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

Hanane Zlitni

October 22, 2018

1 Revision History

Date	Version	Notes
October 18, 2018	1.0	Presentation Draft

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
V&V	Verification and Validation
LoSMS	A Library of Simplex Method Solvers
CA	Commonality Analysis

[symbols, abbreviations or acronyms – you can simply reference the SRS tables, if appropriate —SS]

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 3
  - 5.1.1 Tests for Solving Maximization Linear Programs 3
  - 5.1.2 Tests for Solving Minimization Linear Programs 4
  - 5.1.3 Tests for Faulty Inputs 4
- 5.2 Tests for Nonfunctional Requirements 6
  - 5.2.1 Usability 6
  - 5.2.2 Portability 6
  - 5.2.3 Robustness 7
- 5.3 Traceability Between Test Cases and Requirements 7

6 Static Verification Techniques **7**

7 Appendix **9**
- 7.1 Symbolic Parameters 9
- 7.2 Usability Survey Questions 9

List of Tables

List of Figures

This document describes the system verification and validation plan (V&V) for the LoSMS (Library of Simplex Method Solvers) 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. Finally, 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. [get back to this —HZ]

3.2 Objectives

[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]

3.3 References

- CA

- ”Special situations in Simplex method” paper

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: [reference IM1 in CA —HZ]

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: [reference IM1 in CA —HZ]

3. **T3: No Optimal Solution**

   Control: Automatic

   Initial State: -

   Input: $max\ Z = 2x_1 + x_2$
   $$s.\,t. \quad x_1 - x_2 \leq 10$$

$$\begin{aligned} 2x_1 \; - \; x_2 \; &\leq \; 40 \\ x_1 \; , \; x_2 \; &\geq \; 0 \end{aligned}$$

Output: "This linear program does not have an optimal solution"

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

Test Case Derivation: [reference IM1 in CA —HZ]

5.1.2 Tests for Solving Minimization Linear Programs

1. **T4: Unique Optimal Solution**

   Control: Automatic

   Initial State: -

   Input:

   Output:

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

   Test Case Derivation:

2. **T5: Multiple Optimal Solutions**

   Control: Automatic

   Initial State: -

   Input:

   How test will be performed:

   Test Case Derivation:

3. **T6: No Optimal Solution**

   Control: Automatic

   Initial State: -

   Input:

   Output:

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

   Test Case Derivation:

5.1.3 Tests for Faulty Inputs

1. **T7: 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"

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

   Test Case Derivation: [reference IM1 & IM2 in CA —HZ]

2. **T8: No Linear Constraints**

   Control: Automatic

   Initial State: -

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

   Output: "Error: No linear constraints"

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

   Test Case Derivation: [reference IM1 & IM2 in CA —HZ]

3. **T9: 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"

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

   Test Case Derivation: [reference IM1 & IM2 in CA —HZ]

4. **T10: No Non-negativity Constraints/Negative Decision Variables**

   Control: Automatic

   Initial State: -

   Input: $max\ Z = 2x_1 - 3x_2$
   $s.\,t. \quad x_1 + x_2 \leq 5$
   $\quad 4x_1 - 3x_2 \leq 4$

   Output: "Error: The decision variables must be positive"

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

   Test Case Derivation: [reference T1 in CA —HZ]

5. **T11: Greater Than or Equal to Inequalities**

   Control: Automatic

   Initial State: -

   Input: $max\ Z = 2x_1 - 3x_2$
   $s.\,t. \quad x_1 + x_2 \leq 5$
   $\quad 4x_1 - 3x_2 \geq 4$
   $\quad x_1 , x_2 \geq 0$

   Output: "Error: The inequalities of the main constraints must be of type less than or equal to"

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

   Test Case Derivation: [reference A2 in CA —HZ]

5.2 Tests for Nonfunctional Requirements

5.2.1 Usability

1. **T12: 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)

5.2.2 Portability

1. **T13: Test for the Portability of LoSMS**

   Type: Static [?? —HZ]

   Initial State: -

   Input/Condition: -

   Output/Result: -

   How test will be performed: Running LoSMS on different operating systems

5.2.3 Robustness

1. **T14: Test for the Robustness of LoSMS**

   Type: Dynamic [?? —HZ]

   Initial State: -

   Input/Condition: -

   Output/Result: -

   How test will be performed: [mutation testing? —HZ]

5.3 Traceability Between Test Cases and Requirements

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

6 Static Verification Techniques

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

References

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. Overall, how easy to use do you find LoSMS? [range 1-5 —HZ]
2. Overall, how confident are you that you completed the task successfully? [range 1-5 —HZ]
3. Rate your satisfaction with LoSMS out of 10

   [add more questions —HZ]