. One theory among instructors is that the quick feedback leads to students making quicker connections to correct methodology. A competing theory is that students are perplexed when faced with an empty text-editor in a development environment outside of the browser-based Grader. Slower submission outliers exist in both groups up through approximately four times the average submission time. The remaining slower submission outliers suggest that the struggling students do not see a reduction in assignment duration.

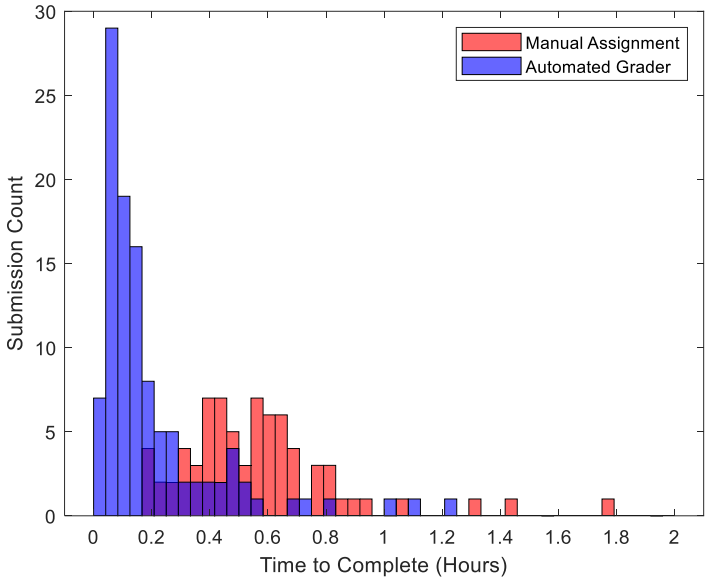

Figure 1: Comparison of time to complete the identical assignment performed as a manual assignment and as an automated Grader assignment. Dark blue areas represent overlapping data between the two groupings.

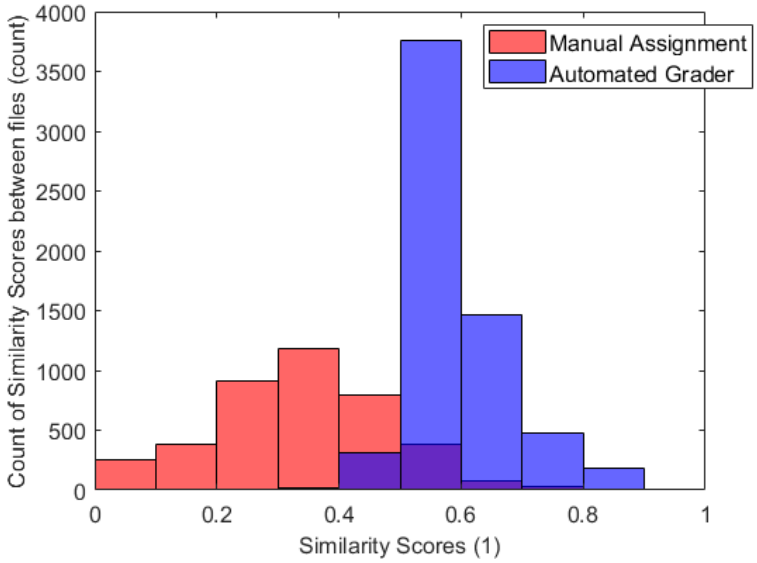

Figure 2: Comparison of similarity of file submissions an identical assignment performed as a manual assignment and as an automated Grader assignment. Dark blue areas represent overlapping data between the two groupings.

The uniqueness summary of all submission is presented in Figure 2. The similarity score represents the ratio of the number of matching lines out of the total number of lines present in a submission. The average similarity score for the manual assignment was 0.35 with a standard deviation of 0.11 and for the automated assignment the similarity score was 0.59 with a standard deviation of 0.08. In general, the submissions in the automated grader environment demonstrated

increased similarity. The reduction in uniqueness between submissions aligns with the previous qualitative concerns voiced about decreased integrity in automated assignments [8]. In these short formative tasks may force students to quickly align to a uniform method of accomplishing a task, further investigation is needed to confirm.

One of the features of an automated assessment is that students can correct misunderstandings and resubmit an assignment without manually interfacing with instructional staff. Figure 3 presents the distribution of submissions for each learner. The ability to iterate through a problem testing a solution for competency multiple times was utilized by 66% of the learners in the automated assessment. As the number of resubmissions increases the number of learners taking advantage of the ability decreases. These results suggest that the internal drive of students to correct their own mistakes is present within the moment. Additionally, qualitatively the instructors have witnessed the power of red and green automated checks driving student's emotions to better their own understanding.

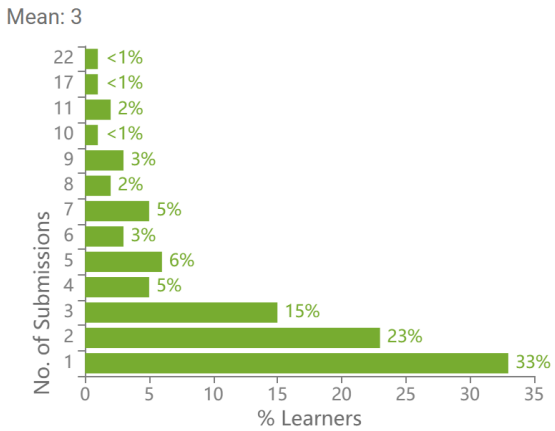

Figure 3: Number of submissions and resubmissions on a formative automated grader submission pool

The previous results demonstrated an increase in completion and a decrease in time spent on the assignment. Another critical aspect of the learning experience is the self-perception of how helpful a material or resource is to the learner. To explore this question at the end of the semester we request that students provide feedback on the components of our course. The completed responses to the survey question "I found the following course materials were helpful to my learning, select all that apply" are presented in Figure 4 as a histogram. The question was asked when the looping assignment was manually graded (96 responses) and when it was automatically graded (98 responses). The automated grader option, MATLAB Grader, ranked as the top 5th most helpful material towards student learning in both years. In the flipped classroom environment, the core learning materials are delivered through pre-class assignments and videos. Of note and interest in these results the helpfulness of the automated assignments was ranked higher than the classical pre-class content.

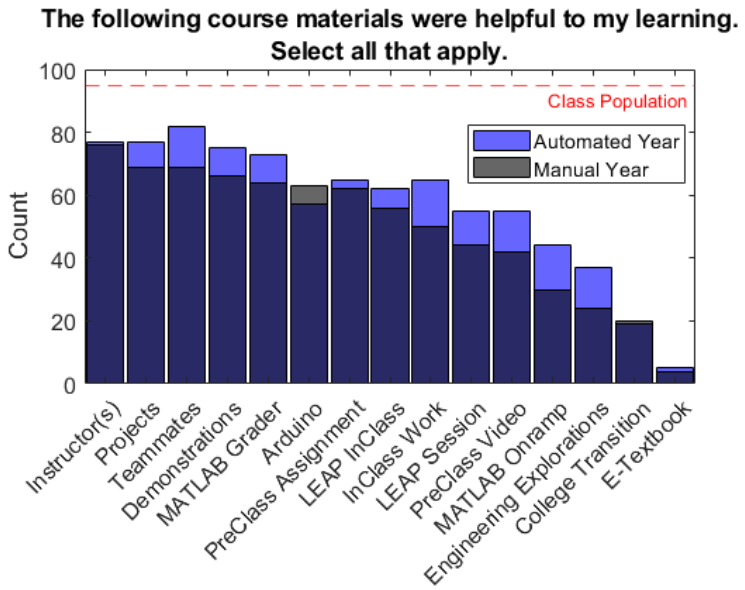

Figure 4: Summary of responses from an end-of-course survey for a first-year engineering course detailing the most helpful learning materials. Dark blue regions identify regions of overlap between the two datasets.

This analysis has several limitations. Firstly, the sample size of 103 and 116 first-year students is relatively small, limiting the generalizability of our findings. Secondly, the current test codes may not be sufficiently robust, potentially enabling students to find workarounds that satisfy the tests without demonstrating true competency of the underlying concepts. In most observed cases, this limitation was tested by learners with exceptional computational competency. Finally, the integration of multiple online learning platforms can lead to inconsistencies in terminology and potential confusion for students.

Summary

Automated assessment of coding problems has been a constant area of exploration and improvement for decades. In this work we discussed how we implemented a series of automated tests to evaluate MATLAB code from an introductory engineering competency perspective. Application of automated assessment reduced the time it takes for students to complete a task and engages students interactively. However, the uniqueness of the solution decreases as to align with the expected automated answer or reference concepts. In an introductory course this may be desirable to reward best coding practices. The iterative multiple attempts approach to solving an introductory problem was utilized by first-year engineers most of the time. The learner perception of automated assessment's helpfulness to their learning was observed to be positive over other materials, such as videos and textbooks.

These observations of an iterative competency-based automatic grading demonstrate improved performance for the first-year engineering learning environment. Future research should investigate whether these positive outcomes, such as improved student completion rates and learning performance, are consistent across different levels of problem complexity. Automated assessment tools have the potential to significantly increase the number and diversity of coding

assignments, thereby enabling instructors to offer more personalized learning pathways for each student within a broad First-year introductory engineering course.

References

[1] C. Douce, D. Livingstone, and J. Orwell, "Automatic test-based assessment of programming: A review," *J. Educ. Resour. Comput.*, vol. 5, no. 3, pp. 4–es, 2005, doi: 10.1145/1163405.1163409.

[2] C. Daly, "RoboProf and an introductory computer programming course," *SIGCSE Bull.*, vol. 31, no. 3, pp. 155–158, 1999, doi: 10.1145/384267.305904.

[3] R. S. Pettit, J. D. Homer, K. M. McMurry, N. Simone, and S. A. Mengel, "Are Automated Assessment Tools Helpful in Programming Courses?," presented at the 2015 ASEE Annual Conference & Exposition, Seattle, Washington, 2015/06/14, 2015. [Online]. Available: https://peer.asee.org/23569.

[4] *MATLAB Grader*. (2024). [Online]. Available: https://www.mathworks.com/products/matlab-grader.html

[5] A. Bartolini, "GIFTS - Integrating MATLAB Grader into an Engineering Computing Course," presented at the 15th Annual First-Year Engineering Experience Conference (FYEE), Boston, Massachusetts, 2024/07/28, 2024. [Online]. Available: https://peer.asee.org/48609.

[6] D. Kosfelder, "GraderPlus - A library for writing test code in MATLAB®-Grader," 2021. [Online]. Available: https://github.com/DavidKosf/GraderPlus

[7] A. D. Silva, "Matlab Grader Utils," 2023. [Online]. Available: https://github.com/alfonsovng/matlab-grader-utils

[8] H. Suleman, "Automatic marking with Sakai," presented at the Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology, Wilderness, South Africa, 2008. [Online]. Available: https://dl.acm.org/doi/10.1145/1456659.1456686.

[9] E. A. Stephan, D. R. Bowman, W. J. Park, W. D. Martin, and M. W. Ohland, *Thinking like an engineer : an active learning approach*, Fifth edition. ed. New York, NY: Pearson, 2022.

[10] Mathworks, *MATLAB Documentation: Test Learner Solutions*, 2025. [Online]. Available: https://www.mathworks.com/help/matlabgrader/ug/testing-learner-solutions.html.

Appendix A - Detailed Example First-Year Engineering Grader Problem

An example detailed summative assessment problem.

Steel is often quenched before it can be used for construction. Quenching hardens the steel but also makes it more brittle. Quenching is carried out by rapidly cooling the steel in a chilled water solution to a specific temperature (In our case we will use 222 Deg. F.), the time it takes for the temperature to drop is critical to the successful quenching process. The collected vectors **minutes** and **temp** contain data recorded from a steel sample exposed to a water chiller used in the quenching process.
How long does it take for this sample to drop in temperature to below the specific temperature? Temperature was measured in Fahrenheit occurred every 1 minute.

Write a MATLAB script below that:

- Has a proper file header with your name, date and email.
- Has useful comments for each line of code created
- Plots the recorded data of chiller temperature versus time.
- Plots on the same graph a horizontal line representing the final quenching threshold of 222 degrees F.
- Plot includes proper axis labels, titles, legends, and data style types (points vs. line).
- Write a for loop to iterate through the data and determine how long, in minutes, the steel is held at a temperature above 222 degrees F. Save the duration in a variable called **QuenchCount**
- Using either a function or loop, determine and save the minimum temperature reached in a variable called **MinTemp**
- Create an output string called **UserStatus** to update the user on the duration of quenching that occurred. Use the format "Quenching occurred for 23 minutes until 222 Deg. F. was reached with a minimum temperature of 131 Deg. F.". (These numbers are examples only)

External Library Files:
mg_SolutionContainsExplicit.m [6]
mg_plotExists.m
mg_getPlotInfo.m
mg_isCurveInPlot
RandomParameters.m [7]
get_temperatures.m [Appendix B]

Reference Solution:

```
% Reference Solution
% Name: Instructor, instructor@mtu.edu
% Date: 01/01/1971
% Program: Steel Quench Time

MNumber = 'M12345678'; % A Default Unique Student Identifier
% Try to use a learner solution MNumber value
MNumber = RandomParameters.get_str_value_from_learner('MNumber').char()
[minutes, temp] = get_temperatures(MNumber); % Update Input Vectors

% Create Plot
figure;
plot(minutes, temp, '*');
hold on;
plot([0 500], [222 222], '-');
xlabel('Time (minutes) [Minutes]');
ylabel('Temperature (temp) [Deg. F]');
title('Quenching Chiller Recorded Temperatures');
legend('Measurements', 'Quenching Limit');

% Analysis
QuenchCount = 0 ;
for i = 1:length(minutes)
    if(temp(i)>222)
        QuenchCount = QuenchCount +1;
    end
end
MinTemp = min(temp);

% Output
UserStatus = sprintf('Quenching occurred for %.f minutes until 222 Deg. F.
was reached with a minimum temperature of %.f Deg. F.', QuenchCount,
MinTemp);
disp(UserStatus)
```

Learner Template

```
% Student Solution Steel Quenching
% Name:
% Email:
% Date:
MNumber = 'M12345678'; % Update with your MNumber

%Load in Data
[minutes, temp] = get_temperatures(MNumber); % This will fetch a unique dataset for your ID
```

Appendix B - Supplemental Script: get_temperature.m file

```
function [myTime, myTemp] = get_temperatures(id)
%get_temperatures.m returns rotating input vectors based upon a input ID
%   This file is meant to be called by a student to populate data vectors
%within the Grader Environment. The function will return consistent vectors
%with a consistent input ID.
% Inputs:
%    id       : a character vector of numeric ID values
% Outputs:
%    myTime  : a numeric vector of incrementing time
%    myTemp     : a numeric vector of selected problem input data

% Build a matrix (m) of possible input vectors
m = [603.9  593.9 583.9 573.9 563.9 553.9 543.9 533.98 523.98 513.98
599.7 589.7 579.7 569.7 559.7 549.7 539.7 529.7 519.7 509.7
.. [data lines omitted] ..
595.1 585.1 575.1 565.1 555.1 545.1 535.1 525.1 515.1 505.1];

mynum = min(str2num(id(end))+1, 9);  % Use last digit of ID String
myTime = [1:400]';                   % Create a Time Vector
myTemp = m(:,mynum);    % Subset the matrix to a single input vector

end
```