Verification and Validation Report: OptLib

Fasil Cheema

April 16, 2024

1 Revision History

Date	Version	Notes
April 15, 2024	1.0	Initial Upload

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
VnV	Verification and Validation
TC	Test Case
MG	Module Guide
MIS	Module Interface Specification
SRS	Software Requirements Specification
FR	Functional Requirement
NFR	Nonfunctional Requirement

Contents

List of Tables

List of Figures

This document provides a detailed summary of the VnV report. We execute and report the results of the tests that should satisfy the requirements in this document. Sections 3 and 4 are the tests for the functional and non functional requirements respectively.

3 Functional Requirements Evaluation

In this section we conduct and report the tests for the functional requirements.

3.1 Dynamic Testing

This section conducts the tests in section 4.1 of the VnV plan.

3.1.1 Input Validation

Tables 1 and 2 are tests for FR1 in the SRS and VnV plan (Cheema, 2024). These tests are designed to validate when the input parameters are valid. After defining the inputs as follows we simply test using the following code:

```
from inputverify import InputVerifier

validator = InputVerifier()
valid, err_msg = validator.inputverify(A_mat, b_vec, c, x_0, H_0, ...
... B_0, step_size, max_s, min_err)
```

if valid is True we pass the test. If valid is False the test is failed.

This is the first test for FR1. We need to also evaluate this for the case where we have invalid inputs. See tables 3, 4.

As expected for the valid inputs we pass all tests. For the invalid tests we fail all tests. The tests not only check for several things. As stated in the MG, MIS (Cheema, 2024) we have to ensure the types are consistent with the parameters. We also need to check for the shapes of the input matrices/vectors. Note that for example b_vec is a 1 by d row vector whereas x_0 is a d by 1 column vector. This is why these tests are critical for the success of the rest of the library in execution.

Table 1: FR 1 - Test 1- Valid Inputs

Parameter	Inputs	Result
A_mat	A_mat = np.array([[0.5,1],[1,1.5]])	Pass
b_vec	b_vec = np.array([[-1,-2]])	Pass
c	0	Pass
x_0	x_0 = np.array([[0],[1]])	Pass
H_0	H_0 = np.array([[1,0],[0,1]])	Pass
B_0	B_0 = np.array([[1,1],[1,1]])	Pass
step_size	5	Pass
max_s	10000	Pass
min_err	0.05	Pass

4 Nonfunctional Requirements Evaluation

This section conducts the tests stated in section 4.2 of the VnV plan.

4.1 Dynamic Testing

4.1.1 PSD Test Full Step

For this test we simply do a full call to the library and utilize the Fletcher Reeves Conjugate Gradient Descent method and obtain a final value:

```
from optlib import optimizer

optimizer = optimizer()
result =  optimizer.FRCG(A_mat, b_vec, c, x_0, ...
..., step_size, max_s, min_err)
```

The oracle final value according to the textbook should be

```
np.array([[9],[3]])
```

We repeat the test for BFGS and DFP with the following function calls:

```
result = optimizer.DFP(A_mat, b_vec, c, x_0, B_0 ...
..., step_size, max_s, min_err)
```

Table 2: FR 1- Test 2 Valid Inputs

Parameter	Inputs	Result
A_mat	A_mat = np.array([[0.5,1,3],[1,1.5,4],[1,0,1]])	Pass
b_vec	b_vec = np.array([[-1,-2,-3]])	Pass
c	0	Pass
x_0	x_0 = np.array([[0],[1],[2]])	Pass
H_0	H_0 = np.array([[1,0,0],[0,1,0],[0,0,1]])	Pass
B_0	B_0 = np.array([[1,1,1],[1,1,1],[1,1,1]])	Pass
step_size	5	Pass
max_s	10000	Pass
min_err	0.05	Pass

```
result = optimizer.BFGS(A_mat, b_vec, c, x_0, H_0...
... , step_size ,max_s, min_err)
```

we obtain the following results:

```
np.array([[1.2222],[0.77778]])
```

and

```
np.array([[1.166667],[0.833333]])
```

for the BFGS and DFP algorithms respectively. This is within the error threshold of the desired result.

4.1.2 PSD Test Exact Step

when doing the test in table 6 with exact step we expect the following:

```
np.array([[1],[1]])
```

In all tests with each algorithm we obtain the desired result.

The function calls

Table 3: FR 1- Test 3 Invalid Inputs

Parameter	Inputs	Result
A_mat	A_mat = np.array([[0.5,3],[1,4],[1,1]])	Fail
b_vec	b_vec = np.array([-1,-2])	Fail
c	'hello'	Fail
x_0	x_0 = np.array([[0],[1],[2],[6]])	Fail
H_0	H_0 = np.array([[1,0,0],[0,1,0],[0,0,1]])	Fail
B_0	B_0 = np.array([[1,1],[1,1]])	Fail
step_size	True	Fail
max_s	-10	Fail
min_err	0.000001	Fail

```
result = optimizer.DFP(A_mat, b_vec, c, x_0 ...
..., -1, max_s, min_err)
result = optimizer.DFP(A_mat, b_vec, c, x_0, B_0 ...
..., -1, max_s, min_err)
result = optimizer.BFGS(A_mat, b_vec, c, x_0, H_0 ...
..., -1, max_s, min_err)
```

with the value of the parameters defined in the table 6.

4.1.3 Exact Step Calculation

We first enumerate the parameters configuration with the desired parameters. We do this in the following function call

```
parameters = paramconfig(A_mat, b_vec, c, x_0, H_0, B_0, step_size, ...
... max_s, min_err)
```

The parameters are found in table 7. We then compute the exact step with the following line:

```
s_0 = compute_gradient(x_0, parameters)
alpha_t = exactStep(x_0, s_0, parameters)
```

The expected result is $\frac{1}{2}$ and we achieve the desired result.

Table 4: FR 1- Test 4 Invalid Inputs

Parameter	Inputs	Result
A_mat	A_mat = np.array([0])	Fail
b_vec	b_vec = np.array([[-1],[-2],[6]])	Fail
c	np.ones((2,2))	Fail
x_0	x_0 = np.array([[0],[1],[2])	Fail
H_0	H_0 = np.array([[1,0],[0,0]])	Fail
B_0	B_0 = np.array([[1,1,4],[1,1,5],[5,6,7]])	Fail
step_size	False	Fail
max_s	10e18	Fail
min_err	-5	Fail

4.1.4 Gradient Calculation

Now we must check if the gradient of the quadratic form is accurate. We again utilize the textbook examples (Boyd and Vandenberghe, 2005). We call the compute_gradient method from the gradcalc module:

```
s_0 = compute_gradient(x_0, parameters)
```

In this example we use parameters specified in table 8. We expect a value of

```
np.array([[-1],[2]]
```

As in the textbook example we obtain this result.

4.1.5 Search Direction Calculation

Now we must check if the search direction for each method is accurate. We again utilize the textbook examples (Boyd and Vandenberghe, 2005). We call the search direction methods for each algorithm:

```
s_t = searchdir.dirBFGS(H_0, x_t, parameters)
s_t = searchdir.dirDFP(B_0, x_t, parameters)
s_t = searchdir.dirFRCG(x_t, x_prev, s_prev parameters)
```

In this example we use parameters specified in table 8. We expect a value of

```
np.array([[2],[1]]
```

Table 5: NFR 1 - Test 5- PSD Matrix Full Step

Parameter	**Inputs**
A_mat	A_mat = np.array([[1,0.5],[0.5,1.5]])
b_vec	b_vec = np.array([[-3,-4]])
c	0
x_0	x_0 = np.array([[0],[1]])
H_0	H_0 = np.array([[1,0],[0,1]])
B_0	B_0 = np.array([[1,1],[1,1]])
step_size	1
max_s	10000
min_err	0.05

for the DFP and the BFGS function calls. As for the FRCG function call we expect the following:

```
np.array([[-1],[2]]
```

As in the textbook example we obtain this result.

4.2 Portability Tests

For this test we simply have to ensure NFR3 and NFR4 can be run. That is on a clean build can we access our library. We have a bash script in the src module to run the tests. We first have a clean build and ensure Python 3.9¿= is installed and numpy 1.19¿= is installed. The test worked on a clean build as long as we had these two dependencies installed.

5 Comparison to Existing Implementation

Although scipy.optimize works well for desired results. The same results were also found for the tests with the textbook examples.

Table 6: NFR 1 - Test 6- PSD Matrix Exact Step

Parameter	**Inputs**
A_mat	A_mat = np.array([[1,0.5],[0.5,1.5]])
b_vec	b_vec = np.array([[-3,-4]])
c	0
x_0	x_0 = np.array([[0],[1]])
H_0	H_0 = np.array([[1,0],[0,1]])
B_0	B_0 = np.array([[1,1],[1,1]])
step_size	-1
max_s	10000
min_err	0.05

6 Unit Testing

Each module was tested and integrated into the project. First we tested the vecmath module with simple numpy arrays to ensure our vector math worked. Then for textbook examples we knew the asnwers to we tested the gradient calculation as in test 4.1.4, then tested exact step calculation as in 4.1.3, then tested search direction calculation as in 4.1.5. We also tested the ParamConfig and InputVerify modules separately on the command line as well as imports of each package before integrating everything together. Finally tested the whole function calls as in 4.1.1 and 4.1.2.

7 Changes Due to Testing

Due to testing we had to carefully evaluate the shape of the numpy matrices. This was a careful decision as sometimes errors are not raised when incompatible matrices are multiplied. For this reason we changed all numpy arrays to be 2d arrays including column and row vectors. We also introduced specialized checks to make sure two vectors/matrices that are operated on are compatible.

Table 7: NFR 1 - Test 7-Exact Step

Parameter	Inputs
A_mat	A_mat = np.array([[1,0.5],[0.5,1.5]])
b_vec	b_vec = np.array([[-3,-4]])
c	0
x_0	x_0 = np.array([[0],[1]])
H_0	H_0 = np.array([[1,0],[0,1]])
B_0	B_0 = np.array([[1,1],[1,1]])
step_size	-1
max_s	10000
min_err	0.05

8 Automated Testing

Please see the shell script in the src file to see the tests conducted. The filename is FRTest1.sh

9 Trace to Requirements

Please refer to the SRS (Cheema, 2024) where traceability matrices can be found to the requirements. This is to have easy access and accountability to the changes in a large project.

10 Trace to Modules

Please refer to the VnV plan, and MG/MIS documents (Cheema, 2024) where traceability matrices can be found to the respective modules. Again traceability for the tests to their respective modules ensures the validity of the final project.

Table 8: NFR 1 - Test 8-Gradient Calculation

Parameter	**Inputs**
A_mat	A_mat = np.array([[1,0.5],[0.5,1.5]])
b_vec	b_vec = np.array([[-3,-4]])
c	0
x_0	x_0 = np.array([[0],[1]])
H_0	H_0 = np.array([[1,0],[0,1]])
B_0	B_0 = np.array([[1,1],[1,1]])
step_size	-1
max_s	10000
min_err	0.05

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.

Table 9: NFR 1 - Test 9-Search Direction Calculation

Parameter	**Inputs**
A_mat	A_mat = np.array([[1,0.5],[0.5,1.5]])
b_vec	b_vec = np.array([[-3,-4]])
c	0
x_t	x_t = np.array([[0],[1]])
H_t	H_t = np.array([[1,0],[0,1]])
B_t	B_t = np.array([[1,0],[0,1]])
step_size	-1
max_s	10000
min_err	0.05
x_prev	np.array([[0],[2]])
s_prev	np.array([[1],[-2]])

Appendix — Reflection

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

1. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)

2. We had many changes. We introduced many more tests to validate the individual modules. This included tests for the modules gradcalc, stepsizecalc, searchdir. The heavy emphasis on these tests is due to how they are heavily employed by the rest of the project. For this reason their validity is critical and more tests were needed to ensure correctness.