System Verification and Validation Plan for ROC: software estimating the radius of convergence of a power series

John M Ernsthausen

October 29, 2020

Contents

1 Symbols, Abbreviations, and Acronyms **iii**

2 Introduction **1**

3 General Information **2**
- 3.1 Summary . . . 2
- 3.2 Objectives . . . 2
- 3.3 Relevant Documentation . . . 3

4 Verification and Validation Plan **3**
- 4.1 Verification and Validation Team . . . 3
- 4.2 SRS Verification Plan . . . 3
- 4.3 Design Verification Plan . . . 4
- 4.4 Implementation Verification Plan . . . 4
- 4.5 Automated Testing and Verification Tools . . . 4
- 4.6 Software Validation Plan . . . 5

5 System Test Description **5**
- 5.1 Tests for Functional Requirements . . . 5
  - 5.1.1 Timing . . . 5
  - 5.1.2 Accuracy . . . 6
- 5.2 Traceability Between Test Cases and Requirements . . . 6

6 Unit Test Description **7**
- 6.1 Unit Testing Scope . . . 7
- 6.2 Tests for Functional Requirements . . . 7
  - 6.2.1 Module 1 . . . 7
  - 6.2.2 Module 2 . . . 7
- 6.3 Tests for Nonfunctional Requirements . . . 7
  - 6.3.1 Module ? . . . 7
  - 6.3.2 Module ? . . . 8
- 6.4 Traceability Between Test Cases and Modules . . . 8

List of Tables

List of Figures

Revision History

Date	Version	Notes
29 October 2020	1.0	First submission

1 Symbols, Abbreviations, and Acronyms

Symbols, abbreviations, and acronyms applicable to ROC are enumerated in Section 1 of the Software Requirements Document (SRS) (Ernsthausen, 2020).

2 Introduction

Construct the series centered at $z_0 \in \mathbb{R}$

$$\sum_{n=0}^{\infty} c_n (z - z_0)^n \tag{1}$$

from a sequence $\{c_n\}$ of real numbers where the n^{th} term in the sequence corresponds to the n^{th} coefficient in the series. We associate a sequence $\{s_n\}$ of partial sums

$$s_n \stackrel{\text{def}}{=} \sum_{k=0}^{n} c_k (z - z_0)^k \tag{2}$$

with the power series. If $\{s_n\} \to s$ as $n \to \infty$, then we say $\{s_n\}$ converges to s. The number s is the sum of the series, and we write s as (1). If $\{s_n\}$ diverges, then the series is said to diverge.

We cannot perform an infinite sum on a digital computer. However, given a tolerance TOL and a convergent power series, there exists an integer N such that, for all $m \geq n \geq N$, $|\sum_{k=n}^{m} c_k| < \text{TOL}$. We assume that we know N. Our software ROC will estimate the radius of convergence R_c from the first N terms in the power series. The coefficients may be scaled with a scaling h to prevent numerical overflow. The default scaling is $h = 1$. Scaling the coefficients is a change of variables $v(z) \stackrel{\text{def}}{=} (z - z_0)/h$ in (1). With the scaled coefficients $\tilde{c}_n \stackrel{\text{def}}{=} c_n h^n$ and the change of variables, (1) transforms to

$$\sum_{n=0}^{\infty} \tilde{c}_n v^n. \tag{3}$$

When r_c is the radius of the circle of convergence of (3), $R_c = h r_c$ is the radius of the circle of convergence of (1). ROC may not compute R_c exactly. These are the assumptions under which the ROC software operates. In the sequel, we denote both scaled and unscaled coefficients by c_n.

Corliss and Chang Chang and Corliss (1982) (CC) observed that the coefficients of (1) follow a few very definite patterns characterized by the location of primary singularities. Real valued power series can only have poles, logarithmic branch points, and essential singularities. Moreover, these singularities occur on the real axis or in complex conjugate pairs. The effects of secondary singularities disappear whenever sufficiently long power series are used. To determine R_c and the order of the singularity μ, CC fit a given finite sequence to a model.

Recall that a primary singularity of (1) is the closest singularity to the series expansion point z_0 in the complex plane. All other singularities are secondary singularities.

This document provides a verification and validation plan for developing ROC.

3 General Information

The scope of this ROC project is limited to top-line analysis. Top-line analysis always applies to any power series (1). It resolves situations where secondary singularities are less distinguishable from primary singularities. However it is less accurate. It does have a convergence analysis Chang and Corliss (1982).

3.1 Summary

In Ernsthausen (2020), we detailed the algorithm implementing top-line analysis.

Module findrc We know the tolerance TOL, the scale h, the integer N, and the sequence $\Big\{c_0, c_1, \ldots, c_N\Big\}$ by assumption, as in Section 2. We extract the last 15 terms from the sequence. With $k = i + N - 14$, obtain the best linear fit $y(k) = mk + b$ in the 2-norm to the points

$$\Big\{(N-14, \log_{10}|c_{N-14}|), (N-13, \log_{10}|c_{N-13}|), \ldots, (N, \log_{10}|c_N|)\Big\}, \tag{4}$$

that is, find m and b such that $\sum_{i=0}^{14} |\log_{10}|c_{i+N-14}| - y(i+N-14)|^2$ is minimized. The approximation problem (4) is the well known *linear least-squares problem* Golub and Van Loan (1989). The model parameter m will be negative whenever the series (1) converges and positive whenever it diverges.The raidus of convergence $R_c = h/10^m$.

Module findmu Start with the series resulting from integration of the given series three times and fit the coefficients with Module **findrc**. If that graph is linear, meaning the minimizer has norm less than TOL, then the slope is accepted and the order of the singularity is 3. If the graph opens upward, then the series is differentiated term-wise to reduce the second derivative of the graph, and a new top-line fit is computed with Module **findrc**. This process is repeated, reducing μ by 1 each time, until the graph opens downward or until seven term-wise differentiations have been tested. If seven term-wise differentiations have been tested and each result in turn proves unsatisfactory, then the final estimate for R_c is reduced by 10 percent for a conservative estimate for R_c and $\mu = -4$ is returned.

We implement Module **findrc** and Module **findrc** in ROC. This document provides plan for verification and validation ROC while implementing these modules in C++.

3.2 Objectives

Our objective it to implement Module **findrc** and Module **findrc** via *test driving* in C++. The implementation must satisfy the requirements enumerated in Ernsthausen (2020).

Assume that the assumptions are satisfied. Recall the requirements

R1: The input sequence should be scaled by h to prevent overflow/underflow. If this is not possible, ROC will find a scale.

R2: ROC should execute as fast as the CC software DRDCV.

R3: ROC should estimate the radius of convergence R_c and the order of singularity μ for the following cases where R_c and μ are known.

The real valued function $1/(z - z_0)^{\mu}$.

The real valued function $1/(1 + 25 * (z - z_0)^2)^{\mu}$.

Top line analysis while solving DAEIVP by the TS method should verify the computed step is within the circle of convergence and compare with the CC algorithm DRDCV.

R4: We must not overestimate R_c. If R_c is overestimated, then the power-series is a divergence sum on the overestimation.

In ODE solving by TS methods, underestimating R_c is acceptable as an underestimation results in a slight increase in computational effort for solving an ODEIVP.

In a nutshell these requirements validate and verify ROC.

3.3 Relevant Documentation

Relevant documentation includes the authors Software Requirements (SRS) Document Ernsthausen (2020), the authors Module Guide (MG, to be written), and the authors Module Interface Specification (MIS, to be written).

4 Verification and Validation Plan

Verification and Validation of ROC includes automated testing at the module level, the system level, and integration level. This document will additionally propose continuous integration.

4.1 Verification and Validation Team

The ROC team includes author John Ernsthausen, fellow students Leila Mousapour, Salah Gamal aly Hessien, Liz Hofer, and Xingzhi Liu as well as Professors Barak Shoshany, Spencer Smith, George Corliss, and Ned Nedialkov. The author appreciates the helpful comments and superior guidance on this project.

4.2 SRS Verification Plan

The SRS document Verification and Validation Plan for ROC will be peer-reviewed by domain expert Leila Mousapour and secondary reviewer Salah Gamal aly Hessien. Prof. Spencer Smith the course instructor and my supervisor Ned Nedialkov will review the SRS document.

The SRS document will be published to GitHub. Defects will be addressed with issues on the GitHub platform.

4.3 Design Verification Plan

The Design documents MG and MIS plan for ROC will be peer-reviewed by domain expert Leila Mousapour and secondary reviewer Xingzhi Liu. Prof. Spencer Smith the course instructor and my supervisor Ned Nedialkov will also review these Design documents.

The MG and MIS documents will be published to GitHub. Defects will be addressed with issues on the GitHub platform.

4.4 Implementation Verification Plan

The Implementation Verification Plan for ROC includes automated testing at the module level. For testing at the module level, the author plans to follow the Test Driven Development software development practices as described by Langr (2013) for *test driving*. Test driving is an interactive process for software development where tests are determined as part of the software development process. However on a high level, some unit tests will be offered in the sequel.

Let us recall test driving Langr (2013). Test driving results in unit tests. A unit test verifies the *behavior* of a code unit, where a *unit* is the smallest testable piece of an application.

A single unit test consists of a descriptive name and a series of code statements conceptually divided into four parts

1. (Optional) statements that set up a context for execution
2. One or more statements to invoke the behavior to be verified
3. One or more statements to verify the expected outcome
4. (Optional) cleanup statements

The first three parts are referred to as *Given-When-Then*. *Given* a context, *When* the test executes some code, *Then* some behavior is verified.

Test Driven Development is used to test drive new behavior into the code in small increments. To add a new piece of behavior into the system, first write a test to define that behavior. The existence of a test that will not pass drives the developer to implement the corresponding behavior. The increment should be the smallest meaningful amount of code, one or two lines of code and one assertion.

The code should be well formatted and free of memory leaks.

4.5 Automated Testing and Verification Tools

Automated Testing and Verification Tools will be extensively used in the development of ROC for automation at the module level, the system level, and integration level. These tools include git a distributed version-control system for tracking changes in source code during software development, cmake for build automation, gtest as a unit testing framework, clang-format for consistent code style formatting, and valgrind to identify memory leaks.

This document will additionally propose to branch out into three new areas for the author. In the DevOps arena, continuous integration will be persued with TravisCI. The continuous integration tool should send an email confirming the success state of integrating the new code into production. The author will explore the application of linters and metrics for code coverage.

4.6 Software Validation Plan

The Software Validation Plan for ROC includes automated testing at the system level and integration level. The code must validate known cases at the system level. The code output will be compared with the output of DRDCV developed by CC for validation at the integration level.

5 System Test Description

System tests are about the public interface. Nonfunctional tests are removed from the discussion because ROC has no nonfunctional requirements at this time.

5.1 Tests for Functional Requirements

Functional requirements at the system level reflect R2 and R3.

5.1.1 Timing

Requirement R2 says that ROC should execute as fast as the CC software DRDCV. The following tests represent a comparison between ROC and DRDCV.

Proportion of time spent finding the stepsize

1. Layne-Watson

   Input: Load a file with the Taylor series solution of the Layne-Watson problem. Each time the local initial value problem was solved there will be a Taylor series of length N and a scale h, which are the required inputs for ROC and DRDCV.

   Output: Difference between time for ROC to solve problem and time for DRDCV to solve problem.

   Test Case Derivation: Two techniques resolving the same data.

   How test will be performed: Automatically.

2. Planetary-Motion

Input: Load a file with the Taylor series solution of the Planetary-Motion problem. Each time the local initial value problem was solved there will be a Taylor series of length N and a scale h, which are the required inputs for ROC and DRDCV.

Output: Difference between time for ROC to solve problem and time for DRDCV to solve problem.

Test Case Derivation: Two techniques resolving the same data.

How test will be performed: Automatically.

5.1.2 Accuracy

Requirement R3 says that the new top line analysis in ROC should compute a R_c and μ which is close to the values computed with the CC algorithm DRDCV. This test treats DRDCV as a pseudo oracle. The comparison will be carried out on the process of solving a DAEIVP by the TS method.

Accuracy in finding the stepsize

1. Layne-Watson

   Input: Load a file with the Taylor series solution of the Layne-Watson problem. Each time the local initial value problem was solved there will be a Taylor series of length N and a scale h, which are the required inputs for ROC and DRDCV.

   Output: Each time the local initial value problem was solved, find R_c and μ with ROC and DRDCV. Compare the results. Expect the results to compare within 10%.

   Test Case Derivation: Two techniques resolving the same data.

   How test will be performed: Automatically.

2. Planetary-Motion

   Input: Load a file with the Taylor series solution of the Planetary-Motion problem. Each time the local initial value problem was solved there will be a Taylor series of length N and a scale h, which are the required inputs for ROC and DRDCV.

   Output: Each time the local initial value problem was solved, find R_c and μ with ROC and DRDCV. Compare the results. Expect the results to compare within 10%.

   Test Case Derivation: Two techniques resolving the same data.

   How test will be performed: Automatically.

5.2 Traceability Between Test Cases and Requirements

In each test, the requirement supported by the test case is stated.

6 Unit Test Description

6.1 Unit Testing Scope

6.2 Tests for Functional Requirements

6.2.1 Module 1

1. test-id1

   Type:

   Initial State:

   Input:

   Output:

   Test Case Derivation:

   How test will be performed:

2. test-id2

   Type:

   Initial State:

   Input:

   Output:

   Test Case Derivation:

   How test will be performed:

3. ...

6.2.2 Module 2

...

6.3 Tests for Nonfunctional Requirements

6.3.1 Module ?

1. test-id1

   Type:

   Initial State:

Input/Condition:

Output/Result:

How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

6.3.2 Module ?

...

6.4 Traceability Between Test Cases and Modules

References

YF Chang and G Corliss. Solving ordinary differential equations using Taylor series. *ACM TOMS*, 8(2):114–144, 1982.

J.M. Ernsthausen. Software requirements specification for ROC: software estimating the radius of convergence of a power series. https://github.com/JohnErnsthausen/roc/blob/master/docs/SRS/SRS.pdf, 2020.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, second edition, 1989.

Jeff Langr. *Modern C++ programming with test-driven development*. The Pragmatic Programmers, LLC, 2013.