Optimization Library: System Verification and Validation Plan

Fasil Cheema

Feb 19, 2024

Revision History

Date	Version	Notes
Feb 19, 2024	1.0	Initial Upload
April 15, 2024	2.0	Revised Version

Contents

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 (Cheema, 2024).

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.

2 General Information

This section provides general information about the OptLib project.

2.1 Summary

The main project as discussed in the SRS (Cheema, 2024), 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 (Cheema, 2024).

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.

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.

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 (Cheema, 2024). 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.

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 achieve (Cheema, 2024). 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.

3.1 Verification and Validation Team

- Author: Fasil Cheema
- Primary Reviewer: Morteza Mirzaei
- Secondary Reviewer: Dr Spencer Smith
- Secondary Reviewer (SRS): Nada
- Secondary Reviewer (VnV): Xinyu
- Secondary Reviewer (MG/MIS): Valerie

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 maintain the quality of the document. We will ensure that we check over grammatical errors such as spelling mistakes, grammar etc. We will also ensure we cover major concepts we should have in our SRS such as proper documentation of the high level concept of the convex optimization.

3.3 Design Verification Plan

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 (primary reviewer Morteza) is going to ensure these calculations are line with the original algorithms.

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.

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. This checks for incorrect code such as incorrect function calls. It can also check for code that ends up not being run or overridden. PyLint also ensures we keep pythonic code this is very good for code readability. We will also have a code inspection and do rubber duck testing. We will also conduct unit testing for modules within the testing scope. This testing will be done by the author by running the PyLint over the code. The author will ensure PyLint does not have any remaining issues (at least ones that cause issues). The author will ensure that PyLint makes sure there is no unused code, incorrect function calls, packages that are not imported, misspelled variables or functions, and any other mistakes in the code. Details for unit testing can be found in section 4.

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. We will also have continuous integration where each module will be individually tested and validated. Then everything comes together and is tested.

3.7 Software Validation Plan

There is a textbook for convex optimization problems (Boyd and Vandenberghe, 2005) 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).

4 System Test Description

This section will contain information on the system tests.

4.1 Tests for Functional Requirements

Functional Tests are given for this document in the SRS (Cheema, 2024). We will have input We will have input tests for our functional requirements in (Cheema, 2024).

4.1.1 Input

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

- FR1: The function and input vector are in the specified dimension which does not exceed maximum defined dimension.
- FR2: The function can be represented in the quadratic form (ie valid parameters)

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 be in the quadratic form. This means the input should be not only in the appropriate dimensionality but also be valid (the format must be a numpy array of 2 dimensions for vectors etc) This section will cover the area of testing related to these 2 FR.

Tests for FR1, FR2

1. test-Default, non-problematic, inputs

   Control: Automatic

   Initial State: Pending

   Input:for matrix $\mathbf{A}$, vector $\tilde{\mathbf{b}}$, 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 $\tilde{\mathbf{b}}$ 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.

   How test will be performed: Import the input verification module and check different values for the input parameters $\mathbf{A}$, $\vec{b}$, c, $\vec{x}_0$, $\mathbf{H}_0$, $\mathbf{B}_0$, step_size, max_s, min_err.

   Check for valid $\mathbf{A}$ (2 dimensional matrix, a square matrix, a numpy array of floats, a numpy array).

   Check for valid $\vec{b}$ (2 dimensional matrix, a row vector of appropriate shape, a numpy array of floats, a numpy array, should match dimensionality of $\mathbf{A}$).

   Check for valid c (a valid float or integer).

   Check for valid $\vec{x}_0$ (a 2 dimensional matrix, a column vector of appropriate shape, a numpy array of floats, a numpy array, should match dimensionality of $\mathbf{A}$).

   Check for valid $\mathbf{H}_0$ (2 dimensional matrix, a square matrix, a numpy array of floats, a numpy array, appropriate dimensionality size matching $\mathbf{A}$).

Check for valid $\mathbf{B}_0$ (2 dimensional matrix, a square matrix, a numpy array of floats, a numpy array, appropriate dimensionality size matching $\mathbf{A}$).

Check for valid step_size (a natural number also checks if below the max number of steps defined in (Cheema, 2024)).

Check for valid max_s (a natural number also checks if below the max number of steps defined in (Cheema, 2024)).

Check for valid min_err (between 0 and 1 and also above the min threshold defined in (Cheema, 2024)).

2. test-Problematic Input

Control: Automatic

Initial State: Pending

Input: for matrix $\mathbf{A}$, vector $\tilde{\mathbf{b}}$, 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 $\tilde{\mathbf{b}}$ 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 $\tilde{\mathbf{b}}$ be the same but we will ensure they are always different. In other words a test for if the matrices are not 2 dimensional matrices, the vectors are not column or row vectors specifically, c is not a scalar, or if matrix dimensions are inconcisent or if an input vector/matrix dimensions are not consistent.

Output: The input verification module should detect a problem and raise an error.

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).

How test will be performed: Import the input verification module and check different values for the input parameters $\mathbf{A}$, $\vec{b}$, c, $\vec{x}_0$, $\mathbf{H}_0$, $\mathbf{B}_0$,

step_size, max_s, min_err.

Check for invalid $\mathbf{A}$ (not 2 dimensional matrix, not a square matrix, not a numpy array of floats, not a numpy array).

Check for invalid $\vec{b}$ (not 2 dimensional matrix, not a row vector of appropriate shape, not a numpy array of floats, not a numpy array, should match dimensionality of $\mathbf{A}$).

Check for invalid c (not a valid float or integer).

Check for invalid $\vec{x}_0$ (not 2 dimensional matrix, not a column vector of appropriate shape, not a numpy array of floats, not a numpy array, should match dimensionality of $\mathbf{A}$).

Check for invalid $\mathbf{H}_0$ (not 2 dimensional matrix, not a square matrix, not a numpy array of floats, not a numpy array, not appropriate dimensionality size matching $\mathbf{A}$).

Check for invalid $\mathbf{B}_0$ (not 2 dimensional matrix, not a square matrix, not a numpy array of floats, not a numpy array, not appropriate dimensionality size matching $\mathbf{A}$).

Check for invalid step_size (not a natural number also checks if above the max number of steps defined in (Cheema, 2024)).

Check for invalid max_s (not a natural number also checks if above the max number of steps defined in (Cheema, 2024)).

Check for invalid min_err (not between 0 and 1 and also below the min threshold defined in (Cheema, 2024)).

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. Wherever the 'textbook' is referenced it is in reference to the following text (Boyd and Vandenberghe, 2005).

4.2.1 Area of Testing NFR

PSD Test 1 Exact step

1. PSD Test 1 Full step

   Type: Manual

   Initial State: Pending

   Input/Condition: $\mathbf{A}$ will be set to a matrix specified. $\tilde{\mathbf{b}}$ will be set to a vector specified. The scalar c will be set to 0. 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 or inverse Hessian, B_0 set to be the identity matrix of the corresponding dimensionality.

   Output/Result: Relative Error of the two vectors (utilizing the norm of a vector). The result will be from a textbook and will correspond to the known value. Final result is a Pass/Fail depending on if the norm value is below the err threshold.

   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. A trusted textbook will supply the example and also provide an expected answer (Boyd and Vandenberghe, 2005). We will 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 set to a matrix specified. $\tilde{\mathbf{b}}$ will be set to a vector specified. The scalar c will be set to 0. 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 or inverse Hessian, B_0 set to 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). Final result is a Pass/Fail depending on if the norm value is below the err threshold.

How test will be performed: We will compute our respective algorithms then conduct the same computation via our trusted source a textbook with the specific answer (also scipy.minmize('bfgs')...) and then compute the relative error (Boyd and Vandenberghe, 2005). 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 3 adaptive Step

   Type: Automatic

   Initial State: Pending

   Input/Condition: $\mathbf{A}$ will be set to a matrix specified. $\tilde{\mathbf{b}}$ will be set to a vector specified. The scalar c will be set to 0. 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 or inverse Hessian, B_0 set to 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). Final result is a Pass/Fail depending on if the norm value is below the err threshold.

How test will be performed: We will compute our respective algorithms then conduct the same computation via our trusted source; a textbook example (also scipy.minmize('bfgs')...) and then compute the relative error (Boyd and Vandenberghe, 2005). 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. Test 4 Exact Step

   Type: Automatic

   Initial State: Pending

   Input/Condition: Given the input parameters as specified from a textbook example we will compute the exact step for each algorithm (Boyd and Vandenberghe, 2005). The result will be compared to the result in the textbook; the oracle answer.

   Output/Result: Final result is a Pass/Fail depending on if the norm value is below the err threshold.

   How test will be performed: Compute the exact step for each algorithm and obtain a value. If the norm between this scalar value and the accepted answer is below the error threshold we make a decision.

5. Test 5 Gradient

   Type: Automatic

   Initial State: Pending

   Input/Condition: Given a vector and input parameters from a textbook example (Boyd and Vandenberghe, 2005).

   Output/Result: Final result is a Pass/Fail depending on if the norm value is below the err threshold.

   How test will be performed: Compute the gradient for each initial vector and corresponding quadratic form (input parameters define this) and obtain a value. If the norm between this column vector and the accepted answer is below the error threshold we make a decision.

6. Test 6 Search Direction

   Type: Automatic

   Initial State: Pending

   Input/Condition: Given a vector and input parameters from a textbook example (Boyd and Vandenberghe, 2005).

   Output/Result: Final result is a Pass/Fail depending on if the norm value is below the err threshold.

   How test will be performed: Compute the search for each initial vector and corresponding quadratic form (input parameters define this) and obtain a value. If the norm between this column vector and the accepted answer is below the error threshold we make a decision.

7. Portability Test 1- Linux

   Type: Manual

   Initial State: Pending

   Input:

   Output:

   How test will be performed: Ensuring we install the dependencies, 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).

8. 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).

4.3 Traceability Between Test Cases and Requirements

Note that TF1 refers to Test Function Requirement 1, and TNF1 refers to Test Non Functional Requirement 1.

	TF1	TF2	TNF1	TNF2	TNF3	TNF4	TNF5	TNF6	TNF7	TNF8
FR1	X	X								
FR2		X								
NFR1			X	X	X	X	X	X		
NFR2		X				X	X	X		
NFR3		X				X	X	X		
NFR4									X	X
NFR5						X	X	X		

Table 1: Traceability Matrix Showing the Connections Between Test Cases and Functional Requirements

References

Stephen P. Boyd and Lieven Vandenberghe. Convex optimization. *Journal of the American Statistical Association*, 100:1097 – 1097, 2005.

Fasil Cheema. System requirements specification. https://github.com/FasilCheema/OptimizationLibrary, 2024.