SpecSearch: System Verification and Validation Plan

Robert White

November 9, 2018

1 Revision History

Date	Version	Notes
2018-10-13	1.0	Creation of first draft for VnV plan presentation.
2018-10-20	1.1	Edit of 1.0. I considered all of the feedback from my presentation and made necessary changes. I also prepared the document for submission.
2018-10-22	1.2	Edit of 1.1. Removed comments, updated test cases, added references, improved writing and added test-NFR2c.
2018-10-23	1.3	Edit of 1.2. Minor corrections and revisions.

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
R	Requirement
NFR	Non-functional Requirement

Refer to the SRS Symbols, Abbreviations and Acronyms for a more complete list.

Contents

List of Tables

This document discusses the verification and validation requirements for SpecSearch. The Project Management Body of Knowledge (PMBOK) guide provides unambigious definitions for verification and validation [Project Management Institute, 2017]. [Use BibTeX; it will make your work much easier —SS] PMBOK defines verification as "the evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process."

PMBOK defines validation as "the assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers" [Project Management Institute, 2017]. After reading this document one should be able to create and run test cases to verify and validate SpecSearch.

3 General Information

3.1 Summary

The software that is going to be tested is called SpecSearch. SpeacSearch [proof read —SS] will search for the spectrum (set of eigenvalues) of a particular lax equation from a lax pair that is compatbilible [spell check —SS] with solutions to the Non-Linear Schrödinger (NLS) Equation. SpecSearch will also use the spectral information to determine the stability of the solutions. Refer to the instance models in the SRS for more details.

3.2 Objectives

The qualities that are most important for SpecSearch are highlighted in this section. Many of these qualities are also presented in the SRS.

My supervisor, Dr. Dmitry Pelinovsky, [LaTeX has a rule that it inserts two spaces at the end of a sentence. It detects a sentence as a period followed by a capital letter. This comes up, for instance, with Dr. Pelinovsky. Since the period after Dr. isn't actually the end of a sentence, you need to tell LaTeX to insert one space. You do this either by Dr. Pelinovsky (if you don't mind a line-break between Dr. and Pelinovsky), or Dr. Pelinovsky (to force LaTeX to not insert a line break). —SS] and me intend to use this program

as a tool to search for the continuous spectrum of the previously mentioned spectral problem. SpecSearch will only provide the approximate location of the spectrum and a finite number of elements in the spectrum. We will use this output as a guide in analytically solving for the entire continuous spectrum. Therefore, the code should be reliable and accurate within my supervisor's standards.

There are various numerical methods that can be used to approximate the derivatives of the eigenfunctions [Graselli and Pelinovksy, 2007]. My supervisor and me intend to experiment with these different methods in an attempt to create a more accurate picture of the spectrum. Therefore, it is important that SpecSeach [proof read - better yet, use the progname command from the SRS —SS] is maintainable and manageable.

SpecSearch will also be used by a team of researchers studying rogue waves. These users may not have a strong software developing background. Therefore, it is important that the code is easy to use and has a simple user interface.

The objectives are summarized in the following points for easy reference. The objectives are to:

- Build confidence in software correctness.
- Ensure maintainability and manageability
- Satisfy the requirements of my thesis supervisor and those outlined in the SRS.
- Verify usability.

3.3 References

[You should explain te purpose behind any reference, not just list them. An important reference is missing. You should list your SRS and give a link to the project GitHub page. —SS]

- Bernard Deconinck and Benjamin L.Segal. The stability spectrum for elliptic solutions to the focusing NLS equation. PhysicaD, 2017.

- Robert White. System Requirements Specification for SpecSearch. Github, 2018.
- Matheus Grasselli and Dmitry Pelinovsky. Numerical Mathematics. Jones and Bartlett Learning, 2007.

4 Plan

4.1 Verification and Validation Team

The verification and validation team consists of my thesis supervisor, Dr. Dmitry Pelinovsky, and me. [You should give your name here. —SS]

4.2 SRS Verification Plan

The SRS verification plan consists of feedback from Dr. Dmitry Pelinovsky (supervisor), Dr. Spencer Smith and CAS 741 classmates. My supervisor will provide feedback regarding mathematical theory, model assumptions, constraints and research goals. Feedback from classmates and Dr. Smith will criticize the document outline, readability and requirements. [You can be more specific about the names of your colleagues, since they are given in the Repos.xlsx file for all reviews. —SS]

4.3 Design Verification Plan

The design verification plan will simply involve inspection of the software by my thesis supervisor. [What about the inspection by myself and your classmates? —SS]

4.4 Implementation Verification Plan

The implementation verification plan consists of two parts. The first part is a software verification checklist. The checklist will be completed by researchers , Dr. Pelinovksy [proof read —SS] and me. The checklist verifies that basic software features have been implemented successfully. For example, it checks if the user was able to fufill [spell check —SS] their responsibilities outlined

in the SRS. This checklist can be found in the 7.4.
[great to see a checklist being proposed. —SS]

The second part involves running the software tests outlined in sections 5 and 6. Unit testing will also be performed.

4.5 Software Validation Plan

This section does not apply to SpecSearch. [You should say why not. —SS]

5 System Test Description

5.1 Tests for Functional Requirements

1. test-Rin1

   Initial State: -

   Input: Table 2 in 7.3.

   Output: Error or pass message. [You put your expected output here. There shouldn't be an or for a deterministic system. —SS]

   How test will be performed: Combinations of inputing non-numerical values as input (such as letters), or numerical values outside of their respective constraints, will be considered. A successful test in these instances will be an error message.
   I will also test cases with each variable in the input having an acceptable numerical value. A successful test in these cases will be a pass message. [You actually need a separate test for every error input error case you want to test. You cannot summarize all of the tests like this. The good news is that you can build a table that lists many tests. All you need to do is show what input varies between each test and show the expected output for that test. —SS]

2. test-Rfind1

   Initial State: -

   Input: Table 2 in 7.3 (Similar to test-Rin1).

Output: Size of spectrum array [It looks like you are talking about more than one test again. For specific inputs that you get to pick, can't you say what the size should be? —SS]

How the test will be performed: Passing inputs from test-Rin1 should produce $4n$ eigenvalues. Unsuccessful cases from test-Rin1 should produce no eigenvalue array or an empty eigenvalue array. We are not testing for accuracy in this test.

3. test-Rcon1

   Initial State: -

   Input: Table 2 in 7.3 (Similar to test-Rin1).

   Output: Approximated Spectrum (Connected or disconnected)

   How test will be performed: This test will check to see if there is a sufficient number of points between the tagged portions of the spectrum. Tagged portions are the explicitly calculated eigenvalues from the previous test. The spectrum array should have $2 * m - 1$ approximated values between these "tagged portions", where m is the scaling parameter. Failed test cases from test-Rin1 should have no data for this section. We are not testing for accuracy in this test.
   The scaling parameter has not been studied yet. It will be incorporated in the final draft of the documents.

4. test-Rplt

   Initial State: -

   Input: Table 2 in 7.3 (Similar to test-Rin1).

   Output: A plot of the spectrum.

   How test will be performed: This will involve visual inspection by my supervisor to see if the spectrum plot is adequate.
   There should be no plot for the failed test cases in test-Rin1.

5. test-Rstl

   Initial State: -

   Input: Table 2 in 7.3 (Similar to test-Rin1).

   Output: Binary Variable (Verification of stability)

How test will be performed: The stability results will be compared with the stability analysis in [Deconinck and Segal, 2017]. My supervisor and me are still working on test cases for Rstl.Test-Rstl will be updated in the final draft of the documents. [This looks like a good test case. You should also work out what you mean by comparing the results. Are you just comparing a Boolean value, or are there other measures being compared? —SS]

[Your tests are not specific enough. There should be one set of inputs for each test and one expected output. —SS]

5.2 Tests for Nonfunctional Requirements

1. test-NFR1

   Type: Static

   Initial State: -

   Input/Condition: SpecSearch MATLAB code

   How test will be performed: The software will be manually read by Dr. Pelinovksy [proof read —SS] and I to see if there is a more effective code structure to allow implementation of new numerical algorithms. [What nonfunctional requirement is this test testing? Reference your SRS and remind the reader which quality you are investigating. —SS]

2. test-NFR2a

   Type: Manual

   Initial State: -

   Input: Integer n (same as test-Rin1).

   Output: Operator Matrix.

   How test will be performed: Test-NFR2a will check if the operator matrix is of the correct form. See 7.5 for more detail. The values from table 2 will be used to inspect A1.

3. test-NFR2b
   [You can generate test numbers automatically with LaTeX —SS]

   Type: Manual [Why is this manual? —SS]

   Initial State: -

   Input: Table 2 in 7.3 (Similar to test-Rin1).

   Output: Spectrum.

   How test will be performed: The output will be tested against the boundary value eigen-values derived analytically in Deconinck and Segal. See details below Table 2 of 7.3.

4. test-NFR2c

   Type: Usability Survey (see 7.2)

   How test will be performed: The survey will be administered to the future users of SpecSearch. Feedback from the survey will be considered when updating SpecSearch. The survey is in 7.2.

5.3 Traceability Between Test Cases and Requirements

	Rin	Rfind	Rcon	Rplt	Rstl	NFR1	NFR2
test-Rin	X						
test-Rfind		X					
test-Rcon			X				
test-Rplt				X			
test-Rstl					X		
test-NFR1						X	
test-NFR2a							X
test-NFR2b							X
test-NFR2c							X

Table 1: Traceability Between Test

6 Static Verification Techniques

- Code inspection : I will go through the code to see if each part is correct with respect to the mathematical theory. In particular I will ensure that:
  - Variables are being used in the right context.
  - Functions from other packages are being used in the right context. Some packages have different standards for constants. For instance, one of the complete elliptic integral functions in MATLAB does not square the inputted variable. Theoretical convention dictates that this constant should be squared.
  - The dimensions of the vectors and matrices are appropriate. For example, multiplication of row with column versus column with row.
  - Equations are translated correctly into MATLAB syntax.
- Code walkthrough: My supervisor and me will go through the code together to ensure that:

  - I correctly implemented the mathematical theory and numerical algorithms.
  - I made the code manageable and maintainable for future use.

- Linter: A linter is a program that checks the source code for programmatic and stylistic errors

[You can remove this section and move the contents to the implementation verification plan. I like that you are proposing inspection and walkthroughs. If you are going to use the term walkthrough you should reference the specific steps you are going to follow. A walkthrough is actually a rigorous set of steps. You can do a simplified version, but if what you intend is just a meeting with your supervisor to discuss the code, you shouldn't call it a walkthrough. —SS]

References

[1] Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK Guide)Sixth Edition. Project Management Institute, 2017.

[2] Caswell and Johnston. Design Verification. University of Calary. 2017.

[3] Bernard Deconinck and Benjamin L.Segal. The stability spectrum for elliptic solutions to the focusing NLS equation. PhysicaD, 2017.

[4] Robert White. System Requirements Specification for SpecSearch. Github, 2018.

[5] Matheus Grasselli and Dmitry Pelinovsky. Numerical Methods. Jones and Bartlett Learning; 1 edition, Aug. 13 2007

7 Appendix

7.1 Symbolic Parameters

7.2 Usability Survey Questions

- How long did it take before you could run the software? How many attempts at running SpecSeach [proof read —SS] 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?

7.3 Data Input Tables

The following data table is for test-Rin1. An "X" indicates no-input or incorrect input. Incorrect input is anything that is non-numerical (i.e. letters or symbols).

Input ID	a	b	k	n	Result
I1	1	1	0.9	100	Pass
I2	X	1	0.9	10	Error
I3	1	X	0.9	100	Error
I4	1	1	X	100	Error
I5	1	1	0.9	X	Error
I6	1	1	1	1.2	Error
I7	1	1	1.2	100	Error
I8	1	-1	0.9	100	Error
I9	-1	2	0.9	100	Pass
I10	1	2	0.99	100	Pass
I11	1	3	0.99	500	Pass
I13	1	1	0	150	Pass
I14	-1	2	0.9	330	Pass
I15	1	1	0.99	170	Pass
I16	1	5	0.999	590	Pass
I17	1	1	0	200	Pass

Table 2: Prescribed input for testing Rin

The following table outlines the boundary value test cases that can be verified from previous literature. A 'V' means that the parameter is free to vary.

For these cases all of the eigenvalues, except for four of them, will be purely imaginary. The remaing four have the form: $\lambda = \frac{\pm}{-}\frac{1}{2}a(1+\sqrt{1-k^2})$ and $\lambda = \frac{\pm}{-}\frac{1}{2}a(1-\sqrt{1-k^2})$ for B1 (see table 3).
As for B2, the eigenvalues will have the form: $\lambda = \frac{1}{2}a(\frac{\pm}{-}k + i\sqrt{1-k^2})$ and $\lambda = \frac{1}{2}a(\frac{\pm}{-}k - i\sqrt{1-k^2})$. [Deonick and segal 2017]

Boundary ID	a	b	k
B1	$\sqrt{b}$	a^2	V
B2	$\sqrt{\frac{b}{k^2}}$	a^2k^2	V
B3	V	V	0

Table 3: Prescribed input for Rin

The plan is to run B1 with a=b=1 and k=0.999. B2 will be run with the same input. The expected eigenvalues were previously mentioned.

7.4 Software Verification Checklist

- Does the software allow the user to fufill their responsibilities (as outlined in the SRS)?
- Does the software fufill its intended responsibilities (see SRS)?
- Was the intended user able to use the software and interpret the output for their research purposes?
- Are any of the system constraints violated (see SRS)?

[A checklist is a great idea, but this checklist is more abstract than you want to be. It is ambiguous because so much of the decision making is left to judgement. You want something that has more questions, but simpler questions. It is great to reference your SRS, but rather than open-ended questions, ask specific questions about specific requirements. If you google "software verification checklist," you should get some useful results and inspiration. —SS]

7.5 Correct Matrix Form

test-NFR2a will ensure that the matrix for which the eigen values are calculated, E, is of the correct form. For any given input n, the matrix is 4n by 4n. It is of the form $[A1A2; A2(-A1)]$ where $A1, A2$ is the first row and $A2 - A1$ is the second row. A1 and A2 are matrices. A2 is a diagonal 2n by 2n matrix and A1 is 2n by 2n and defined as follows: A1(i,j) = n/2 for (1,2),(2,3),... (2n-1,2n) and A1(i,j)=-n/2 for (2,1),(3,2) ... (2n,2n-1). A1(1,2n)=-n/2 and A1(2n,1)=n/2. //

For example for arbitrary n it should have the form:

$$A1 = \begin{bmatrix} 0 & \frac{n}{2} & 0 & 0 & 0 & -\frac{n}{2} \\ -\frac{n}{2} & 0 & \frac{n}{2} & 0 & 0 & 0 \\ 0 & -\frac{n}{2} & 0 & \frac{n}{2} & 0 & 0 \\ 0 & 0 & -\frac{n}{2} & 0 & \frac{n}{2} & 0 \\ 0 & 0 & 0 & -\frac{n}{2} & 0 & \frac{n}{2} \\ \frac{n}{2} & 0 & 0 & 0 & -\frac{n}{2} & 0 \end{bmatrix}$$