Optimization Library: System Verification and Validation Plan

Fasil Cheema

February 19, 2024

Revision History

Date	Version	Notes
Feb 19, 2023	1.0	Initial Upload

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test
VnV	Verification and Validation
FR	Functional Requirement
NFR	Non-Functional Requirement
MG	Module Guide
MIS	Module Interface Specification

For further symbols and acronyms that are prevalant throughout the project please reference the SRS (Author, 2019).

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Author, 2019) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document is the system Verification and Validation plan (Syst VnV plan). This document introduces the plan to verify the correctness and the validity of the software being developed. For the VnV plan we will state the FR and the NFR, and try to map out a plan to successfully achieve success on all necessary requirements. We will state all necessary details in this document for a reader to be able to map out and build their own system tests for the corresponding software. The primary objective this document seeks to accomplish is state a plan that can test the compliance with the NFR and FR (verification). Also, to ensure that the software will meet the expectations of users (validation). Since this is a library, further a numerical library, the primary tests will be accuracy of the software and reliability to give relatively accurate solutions to users' specification. ..... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

2 General Information

2.1 Summary

The main project as discussed in the SRS (Author, 2019), is to build a library of function optimizers. Specifically, we will focus on function minimization and restrict our library to a few solvers (see SRS). We will focus on the David-Fletcher-Powell (DFP), Fletcher-Reeves conjugate gradient (FRCG), and Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithms. These methods are split between two classes of function optimizers; Conjugate-Gradient (Fletcher-Reeves) and Quasi-Newton methods (DFP and BFGS). Both of these classes still share the common idea of a line search. For further detail on the families of solvers and optimization please reference the SRS (Author, 2019).

Our library will be used in the following manner; a user will call a specific minimizer listed prior, the user will provide the function required to be minimized (in the appropriate format) with the specified parameters for the minimizer. The minimizer will return a solution based off the parameters given. [Say what software is being tested. Give its name and a brief overview of its general functions. —SS]

2.2 Objectives

It is important to note these optimizers are not oracles; as such we do not expect them to return the correct global minimum each time. The 'correct' solution we seek is the output expected from running the specific optimizer with the specified parameters. Therefore, the expected stated primary objective of finding a 'correct' solution comes with some caveats. We will therefore prioritize our libraries accuracy to their respective algorithms. We wish to build a library of function minimizers that faithfully follow the computation of the original algorithm design, even if that is an incorrect solution. To summarize we wish to build confidence in the software's faithfulness to the original algorithm's expected solutions. We will also expect the user to be able to call the function and the library to execute several modules working in unison without a hitch. We will also wish to demonstrate usability for the expected users of this software. We will not ensure this is the most efficient realization of these algorithms; as this is out of the scope of this project. There are numerous high-computing libaries that have created the algorithms we are building and we do not wish to compete with them. We will also expect that the external libraries from which we will verify our solutions' 'correctness' are reliable and have been thoroughly tested. We will also hold this standard for the textbooks from which we will have test cases for our library.

[State what is intended to be accomplished. The objective will be around the qualities that are most important for your project. You might have something like: "build confidence in the software correctness," "demonstrate adequate usability." etc. You won't list all of the qualities, just those that are most important. —SS]

[You should also list the objectives that are out of scope. You don't have the resources to do everything, so what will you be leaving out. For instance, if you are not going to verify the quality of usability, state this. It is also worthwhile to justify why the objectives are left out. —SS]

[The objectives are important because they highlight that you are aware of limitations in your resources for verification and validation. You can't do everything, so what are you going to prioritize? As an example, if your system depends on an external library, you can explicitly state that you will assume that external library has already been verified by its implementation team. —SS]

2.3 Relevant Documentation

The SRS, MG, and MIS are relevant documents to the VnV and will encompass the complete documentation for the Optimization Library. The SRS will contain a high-level explanation of the purpose, execution, and ideas behind this project. It will also define common themes between the different minimizers, and give a high-level walkthrough of function optimization. This document is great for understanding certain design decisions and the purpose of each module. The MG and MIS documents are important to illustrate the purpose of each module and how they interact with each other. These documents coupled with the VnV should allow a reader to go through the whole software development process for this project. [Reference relevant documentation. This will definitely include your SRS and your other project documents (design documents, like MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. —SS]

Author (2019)

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

3 Plan

In this section we provide a roadmap for our VnV. We wish to ensure the valdity of our work as such we will have a team reviewing our documentation ensuring our ideas are correct and in line with the goals we seek to acheive (Author, 2019). We will also introduce multiple tests that are expansive enough to give confidence to future users that our library will do what we expect it to do. [Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

- Author: Fasil Cheema
- Primary Reviewer: Morteza Mirzaei
- Secondary Reviewer: Dr Spencer Smith
- Secondary Reviewer: Xinyu

[Your teammates. Maybe your supervisor. You should do more than list names. You should say what each person's role is for the project's verification. A table is a good way to summarize this information. —SS]

3.2 SRS Verification Plan

For the SRS verification plan a critical component will be taking criticisms from the reviewers of the SRS. We will implement these criticisms into our updated document. We will also have a checklist for the SRS to ensure we maintain the quality of the document. This checklist will contain information to check over grammatical errors such as spelling mistakes, grammar etc. We will also ensure the checklist covers major concepts we should have in our SRS such as proper documentation of the high level concept of the convex optimization.

[List any approaches you intend to use for SRS verification. This may include ad hoc feedback from reviewers, like your classmates, or you may plan for something more rigorous/systematic. —SS]

[Maybe create an SRS checklist? —SS]

3.3 Design Verification Plan

[Plans for design verification —SS]

This section introduces the plan for design verification. For this plan we will have team members reviewing the high level calculations of the functions are correct. Domain experts are going to ensure these calculations are line with the original algorithms.

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.4 Verification and Validation Plan Verification Plan

We will ensure the validity of the plan by checking with our reviewers and their feedback. The VnV feedback will be implemented into the plan accordingly. We will also have a checklist to ensure the VnV plan covers all bases from minor things such as spelling quality, to major high level tests that need to be verified. [The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

[The review will include reviews by your classmates —SS]
[Create a checklists? —SS]

3.5 Implementation Verification Plan

The verification plan will be done by testing all the FR and NFR. The tests can be found in section 4. In addition, we will check over our code built in Python with a linter; PyLint. We will also have a code inspection and do rubber duck testing. We will also conduct unit testing for modules within the testing scope. Details for unit testing can be found in section 4.

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

3.6 Automated Testing and Verification Tools

In this section we will use several tools for automated testing. We will use the PyTest (Unittest) framework. This will allow multiple unit tests to be created and conducted. We will also employ PyLint as the linter of choice for ensuring we avoid unnecessary mistakes in the code. [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. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

3.7 Software Validation Plan

There is a textbook for convex optimization problems (**?**) where known solutions for their respective algorithms are known. This will be useful to verify

if our algorithm will output the correct solution when given certain parameters. There are also libraries that have trusted builds of the minimizers we seek to make. Scipy comes built in with a minimizer function that allows us to use any specific minimizer we want including the ones presented in this doc (DFP,BFGS, FRCG). [If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

4 System Test Description

4.1 Tests for Functional Requirements

Functional Tests are given for this document in the SRS (Author, 2019). We will have input We will have input tests for our functional requirements in (Author, 2019). [Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

4.1.1 Input

These tests will attempt to verify the following requirements also found in the SRS:

- R1: The function and input vector are in the specified dimension which does not exceed maximum defined dimension.

- R2: The function will notify user of invalid data size (matrix shape mismatch)

4.1.2 Area of Testing1-Functional Requirements 1,2

FR 1 states that the matrices, and vectors in a function we seek to minimize should all be of appropriate size. That is, we should not have an error when attempting to conduct matrix operations. Consequently, FR2 states that the function shall notify the user in the case where this occurs; a size mismatch. This section will cover the area of testing related to these 2 FR. [It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Tests for FR1, FR2

1. test-Default, non-problematic, inputs

   Control: Automatic

   Initial State: Pending

   Input:for matrix $\mathbf{A}$, vector $\vec{\mathbf{b}}$, scalar c We have for $\mathbf{A}$: this matrix will be the identity matrix ranging from dimensions 1 to our max dimension 6. We will have $\vec{\mathbf{b}}$ also be a vector of 1s ranging from dimensions 1 to 6 as well. Finally we will have the scalar c be set to 1 for all the tests.

   Output: Valid (True)

   Test Case Derivation: The size mismatch detector should not have an issue accepting these inputs. They should all pass. [Justify the expected value given in the Output field —SS]

   How test will be performed: Utilizing PyTest we will set the unit tests with the appropriate dimensions as specified above.

2. test-Problematic Input

   Control: Automatic

   Initial State: Pending

Input: for matrix $\mathbf{A}$, vector $\vec{\mathbf{b}}$, scalar c We have for $\mathbf{A}$: this matrix will be the identity matrix ranging from dimensions 1 to our max dimension 6. We will have $\vec{\mathbf{b}}$ also be a vector of 1s ranging from dimensions 1 to 6 as well. Finally we will have the scalar c be set to 1 for all the tests. For this test we will not have the dimension of $\mathbf{A}$ and $\vec{\mathbf{b}}$ be the same but we will ensure they are always different. In other words we will try all possible permutations of dimensions for both the matrix and vector where the dimensions of both are not equal.

Output: The size mismatch detector should detect a problem and raise an error. [The expected result for the given inputs —SS]

Test Case Derivation: This is an invalid size and further in the library there will be invalid matrix operations (cannot do matrix multiplication for matrices of invalid sizes). [Justify the expected value given in the Output field —SS]

How test will be performed: Utilizing PyTest we will set the unit tests with the appropriate dimensions as specified above.

4.2 Tests for Nonfunctional Requirements

The main test for the NFR will be related to accuracy. We will like to ensure that our solution from the algorithm of choice is within the specified accuracy parameter to the solution from the corresponding trusted solution. [The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

4.2.1 Area of Testing1

PSD Test 1 Full step

1. PSD Test 1 Full step

   Type: Automatic

Initial State: Pending

Input/Condition: $\mathbf{A}$ will be the identity matrix ranging from 1 to 6 dimensions. $\vec{\mathbf{b}}$ will be a vector of 1s ranging from 1 6 dimensions along with $\mathbf{A}$. The scalar c will be set to 1. We will use one of each algorithm (DFP, FRCG, and BFGS) with a full step (step size set to 1), an initial starting choice of the zero vector for the corresponding dimension. For the quasi-newton methods we will have an initial Hessian, H_0, set tot be the identity matrix of the corresponding dimensionality.

Output/Result: Relative Error of the two vectors (utilizing the norm of a vector).

How test will be performed: We will compute our respective algorithms then conduct the same computation via our trusted source (scipy.minmize('bfgs')...) and then compute the relative error. This is done by taking the norm of the difference of the two vectors over the norm of the 'true' solution. If this is below the threshold we specify: accuracy parameter ($\epsilon_{acc} = 1\%$)

2. PSD Test 2 adaptive Step

   Type: Automatic

   Initial State: Pending

   Input/Condition: $\mathbf{A}$ will be the identity matrix ranging from 1 to 6 dimensions. $\vec{\mathbf{b}}$ will be a vector of 1s ranging from 1 6 dimensions along with $\mathbf{A}$. The scalar c will be set to 1. We will use one of each algorithm (DFP, FRCG, and BFGS) with an adaptive step size, an initial starting choice of the zero vector for the corresponding dimension. For the quasi-newton methods we will have an initial Hessian, H_0, set tot be the identity matrix of the corresponding dimensionality. In this case the adaptive step size requires the individual algorithm to calculate the step size at each iteration. This is catered for each individual algorithm and adds another degree of complexity.

   Output/Result: Relative Error of the two vectors (utilizing the norm of a vector).

   How test will be performed: We will compute our respective algorithms then conduct the same computation via our trusted source (scipy.minmize('bfgs')...) and then compute the relative error. This

is done by taking the norm of the difference of the two vectors over the norm of the 'true' solution. If this is below the threshold we specify: accuracy parameter ($\epsilon_{acc} = 1\%$)

3. non-PSD Test 2 adaptive Step

   Type: Automatic

   Initial State: Pending

   Input/Condition: $\mathbf{A}$ will be the identity matrix ranging from 1 to 6 dimensions, except the first entry will be -1000. This makes the matrix non-PSD this is to violate the assumptions of our algorithms which require a PSD matrix to converge to a global minima. $\vec{\mathbf{b}}$ will be a vector of 1s ranging from 1 6 dimensions along with $\mathbf{A}$. The scalar c will be set to 1. We will use one of each algorithm (DFP, FRCG, and BFGS) with an adaptive step size, an initial starting choice of the zero vector for the corresponding dimension. For the quasi-newton methods we will have an initial Hessian, H_0, set tot be the identity matrix of the corresponding dimensionality. In this case the adaptive step size requires the individual algorithm to calculate the step size at each iteration. This is catered for each individual algorithm and adds another degree of complexity.

   Output/Result: Relative Error of the two vectors (utilizing the norm of a vector).

   How test will be performed: We will compute our respective algorithms then conduct the same computation via our trusted source (scipy.minmize('bfgs')...) and then compute the relative error. This is done by taking the norm of the difference of the two vectors over the norm of the 'true' solution. If this is below the threshold we specify: accuracy parameter ($\epsilon_{acc} = 1\%$)

4. Portability Test 1- Linux

   Type: Manual

   Initial State: Pending

   Input:

   Output:

How test will be performed: Using the conda environment, on a fresh build of ubuntu 22.0 we wish to first install the environment then run the library with the first test for the NFR (PSD Test 1 Full step).

5. Portability Test 2- Windows

   Type: Manual

   Initial State: Pending

   Input:

   Output:

   How test will be performed: Using the conda environment, on windows 11 we wish to first install the environment then run the library with the first test for the NFR (PSD Test 1 Full step).

4.2.2 Area of Testing2

...

4.3 Traceability Between Test Cases and Requirements

	FR1	FR2	NFR1	NFR2	NFR3	NFR4	NFR5	
TM:CON	X							
GD??	X							
DD??		X						
DD??			X					
DD??				X				
DD??					X			
IM??		X	X	X		X		
IM??		X	X	X			X	
IM??		X	X	X	X			X

Table 1: Traceability Matrix Showing the Connections Between Items of Different Sections

[Provide a table that shows which test cases are supporting which requirements. —SS]

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[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]

5.2 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]

5.2.1 Module 1

[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-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   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:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Author Author. System requirements specification. https://github.com/..., 2019.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

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

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

Appendix — Reflection

[This section is not required for CAS 741 —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage etc. You should look to identify at least one item for each team member.

2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?