Runge-Kutta (RK) Generator

Alexander Schaap

December 19, 2017

1 Revision History

The latest version can be found at https://github.com/aschaap/cas741.

Date	Version	Notes
October 16	1.0	Initial draft for presentation
October 25	1.1	Processed feedback and completed report
December 18	1.2	Final version, processed issues and new feedback

2 Symbols, Abbreviations and Acronyms

Also see the Table of Symbols in the SRS at https://github.com/aschaap/cas741.

symbol	description
ODE	Ordinary Differential Equation
RK	Runge-Kutta
T	Test

Contents

3 General Information

This document is a test plan for the RK Generator family of ODE solvers.

3.1 Purpose

This document specifies the (black-box) verification and validation tests for the RK Generator. The intended audience are testers aiming to test the system and developers maintaining the software. This document is expected to be updated when development of the system proceeds.

3.2 Scope

The scope of the testing of the RK Generator is restricted to correctness, in some cases within an expected margin of error. Performance, while implied, should also be measured. The implementation could be compared to an unstaged version of the same code (plain OCaml code rather than MetaOCaml; no generation). Alternatively, a comparison against MATLAB would give an indication of performance compared to the competition. This is a secondary goal though. The tests in this document follow from the SRS available at `https://github.com/aschaap/cas741`. [An explicit web-link to your GitHub repo would be nice. —SS] [I've added one to the revision history as well as here. —AS] [Reference your SRS document —SS][Done. —AS]

3.3 Overview of Document

The structure of this documents is based on Smith (2006). The next section will outline the software to test, the testing team and the proposed approach. Subsequently, system tests are covered, and ultimately unit tests will be added in the section after that.

4 Plan

This section begins with a brief description of the software being tested, followed by a succinct description of the test team before discussing the automated approach proposed, the tools expected to accomplish this, and the manual testing necessary.

4.1 Software Description

The RK Generator is a metaprogramming approach to a library of ODE solvers. Given a choice of RK method, an ODE, an interval, a step size and an initial condition, it generates a function that will provide values for input on the given interval. This function will be unique to the inputs provided. During the creation of this function, metaprogramming is also employed for performance reasons. The parent program that called the generator can subsequently use the generated code to solve specific points within the interval.

4.2 Test Team

The test "team" will comprise Alexander Schaap.

4.3 Automated Testing Approach

Automated tests exist primarily in the form of unit tests, though some "system" tests will do black-box verification.

4.4 Verification Tools

For unit tests, OUnit seems an obvious choice. For automating compilation and running unit tests as a primitive form of continuous integration, Make is sufficient. Unit tests give us regression testing for free. A tool exists to measure code coverage. It is called bisect_ppx and appears actively maintained. It remains to be seen how useful it is for MetaOCaml code. While code coverage has its weaknesses, it should suffice because there is little branching in this software.

4.5 Non-Testing Based Verification

Ideally, unit tests capture all the desired behaviour. However, manual inspection of the generated code by the developer will be required. The manual assessment should determine whether modifications to the generator are beneficial to either making the code more readable or improving the results in terms of accuracy. While readability isn't strictly a requirement, simple (and therefore more readable) code is often a decent indicator of correctness in metaprogramming. A common approach used for metaprogramming is

outputting generated code to text and comparing it to a previous version. Any differences are subject to manual inspection. Note that the correctness of the previous code is not guaranteed; this is also subject to manual inspection. [In your case having a build infrastructure where you can do a diff between the generated code and some "golden" version of the generated code, would be nice. This won't tell you if your generated code is correct, but it will tell you if you generator starts creating different code. This either means you have introduced an error, or you have a new "golden" version to put in place. This is the approach used for ggk and for Drasil. —SS] [I've added a summary of this approach. —AS]

5 System Test Description

Also known as black-box tests, system tests verify the software as a whole. This is done without assumed knowledge of the internal workings. This section covers the system tests along with non-functional tests.

5.1 Tests for Functional Requirements

The intent is to test using problems with known solutions to get around oracle problem. Results of other ODE solvers such as MATLAB or Octave can be used for comparison purposes in case of unknown solutions. Currently, all tests have known solutions. [I would like to see some tests where you compare to MATLAB or Octave as the pseudo oracle. This will provide a greater variety of tests. —SS] [I'm afraid I'll have to move that to phase 2; I've updated the corresponding section to reflect this. —AS]

It is assumed that these tests test both the generator and the generated code. The inputs therefore are for the generator, but inputs are also needed for the generated code. Since the generator creates a function that takes one input, the input for generated code is provided one at a time (making the generated code run multiple times, which should be the typical use case).

5.1.1 RK4 & RK2

These tests are to be repeated for each method. The expected results for each method are described separately.

Solution Correctness Tests

1. Low-order-ODE[1]

   Type: Functional, Dynamic

   Initial State: N/A

   Input (to generator): $x'(t) = 5x - 3 \quad x(2) = 1, \quad 2 \leq t \leq 3, \quad h = 0.1$

   Input (to generated code): t = 2, 3, 0, -1024, 2.2, 2.5, 23 [What are these numbers? The final value of t, or something else? —SS] [I've added t to make this unambiguous. —AS]

   Output (RK4): printed values for the user to compare to the exact solution $x(t) = \frac{2}{5}e^{5(t-2)} + \frac{3}{5}$ for the same inputs to see whether they are acceptably close.

   Output (RK2): similar to RK4, but with a higher margin of error in most cases (should be within$O(h^3)$).

   How test will be performed: as part of the automated tests executed by the Makefile.

   [Rather than make the vague statement that you are taking floating point errors into account, I think your goal should be to summarize the relative error for each test case. You can make a table of errors. Rather than specify a priori what the error should be, the job of your testing is to simply describe the results. Judging the acceptability of the errors is something that can be done by potential users of the software. —SS] [I've changed the test cases to reflect this change. —AS]

2. High-order-ODE[2]

   Type: Functional, Dynamic

   Initial State: N/A

   Input (to generator): $x'(t) = 7x^2t^3 \quad x(2) = 3, \quad 2 \leq t \leq 4, \quad h = 0.1$ [Why not just write $x'(t) = 7x^5$? (product rule for exponents). Is there

[1] Adapted from: http://mathinsight.org/ordinary_differential_equation_introduction_examples

[2] Adapted from: http://mathinsight.org/ordinary_differential_equation_introduction_examples

supposed to be a t on the right hand side of the equation? Looking into this further, I believe you mean $7x^2t^3$. —SS] [Yes, my mistake. Thank you for pointing this out, I've corrected it. —AS]

Input (to generated code): t = 2, 3, 0, 2.5, -89, 8192

Output (RK4): printed values close to the exact solution $x(t) = \frac{-1}{\frac{7}{4}x^4 - \frac{85}{3}}$; the error should be within $O(h^5)$. The user can then decide whether these results are acceptable.

Output (RK2): similar to that of RK4, but with the larger margin of error (of $O(h^3)$).

How test will be performed: as part of the automated tests executed by the Makefile.

3. Stiff-ODE

   Type: Functional, Dynamic

   Initial State: N/A

   Input (to generator): $y'(t) = -15y(t), \quad y(0) = 1, \quad 0 \leq t \leq 1, \quad h = 0.1$.

   Input (to generated code): t = -1, 0, 0.4, 1, 1.5

   Output (RK4 & RK2): A printed inaccurate numeric result that deviates significantly from the exact solution of $y(t) = e^{-15t}$ with $y(t) \to 0$ as $t \to \infty$ (or even as $t \to 1$), also for the user to manually inspect.

   How test will be performed: as part of the automated tests executed by the Makefile.

[I agree with this. You should mention it under non-testing based verification. You should also give more detail on how it is going to be done, who is going to do it, etc. Ideally, there will be a specific approach for the verification, including a pointer to the specific algorithm that the final code should resemble. —SS] [I've moved these sentences and expanded on the topic under non-testing based verification. —AS] [Does the ... mean something? Are more details coming? —SS] [No, I removed them. —AS]

5.2 Tests for Nonfunctional Requirements

The implied nonfunctional requirement to be tested is performance. There is no "wrong" result though. The goal is to provide insight into the performance in comparison to obvious alternatives.

5.2.1 Performance Tests

This section discusses tests that compare the performance of the implementation to that of existing implementations. These tests are part of phase two, which will commence in January 2018, due to performance being a secondary goal and the limited time in which everything must be done.

Calculation Speed Tests

1. Comparison against unstaged (plain OCaml) code

   Type: Non-Functional, Dynamic

   Initial State: N/A

   Input/Condition: Input to any one of the functional tests will be provided to both versions of the code, but the objective is to compare running times

   Output/Result: identical outputs that correspond to the expected output of the respective test chosen, but the staged (MetaOCaml) code should be faster

   How test will be performed: via Make and OUnit

2. Comparison against MATLAB

   Type: Non-Functional, Dynamic

   Initial State: N/A

   Input: Input to any one of the functional tests above will be provided to both MATLAB and the generator (note that MATLAB could require a different input format)

   Output: Similar output but with the expectation that MATLAB may be more accurate. MATLAB will also likely be faster (excluding start-up time).

How test will be performed: via Make

[More detail needs to be provided. How are the results going to be summarized? What equation is going to be used to compare the generated code to the "competition"? Percent relative change seems like a good summary statistic to me. Will there be a bar graph showing the results, or some other approach? —SS]

[For many of your tests they might be too simple to show much of a difference in performance. You will need a long integration time for differences to be observable. —SS]

...

5.3 Traceability Between Test Cases and Requirements

N/A - No requirements, just a commonality analysis (which implies requirements).

6 Unit Testing Plan

Unit tests can be divided into two distinct parts. The first set verifies portions of the generator, while the second part has to verify the generated code.

One could easily assume that verifying a generator is similar to verifying other programs, and that tools can be reused for this purpose. However, a common approach at the moment is to write generated code to a file and use "diff" to compare the two. By comparing the generator's output to previous generator output for identical inputs the tester can manually determine by careful examination whether any differences are appropriate.

These tests will have to be defined after implementation.

References

W. Spencer Smith. Systematic development of requirements documentation for general purpose scientific computing software. In *Proceedings of the 14th IEEE International Requirements Engineering Conference, RE 2006*, pages 209–218, Minneapolis / St. Paul, Minnesota, 2006. URL http://www.ifi.unizh.ch/req/events/RE06/.