Test Plan for the Library of ODE Solvers (LODES)

Paul Aoanan

December 9, 2017

1 Revision History

Date	**Version**	**Notes**
October 24, 2017	1.0	Initial draft.
December 9, 2017	1.1	Initial revision.

2 Symbols, Abbreviations and Acronyms

The following table lists the symbols, abbreviations and acronyms used in the Test Plan. The software library's Commonality Analysis (CA) tables provide supplementary items in addition to the ones listed below.

symbol	**description**
CA	Commonality Analysis
IDE	Integrated Development Environment
IVP	Initial Value Problem
ODE	Ordinary Differential Equation
LODES	Library of ODE Solvers
SRS	Software Requirements Specification
T	Test
O	Output

Table 1: Symbols, Abbreviations, and Acronyms used in the Test Plan

Contents

List of Tables

3 General Information

The following section provides an overview of the Test Plan for the Library of Ordinary Differential Equation (ODE) Solvers.
[You do not need to end your paragraphs with this symbol. A hard return and a newline will end the paragraph. —SS] [Thank you - noted. —PA]

The Test Plan contains the verification and validation activities outlined for LODES (). The development of LODES follows the traditional waterfall model, wherein the software requirements and expectations are laid out in the Software Requirements Specification (SRS).

The Test Plan contains the system test cases which are used to verify and validate the SRS functional and non-functional requirements of LODES. Furthermore, the Test Plan also contains unit test cases which are used to verify the functionality laid out in the Module Guide and Module Interface Specification design documents

Section 3.1 explains the purpose of this document, Section 3.2 the scope of the system, and Section 3.3 the document overview.

[This "roadmap" doesn't really make sense. You start with describing the "following section" and then you introduce "this section." Is the following section 3.1 or 4? Some kind of introduction before the "roadmap" would also make sense. You can put this document in context. What is the project? What other documentation exists. In particular, you should point to the SRS. You should also give the GitHub url where the project is located. —SS] [Comments addressed above. —PA]

3.1 Purpose

The main purpose of this document (the Test Plan) is to describe the verification and validation process that will be used to test the functionality of LODES. This document closely follows the requirements and governs the subsequent testing activities. This document is intended to be used as a reference for all testing and will be used to increase confidence in the software implementation. [I deleted the extra newline you inserted. You are nearly always best to go with the LaTeX default formatting and not start manually inserting newlines. —SS] [Noted - thank you. —PA]

This document will be used as a guide and starting point for the Test Report. The test cases listed in this document will be executed and the output will be analyzed to uncover errors, increase confidence and correctness in the software.

3.2 Scope

The scope of the testing is limited to the Library of ODE Solvers. Given the appropriate inputs, each program in LODES is intended to find the solution to an Initial Value Problem (IVP).

Due to time and cost constraints, the scope of testing is limited to automated unit and manual system verification and validation activities.

Static testing will be briefly described and will be left to the developer and verifier to perform with due diligence.

3.3 Overview of Document

The following sections provide more detail about the testing of LODES. Information about the testing process is provided and the software specifications that were discussed in the Commonality Analysis are stated. The evaluation process that will be followed during testing is outlined and test cases for both the system testing and unit testing are provided.

4 Plan

This section provides a description of the software that is being tested, the team that will perform the testing, the approach to automated testing, the tools to be used for verification, and the non-testing based verification.

4.1 Software Description

The software being tested is the Library of ODE Solvers. Given the ODE, initial values of x and y, and the final value of x, the programs calculate the final value of y through the use of numerical methods.

4.2 Test Team

The test team that will execute the test cases, write and review the Test Report consists of:

- Paul Aoanan
- To be determined (The test report will be reviewed by an independent individual)

4.3 Automated Testing Approach

Automated unit testing will be implemented for LODES as described in Section 4.4. A combination of the Unit Testing Framework and MATLAB's Test Class will be used for automated testing.

4.4 Verification Tools

The verification tools to be used will be the following:

1. Unit Testing Framework
   A Unit Testing Framework designed in MATLAB that will compare MATLAB's own functional programs with LODES' running the same inputs will be implemented.

The following equation [say equation, not algorithm —SS] [Reworded. —PA] will be implemented to compare the results:

$$\epsilon_{\text{relative}} = \frac{\|\text{Result}_{\text{MATLAB}} - \text{Result}_{\text{LODES}}\|}{\|\text{Result}_{\text{MATLAB}}\|}$$

[The above equation will give you an error for every instant of time where you do the calculations; that is, you will have an error for each time step. This could let you build some nice plots, but the vector of numbers on its own is too much for a person to take in. You should also have a scalar value to assess the error. Specifically, you should take the norm of your $\epsilon_{\text{relative}}$ vector. Any vector norm is fine, as long as you say which one you are using. —SS] [The equation has been modified to take the norm of the vector instead. The testing software will also have the ability to output the plot of the error percent of each step. —PA]

2. MATLAB's Test Class
   MATLAB's Test Class (matlab.unittest.testcase) will be used for providing the framework for automated system testing. The documentation can be referenced in MATHWORKS.

3. Static Analyer
   [proof read —SS] [Proof read. —PA] The MATLAB IDE will be used for program debugging and for checking syntax errors.

4. Continuous Integration
   The source code and the project repository is located in GitHub at: `https://github.com/aoananp/cas741/`. It provides the Build Server functionality to fully maintain and document the software through its lifecycle. As well, it provides the compare functionality for future regression testing and analysis of code updates. [How does the build server find Matlab? This is something you should look into sooner than later, since it might require something special to deal with the Matlab licensing. —SS] [MATLAB 2017a has a built-in Build Server that seamlessly integrates with Git. —PA]

5. Code Coverage Tool
   Due to the commercial nature of MATLAB, only commercial code coverage tools are viable for use due to the maturity, increased confidence, and detailed documentation that they offer. Other coverage tools may be considered, but no code coverage tool will be considered in the scope of this test plan due to budget constraints. [Okay – good to be explicit. —SS]

4.5 Non-Testing Based Verification

LODES will undergo the following non-testing based verification activities:

Code Inspection

LODES will undergo an initial desk review by an independent body. The code will be perused for syntax errors and correct program calls. This code inspection activity provides the initial sanity check for the developer, the reviewer, and the software. The code inspection checklist is provided in the Appendix Section 7.1.

[You should be more specific about how you are going to run the code inspection. Probably you want to give a checklist for the reviewer to follow. A google search will find you many checklists that could give you a good starting point. The checklist should be included in your appendix. —SS] [The code inspection checklist is provided in Section 7.1 —PA]

Code Walkthrough

LODES will undergo code walkthrough by the developer and an independent body. They will jointly review the code and reference the Commonality Analysis for algorithm adherence. This activity will also involve logic analysis, loop and recursion boundary tracing (using by-hand test cases), passing of variables and references, and if the code is programmatically correct. The Walkthrough Report will contain the results of a visual inspection of the code involved in the project.

[Great to see that you have additional details. If I have any comments on the walkthrough, I'll make them when I review that section. —SS] [The code inspection checklist is shown in Section 7.1. The walkthrough report will include the details of each specific iteration of the code and will have to based on the actual code itself. The Commonality Analysis and the Unit Test plan will serve as the heuristic guides in carrying out the code walkthrough. —PA]

Symbolic Execution

Generally, symbolic execution will be performed using the boundary input conditions. Test conditions will be analyzed and executed by-hand. Generally, inputs in, on, and around the boundary conditions are chosen. [Are you doing symbolic execution? There isn't enough information here. What software tool are you using? —SS] [In the interest of time, details of symbolic execution testing and and its execution will be covered in Revision 2 of the document and the next iteration of the waterfall model. —PA]

[I am glad to see code inspection, code walkthroughs and symbolic execution mentioned. Your implementation will be fairly straightforward. To represent a graduate level of effort, you need these "extras." —SS]

5 System Test Description

System testing will be executed to provide increased confidence that LODES will achieve the goals defined in the Commonality Analysis. It uses a "black box" [quotes in LaTeX are done as "quotes." That is, there is a different symbol for the opening and closing quotation marks. —SS] approach wherein it tests the system as a whole through the use of input and output analysis.

5.1 Tests for Faulty Input

5.1.1 Input

The input will be based on the Assumptions table in the Commonality Analysis. Each test will correspond to an entry from the assumptions item whilst altering a specific input variable to a non-permissible value. The list of inputs is in order with the entries in the table.

Table 2: Faulty Input Test Cases

Number	Input	Expected Outcome
01	ODE Function Call $\notin$ {euler741, trap741, heun741, rk4741} $\cup$ MATLAB functions	error: undefined function call
02	$f(x,y) = y'' + y' + x + 2$	success: false; the ODE is of second order and the software is outside of the assumptions
03	$f(x,y) = y' + 1$	success: false; the form shall be of $y' = f(x,y)$
04	$f(x,y) = (dx/dy) + 1$	success: false; the form shall be of $y' = f(x,y)$
05	$f(x,y) = (y+1)/x$	success: false; the ODE shall be linear
06	$f(x,y) = y/(x-3)$	success: false; the ODE shall be linear
07	Boundary Value Problem	success: false; BVPs are out of scope of the software
08	$h = -1$	success: false; h shall be greater than 0
09	$h = 0$	success: false; h shall be greater than 0
10	$x_0 = i$	success: false; inputs shall be $\in \mathbb{R}$
11	$x_0 = [0,1]$	success: false; input shall be a single number
12	$y_0 = i$	success: false; inputs shall be $\in \mathbb{R}$
13	$y_0 = [0,1]$	success: false; input shall be a single number
14	$x_k = i$	success: false; inputs shall be $\in \mathbb{R}$
15	$x_k = [0,1]$	success: false; input shall be a single number

[I would like more information in this table. Why is the success false? I assume that for 02 this is because it is a second order ode, but you should make this explicit. Also, I might have missed something new in Matlab, but how exactly are derivatives represented? Are you using symbolic programming, like for Maple? If you are just passing a function, then y'' won't have any semantic meaning. —SS] [Clarifications to the table have been made. —PA]

5.2 Tests for Functional Requirements

5.2.1 Calculation Tests

Euler's Method

1. **T-1: Simple Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = y, h = 2, x_0 = 0, y_0 = 1, x_k = 2$

   Output: $y_k = 3$, success = true

   How test will be performed: Automated system test [For these test cases, it would be nice to see the symbolic form of the closed-form solution. As it is right now, a reviewer would have to derive the closed-form solution and then verify if your output is correct. Your test cases might actually come from symbolically solving the algorithm. If this is the case, you really need to make it clear. —SS] [The test will be performed as such. The listed inputs are for the tester to input to the program. The output will be verified when compared with the MATLAB solution. —PA]

2. **T-2: Simple-Iterative Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = y, h = 0.5, x_0 = 0, y_0 = 1, x_k = 2$

   Output: $y_k = 5.0625$, success = true

   How test will be performed: Automated system test

3. **T-3: Non-linear Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = sin(x) - y^2, h = 5, x_0 = 0, y_0 = 1, x_k = 5$

   Output: $y_k = -4$, success = true

   How test will be performed: Automated system test

4. **T-4: Non-linear Iterative Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = sin(x) - y^2, h = 1, x_0 = 0, y_0 = 1, x_k = 5$

   Output: $y_k = -0.669532$, success = true

   How test will be performed: Automated system test

Trapezoid Method

1. **T-5: Simple Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = y, h = 2, x_0 = 0, y_0 = 1, x_k = 2$

   Output: $y_k = 3$, success = true

   How test will be performed: Automated system test

2. **T-6: Simple-Iterative Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = y, h = 0.5, x_0 = 0, y_0 = 1, x_k = 2$

   Output: $y_k = 5.0625$, success = true

   How test will be performed: Automated system test

3. **T-7: Non-linear Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = sin(x) - y^2, h = 5, x_0 = 0, y_0 = 1, x_k = 5$

   Output: $y_k = -4$, success = true

   How test will be performed: Automated system test

4. **T-8: Non-linear Iterative Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = sin(x) - y^2, h = 1, x_0 = 0, y_0 = 1, x_k = 5$

   Output: $y_k = -0.6695$, success = true

   How test will be performed: Automated system test

Heun's Method

1. **T-9: Simple Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = y, h = 2, x_0 = 0, y_0 = 1, x_k = 2$

   Output: $y_k = 5$, success = true

   How test will be performed: Automated system test

[It looks like you are repeating inputs for different tests. I think this is a great idea, since there really is no reason to make up different arbitrary tests for each method. However, repeating all of this information and leaving it to the reader to deduce the differences isn't really fair to them. I suggest that instead of presenting each test separately, you build a table that summarizes all of the related tests. It would take less space and it would be easier to review. —SS]

[Noted. This will be addressed in the next revision of the Test Plan and the next waterfall iteration. For now, the focus is on test execution - the document's content is valid for now for our purposes. —PA]

2. **T-10: Simple-Iterative Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = y, h = 0.5, x_0 = 0, y_0 = 1, x_k = 2$

   Output: $y_k = 6.972900$, success = true

   How test will be performed: Automated system test

3. **T-11: Non-linear Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = sin(x) - y^2, h = 5, x_0 = 0, y_0 = 1, x_k = 5$

   Output: $y_k = -43.897311$, success = true

   How test will be performed: Automated system test

4. **T-12: Non-linear Iterative Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = sin(x) - y^2, h = 1, x_0 = 0, y_0 = 1, x_k = 5$

   Output: $y_k = -1.101197$, success = true

   How test will be performed: Automated system test

Runge-Kutta Method

1. **T-13: Simple Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = y, h = 2, x_0 = 0, y_0 = 1, x_k = 2$

   Output: $y_k = 7$, success = true

   How test will be performed: Automated system test

2. **T-14: Simple-Iterative Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = y, h = 0.5, x_0 = 0, y_0 = 1, x_k = 2$

   Output: $y_k = 7.383970$, success = true

   How test will be performed: Automated system test

3. **T-15: Non-linear Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = sin(x) - y^2, h = 5, x_0 = 0, y_0 = 1, x_k = 5$

   Output: $y_k = -1702.845129$, success = true

   How test will be performed: Automated system test

4. **T-16: Non-linear Iterative Case**

   Type: Functional, Automated, System

   Initial State: Not applicable

   Input: $f(x, y) = sin(x) - y^2, h = 1, x_0 = 0, y_0 = 1, x_k = 5$

   Output: $y_k = -1.028367$, success = true

   How test will be performed: Automated system test

[Your outputs are unlikely to be an exact match. There will be some epsilon error. You mention this at the start, but it isn't clear that it is applied here. Part of your output could be a summary of the observed errors for each test output in comparison to the ideal output. —SS]

[Are you just taking one step for each of these tests? You really also need tests that will predict the ODE values for a sequence of steps. —SS]

[These tests are the comparing the results of the ODE to the solutions worked out by hand. The testing method to be used is by relying on the MATLAB's solution as the oracle and taking the norm of the relative error vector - this is described early on in the document. The test report will indeed be a summary of observed errors. The outputs are left here for reference and for discussion in the Test Report. —PA]

5.3 Tests for Nonfunctional Requirements

5.3.1 Performance Requirements

Speed

1. **T-17: Speed Benchmark**

   Type: Non-Functional, Automated, Performance

   Initial State: Not Applicable

   Input/Condition: $f(x, y) = sin(x) - y^2, h = 1E - 5, x_0 = 0, y_0 = 1, x_k = 5$

   Output/Result: $\sigma_{\text{LODES}} \leq mult * \sigma_{\text{MATLAB}}$ (LODES' runtimes shall be no more than *mult* times that of MATLAB's. For now, $mult = 4$ is defined.) [Use a symbolic constant here, so that it is easy to change. Also, this requirement should be in the SRS. Is it there? If not, you should add it. —SS] [This requirement is in the CA. —PA]

   How test will be performed: Using MATLAB's Run and Time functionality, the execution time of a program will be measured and compared through program calls to the respective MATLAB and LODES functions.

[Is this just one step of the ODE solver? I don't think you'll be able to measure any difference between the different methods with one step. Everything will happen too fast. You should have a long integration time to let the tiny speed differences accumulate. —SS]

[This is through all the steps in the ODE solver, executing from x_0 to x_k and y_0 to y_k. —PA]

5.3.2 Results Analysis

Benchmark Results

1. **T-18: MATLAB Benchmark**

   Type: Non-Functional, Automated, Precision

   Initial State: Not Applicable

   Input/Condition: $f(x, y) = sin(x) - y^2, h = \text{(VARIABLE)}, x_0 = 0, y_0 = 1, x_k = 5$

   Output/Result: The $\epsilon_{\text{relative}}$ vs. h plot. The intermediate y values (Result) will be compared according to the following formula of the relative error norm -

$$\epsilon_{\text{relative}} = \frac{\text{Result}_{\text{MATLAB}} - \text{Result}_{\text{LODES}}}{\text{Result}_{\text{MATLAB}}}$$

   How test will be performed: The h (step-size) values will be varied across the range (0, 1000]. $\epsilon_{\text{relative}}$ will be plotted against h.

[Good idea for a plot, but you'll need to run the simulations long enough to observe differences as mentioned above. —SS]

[I don't think I see a test related to accuracy where you compare the different ODE solvers to a known closed form solution over time. In your original planning for this project you were thinking that the purpose of your program was to do this comparison. I persuaded you to instead focus on a library of ODE solvers, suggesting that you can return to this idea for testing. It is time now to return to your original idea. :-) It would be a great plot to show

how the error differs between your different methods. —SS] [For testing, the MATLAB ODE solver will serve as the oracle. The comparisons will be made against it and observations against the desktop execution (outputs described in the Functional Tests) of the solution will be described. —PA]

5.4 Traceability Between Test Cases and Requirements

The following table shows the traceability mapping for the test cases laid out in this Test Plan to the requirements described in the Commonality Analysis.

Table 3: Requirements Traceability Matrix

Test Number	**CA Requirements**
T1	IM1, O1, O2, O3, O4, O5
T2	IM1, O1, O2, O3, O4, O5
T3	IM1, O1, O2, O3, O4, O5
T4	IM1, O1, O2, O3, O4, O5
T5	IM2, O1, O2, O3, O4, O5
T6	IM2, O1, O2, O3, O4, O5
T7	IM2, O1, O2, O3, O4, O5
T8	IM2, O1, O2, O3, O4, O5
T9	IM3, O1, O2, O3, O4, O5
T10	IM3, O1, O2, O3, O4, O5
T11	IM3, O1, O2, O3, O4, O5
T12	IM3, O1, O2, O3, O4, O5
T13	IM4, O1, O2, O3, O4, O5
T14	IM4, O1, O2, O3, O4, O5
T15	IM4, O1, O2, O3, O4, O5
T16	IM4, O1, O2, O3, O4, O5
T17	NFR1
T18	Future NFR2

6 Unit Testing Plan

6.0.1 ODE String Parser

Parser Functionality

1. **T-19: Simple**

   Type: Functional, Manual, Unit

   Initial State: Not Applicable

   Input/Condition: $f(x, y) = ax^2 + by^2 - cx - dy - 1$

   Output/Result: Machine-interpreted $f(x, y)$

How test will be performed: $f(x, y)$ will be passed into the parser function and the output shall be a machine-interpreted function.

[Are you implementing a parser? I thought that Matlab was just taking care of this? —SS] [MATLAB will take care of the parser - I figured since it was part of my requirement, that I shall test it as well. —PA]

2. **T-20: Trigonometric**

   The test case is removed in Release 0.

[I don't understand why there is a trigonometric unit test? You aren't planning on implementing trigonometric functions (I don't think) and you don't need to test Matlab. —SS]

[This test case is removed. —PA]

[For the unit tests you really shouldn't try to be specific until you have done your design. This section was intended to be an overview of your plans for selecting unit test cases. —SS]

Function Iteration

1. **T-21: Program Iteration**

   Type: Functional, Manual, Unit

   Initial State: Not Applicable

   Input/Condition: $f(x, y) = y, h = .1, x_0 = 0, y_0 = 1, x_k = 1$

   Output/Result: y_k through each time step

   How test will be performed: Each iteration of the loops or recursions in the code will be observed in run-time. Each variable will be monitored prior to loop or function re-entry. This will be done through all four methods (Euler, Trapezoid, Heun, and Runge-Kutta).

7 Appendix

7.1 Code Inspection Checklist

1. Are there defects in the code?
2. Does the code adhere to MATLAB coding standards and guidelines (syntax, function calls, etc.)?
3. Does the code sensibly apply the ODE solvers as defined in the Commonality Analysis?
4. Does the code contain spelling errors?
5. Do the recursion/loops have the correct terminating cases according to the formulas defined in the Commonality Analysis?
6. Does the code have the correct inputs and outputs? [What does this mean? Do you mean whether it agrees with the requirements? —SS] [It shall agree with the inputs and outputs listed in the functional requirements listed above through code walkthrough. —PA]
7. Do the datatypes in the code adhere to ones defined in the requirements and specifications?
8. Does the code satisfy the ranges and assumptions defined in the Commonality Analysis?

[This isn't actually a code walkthrough procedure. This is a code inspection checklist. A walkthrough is when a group of reviewers "play computer" and go through the code and its algorithms by hand. —SS] [Noted - that was the intent of the section :-). The section has been renamed. —PA]

7.2 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

symbol	**unit**	**description**
ϵ	none	The measure of the difference/error between results obtained with LODES and MATLAB.
σ	seconds	The measure of time a program executes.

7.3 Usability Survey Questions

No usability study will be performed for LODES.