Student Usage of Small Auto-graded MATLAB Coding Exercises

Dr. Alex Daniel Edgcomb, Zybooks

Alex Edgcomb finished his PhD in computer science at UC Riverside in 2014. Alex works with zyBooks.com, a startup that develops interactive, web-native textbooks in STEM. Alex has also continued working as a research specialist at UC Riverside, studying the efficacy of web-native content for STEM education.

Dr. Nikitha Sambamurthy, Zybooks

Nikitha Sambamurthy completed her Ph.D. in engineering education at Purdue University in 2017. Nikitha works with zyBooks, a startup that develops interactive, web-native textbooks for college courses in STEM (science, technology, engineering, and math) disciplines.

Mr. Dasharath Gulvady, MathWorks
Santosh Kasula, MathWorks

Santosh Kasula is a Software Engineering Manager for Online Learning Products at MathWorks. MathWorks is the leading developer of mathematical computing software. MATLAB, the language of technical computing, is a programming environment for algorithm development, data analysis, visualization, and numeric computation. Simulink is a graphical environment for simulation and Model-Based Design for multidomain dynamic and embedded systems. Engineers and scientists worldwide rely on these product families to accelerate the pace of discovery, innovation, and development in automotive, aerospace, electronics, financial services, biotech-pharmaceutical, and other industries.

Student Usage of Small Auto-Graded MATLAB® Coding Exercises

Abstract

Instructors are increasingly using small auto-graded coding exercises with immediate feedback to help students learn the MATLAB programming language. Such exercises may require students to write 3 - 10 lines of code. We analyzed student usage of 38 instances of MATLAB coding exercise instances across 1,435 students from seven courses at different universities to determine how students are using the automated MATLAB assessment tool. When instructors suggested completing the exercise (not necessarily requiring or awarding points), we found that student completion rates were on average 83%, with an average per exercise ranging from 64% to 95%. We found that students spent 7.8 minutes on average, matching the 3–10 minutes expected by the exercise authors. We found that students made 4.5 attempts on average per exercise. For some harder exercises, the averages were higher at 12.5 attempts on average and 10.4 minutes on average, suggesting that students were indeed putting forth good effort. Further, we analyzed the students' wrong submissions of exercises that had a high average number of tries. We identified common mistakes by students and shared our best practices for authoring coding exercises.

Introduction

Numerous college-level engineering courses introduce MATLAB® [1][2][3][4][5]. For other programming languages, research shows that small auto-graded coding exercises improve student learning in introductory programming courses [6][7][8][9][10][11][12][13]. Figures 1 - 5 provide examples of such exercises. Not surprisingly, MATLAB instructors have started assigning MATLAB coding exercises.

Though researchers have found MATLAB to be pedagogically valuable [14][15], best practices for incorporating small auto-graded exercises into a course are still being investigated. Toward contributing to that investigation, we analyzed student usage of MATLAB coding exercises across several universities, looking at usage patterns for completion rate, number of tries, and time spent on MATLAB coding exercises. We also analyzed student mistakes and potential misconceptions. Further, we suggest best practices for authoring MATLAB coding exercises.

Background

Many universities offer an introductory MATLAB course to teach basic programming and problem-solving skills [16]. Such courses typically incorporate problem-based learning, where students are provided open-ended problems with limited to no guidance on how to achieve the learning objectives [17]. MATLAB has been found to be pedagogically valuable [14][15]. Devens introduced discipline-specific MATLAB programming in a freshman engineering course with 20 students, which included a short homework, called a micro-challenge, and a week-long

programming assignment, in a freshman engineering course with 20 students [14]. Students performed proficiently in the course and felt much more confident in their computing abilities, and felt the course was important and useful to both current studies and future careers. Tilbury developed web-based MATLAB learning materials in the domain of automatic controls; the learning materials were coupled with MATLAB homework [15]. Tilbury found that student behavior while working on MATLAB homework included frequent quick references to the learning material.

Researchers have also analyzed student learning and usage of small auto-graded coding exercises in introductory programming courses that are not based on MATLAB [6][7][8][9][10][11][12][13]. Edgcomb found that students completed 25% of assigned exercises when no points were awarded, but students completed 62% when only a small number of points were awarded (2 out of 100 course points) [9]. Spacco analyzed student program submissions from three institutions and five semesters, finding that harder exercises tended to yield lower student scores but higher chances of correctly compiling code [11]. Dyke analyzed the usage of a coding editor by 124 students in a programming course, finding significant correlations between good programming habits and exam grades [7]. For example, the use of editor functionalities, e.g., auto-complete, was positively correlated (r-value = 0.623; p-value < 0.01) with exam grades. Denny [6] found that students who invented programming exercises in preparation for an exam performed significantly better on the exam and felt that the process of inventing had contributed to their learning.

Toward contributing to the on-going investigation into MATLAB coding exercises, this paper analyzes students usage of such exercises and identifies potential best practices for authoring such exercises.

MATLAB® Coding Exercises

The MATLAB coding exercises in this analysis are integrated into learning material [18]. The learning material consists of modules that are each designed to take 10-15 minutes to complete, not including the coding exercises. One or more coding exercises are located at the end of each module. Table 1 has examples of MATLAB coding exercises. Each exercise includes a caption, prompt, starter code, and list of assessments.

Table 1: Example of MATLAB® coding exercises that were harder for students.

Caption	MATLAB Coding exercise

Function definition: Volume of a pyramid	*Prompt*
	Define a function CalculatePyramidVolume with inputs baseLength, baseWidth, and pyramidHeight. The function returns pyramidVolume, the volume of a pyramid with a rectangular base. Relevant geometry equations:
	* Volume = base area * height * 1/3
	* Base area = base length * base width
	
	Starter code
	`% Define a function CalculatePyramidVolume`
	`    % Function inputs: baseLength, baseWidth, and pyramidHeight`
	`    % Function output: pyramidVolume`
	
	Assessments
	Check if function definition exists
	Check functions input and output arguments
	Check if CalculatePyramidVolume(1, 1, 1) returns 0.3333
	Check if CalculatePyramidVolume(5.8, 4.0, 6.0) returns 46.4000

Function call in expression: Reduced pricing	*Prompt*
	Write a single statement that assigns totalCost with the discounted cost of item 1 and item2. Use the function DiscountedPrice to determine the cost of each item. Hint: Call DiscountedPrice() twice in an expression.
	
	Ex: If the first item is $10 and 50% off, and the second item is $20 and 40% off, then totalCost is $17 (i.e. $5 + $12)
	
	DiscountedPrice is already provided
	Function saleItemCost = DiscountedPrice(originalItemCost, discountedRate)
	% originalItemCost: Original cost of an item in dollars
	% discountedRate: Discount rate as a decimal value
	end
	
	Starter code
	`function totalCost = CarTotal(item1Cost, item1Discount, item2Cost, item2Discount)`
	`    % Assign totalCost with the discounted cost of items 1 and 2`
	`    totalCost = 0`
	`End`
	
	Assessments
	Check if CartTotal(10, 0.5, 20, 0.4) returns 17
	Check if CartTotal(45, 0.1, 15, 0.75) returns 44.2500

<table>
<tr>
<td>If branching: Bridge toll</td>
<td>Prompt
Complete the example to calculate finalToll. The base toll for a bridge is baseToll. If the vehicle's weight is over 5,000 pounds, then an additional 4 dollars is added to the toll. Lastly, the toll amount is doubled due to heavy traffic.

Starter code
<pre>function finalToll = CalculateToll(baseToll, vehicleWeight)
% baseToll: The base toll to cross a bridge in dollars
% vehicleWeight: Weight of a vehicle in pounds

    % Assign finalToll with baseToll
    finalToll = 0;

    % If vehicle's weight is over 5,000 pounds, increase
toll by 4 dollars

    % Double toll amount due to heavy traffic

end</pre>
Assessments
Check if CalculateToll(10, 3500) returns 20
Check if CalculateToll(15, 6200) returns 38</td>
</tr>
<tr>
<td>Expression with multiple exponents: Computing wind chill</td>
<td>Prompt
On a windy day, a temperature of 15 degrees may feel colder, perhaps 7 degrees. The formula below calculates the "wind chill," indicating the temperature that is felt based on the actual temperature T (in Fahrenheit) and wind speed W (in miles per hour).

windChill = 35.7 + 0.6T - 35.7W^0.16 + 0.43TW^0.16

Write a statement that assigns windChill with the temperature felt given temperatureFahrenheit and windSpeed.

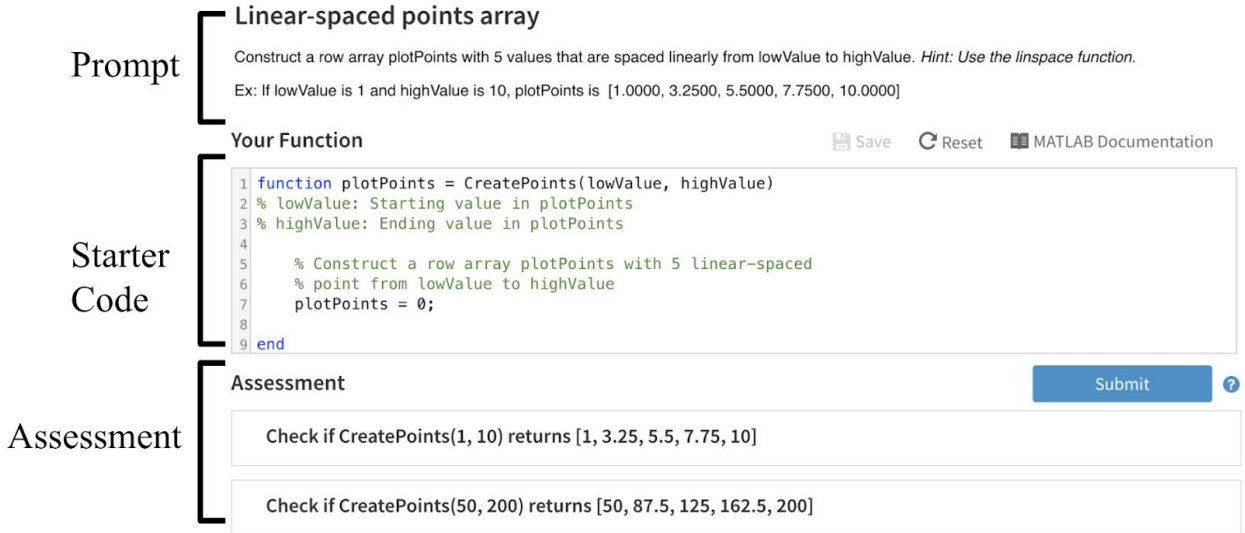
Start code
<pre>temperatureFahrenheit = 32
windSpeed = 10

% Assigns windChill with the temperature felt given
temperatureFahrenheit and windSpeed
windChill = 0</pre>
Assessments
Check if variables temperatureFahrenheit and windChill exists
Check windChill's value</td>
</tr>
</table>

Coding Environment

The MATLAB coding exercises in this analysis were written in an automated MATLAB assessment homework [19] system. We describe that system to give context into the student and authoring experience, as well as context for the best practices. The automated MATLAB assessment homework system supports two types of exercises: function and script. As shown in Figure 1, a function type of MATLAB coding exercise includes from top to bottom: A title, a prompt, a coding area (labeled "Your Function"), a development area (labeled "Code to call your function"), a "Run Function" button, and a "Submit" button. The coding area may have starter code, such as shown in Figure 1. The coding area must define a function because calling that function is how assessment is done. A student can edit the code in the development area, enabling the student to write a custom test. Clicking the "Submit" button causes the student's code to be assessed via a test vector; two such tests are shown at the bottom of Figure 1. Once all test vectors pass, the exercise is marked as completed, and the completed status remains even if the student submits code that does not pass all test vectors. There is no limit to the number of submissions that a student may make. Compared with a function type, a script type of MATLAB coding exercise has no development area.

Figure 1: MATLAB coding exercise (function type) includes: Prompt, coding area (initially, with starter code), and assessment ("Submit" button, tests student's code; unlimited submissions).



Methods

Our goal was to better understand student usage of MATLAB coding exercises. We collected student submissions from such exercises integrated in a zyBook. We then computed the metrics defined in the "Metrics: Definitions" section below across each activity using the student

submissions. The "Metrics: Data" section below details the computed metrics. We then discussed the hardest exercises in order to identify potential issues and suggest improvements.

Collection of Student Submissions of MATLAB Coding Exercises

We sought to identify a large sample of students who were assigned MATLAB coding exercises. Further, we sought to identify a large sample of coding exercises, so that we could analyze the student usage of the coding exercises.

First, we identified courses using zyBooks with the following inclusion criteria and rationale:

- The course used the Introduction to MATLAB zyBook during the Spring 2017 semester
    - Used to control for time of the year. Fall student demographics often differ from Spring demographics. Further, we started collecting data at end of 2016, so Spring 2017 semester was the largest semester of students we had on record at the time of this publication.
- The course using the zyBook had 75 or more students
    - Used to mitigate the effects of outlier course students. That is, having more students in the class gives a better estimate of how well the average student would perform in the course for any given year. Thus, smaller courses tend to have more variability in student performances.
- The students completed at least 50% of the coding exercise across any five chapters in the course using the zyBook
    - Used to predict that the coding exercises were assigned, or at least strongly suggested, by instructors. Course syllabi are not commonly posted openly on the internet anymore; instead, more commonly behind a Learning Management System that requires credentials. So, we could not directly find syllabi.

These criteria yielded seven courses using zyBooks with a total of 1,435 students. The courses include four-year teaching and research universities. To protect privacy, we withhold identifiable data about the courses.

Second, from these courses using zyBooks, we identified coding exercises that were completed by at least 50% of the students in that zyBook. This criterion yielded 38 coding exercises, of which about half were specifically about matrix manipulation and the other half on other basic concepts of MATLAB, such as variables, branching, loops, and functions.

Metrics: Definitions

We defined the following metrics to help analyze student usage of the coding exercises. First, we defined student-level metrics for each student and coding exercise. For example, the completion metric tells us whether a particular student completed a particular coding exercise. Second, we

defined exercise-level metrics for all students on a particular coding exercise. For example, completion rate tells us the percentage of students who completed the exercise.

Student-level: For each student, the following metrics were applied to each coding exercise:

- Completion: Whether the student completed the exercise. Value of 1 was assigned if the student completed the code exercise by submitting a correct answer at some point for that exercise. Otherwise, value of 0 was assigned.
- Number of tries: The total number of submissions for a particular coding exercise. Each time a student submitted code for a coding exercise, we interpreted that submission as one more try of that coding exercise by that student. We stopped counting once a correct submission was made for that coding exercise by that student.
- Time spent: The estimated number of minutes that a student spent on a particular coding exercise. Time spent is the sum of the time-between coding exercise submissions. For example, if a student submitted at 5:00pm and then re-submitted at 5:02pm, the time-between was two minutes. A time-between that was greater than 10 minutes was excluded.

Exercise-level: The following metrics were applied to each coding exercise:

- Completion rate: The percentage of students who completed the coding exercise.
- Average number of tries: Of students who completed the exercise, the sum of each student's number of tries divided by the number of students.
- Average time spent: Of students who completed the exercise, the sum of each student's time spent divided by the number of students.

Metrics: Data

Table 2 shows the exercise-level metrics across all 38 coding exercises. The average completion rate was 83% (range 64% - 95%). A completion rate below 100% is not unexpected as some students may drop the course or just not do homework. The average time spent was 7.8 minutes (range 3.4 - 10.4). The average of the average number of tries was 4.5 (range 2.0 - 12.5).

Table 2: Exercise-level metrics across all 38 coding exercises. Students put forth significant effort, usually spending many minutes per exercise and trying multiple times.

	Completion rate	Average number of tries	Time spent (mins)
Average	83%	4.5	7.8
Min	64%	2.0	3.4
Max	95%	12.5	10.4

Table 3 lists the exercise-level metrics for each of the 38 coding exercises. The correlation between the completion rate and average number of tries was R=-0.23, between completion rate and time spent was R=-0.48, and between average number of tries and time spent was R=0.41. These correlations seem to make sense because we would expect the completion rate to decrease when the average number of tries or time spent were high. Also, we would expect a higher number of tries to be associated with more time spent.

The exercises that took the most time tended to require the user to write branching logic or create a function. The exercises that took the least time tended to be variable assignment and simple expression writing. The mid-range time spent tended to be related to arrays.

Table 3: Exercise-level metrics for each of the 38 coding exercises, ordered by time spent (smallest to largest).

Coding exercise title	Completion rate	Average number of tries	Time spent (mins)
Circle area using pi	93%	3.0	3.4
Comments	90%	5.1	4.6
Declaring a character	83%	3.4	5.4
Compute an expression	92%	2.4	6.0
Exponent expression: Population growth	85%	4.4	6.1
Indexing the last element: Print queue	75%	3.9	6.1
Switch statement to convert letters to Greek letters	72%	4.0	6.6
Check bounds	77%	4.0	6.6
Equality check: Number of bricks	81%	4.2	6.6
Construct an array	87%	5.0	6.7
Distance between 2 points: Exponents and square-roots	84%	2.9	7.0
Linear-spaced points array (*shown in Figure 1*)	73%	3.4	7.0
Writing a numeric expression	91%	2.4	7.0
Assignment statements based on input value	86%	4.1	7.1
Double colon operator: Counting up	82%	3.3	7.3
Plus x: A first function	90%	3.7	7.4
If-elseif-else: Medication dosage by weight	78%	5.1	7.4
Arithmetic array operations	83%	4.5	7.6
Function definition: Double down	88%	5.1	7.7
Double colon operator: Increment by x	82%	2.6	7.9
Indexing an array element	84%	3.1	8.1

If-else branching: Range	76%	4.1	8.2
Integer indexing array: Weekend box office	79%	6.4	8.3
Double colon operator: Counting down	82%	2.0	8.4
Concatenating arrays	76%	2.4	8.6
If-else branching	79%	4.0	8.6
Variable assignment	94%	4.0	8.6
Multiple variables: Curving an exam score	94%	2.4	8.8
Indexing the array: Shift right with variable sized arrays	64%	12.5	9.1
Integer indexing array: Shift left	75%	8.8	9.2
Indexing the array: Moving values	76%	6.1	9.3
Multiple if statements	75%	5.0	9.5
Fahrenheit to Celsius using multiple statements	91%	5.3	9.7
Assigning a sum	95%	3.4	9.7
Expression with multiple exponents: Computing wind chill	86%	4.5	9.7
If branching: Bridge toll	80%	8.0	10.0
Function call in expression: Reduced pricing	77%	4.7	10.1
Function definition: Volume of a pyramid	80%	9.3	10.4

Discussion on Hardest Exercises

Toward better understanding areas of improvement, we looked at student submissions for the hardest coding exercises, as defined as those having the highest average time spent. For each exercise, we identify common mistakes by students and discuss ways to mitigate such mistakes.

The coding exercise "Function definition: Volume of a pyramid" (refer back to Table 1) had an average time spent of 10.4 minutes. This coding exercise is intended for students to practice declaring a function. One common mistake was the misspelling of variable names, specifically, the word "pyramid". A suggestion might be to use a more common word, for example by modifying the problem to be about the area of a rectangle. Another common mistake was incorrect function declaration syntax, such as forgetting to put "Function" in front, doing the calculation in the declaration, and forgetting the equal sign. Such incorrect syntax is expected, as the intention is for students to practice declaring a function. We would suggest providing an example in the instructions and bolding the variable and function names. Further, we noticed that the given equations were computationally backwards, i.e., the volume equation uses the base area but the base area equation is given after the volume equation. Also, the given equations could use the actual variable names that students are instructed to use.

For the coding exercise "Function call in expression: Reduced pricing" (shown in Table 1), a common mistake was to call CartTotal instead of DiscountedPrice. An improvement may be to give an example of calling DiscountedPrice. Another common mistake was to compute the saleItemCost incorrectly. One improvement may be to give the example using precise equations instead of words. An example of calling CartTotal should be given in the instructions, and the instructions should bold the variable and function names.

For the coding exercise "If branching: Bridge toll" (shown in Table 1), a common mistake was to forget to close an if-else statement with an "end", which is a logical error, though not a syntactical error. Another common mistake was to write "else" instead of "end". A third common mistake was to incorrectly type a variable name. One improvement would be to add an example call to code, along with expected output, in the description. Another improvement would be to use the variable name in the description for each variable.

For coding exercise "Expression with multiple exponents: Computing wind chill" (shown in Table 1), the common mistakes were to use the wrong order of operations and to not put the whole equation. Suggested improvements include adding an example and using variable names in the description, and having each assessment specify what is called and what is expected.

Suggested Best Practices: Authoring MATLAB Coding Exercises

This section describes what we have found to be best practices for authoring MATLAB coding exercises. These best practices were developed through experience by writing hundreds of coding exercises and reviewing thousands of student submissions and feedback on those exercises, as well as iterating on those exercises over a few years.

Most coding exercises described in this paper were authored following these best practices.

List of best practices:

- The coding exercise should take no more than 10 - 15 minutes to complete for most students. Typically, we aim for requiring 3-5 lines of code.
- The title of the exercise is descriptive and includes the concept being practiced.
- The prompt describes the problem including any restrictions, sometimes provides a hint (especially for more challenging problems), and always provides an example with arguments and return value. Function and variable names are bold to distinguish code from terms. Vague language is avoided, such as uses of "it". The language is clear and concise. If a new concept is introduced (e.g. calculating interest), then that concept is described in one to two sentences.
- The coding area always has starter code, which include comments that explain where the student's code should go and a summary of the function arguments, return value, and

variable names. For a function type of exercise, typically, the function declaration is typically given, unless the problem is about writing a function declaration. Sometimes function variables are also pre-defined. The starter code often contains comments with instructions to remind the student of the key points from the prompt, which enables the student to remain working in the coding area and not need to return to the prompt.

- The development area is set to the first assessment.
- Each assessment clearly states what is being tested, including the expected result. In the case of a function type of exercise, each assessment states what the arguments are and what the expected return value is. Two types of assessments are used: Verify that the return value is the expected value; and less commonly, verify that a particular keyword or identifier was used. An exercise usually has three to five assessments, and always at least 2. When possible, assessments incrementally build on each other, such as one assessment checking the existence of a variable and a second assessment checking the value of that variable. Such increments are typically small differences.

Conclusion

Instructors are increasingly using small auto-graded coding exercises with immediate feedback to help students learn the MATLAB programming language. Our analysis shows that students put forth good effort on such exercises: Students averaged 7.8 minutes per exercise (the intended time is 5 - 10 minutes) with 4.5 attempts on average and an 83% completion rate on average. For one exercise, the average time spent was 10.4 minutes; on another, the average number of attempts was 12.5. Our analysis identified areas of improvement for more difficult exercises: common student mistakes and potential areas of confusion caused by the exercise. We suggest resolutions for each area. We also provide a list of our best practices for authoring MATLAB coding exercises.

Acknowledgements

This material is based upon work supported by the National Science Foundation under SBIR Grant No. 1430537.

References

[1] Agrawal, J., Y. Lee, and O. Farook. Digital Communication as the First Course in Undergraduate Telecommunication Engineering Technology Program. Proceedings of ASEE Annual Conference, 2017.
[2] Farook, O., J. Agrawal, A. Kulatunga, A. Ahmed, W. Yu, Y. Lee, and H. Alibrahim. Freshman Experience Course in Electrical and Computer Engineering Technology Emphasizing Computation, Simulation, Mathematical Modeling, and Measurements. Proceedings of ASEE Annual Conference, 2017.
[3] Polasik, A. Successes and Lessons Learned in an Undergraduate Computational Lab Sequence for Materials Science and Engineering. Proceedings of ASEE Annual Conference, 2017.

[4] Rihana-Abdallah, A. and J. Lynch. Using Matlab-generated Numerical Solutions in an Environmental Engineering Class to Predict the Fate and Transport of Contaminants. Proceedings of ASEE Annual Conference, 2017.
[5] Wheatley, B., T. Donahue, and K. Catton. An Active Learning Environment to Improve First-Year Mechanical Engineering Retention Rates and Software Skills. Proceedings of ASEE Annual Conference, 2017.
[6] Denny, Paul, Diana Cukierman, and Jonathan Bhaskar. Measuring the effect of inventing practice exercises on learning in an introductory programming course. Proceedings of the 15th Koli Calling Conference on Computing Education Research. ACM, 2015.
[7] Dyke, Gregory. Which aspects of novice programmers' usage of an IDE predict learning outcomes. Proceedings of the 42nd ACM technical symposium on Computer science education. ACM, 2011.
[8] Kumar, Amruth N. Results from the evaluation of the effectiveness of an online tutor on expression evaluation. ACM SIGCSE Bulletin 37.1 (2005): 216-220.
[9] Edgcomb, A., F. Vahid, R. Lysecky, and S. Lysecky. An Analysis of Incorporating Small Coding Exercises as Homework in Introductory Programming Courses. Proceedings of ASEE Annual Conference, 2017.
[10] Helminen, Juha, Petri Ihantola, and Ville Karavirta. "Recording and analyzing in-browser programming sessions." Proceedings of the 13th Koli Calling International Conference on Computing Education Research. ACM, 2013.
[11] Spacco, Jaime, Paul Denny, Brad Richards, David Babcock, David Hovemeyer, James Moscola, and Robert Duvall. "Analyzing student work patterns using programming exercise data." Proceedings of the 46th ACM Technical Symposium on Computer Science Education. ACM, 2015.
[12] Safei, Suhailan, Abdul Samad Shibghatullah, and Burhanuddin Mohd Aboobaider. "A Perspective Of Automated Programming Error Feedback Approaches In Problem Solving Exercises." Journal of Theoretical and Applied Information Technology 70.1 (2014): 121-129.
[13] Queirós, Ricardo Alexandre Peixoto, and José Paulo Leal. "PETCHA: a programming exercises teaching assistant." Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education. ACM, 2012.
[14] Devens, P. E. MATLAB & freshman engineering. In Proceedings of the American Society for Engineering Education Annual Conference & Exposition (ASEE'99), 1999.
[15] Tilbury, D. and W. Messner. Development and integration of Web-based software tutorials for an undergraduate curriculum: Control tutorials for MATLAB. Frontiers in Education Conference, 1997. 27th Annual Conference. Teaching and Learning in an Era of Change. Proceedings.. Vol. 2. IEEE, 1997.
[16] J. P. Hoffbeck, H. E. Dillon, R. J. Albright, Wayne Lu and T. A. Doughty, "Teaching programming in the context of solving engineering problems," *2016 IEEE Frontiers in Education Conference (FIE)*, Erie, PA, USA, 2016, pp. 1-7.
[17] Fee, Samuel B., and Amanda M. Holland-Minkley. "Teaching computer science through problems, not solutions." Computer Science Education 20.2 (2010): 129-144.
[18] zyBooks. http://www.zybooks.com/. Accessed: April 2018.
[19] MATLAB® Grader™ by The MathWorks, Inc. https://www.mathworks.com/. Accessed: April 2018.