Module Interface Specification for Optimization Library

Fasil Cheema

March 18, 2024

1 Revision History

Date	Version	Notes
March 17, 2024	1.0	Initial Upload

2 Symbols, Abbreviations and Acronyms

See SRS Documentation which can be found at https://github.com/FasilCheema/OptimizationLibrary/blob/main/docs/SRS/SRS.pdf

symbol	description
$\mathbf{A}$	$\mathbf{A}$ is the $n \times n$ matrix that is the leading term in the function (which can be expressed in the
b	b is the $n \times 1$ column vector that is the second term in the function (which is in the quadratic
c	c is the constant (real number) that is the third term in the function (which can be in the qua
f	f is the function of interest which we are trying to minimize.
n	n is a natural number and is the dimensionality of our input variable x, and hence the dimens
x	x is the $n \times 1$ column vector that is the input variable that goes into our function.
x^*	x^* is the $n \times 1$ column vector that is optimal (value of x that when input into our function re
x_0	x^* is the $n \times 1$ column vector that is the initial (value of x that is input into our minimizer).

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for OptLib, a library of optimization solvers.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/FasilCheema/OptimizationLibrary.

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 OptLib.

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 OptLib 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, ProgName 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	
Behaviour-Hiding	Output Verification
	Vector Math Module
	Control Module
	Step-Size Calculator
	Input Parameter Format
	Specification Parameters Module
Software Decision	Optimization Solver

Table 1: Module Hierarchy

6 MIS of Control Module

6.1 Module

Main

6.2 Uses

- Output Verification
- Vector Math Module
- Specification Parameters Module
- Hardware Hiding Module
- Optimization Solver
- Hardware hiding module

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
Main	-	-	-

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

6.4.4 Access Routine Semantics

main():

- transition: the control module oversees the order of execution of the different modules:
  - Select the function to use to minimize via the Optimization Solver Module
  - The Input Parameter Module then verifies if the parameters are in the appropriate format.
  - If no exceptions occur, the Step Size Calculator Module computes the appropriate step size.
  - The Optimization Solver Module continues with the specific computation given the step size, the Vector Math Module will be heavily employed.
- output: None (returns to the program that called the function)
- exception: Various exceptions can occur in sub modules

6.4.5 Local Functions

None

7 MIS of Vector Math Module

7.1 Module

vector

7.2 Uses

- Hardware Hiding Module

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
vectorMath	vectors/matrices of dimensions $\mathbb{R}^{n\times m}$	vector/matrix of type $\mathbb{R}^{n\times m}$	Dimension Error Type Error

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

None

7.4.3 Assumptions

All input vectors/matrices are of the correct type and are verified by the control module.

7.4.4 Access Routine Semantics

vectorMath(*funcname,params*):

- transition: The vector math is calculated as follows:

  - Takes a matrix/matrices as input and a specified operation

  - Conducts said operation on given matrix/matrices

- output: outputs a matrix/vector in an appropriate format.
  out := matrix **x**

- exception: exc:=

  - Dimension Error: for operations such as matrix multiplication and addition ensuring the input matrices are of appropriate dimensions.

  - Type Error: Ensure the input to the function is in the appropriate format *funcname, params*

7.4.5 Local Functions

None

8 MIS of Template

8.1 Module

Main

8.2 Uses

- Output Verification
- Vector Math Module
- Specification Parameters Module
- Hardware Hiding Module
- Optimization Solver
- Hardware hiding module

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
Main	-	-	-

8.4 Semantics

8.4.1 State Variables

param := sequence of (
$s : \mathbb{R}$, step size

[Not all modules will have state variables. State variables give the module a memory. —SS]

8.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

8.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

8.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

8.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

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.

9 Appendix

[Extra information if required —SS]

10 Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design. Please answer the following questions:

1. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

2. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)