Module Interface Specification for ROC

John M Ernsthausen

December 5, 2020

1 Revision History

Date	Version	Notes
22 November 2020	1.0	First submission

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at
https://github.com/JohnErnsthausen/roc/blob/master/docs/SRS/SRS.pdf.

Contents

3 Introduction

The following document details the Module Interface Specifications for the implemented modules in library software ROC estimating the radius of convergence of a power series. It is intended to ease navigation through the program for design and maintenance purposes.

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

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 ROC.

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 ROC 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, ROC 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	Input Format Module Output Format Module
Behaviour-Hiding Module	Input Format Module Output Format Module Parameters Module Real pole module Complex conjugate pair of poles module Resolve nearest pole in hard to resolve case module Pole identification module
Software Decision Module	Solver Module

Table 1: Module Hierarchy

6 MIS of Input Format Module

The secrets of this module are input acquisition via hardware (R1), input acquisition via software (R2), validate input format (R3), validate input type (R4), inputs should satisfy the assumptions (R5), and inputs should be scaled to prevent overflow/underflow (R6). This module handles the data structure for input coefficients and the scaling of the input coefficients, how the values are input (software/hardware), and how the values are verified (OS). The load and verify secrets are isolated to their own methods.

6.1 Module

IN

6.2 Uses

None

6.3 Syntax

Name	In	Out	Exceptions
load_coeffs	string	vector<double>	FileError
verify_coeffs	-	-	FileError

6.4 Semantics

6.4.1 Environment Variables

inputFile: sequence of string *#f[i] is the ith string in the comma separated value text file f*

6.4.2 Assumptions

The coefficients must be ordered according to the convention from lowest order to highest order. The number of coefficients follows. Finally the scaling follows.

6.4.3 Access Routine Semantics

load_coeffs(s):

- transition: The filename s is first associated with the file f. inputFile is used to modify a coefficients vector of type double, size variable of type integer, and a scaling variable of type double. This file can be processed by ROC as each line is read. In this way, the option of input by software and input by hardware are handled identically.
- verify_coeffs() is handled by the OS plus a validation that the coefficients vector has the expected size.
- exception: exc := a file name s cannot be found OR the format of inputFile is incorrect $\Rightarrow$ FileError

7 MIS of Output Format Module

The secrets of this module are output via hardware (R7), output via software (R8), validate input format (R9), validate input type (R10), This module handles the data structure for output coefficients, how the values are output (software/hardware), and how the values are verified (OS).

7.1 Module

OUT

7.2 Uses

None

7.3 Syntax

Name	In	Out	Exceptions
write_out	double	hardware/software	FileError

7.4 Semantics

7.4.1 Environment Variables

outputFile: sequence of string *#f[i] is the ith string in the comma separated value text file f*

7.4.2 Assumptions

The radius of convergence R_c and order of singularity μ must be ordered according to a convention.

7.4.3 Access Routine Semantics

write_out(s):

- transition: The filename s is first associated with the file outputFile. This file can be appended by ROC as each line is read. In this way, the option of output by software and output by hardware are handled identically.
- verify_out() is handled by the OS.
- exception: exc := a file name s cannot be found ⇒ FileError

8 MIS of Parameters Module

The secrets of this module are parameter acquisition, format, type, distribution, and constraints. The author is thinking of constants in a header file.

8.1 Module

PARAM

8.2 Uses

None

Using $ $ for italics looks odd (it is formatted to mean T·H·R·E···)
Inside $ $ use \mathit, or \text, or \mbox

You should define the constants here

8.2.1 Exported Constants

$THREETERM\_NUSE$ of type integer. $THREETERM\_NUSE$ must be less than the size of the coefficient vector minus 10.
$SIXTERM\_NUSE$ of type integer. $SIXTERM\_NUSE$ must be less than the size of the coefficient vector minus 10.

9 MIS of Solver Module

Algorithm to find the distance to the nearest real pole (R16), nearest complex conjugate pair of poles (R17), and nearest pole in hard to resolve case (R18). This solver is required to solve the linear least squares problem.

You don't have semantics for any of your modules

9.1 Module

SOLVER

9.2 Uses

The RealPoleModule discussed in Section 10, ComplexConjugatePoleModule discussed in Section 11, and the hard to resolve case module discussed in Section 12 set up their respective problem for this module to resolve.

9.3 Syntax

The syntax depends on the pole to resolve and the algorithm used, the linear least squares solver is the QRFactorization in all cases.

10 MIS of Real Pole Module

Find the distance to the nearest real pole (R16).

10.1 Module

REALPOLE

10.2 Uses

IN Section 6, OUT Section 7, PARAM Section 8, SOLVER Section 9

Uses should be used by the specification - I don't see anything for these modules involved.

10.3 Syntax

Name	In	Out	Exceptions
REALPOLE	vector<double>, size, scale	R_c, μ, error	SOLVER_EXCEPTION

11 MIS of Complex Poles Module

Find the distance to the nearest complex conjugate pair of poles (R17).

11.1 Module

COMPLEXPOLES

11.2 Uses

IN Section 6, OUT Section 7, PARAM Section 8, SOLVER Section 9

11.3 Syntax

Name	In	Out	Exceptions
COMPLEXPOLES	vector<double>, size, scale	R_c, μ, error	SOLVER_EXCEPTION

12 MIS of Top-Line Module

Find the distance to the nearest complicated pole (R18).

12.1 Module

TOPLINEANALYSIS

12.2 Uses

IN Section 6, OUT Section 7, PARAM Section 8, SOLVER Section 9

12.3 Syntax

Name	In	Out	Exceptions
TOPLINEANALYSIS	vector<double>, size, scale	R_c, error	SOLVER_EXCEPTION

13 MIS of ROC Module

Find the Radius of convergence R_c and order of singularity μ.

13.1 Module

ROC

13.2 Uses

REALPOLE Section 10, COMPLEXPOLES Section 11, TOPLINEANALYSIS Section 12

13.3 Syntax

This is a type — these are not types

Name	In	Out	Exceptions
ROC	vector<double>, size, scale	R_c, μ, error	ROC_EXCEPTION

You should list the input & output types

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.

Your MIS isn't complete without the semantics. The MIS should be an abstract view of the code. In your case my recommendation is to start with your code and abstract out the details. For instance, rather than vector<double> say seq of R. When is the module of factorization? This module is nothing!

You don't want any of the pointer related Syntax or C++ Syntax in your MIS - just make it a mathematical document.