

# Optimization Library: System Verification and Validation Plan

Fasil Cheema

Feb 19, 2024

## Revision History

Date	Version	Notes
Feb 19, 2024	1.0	Initial Upload

# Contents

<b>1</b>	<b>Symbols, Abbreviations, and Acronyms</b>	<b>iii</b>
<b>2</b>	<b>General Information</b>	<b>1</b>
2.1	Summary . . . . .	1
2.2	Objectives . . . . .	2
2.3	Relevant Documentation . . . . .	2
<b>3</b>	<b>Plan</b>	<b>3</b>
3.1	Verification and Validation Team . . . . .	3
3.2	SRS Verification Plan . . . . .	3
3.3	Design Verification Plan . . . . .	3
3.4	Verification and Validation Plan Verification Plan . . . . .	4
3.5	Implementation Verification Plan . . . . .	4
3.6	Automated Testing and Verification Tools . . . . .	4
3.7	Software Validation Plan . . . . .	4
<b>4</b>	<b>System Test Description</b>	<b>5</b>
4.1	Tests for Functional Requirements . . . . .	5
4.1.1	Input . . . . .	5
4.1.2	Area of Testing1-Functional Requirements 1,2 . . . . .	5
4.2	Tests for Nonfunctional Requirements . . . . .	6
4.2.1	Area of Testing1 . . . . .	6
4.3	Traceability Between Test Cases and Requirements . . . . .	9

# 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 prevalent 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

### 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 libraries 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. 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.

[Cheema \(2024\)](#)

## 3 Plan

In this section we provide a roadmap for our VnV. We wish to ensure the validity 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: Xinyu

### 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.

### 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 are 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. 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.

### 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.

### 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

### 4.1 Tests for Functional Requirements

Functional Tests are given for this document in the SRS ([Cheema, 2024](#)). 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:

- 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.

#### 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.

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.

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: 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.

### 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.minimize('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.minimize('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 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).

How test will be performed: We will compute our respective algorithms then conduct the same computation via our trusted source (`scipy.minimize('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).

### 4.3 Traceability Between Test Cases and Requirements

	Test F1	Test F2	Test NF1	Test NF2	Test NF3	Test NF4	Test NF5
FR1	X	X					
FR2							
NFR1			X	X	X		
NFR2							
NFR3							
NFR4						X	X
NFR5							

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.