The MATLAB Grader Test Generator: A Teacher's Tool for Generating Autograding Tests of MATLAB Scripts*

Robin T. Bye[a]
Cyber-Physical Systems Laboratory, Department of ICT and Natural Sciences, NTNU - Norwegian University of Science and Technology,

Keywords: Assessment, Autograding, Instant Feedback, e-Learning, Learning Management System, MATLAB Grader.

Abstract: MATLAB Grader is a relatively new addition to the MATLAB software family that enables autograding and immediate student feedback of coding assignments. However, manually configuring the assessment tests in MATLAB Grader can be a tedious task for a teacher, especially for larger assignments where more than 100 variables, say, need to be checked. In this paper, I present the MATLAB Grader Test Generator, a tool consisting of both a graphical app and an all-in-one test script that can aid the teacher in this process. The test generator tool attempts to circumvent inherent limitations in MATLAB Grader whilst being free, open-source, and relatively easy to customize for a technically skilled teacher. The tool is available on the MATLAB File Exchange, where it was awarded the File Exchange Pick of the Week in April, 2019.

1 INTRODUCTION

MATLAB is both a high level programming language and a multi-paradigm numerical computing platform that is extensively used in both industry and academia, with more than 4 million users worldwide (MathWorks, 2020a). Common areas of application include artificial intelligence, machine learning, data analysis, control systems, signal processing, computer vision, optimisation, to list a few. A recent and useful pedagogical addition to MATLAB is MATLAB Grader[1] (originally released as Cody Coursework), which is an online browser-based environment where teachers can create and share MATLAB coding assignments with their students. As noted in the documentation (MathWorks, 2020b), teachers can either enrol their students in their course on the MATLAB Grader website, or, if supported, they can integrate MATLAB Grader in their university's learning management system (LMS), which currently include Blackboard, Canvas, and Moodle, and any other LMS that is compliant with version 1.1 of the Learning Tools Interoperability (LTI) standard of the IMS Global Learning Consortium (2020).

Being online, students can work with the assignments anywhere, anytime, and submit solutions as many times as they please. The submitted solutions of students are autograded on the fly, given that the teacher has created a number of suitable tests (called "assessments" in MATLAB Grader) beforehand. Upon autograding, a student will get immediate feedback about the passed or failed tests, and optionally, also the correct answers or hints about what needs to be changed in the student's code.

Likely because of its recent availability, there are virtually no reports in the scientific literature regarding MATLAB Grader. A search on Google Scholar for the phrase 'MATLAB Grader' returns only about 15 written works, none of which have been cited by others.

Nevertheless, the ability of MATLAB Grader to offer repeated submissions, autograding, and immediate feedback is highly popular and motivating for students and increase the degree of completeness and correctness of the students' submitted work, as we have noted in our Cyber-Physical Systems Laboratory[2] (CPS Lab) from work on both flipped classroom teaching approaches (Bye, 2018, 2017) and from teaching electrical and computer engineering undergraduate courses (Bye and Osen, 2019; Osen and Bye, 2018). Our findings are further cemented

[a] https://orcid.org/0000-0002-6063-1264

*This work was partially funded by NTNU Excited, a university centre for excellent IT education.

[1] https://grader.mathworks.com

[2] https://www.ntnu.no/blogger/cpslab/

Bye, R.
The MATLAB Grader Test Generator: A Teacher's Tool for Generating Autograding Tests of MATLAB Scripts.
DOI: 10.5220/0009517904060413
In *Proceedings of the 12th International Conference on Computer Supported Education (CSEDU 2020) - Volume 2*, pages 406-413
ISBN: 978-989-758-417-6

in the relevant literature (e.g., Gramoli et al., 2016; Ala-Mutka, 2005; Cheang et al., 2003; Venables and Haywood, 2003), with a particular interesting topic being that of autogenerating more detailed and advanced feedback than simply right or wrong answers (Singh et al., 2013).

When creating a coding assignment in MATLAB Grader, the teacher must create a solution script and an accompanying learner template script, the latter of which has selected variables with the correct answer removed. The task for students is then to ’fill in the blanks’ in the template, performing intermediate operations and calculations if needed, and submit the script as their solution.

In addition to making the scripts, the teacher must configure a number of tests manually in the web browser, using pull-down menus and entering text defining which variables should be tested for correctness, and optionally adding some corrective feedback comments or hints for incorrect answers. The backend of MATLAB Grader will then compare (autograde) the values of the variables in the student’s script with those of the teacher’s solution script and provide feedback to the student.

From the teacher’s point of view, a major drawback with this procedure is the lack of an application programming interface (API). Manually configuring the tests is a tedious process, especially for larger coding assignments. Moreover, it is common in the process that the teacher decides that some variables need renaming, reordering, or removal; new variables must be added; or new tests must be added or old tests removed. In these cases, the teacher must manually work through the test configuration environment in order to perform such updates, clicking on pull-down menus and updating information. With a well-designed API and/or a suitable backend, a default set of tests could have been autogenerated on the basis of the teacher’s solution script and template script, selecting variables common in both scripts as default comparison tests. The teacher would then only have to update and maintain these two scripts. Optionally, a feature could be implemented that also gives the teacher an option to only test a subset of the default tests.

The MATLAB Grader Test Generator has been developed as an attempt to circumvent the lack of such an API and backend and ease the burden of creating coding assignments in MATLAB Grader. The tool is open-source and freely available on the MATLAB File Exhange[3] and on GitHub (Bye, 2020) and was selected by MathWorks as the File Exchange Pick of the Week[4] in April, 2019.

In the following, I present the method used for developing the tool (Section 2), demonstrate the end result and example usage (Section 3), and discuss limitations and room for improvement, before some concluding remarks (Section 4).

2 METHOD

For illustration, consider a toy example, where students should calculate the area and circumference of a circle, a right triangle, and a rectangle, respectively. The teacher’s reference solution and the students’ learner template are given in Listings 1 and 2, respectively.

Listing 1: Example reference solution.

```matlab
%% Reference Solution
% Circle
r = 2;
areaCircle = pi*r^2;
circumferenceCircle = 2*pi*r;

% Right triangle
a = 3; b = 4; c = 5;
areaTriangle = (1/2)*a*b;
circumferenceTriangle = a+b+c;

% Rectangle
x = 6; y = 7;
areaRectangle = x*y;
circumferenceRectangle = (2*x)+(2*y);
```

Listing 2: Example learner template.

```matlab
%% Learner Template
% Circle
r = 2;
areaCircle = 'enter answer here';
circumferenceCircle = 'enter answer here';

% Right triangle
a = 3; b = 4; c = 5;
areaTriangle = 'enter answer here';
circumferenceTriangle = 'enter answer here';

% Rectangle
x = 6; y = 7;
areaRectangle = 'enter answer here';
circumferenceRectangle ='enter answer here';
```

In earlier versions of MATLAB Grader, the teacher was given a default testing code snippet in which the default variable name, x, to be tested had to be replaced two places in the code (see Listing 3).

[3]https://se.mathworks.com/matlabcentral/fileexchange

[4]https://blogs.mathworks.com/pick/2019/04/12/matlab-grader-test-assessment-generator/

Listing 3: MATLAB Grader testing code snippet.

```
% Get reference solution for x.
xReference = referenceVariables.x;

% Compare with learner solution.
assessVariableEqual('x', xReference);
```

This process had to be repeated for every variable to be tested. Whilst the web user interface of MATLAB Grader has improved since the development of the tool presented here, the teacher still has to click to add a test, optionally give it a name, write the name of the variable to be tested (e.g., `areaCircle` in the toy example), and repeat for the next variable.

The MATLAB Grader Test Generator has been developed to improve this process and comes with two main modes of operation: (i) an app with a graphical user interface (GUI) for autogenerating assessment code for multiple tests that can be pasted into the MATLAB Grader web interface; and (ii) an all-in-one script that tests all variables within a single MATLAB Grader test. The two modes are further described below.

2.1 The App

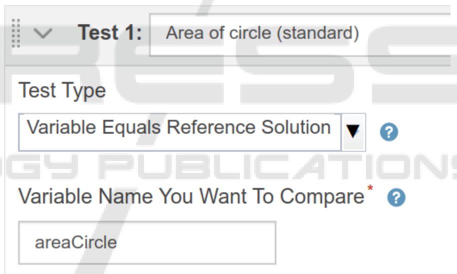
The main goal when developing the test generator app was to autogenerate a list of testing code snippets to be inserted into MATLAB Grader, each corresponding to a variable to be tested. The app achieves this by comparing the solution script and template script and identifying variables common for both scripts. This list of variables is further post-processed by removing variables that the teacher identifies in an exclusion list. Finally, a text file is produced with code snippets separated by double blank lines for readability and easy selection of text for copy and paste operations. Each code snippet corresponds to test code for a variable and can be pasted into the MATLAB Grader assessment environment. In earlier versions of MATLAB Grader, performing such a sequence of copy and paste operations was significantly faster (in the ballpark of 2–3 times faster) than configuring the test using the built-in functionality in MATLAB Grader, especially for larger problems with more than 100 variables, say. With recent improvements in the user interface of MATLAB Grader, the speed-up advantage is smaller and just a little faster but still has the advantage that performing the copy and paste operations requires a smaller cognitive load and concentration on behalf of the teacher.

2.2 All-in-One Test Script

As an alternative to the app solution above, an assessment script has been developed that can be inserted into a single MATLAB Grader test, testing all variables of interest in one go. The script compares all common variables in the student's submission with those of the teacher's solution, excluding any variable in the solution script that contains the character string `exclude` in its variable name. The number of correct test answers are added up and printed together with the total number of tests. The speed-up when compared to configuring each test individually is vast, and approximately equal to an n-times improvement, where n is the number of variables to be tested. The script is included in the appendix of this paper.

3 RESULTS

An example of using the standard method for creating tests in MATLAB Grader is shown in Figure 1, where the test is configured for the variable `areaCircle` from the toy example presented previously. Running


Test 1: Area of circle (standard)

Test Type

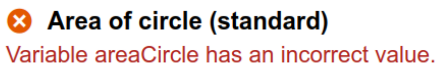
Variable Equals Reference Solution

Variable Name You Want To Compare*

areaCircle

Figure 1: Standard method for creating a test in MATLAB Grader.

this test will either show a green check symbol indicating that the answer was correct, or a red cross and a message saying the answer was incorrect, as depicted in Figure 2 (the student used the incorrect formula $r \cdot \pi^2$ for the area of a circle). Note that the option 'Pre-test' must be selected, otherwise even this simple message will not be displayed.


Area of circle (standard)

Variable areaCircle has an incorrect value.

Figure 2: Standard message for wrong answer in MATLAB Grader.

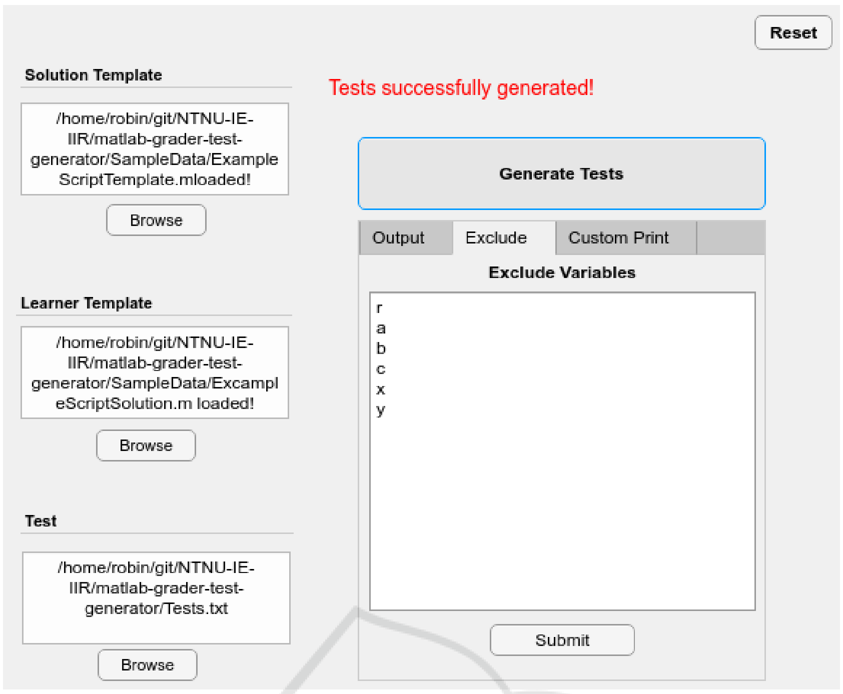
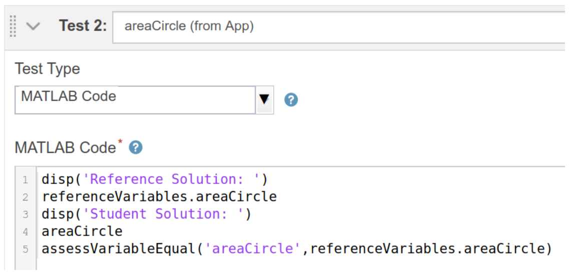

Reset
Solution Template
/home/robin/git/NTNU-IE-IIR/matlab-grader-test-generator/SampleData/ExampleScriptTemplate.mloaded!
Browse
Learner Template
/home/robin/git/NTNU-IE-IIR/matlab-grader-test-generator/SampleData/ExcamplеScriptSolution.m loaded!
Browse
Test
/home/robin/git/NTNU-IE-IIR/matlab-grader-test-generator/Tests.txt
Browse
Tests successfully generated!
Generate Tests
Output
Exclude
Custom Print
Exclude Variables
r
a
b
c
x
y
Submit

Figure 3: MATLAB Grader Test Generator user interface.

3.1 Using the App

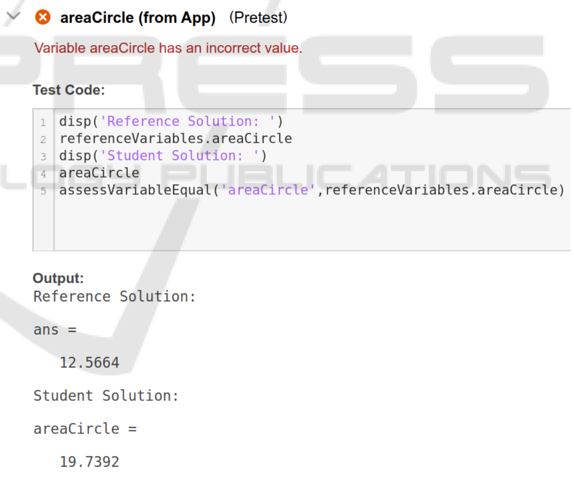
A screenshot of the MATLAB Grader Test Generator app is provided in Figure 3. The user selects a solution script (here: Listing 1), a template script (here: Listing 2), and a location of a text file on the computer where the autogenerated list of testing code snippets should be stored (here: `\Tests.txt`). Variables not to be tested can be entered in the Exclude tab (here: variables `r`, `a`, `b`, `c`, `x`, `y`). Upon clicking Generate, the app appends a list of testing code snippets into the text file, each of which can be pasted into MATLAB Grader. The code snippet for testing the variable `areaCircle` is shown in Figure 4.

Running the test (again, the option 'pre-test' must be checked), will result in the output shown in Figure 5.



```
disp('Reference Solution: ')
referenceVariables.areaCircle
disp('Student Solution: ')
areaCircle
assessVariableEqual('areaCircle',referenceVariables.areaCircle)
```

Figure 4: Inserting testing code snippet in MATLAB Grader.



Variable areaCircle has an incorrect value.

```
disp('Reference Solution: ')
referenceVariables.areaCircle
disp('Student Solution: ')
areaCircle
assessVariableEqual('areaCircle',referenceVariables.areaCircle)
```

```
Reference Solution:

ans =

   12.5664

Student Solution:

areaCircle =

   19.7392
```

Figure 5: Running testing code snippet in MATLAB Grader.

3.2 Using the All-in-One Test Script

Using the all-in-one testing script requires the user to simply paste the script into the testing code window for a single test in MATLAB Grader (not shown due to space considerations). For an example student submission with all answers correct except for the area of the circle, the results in the output for all variable tests will be shown in the single-test output window in MATLAB Grader as depicted in Listing 4.

Listing 4: Output from all-in-one testing script.

```
areaCircle =
   1.9739e+01
ans =
   1.2566e+01
msg =
    'areaCircle has an incorrect value.'
Test 1 of 6: NOT PASSED!
________________________________________

areaRectangle =
    42
ans =
    42
Test 2 of 6: PASSED!
________________________________________

areaTriangle =
     6
ans =
     6
Test 3 of 6: PASSED!
________________________________________

circumferenceCircle =
   1.2566e+01
ans =
   1.2566e+01
Test 4 of 6: PASSED!
________________________________________

circumferenceRectangle =
    26
ans =
    26
Test 5 of 6: PASSED!
________________________________________

circumferenceTriangle =
    12
ans =
    12
Test 6 of 6: PASSED!
________________________________________
________________________________________

Total Score: 5 / 6
________________________________________
```

4 DISCUSSION

There are advantages and drawbacks of the three approaches presented above. Using the standard built-in method for generating tests in MATLAB Grader is generally preferable for smaller coding problems and when the teacher do not want to display correct answers (and student answers) to students. In addition, if MATLAB Grader is linked to an LMS such as Blackboard, the points scored on each test are accumulated and passed on to the grading system in the LMS.

Using the MATLAB Grader Test Generator app has the additional advantage of autogenerating more feedback to students in the form of displaying both the correct solution and the student solution, and, in earlier versions of MATLAB Grader, enabled much faster configuration of tests, especially for larger problems with many variables. With the recently improved interface of MATLAB Grader, the speed advantage is nearly vanished but there is still the advantage of less cognitive load required when performing a series of copy and paste operations as compared to manually typing in a series of unique variable names.

Moreover, although not shown, it is fairly easy to modify and extend the app to display even more information, for example it is trivial to display the error (absolute and/or percentage), the correct formula for each variable (if desired), or other feedback to students.

Using the all-in-one test script has one major advantage over the other two approaches, namely a huge speed-up. The script can be pasted into a single-test configuration window for any MATLAB Grader problem and will automatically test all common variables in the student's solution versus the teacher's solution. Hence, the teacher can reduce the time for configuration of tests to approximately the fraction $1/n$ compared to configuring n tests separately. In addition, as for the app, the script can easily be modified to include more feedback information to students.

However, the all-in-on script has one major drawback. Configuring all variable tests inside a single MATLAB Grader test means that if one or more 'internal' tests fail, the overall MATLAB Grader test will report a failed test (score of zero) to an external LMS. Hence, the teacher must manually inspect each student's submission and the reported score, and update the gradebook in the LMS.

4.1 Future Work

Ideally, to better enable the work-around functionality described in this paper, MathWorks should either improve the MATLAB Grader platform with an API that allows teachers to customize and autogenerate tests, and/or a suitable backend for the same.

Another option is to investigate whether one can design a computer program that can parse the text file with code snippets generated by the MATLAB Grader Test Generator app and can interact with the MATLAB Grader website and configure the tests automatically. In our CPS Lab, we have done some preliminary examinations using various form-filling software freely available but so far we have been unsuccessful with this approach. Furthermore, such a 'hacked' solution would easily become broken the moment the implementation of the MATLAB Grader website changes.

Finally, it should be noted that MATLAB Grader is still a quite new platform with limited adoption amongst teachers. it will be interesting to hear user stories from teachers trying out the tool in the future.

5 CONCLUSIONS

I have presented the MATLAB Grader Test Generator, a free and open-source tool for simplifying and speeding-up the somewhat laborious process of configuring the autograding environment in MATLAB Grader. The tool consists of a graphical app and a script, both of which have their merits as presented above. With some basic programming skills, teachers can customize both the app and the script to their liking, e.g., for adding autogenerated feedback information to students. Whereas the tool successfully circumvents current limitations in the MATLAB Grader platform, future updates in MATLAB Grader might break functionality. Additionally, there might be a technical threshold preventing some teachers from delving into using and customizing the tool. Hence, MathWorks should instead improve the MATLAB Grader platform, offering both a well-documented API and a more extensive variety of options for configuring assessment tests, including autogeneration of tests on the basis of a solution script and learner template script provided by the teacher.

ACKNOWLEDGEMENTS

I wish to thank Sivert Løken, a graduate of the bachelor of automation engineering programme at NTNU in Ålesund, for his code development in this project. In addition, I wish to thank my colleague Ottar L. Osen for his useful input during the development phase and the write-up of this paper. Finally, I am thankful for the financial support from NTNU Excited[5] a Norwegian university centre for excellent IT education.

REFERENCES

Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102.

Bye, R. T. (2017). The Teacher as a Facilitator for Learning: Flipped Classroom in a Master's Course on Artificial Intelligence. In *Proceedings of the 9th International Conference on Computer Supported Education — Volume 1: CSEDU)*, pages 184–195. INSTICC, SCITEPRESS.

Bye, R. T. (2018). A Flipped Classroom Approach for Teaching a Master's Course on Artificial Intelligence. In Escudeiro, P., Costagliola, G., Zvacek, S., Uhomoibhi, J., and McLaren, B. M., editors, *Computer Supported Education: CSEDU 2017 — Revised Selected Best Papers*, volume 865 of *Communications in Computer and Information Science (CCIS)*, pages 246–276. Springer International Publishing.

Bye, R. T. (2020). MATLAB Grader Test Generator. https://github.com/NTNU-IE-IIR/matlab-grader-test-generator.

Bye, R. T. and Osen, O. L. (2019). On the Development of Laboratory Projects in Modern Engineering Education. In *Proceedings of the IEEE Global Engineering Education Conference (EDUCON 2019)*, pages 1327–1334.

Cheang, B., Kurnia, A., Lim, A., and Oon, W. (2003). On automated grading of programming assignments in an academic institution. *Computers & Education*, 41(2):121–131.

Gramoli, V., Charleston, M., Jeffries, B., Koprinska, I., McGrane, M., Radu, A., Viglas, A., and Yacef, K. (2016). Mining autograding data in computer science education. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '16, New York, NY, USA. Association for Computing Machinery.

IMS Global Learning Consortium (2020). Learning Tools Interoperability. http://www.imsglobal.org/activity/learning-tools-interoperability, accessed on 2019-02-03.

MathWorks (2020a). Company Overview. https://se.mathworks.com/company/aboutus.html, accessed on 2020-02-03.

MathWorks (2020b). MATLAB Grader Documentation. https://www.mathworks.com/help/matlabgrader/, accessed on 2019-02-03.

Osen, O. L. and Bye, R. T. (2018). Observations and Reflections on Teaching Electrical and Computer Engineering Courses. In Escudeiro, P., Costagliola, G., Zvacek, S., Uhomoibhi, J., and McLaren, B. M., editors, *Computer Supported Education: CSEDU 2017 — Revised Selected Best Papers*, volume 865 of *Communications in Computer and Information Science (CCIS)*, pages 363–389. Springer International Publishing.

Singh, R., Gulwani, S., and Solar-Lezama, A. (2013). Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, page 15–26, New York, NY, USA. Association for Computing Machinery.

Venables, A. and Haywood, L. (2003). Programming students need instant feedback! In *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20*, ACE '03, page 267–272, AUS. Australian Computer Society, Inc.

[5]https://www.ntnu.edu/excited

APPENDIX

The all-in-one test script presented in this paper is included in Listing 5.

Listing 5: All-in-one testing script.

```matlab
% Excludes every variable containing the terms found inside the parenthesis.
tempnames = who('-regexp','(?i)^(?=\w*(fig|exclude))\w*');
% Exclude variables not to check.
% Primarily variables not common in workspace and reference solution.
% Place variable here if you get non-existing reference error.
% Figures are not testable.
noCheck =  {'r','a','b','c','x','y',...
    'tempnames','noCheck','noCheckPat','referenceVariables','seed', 'ii'};
for ii = 1:length(tempnames)
    noCheck{end+1} = char(tempnames(ii));
end
% Combines noCheck and tempnames to a regex pattern.
noCheckPat = (['^(?!(' strjoin(noCheck, '|') ')$).']);
% Import variables from virtual workspace to a struct.
% Excluding figures and all variables with keyword exclude and some static variables.
varnames = whos('-regexp', noCheckPat);
tol = 0.001;      % Set tolerance, 0.001 = 0.1% relative tolerance.
format compact    % Set output format. Either 'compact' or 'loose'.
format shortE     % Set output format. See documentation for various types.
nodisplays = {}; % Excluding variables to not display due to length, readability etc.
counterTP = 0;    % Counter for tests passed
counterER = 0;    % Counter for errors
counterNC = 0;    % Counter for noCheck variables
for ii = 1:numel(varnames)
    if ismember(varnames(ii).name, noCheck)
        disp(['Test ' num2str(ii) ' of ' num2str(length(varnames))])
        disp(['-Not testable-'])
        counterNC = counterNC + 1;
        disp('--------------------------------------------------------- ')
        continue;
    else
        try
            % for ii = 1 -> referenceVariables.numg
            refV = ['referenceVariables.' varnames(ii).name];
            % Returns a [n m] vector where n is the number of lines the variable contains.
            n = size(eval(refV));
            ns = varnames(ii).size;
            % evalc returns character array of variable.
            % Limit output to 100 char and 3 lines.
            if (ismember(varnames(ii).name, nodisplays) || ...
                    (length(evalc(refV)) > 100) || ...
                    (n(1) > 4) || (ns(1) > 4) || ...
                    (length(evalc(varnames(ii).name)) > 100))
                disp([varnames(ii).name ' ='])
                disp(' ... answer too long to display')
            else
                eval(varnames(ii).name)
                eval(refV)
            end
            assessVariableEqual(varnames(ii).name, eval(refV), ...
                'RelativeTolerance', tol)
            disp(['Test ' num2str(ii) ' of ' num2str(length(varnames)) ...
                ': PASSED!'])
            counterTP = counterTP + 1;
            disp('--------------------------------------------------------- ')

        catch ME
```

```
            msg = getReport(ME, 'basic');
            if contains(msg, 'Variable')
                msgFiltered = extractAfter(msg, 'Variable');
                msg = strtrim(msgFiltered)
            else
                msg
            end
            switch ME.identifier
                case 'MATLAB:nonExistentField'
                    continue;
                case 'AssessmentToolbox:Feedback:SizeMismatch'
                    disp(['Test ' num2str(ii) ' of ' num2str(length(varnames)) ...
                        ': NOT PASSED!'])
                    disp('—————————————————————————————————————— ')
                    counterER = counterER +1;
                    continue;
                case 'AssessmentToolbox:Feedback:ValueMismatch'
                    disp(['Test ' num2str(ii) ' of ' num2str(length(varnames)) ...
                        ': NOT PASSED!'])
                    disp('—————————————————————————————————————— ')
                    counterER = counterER +1;
                    continue;
                case 'AssessmentToolbox:Feedback:DataTypeMismatch'
                    disp(['Test ' num2str(ii) ' of ' num2str(length(varnames)) ...
                        ': NOT PASSED!'])
                    disp('—————————————————————————————————————— ')
                    counterER = counterER +1;
                    continue;
                otherwise
                    msg = getReport(ME, 'extended')
                    rethrow(ME)
            end
        end
    end
end
studentScore = num2str(counterTP + counterNC);
totalScore = num2str(counterTP + counterER + counterNC);
disp('—————————————————————————————————————— ')
disp(['Total Score: ' studentScore ' / ' totalScore])
disp('—————————————————————————————————————— ')
if (studentScore == totalScore)
    disp(['All tests passed'])
else
    msgID = 'CODY:InsufficientScore';
    msg = 'Not all answers are correct';
    baseException = MException(msgID, msg);
    throw(baseException)
end
```