Unit Verification and Validation Plan for SpecSearch

Robert E. White

December 2, 2018

1 Revision History

Date	Version	Notes
2018-11-25	1.0	Creation of first draft.
2018-11-29	1.1	Update of 1.0. Completed tracebility, added references, deleted list of tables and updated section 2.
2018-12-02	1.2	Minor edits before submission.

Contents

List of Tables

List of Figures

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
SRS	System Requirements Specfication
MG	Module Guide
MIS	Module Interface Specification
VnV	Verification and Validation
M	Module
R	Requirement

See SRS (White (2018c)), MG (White (2018a)) and MIS (White (2018b)) documentation at https://github.com/whitere123/CAS741_REW for a more complete table.

Units are small components of source code. Unit testing involves testing the individual units of the software package. A unit Verification and Validation plan consists of outlining test cases for particular units of a software's modules. These tests are created to ensure that the units satisfy the software's functional and nonfunctional requirements. The tests can be traced to a particular module. The module should be traced to a particular requirement.

3 General Information

3.1 Purpose

The software that is being unit tested is called SpecSearch. SpecSearch is scientific computing software that finds and plots the spectrum of a matrix operator that appears in a LAX pair compatible with solutions of the Non-Linear Schrödinger equation for three different numerical algorithms and two boundary solutions. The modules are listed and described in the MG (White (2018a)) and MIS (White (2018b)). The requirements are outlined in the SRS (White (2018c)). All of these documents can be found in https://github.com/whitere123/CAS741_REW.

3.2 Scope

A lot of the modules in SpecSearch are built-in MATLAB functions. These modules are outside of the scope for the Unit Verification and Validation Plan. They include: Plotting, Eigenvalue and Eigenvector solver, Matrix, Elliptic Functions, Elliptic Integrals and Linspace.
Spectrum Error Equations will not be tested as it simply calculates the difference between two values. These values, which are derived from other modules, will have already been tested. Control will not be tested because it simply coordinates the other modules.
Input Parameters and Output Format have a low testing priority. This is because these modules are not performing any rigorous calculations. They simply involve storing or reshuffling data.

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

4 Plan

4.1 Verification and Validation Team

The Verification and Validation team consists of my supervisor, Dr. Dmitry Pelinovsky, and I, Robert White.

4.2 Automated Testing and Verification Tools

Matlab provides a unit testing framework. I will write unit-based scripts for testing the modules in SpecSearch. The functions and classes that automate this process are detailed in the following link: https://www.mathworks.com/help/matlab/script-based-unit-tests.html.

The functions of particular importance are: throw, catch and assert. The throw and catch will also be used for the error tests in the SystVnVPlan.

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. —SS]

4.3 Non-Testing Based Verification

The non-testing based verfication will involve a code inspection by my thesis supervisor. He will inspect each of the modules to guarentee that the physical equations have been implemented succesfully, verify that the software is designed to be maintainable and manageable and complete the two surveys in the appendix 6.

[List any approaches like code inspection, code walkthrough, symbolic execution etc. Enter not applicable if you do not plan on any non-testing based verification. —SS]

5 Unit Test Description

The modules that are being unit tested are specificed in the MIS (https://github.com/whitere123/CAS741_REW). This plan was created to ensure that each module is behaving accurately and that each module is satisfying the appropriate requirements. Multiple tests may be employed for larger modules (such as Numerical Parameters) to ensure that the different components are behaving appropriately.

All of the inputs for the automated white box unit tests are listed in Table 4 of the System VnV Plan https://github.com/whitere123/CAS741_REW. From now on this table will be denoted by "Input table". An automated boundary value analysis will be performed for two of the modules. Values of k approaching the two boundaries of the (0,1) constraint will be considered. We will see how these modules behave in the limit. The prescribed inputs for

these tests are in 6. From now on this table will be denoted by "Boundary Table".

5.1 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.1.1 Input Parameters Module

The following tests were created to ensure that the input parameters module is storing the correct values specified by the user. This first step is critical as all of the other modules rely on these quantities. This is also the first requirement outlined by the SRS (https://github.com/whitere123/CAS741_REW).
The exception tests are covered in the System VnV Plan and the module is explained mathematical in the MIS (https://github.com/whitere123/CAS741_REW).

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-InParams-N

   Type: Automaic

   Initial State:

   Input: Input Table.

   Output: assert=True

   Test Case Derivation: The environment variable N inputted by the user should match the state variable from InParams.N .

   How test will be performed: Natural number values ranging from 50 to 200 will be inputed in the environment. This value should be exactly equal to the state variable InParams.N . An assert statement will return true if these are equal.

1. test-InParams-k

   Type: Automaic

   Initial State:

   Input: Input Table.

   Output: assert=True

Test Case Derivation: The environment variable k inputted by the user should match the state variable meant to store the value of k.

How test will be performed: Real numbers between 0 and 1 will be inputed in the environment for k. This value should be exactly equal to the state variable InParams.k . An assert statement will return true if these are equal.

5.1.2 Numerical Parameters Module

The following tests were created to build confidence in the Numpar state variables. The exaxt values of each component of the state variables will not be checked individually. This is because they are created with the input parameters and built-in matlab functions. The Matlab functions are outside of the scope of this plan. The input parameters have already been tested in the previously mentioned tests 1. We only want to ensure that the input variables were correctly entered into the MATLAB functions.
These tests enforce the requirement of finding the spectrum. This is because the numerical parameters will build the matrix whose spectrum we are interested in.

1. test-NumParams-Dom

   Type: Automaic

   Initial State:

   Input: Input Table.

   Output: assert=True

   Test Case Derivation: The domain is created using the linspace matlab function. This function accepts two endpoints and a step size as input. The outputted vector should have endpoints equal to the inputted endpoints. This test will ensure that the domain has the desired endpoints, $+/-xend$.

   How test will be performed: The variable NumParams.Dom(1) and NumPars.Dom(end) should be equal to -NumParams.xend and +NumParams.xend respectively. We are assuming that linspace correctly fills in the gaps.

2. test-NumParams-Dom-Bound

   Type: Automaic

   Initial State:

   Input: Boundary Table.

   Output: assert=True

Test Case Derivation: Is the same as test-NumParams-Dom. The only exception is that integrand in the integral that defines NumParams.xend diverges near one of the end points of integration when k is close to 1.

How test will be performed: The variable NumParams.Dom(1) and NumPars.Dom(end) should be equal to -NumParams.xend and +NumParams.xend respectively. The module should behave normally in the limit.

3. test-NumParams-EllipMat

   Type: Automaic

   Initial State:

   Input: Input Table.

   Output: assert=True

   Test Case Derivation: The $Numpars.ellipjMat$ state variable is created by applying the built in matlab diag function to $Numpars.ellipjdn$. To ensure that the diag function was successfully implemented with $Numpars.ellipjdn$ I will check that at least one of the ellipjdn values corresponds to the appropriate diagonal element. The other elements should follow since the diag function is out of the scope of testing.

   How test will be performed: For the tested points in input tables we will check that assert ($Numpars.ellipjdn(2) == Numpars.EllipMat(2,2)$) returns true.

5.1.3 Spectrum Matrix Module

The following tests were created to ensure that the spectrum matrix state variables are of the correct form. We are not concerned with the specific values as we are having faith in matlabs built in functions. We are only checking that we correctly glued the numerical parameters toegether to form the spectrum matrix.
These tests were built to enforce the requirement of finding the spectrum. This is because we are calculating the spectrum of the state variables in this module.

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-SpecMAT-Tr1

   Type: Automaic

   Initial State:

   Input: Input Table.

   Output: assert=True

Test Case Derivation: From instance model 1 of the SRS (https://github.com/whitere123/CAS741_REW) it follows that the spectral matrices are of the form:

$$SpecMat_x = \begin{bmatrix} NUM_x & -D_x \\ -D_x & -NUM_x \end{bmatrix}$$

Clearly this matrix is equal to its own transpose.

How test will be performed: We will check to see if each spectral matrix is equal to its own transpose. This will be done by checking that $assert(SpecMat.SpecMat_x == SpecMat.SpecMat_x^T)$ is true for $x : (O(8), O(10), O(12))$.

2. test-SpecMAT-Tr2

Type: Automaic

Initial State:

Input: Input Table.

Output: assert=True

Test Case Derivation: From the specification of the Spectrum Matrix Module in the MIS (https://github.com/whitere123/CAS741_REW) it follows that the numerical derivative approximation matrices NUM_x for $x : (O(8), O(10), O(12))$ are equal to negative of their own transpose.

How test will be performed: We will check to see if each matrix is equal to neagtive its own transpose. This will be done by checking that $assert(SpecMat.NUM_x == -SpecMat.NUM_x^T)$ is true for $x : (O(8), O(10), O(12))$.

5.1.4 Plotting Module

1. test-Plotting-Inspect

Type: Manual

Initial State:

Input: Output Parameters

Output: 6 spectral plots on the complex plane

Test Case Derivation: There are a total of six spectral matrices. Each corresponds to one of two boundary solutions to the Non-Linear Schrö dinger equation and one of three numerical algorithms.

How test will be performed: My supervisor will inspect the plots to see if they resmble the theory from Deconinck and L.Segal He will check if the theoretical values superimposed on the plot overlap with the appropriate numerical eigenvalues.

1. test-Plotting-Inspect-Bound

   Type: Manual

   Initial State:

   Input: Boundary Input

   Output: 6 spectral plots on the complex plane

   Test Case Derivation: Is the same as test-Plotting-Inspect. The only exception is that the domain will have a very large bound for k close to 1.

   How test will be performed: My supervisor will inspect the plots to see if they resmble the theory from Deconinck and L.Segal He will check if the theoretical values superimposed on the plot overlap with the appropriate numerical eigenvalues.

5.2 Tests for Nonfunctional Requirements

Planning for nonfunctional tests of units will not be relevant to SpecSearch. For system tests related to Nonfunctional requirements please see the System Verification and Validation Plan at https://github.com/whitere123/CAS741_REW.

5.3 Traceability Between Test Cases and Modules

This section shows the traceability between the modules and the test cases.

Test Case	**Modules**
Rin	M1, M2, M13
Rfind	M4, M6, M7, M8, M9, M10,M11, M13
Rplt	M3, M11, M13

Table 1: Trace Between Test Cases and Modules

References

Bernard Deconinck and Benjamin L.Segal. The stability spectrum for elliptic solutions to the focusing nls equation. *PhysicaD*.

Robert White. *Module Guide*. 2018a.

Robert White. *Module Interface Specification*. 2018b.

Robert White. *System Requirements Specification*. 2018c.

6 Appendix

6.1 Software Verification Checklist

- Did any of the inputs you entered provide suprising results? If yes, what were they?
- Were you able to identify which numerical algorithm and wave solution the plot represented?
- Were all of the plots legible?
- Was the output useful for your research?
- Was it clear how to input the variables?

6.2 Usability Survey Questions

- How long did it take before you could run the software? How many attempts at running SpecSeach did it take before you understood how to properly use it and interpret the output?
- Was this program useful for your research and were you able to interpret the results?
- What aspects of this software do you feel need improvement?
- How does this program compare with other software that finds this particular spectrum?
- Was it clear how and where to input the variables?
- Were the plots and stability results clear?

6.3 Boundary Value Analysis Table

Input ID	k	N	P	Result
I1	0.9	100	2	Pass
I2	0.99	120	2	Pass
I3	0.9999	500	2	Pass
I4	0.99999	550	2	Pass
I5	0.999999	2	X	Pass
I6	0.01	700	2	Pass
I7	0.0001	100	2	Pass
I8	0.000001	400	4	Pass
I9	0.0000001	500	4	Pass

Table 2: Boundary Value Inputs

6.4 Symbolic Parameters

None.