

A Library of Simplex Method Solvers: System Verification and Validation Plan

Hanane Zlitni

October 22, 2018

1 Revision History

Date		Version	Notes
October 2018	22,	1.0	First Draft

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
V&V	Verification and Validation
LoSMS	Library of Simplex Method Solvers
CA	Commonality Analysis
SRS	Software Requirements Specification
A	Assumption
IM	Instance Model
<i>s. t.</i>	Subject to
Z	Optimal solution of the objective function
x_1, x_2, x_3	Decision variables

Contents

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	ii
3	General Information	1
3.1	Summary	1
3.2	Objectives	1
3.3	References	1
4	Plan	2
4.1	Verification and Validation Team	2
4.2	SRS Verification Plan	2
4.3	Design Verification Plan	2
4.4	Implementation Verification Plan	2
4.5	Software Validation Plan	2
5	System Test Description	2
5.1	Tests for Functional Requirements	2
5.1.1	Tests for Solving Maximization Linear Programs	2
5.1.2	Tests for Solving Minimization Linear Programs	4
5.1.3	Tests for Faulty Inputs	5
5.2	Tests for Nonfunctional Requirements	6
5.2.1	Usability	6
5.2.2	Portability	7
5.2.3	Accuracy	7
5.2.4	Correctness & Performance	7
5.2.5	Stability	8
5.3	Traceability Between Test Cases and Requirements	8
6	Static Verification Techniques	9
7	Appendix	11
7.1	Symbolic Parameters	11
7.2	Usability Survey Questions	11

List of Tables

1	Traceability Between Test Cases and Requirements	8
---	--	---

List of Figures

This document describes the system verification and validation (V&V) plan for the Library of Simplex Method Solvers (LoSMS) tool. It is based on the tool’s commonality analysis (CA) that can be found, along with the full documentation of LoSMS, in the following link: <https://github.com/hananezlitni/HZ-CAS741-Project>.

The V&V plan starts by providing general information about the tool and this document in Section 3. Then, Section 4 provides additional details about the plan, which includes information about the V&V team, the SRS, design and implementation verification plans and the software validation plan. This is followed by the system test description in Section 5, which consists of tests for the tool’s functional and nonfunctional requirements and traceability between test cases and requirements. The document is concluded by Section 6 which describes the techniques for static verification.

3 General Information

3.1 Summary

The software under test, LoSMS, is a general-purpose program family that facilitates obtaining the optimal solution of a linear program, using the simplex method, given the objective function, the objective function goal (maximization or minimization) and the linear constraints. Since the simplex algorithm is widely used in various fields, LoSMS is intended to be used by people from different backgrounds to help them optimize parameters of their choice.

3.2 Objectives

The objective of this verification and validation plan is to build confidence in the correctness of the LoSMS tool (i.e. it produces the correct output for the corresponding inputs), while providing satisfactory usability.

3.3 References

Different sections in this document refer to the tool’s CA (Zlitni (2018)).

4 Plan

4.1 Verification and Validation Team

The verification and validation team consists of one member: Hanane Zlitni.

4.2 SRS Verification Plan

The CA for the LoSMS tool will be verified by getting feedback from Dr. Spencer Smith and my CAS 741 classmates.

4.3 Design Verification Plan

LoSMS's design documents will be verified by getting feedback from Dr. Spencer Smith and my CAS 741 classmates.

4.4 Implementation Verification Plan

The implementation of the LoSMS tool will be verified statically by performing code review with Dr. Spencer Smith and my CAS 741 classmates and dynamically by executing the test cases detailed in this plan and the unit V&V plan using testing frameworks (e.g. JUnit/PyUnit).

4.5 Software Validation Plan

Not applicable for LoSMS.

5 System Test Description

System testing for the LoSMS tool ensures that the correct inputs produce the correct outputs. The test cases in this section are derived from the instance models and the requirements detailed in the tool's CA.

5.1 Tests for Functional Requirements

5.1.1 Tests for Solving Maximization Linear Programs

1. **T1: Unique Optimal Solution**

Control: Automatic

Initial State: -

Input: $\max Z = 2x_1 - 3x_2 + x_3$

$$\begin{aligned} s. t. \quad & x_1 + x_2 + x_3 \leq 10 \\ & 4x_1 - 3x_2 + x_3 \leq 3 \\ & 2x_1 + x_2 - x_3 \leq 10 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Output: $Z = 3$, occurring when $x_1 = 0, x_2 = 0, x_3 = 3$

How test will be performed: Unit testing using JUnit/PyUnit

Test Case Derivation: IM1 in [Zlitni \(2018\)](#)

2. T2: Multiple Optimal Solutions

Control: Automatic

Initial State: -

Input: $\max Z = 3x_1 + 2x_2$

$$\begin{aligned} s. t. \quad & 3x_1 + 2x_2 \leq 180 \\ & x_1 \leq 40 \\ & x_2 \leq 60 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Output: $Z = 180$, occurring when $x_1 = 40, x_2 = 30$ & $x_1 = 20, x_2 = 60$

How test will be performed: Unit testing using JUnit/PyUnit

Test Case Derivation: IM1 in [Zlitni \(2018\)](#)

3. T3: No Optimal Solution

Control: Automatic

Initial State: -

Input: $\max Z = 2x_1 + x_2$

$$\begin{aligned} s. t. \quad & x_1 - x_2 \leq 10 \\ & 2x_1 - x_2 \leq 40 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Output: “This linear program does not have an optimal solution”, or a corresponding exception

How test will be performed: Unit testing using JUnit/PyUnit

Test Case Derivation: IM1 in [Zlitni \(2018\)](#) and [Niu](#)

5.1.2 Tests for Solving Minimization Linear Programs

1. T4: Unique Optimal Solution

Control: Automatic

Initial State: -

Input: $\min Z = -2x_1 + 3x_2$

$s. t.$ $3x_1 + 4x_2 \leq 24$

$7x_1 + 4x_2 \leq 16$

$x_1, x_2 \geq 0$

Output: $Z = -4.57$, occurring when $x_1 = 2.29$, $x_2 = 0$

How test will be performed: Unit testing using JUnit/PyUnit

Test Case Derivation: IM2 in [Zlitni \(2018\)](#)

2. T5: No Optimal Solution

Control: Automatic

Initial State: -

Input: $\min Z = 3x_1 + 14x_2$

$s. t.$ $-x_1 - 5x_2 \leq -6$

$-x_1 - 4x_2 \leq -5$

$-x_1 - 3x_2 \leq -4$

$-x_1 - 2x_2 \leq -5$

$-x_1 - x_2 \leq -6$

$x_1, x_2 \geq 0$

Output: “This linear program does not have an optimal solution”, or a corresponding exception

How test will be performed: Unit testing using JUnit/PyUnit

Test Case Derivation: IM2 in [Zlitni \(2018\)](#)

5.1.3 Tests for Faulty Inputs

1. T6: No Objective Function

Control: Automatic

Initial State: -

Input: \min

$$\begin{aligned} s. t. \quad & x_1 + x_2 + x_3 \leq 10 \\ & 4x_1 - 3x_2 + x_3 \leq 3 \\ & 2x_1 + x_2 - x_3 \leq 10 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Output: “Error: No objective function”, or a corresponding exception

How test will be performed: Unit testing using JUnit/PyUnit

Test Case Derivation: IM1 & IM2 in [Zlitni \(2018\)](#)

2. T7: No Linear Constraints

Control: Automatic

Initial State: -

Input: $\max Z = 2x_1 - 3x_2 + x_3$

Output: “Error: No linear constraints”, or a corresponding exception

How test will be performed: Unit testing using JUnit/PyUnit

Test Case Derivation: IM1 & IM2 in [Zlitni \(2018\)](#)

3. T8: No Objective Function Goal

Control: Automatic

Initial State: -

Input: $Z = 2x_1 - 3x_2 + x_3$

$$\begin{aligned} s. t. \quad & x_1 + x_2 + x_3 \leq 10 \\ & 4x_1 - 3x_2 + x_3 \leq 3 \\ & 2x_1 + x_2 - x_3 \leq 10 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Output: “Error: No objective function goal”, or a corresponding exception

How test will be performed: Unit testing using JUnit/PyUnit

Test Case Derivation: IM1 & IM2 in [Zlitni \(2018\)](#)

4. **T9: No Non-negativity Constraints/Negative Decision Variables**

Control: Automatic

Initial State: -

Input: $\max Z = 2x_1 - 3x_2$

$s. t.$ $x_1 + x_2 \leq 5$
 $4x_1 - 3x_2 \leq 4$

Output: “Error: The decision variables must be positive”, or a corresponding exception

How test will be performed: Unit testing using JUnit/PyUnit

Test Case Derivation: Theoretical Model 1 in [Zlitni \(2018\)](#)

5. **T10: Greater Than or Equal to Inequalities**

Control: Automatic

Initial State: -

Input: $\max Z = 2x_1 - 3x_2$

$s. t.$ $x_1 + x_2 \leq 5$
 $4x_1 - 3x_2 \geq 4$
 $x_1, x_2 \geq 0$

Output: “Error: The inequalities of the main constraints must be of type less than or equal to”, or a corresponding exception

How test will be performed: Unit testing using JUnit/PyUnit

Test Case Derivation: A2 in [Zlitni \(2018\)](#)

5.2 Tests for Nonfunctional Requirements

5.2.1 Usability

1. **T11: Test for the Usability of LoSMS**

Type: Usability Testing

Initial State: -

Input/Condition: -

Output/Result: -

How test will be performed: Asking participants to try the tool then answer the usability survey questions (see Appendix 7.2). The goal is to ensure that the library provided the services the users requested and that they are satisfied with the results obtained.

5.2.2 Portability

1. T12: Test for the Portability of LoSMS

Type: Static

Initial State: -

Input/Condition: -

Output/Result: -

How test will be performed: Running LoSMS on Mac, Windows and Linux operating systems

5.2.3 Accuracy

1. T13: Test for the Accuracy of the Outputs

Type: Dynamic

Initial State: -

Input/Condition: -

Output/Result: -

How test will be performed: I plan to report the relative error of the expected output for each test case detailed in this document

5.2.4 Correctness & Performance

1. T14: Test for the Correctness & Performance of LoSMS

Type: Parallel Testing

Initial State: -

Input/Condition: -

Output/Result: -

How test will be performed: I plan to make a comparison between LoSMS and MatLab to evaluate the correctness and performance of LoSMS

5.2.5 Stability

1. T15: Test for the Stability of LoSMS Under Heavy Load

Type: Stress Testing

Initial State: -

Input/Condition: -

Output/Result: -

How test will be performed: Use the library to solve problems with great number of inputs and observe how it would behave

5.3 Traceability Between Test Cases and Requirements

The following table describes the mapping between the test cases and requirements.

Test Case Number	Requirements
T1	R1, R4, R5, R6, R7
T2	R1, R4, R5, R6, R7
T3	R1, R4, R5, R6, R8
T4	R1, R4, R5, R6, R7
T5	R1, R4, R5, R6, R8
T6	R1, R2, R3
T7	R1, R2, R3
T8	R1, R2, R3
T9	R1, R2, R3
T10	R1, R2, R3

Table 1: Traceability Between Test Cases and Requirements

6 Static Verification Techniques

Static verification of the LoSMS library implementation will be performed using code review with Dr. Spencer Smith and my CAS 741 classmates.

References

- Shun-Chen Niu. Special situations in the simplex algorithm.
URL <https://www.utdallas.edu/~scniu/OPRE-6201/documents/LP10-Special-Situations.pdf>.
- Hanane Zlitni. Commonality analysis of a library of simplex method solvers, 2018. URL <https://github.com/hananezlitni/HZ-CAS741-Project/blob/master/docs/SRS/CA.pdf>.

7 Appendix

This section provides additional content related to this system V&V plan.

7.1 Symbolic Parameters

There are no symbolic parameters used in this document.

7.2 Usability Survey Questions

1. Did LoSMS successfully provide all the services you requested?
(Yes / No)
2. How confident are you that LoSMS provided you with the correct results?
(1 / 2 / 3 / 4 / 5) ; *(1) being not confident at all and (5) being very confident*
3. How satisfied are you with the library's response time?
(1 / 2 / 3 / 4 / 5) ; *(1) being not satisfied at all and (5) being very satisfied*
4. How likely are you to recommend LoSMS to a friend?
(1 / 2 / 3 / 4 / 5) ; *(1) being very unlikely and (5) being very likely*
5. Rate your overall satisfaction with LoSMS out of 10
6. Please provide us with general comments, if any