Module Interface Specification for a Library of Simplex Method Solvers (LoSMS)

Hanane Zlitni

November 24, 2018

1 Revision History

Date	Version	Notes
December 06, 2018	1.2	Applied Olu Owojaiye's Comments Posted on GitHub
December 03, 2018	1.1	Corrected solveLP() and pivot() pseudo-code in 8.4.4 and 8.4.5
November 24, 2018	1.0	First draft

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/hananezlitni/HZ-CAS741-Project/blob/master/docs/SRS/CA.pdf.

The following are additional symbols, abbreviations or acronyms used in this document:

symbol	**description**
Z	Optimal solution(s) of the objective function
K	The points where the optimal solution(s) occur
Z'	The negation of the objective function
n	A number in $[0, \mathbb{N})$ representing the rows in the simplex tableau
m	A number in $[0, \mathbb{N})$ representing the columns in the simplex tableau
x	A number in $\mathbb{N}$ representing the size of the list of optimal solutions

Contents

3 Introduction

The following document details the Module Interface Specifications for the Library of Simplex Method Solvers (LoSMS) tool.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at `https://github.com/hananezlitni/HZ-CAS741-Project`.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by LoSMS.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of LoSMS uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, LoSMS uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	
Software Decision Module	Input Tableau Simplex Method Solver Output

Table 1: Module Hierarchy

6 MIS of the Input Module

6.1 Module

input

6.2 Uses

None

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
readInputs	$\mathbb{R}^m$, $\mathbb{R}^{n*m}$, lctEnum^{n-1}, gEnum	-	MISSING_INPUT
validateInputs	-	-	INVALID_INPUT
getObjcFunc	-	$\mathbb{R}^m$	-
getLCs	-	$\mathbb{R}^{n*m}$	-
getLCsType	-	lctEnum^{n-1}	-
getGoal	-	gEnum	-
setObjcFunc	$\mathbb{R}^m$	-	-
setLCs	$\mathbb{R}^{n*m}$	-	-
setLCsType	lctEnum^{n-1}	-	-
setGoal	gEnum	-	-

[Explanation: —HZ]

[1. *lctEnum*: an enumerated type that can be 0 ($\leq$), 1 ($=$) or 2 ($\geq$) depending on the type of each linear constraint. *lctEnum*$^{n-1}$ is an array in which the first element is the type of the first linear constraint, the second element is the second linear constraint type and so on. Its length is the number of rows of the tableau minus the first row (because we want to exclude the objective function). —HZ]

[2. *gEnum*: an enumerated type representing the LP goal: 0 (min) or 1 (max). —HZ]

[This seems like important information to put in comments. Comments are not displayed with the final documentation, so important information needs to be included with the main text. —SS]

[For these types you are defining, *lctEnum* and *gEnum*, please do not use integers that map to something meaningful. Mathematically you can define a set for elements of this type. A language like Python gives you enumerated types, so you can implement your new types easily. Hoffmann and Strooper show simple examples of defining new types. You will have something like $gEnum = \{\text{Min}, \text{Max}\}$. You can export these types, so that other modules can use them. If there are many new types like this, you can create a new modules whose purpose is to export the "global" types that you will be using. —SS]

[My comments about these explanation comments and types applies throughout your document. —SS]

6.4 Semantics

6.4.1 State Variables

- objcFunc:$\mathbb{R}^m$
- LCs:$\mathbb{R}^{n*m}$
- LCsType:lctEnum^{n-1} ; where lctEnum $= \{0, 1, 2\}$
- goal:gEnum ; where gEnum $= \{0, 1\}$

6.4.2 Environment Variables

None

6.4.3 Assumptions

None

6.4.4 Access Routine Semantics

getObjcFunc():

- transition: -
- output: $out :=$ objcFunc
- exception: -

getLCs():

- transition: -
- output: $out :=$ LCs
- exception: -

getLCsType():

- transition: -
- output: out := LCsType
- exception: -

getGoal():

- transition: -
- output: out := goal
- exception: -

setObjcFunc($newObjcFunc$):

- transition:
  objcFunc := newObjcFunc
- output: -
- exception: -

setLCs($newLCs$):

- transition:
  LCs := newLCs
- output: -
- exception: -

setLCsType($newLCsType$):

- transition:
  LCsType := newLCsType
- output: -
- exception: -

setGoal($newGoal$):

- transition:
  goal := newGoal
- output: -

- exception: -

readInputs(*obFunc*, *lnrConstr*, *lnrConstrT*, *goal*):

- transition: -
- output: -
- exception: $exc :=$

  At least one input missing $\Rightarrow$ MISSING_INPUT

validateInputs():

- transition: -
- output: -
- exception: $exc :=$

  $\neg$(Last element of LCsType == 2 AND Last element of LCs == 0) $\Rightarrow$ INVALID_INPUT
  [This is math, not a programming language, so tests for equality just use = —SS]

  [Explanation: —HZ]

  [I'm checking if the non-negativity constraint is present in the LCs matrix. If it isn't, an exception will be generated. —HZ]

6.4.5 Local Functions

None

7 MIS of the Tableau Module

7.1 Template Module

tableauADT

7.2 Uses

input

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
TableauT	$\mathbb{R}^m, \mathbb{R}^{n*m}$	TableauT	-
toCanonical	-	-	-
getTableau	-	TableauT	-
getLCsType	-	lctEnum^{n-1}	-
getGoal	-	gEnum	-
updateTableau	operEnum	-	-
setGoal	gEnum	-	-
setLCsType	lctEnum^{n-1}	-	-
setWasMin	boolean	-	-

[Explanation: —HZ]

[*TableauT* is an abstract data type that represents a matrix in which the first row is the coefficients of the objective function and the rest of the rows are the coefficients of the linear constraints. The constructor will receive the coefficient array of the objective function and the coefficient matrix of the LCs and will form the simplex tableau. —HZ]

[*operEnum* is an enumerated type that represents the type of operation that will be performed on the tableau to update its values. —HZ]

7.4 Semantics

7.4.1 State Variables

- sTableau:TableauT

- LCsType:lctEnum^{n-1} ; where lctEnum $= \{0, 1, 2\}$
- goal:gEnum ; where gEnum $= \{0, 1\}$
- wasMin:boolean

[You shouldn't be defining types here. Seeing that these types are coming up again, I suggest you add a module to your design that exports types. —SS]

[Why do you have two modules with the same state variables (goal etc.). This makes it seem like the design is not completely thought out. Why can't your Tableau module use your input module to get the values it needs? Or maybe you don't need an input module. As we discussed in class, most of your design could be encapsulated in the tableau module. —SS]

[Explanation: —HZ]

[*wasMin* is a way to tell whether the original LP was a min problem or not. If it's a min problem, *goal* state variable will be changed to max but information about what it was originally would be lost. Therefore, *wasMin* will be set to "True" if the original LP is a min problem and will be checked after the optimal solution is calculated in the solver module. If *wasMin* is true, Z will be multiplied by -1 because that's the optimum of a min problem. —HZ]

7.4.2 Environment Variables

None

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

new TableauT(*objcFunc, lnrConstr*):

- transition: -
- output: *out* := self

  The output consists of a matrix in which the first row is the coefficients of the objective function and the rest of the rows are the coefficients of the linear constraints.
- exception: -

toCanonical():

- transition:

1. ¬(self.getGoal() == 1) ⇒ self.setWasMin(True)
   ⇒ self.setGoal(1)
   ⇒ self.updateTableau(0)
2. (self.getLCsType() contains 0) ⇒ self.setLCsType([1,1,1,...,1])
   ⇒ self.updateTableau(1)

- output: -
- exception: -

getTableau():

- transition: -
- output: *out* := sTableau
- exception: -

getLCsType():

- transition: -
- output: *out* := LCsType
- exception: -

getGoal():

- transition: -
- output: *out* := goal
- exception: -

updateTableau(*operation*):

- transition: *operation* is of type operEnum, where operEnum = {0,1} for negating the objective function row in sTableau and adding slack variables to sTableau, respectively.

  1. (operation == 0) ⇒ multiply the first row of sTableau by -1
  2. (operation == 1) ⇒ add slack variables coefficients to sTableau

  [Use a conditional rule to express this —SS]

- output: -
- exception: -

setGoal(*newGoal*):

- transition:
  self.goal := newGoal
- output: -
- exception: -

setLCsType(*newLCsType*):

- transition:
  self.LCsType := newLCsType
- output: -
- exception: -

setWasMin(*boolValue*):

- transition:
  self.wasMin := boolValue
- output: -
- exception: -

7.4.5 Local Functions

None.

8 MIS of the Simplex Method Solver

8.1 Module

simplexSolver

8.2 Uses

tableauADT

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
solveLP	TableauT, boolean	-	NO_OPTIMAL_SOLUTION
getZ	-	R^x	-
getK	-	R^m	-
setZ	R^x	-	-
setK	R^m	-	-

8.4 Semantics

8.4.1 State Variables

- $Z : \mathbb{R}^x$
- $K : \mathbb{R}^m$

[Add a brief statement on what each state variables is. I remember what Z is, but I forget what K means. —SS] [Using the types $\mathbb{R}^x$ is confusing, since x usually represents a real value. —SS] [Would it be easier if you added a solver method to your tableau module? —SS]

8.4.2 Environment Variables

None

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

solveLP(*tableau*, *wasMin*):

- transition:
  1. findPivot(*tableau*)
  2. pivot(*tableau*, *pivotRow*, *pivotColumn*)
  3. Repeat 1 and 2 until there are no negative values in the last row (excluding the last column)
  4. setZ(*optimalSolution*)
  5. setK(*points*)
- output: -
- exception: $exc :=$
  $(Z == \text{NULL}) \Rightarrow$ NO_OPTIMAL_SOLUTION

getZ():

- transition: -
- output: $out := Z$
- exception: -

getK():

- transition: -
- output: $out := K$
- exception: -

setZ(*newZ*):

- transition:
  $Z :=$ newZ
- output: -
- exception: -

setK(*newK*):

- transition:
  $K :=$ newK
- output: -
- exception: -

8.4.5 Local Functions

```
findPivot(tableau):
    start
        for each entry in tableau except for the last column
            min(negative entry)
            if entry is found:
                pivotColumn = j
                #j is the column where the most negative entry was found
        for each element in column pivotColumn and constant in the last column
            min(element / constant)
            pivotRow = i
            #i is the row where the most minimum ratio was found
        return pivotRow, pivotColumn
    end

pivot(tableau, pivotRow, pivotColumn):
    start
        for each row in pivotColumn
            perform a row operation to make entry 0
        return tableau
    end
```

9 MIS of the Output Module

9.1 Module

output

9.2 Uses

simplexSolver

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
output	-	-	-

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

screen: The device screen of the driver program's user

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

output():

- transition: Display to the environment variable the optimal solution(s) and the points where they occur by calling *simplexSolver.getZ()* and *simplexSolver.getK()*.
- output: -
- exception: -

9.4.5 Local Functions

None.

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

10 Appendix

There are no additional information to provide.