Module Interface Specification for FSL

Bo Cao

November 24, 2019

1 Revision History

Date	Version	Notes
Nov. 26, 2019	0.99	First Draft

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/caobo1994/FourierSeries/blob/master/docs/SRS/SRS.pdf. We also define the following acronyms for the scope of this document

Acronym	Full Text
OOR	Out of range

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**

3 Introduction **1**

4 Notation **1**

5 Module Decomposition **1**

6 MIS of Infrastructure **3**
- 6.1 Module . . . 3
- 6.2 Uses . . . 3
- 6.3 Syntax . . . 3
- 6.4 Exported Data Type . . . 3

7 MIS of Data Definition **3**
- 7.1 Module . . . 3
- 7.2 Uses . . . 3
- 7.3 Syntax . . . 3
  - 7.3.1 Exported Constants . . . 3
  - 7.3.2 Exported Data Type . . . 3
  - 7.3.3 Exported Access Programs . . . 4
- 7.4 Semantics . . . 4
  - 7.4.1 Assumptions . . . 4
  - 7.4.2 Access Routine Semantics . . . 4

8 MIS of Linear Solver **4**
- 8.1 Module . . . 4
- 8.2 Uses . . . 5
- 8.3 Syntax . . . 5
  - 8.3.1 Export Access Programs . . . 5
- 8.4 Semantics . . . 5
  - 8.4.1 Assumptions . . . 5
  - 8.4.2 Access Routine Semantics . . . 5

9 MIS of Integral **5**
- 9.1 Module . . . 5
- 9.2 Uses . . . 5
- 9.3 Syntax . . . 5
  - 9.3.1 Export Access Programs . . . 5
- 9.4 Semantics . . . 6

9.4.1 Access Routine Semantics 6

10 MIS of Conversion **6**
10.1 Module 6
10.2 Uses 6
10.3 Syntax 6
10.3.1 Export Access Programs 6
10.4 Semantics 6
10.5 Access Routine Semantics 6

11 MIS for Transformation **7**
11.1 Module 7
11.2 Uses 7
11.3 Syntax 7
11.3.1 Export Access Programs 7
11.4 Semantics 7
11.5 Access Program Semantics 7

12 MIS of Basic Operation **7**
12.1 Module 7
12.2 Uses 8
12.3 Syntax 8
12.3.1 Export Access Programs 8
12.4 Semantics 8
12.4.1 Access Program Semantics 8

13 MIS for Advanced Operation **9**
13.1 Module 9
13.2 Uses 9
13.3 Syntax 9
13.3.1 Export Access Programs 9
13.4 Semantics 9
13.4.1 Access Routine Semantics 9

14 Appendix **12**

3 Introduction

The following document details the Module Interface Specifications for Fourier Series Library (FSL).

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/caobo1994/FourierSeries/.

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 FSL.

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 FSL 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, FSL 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. For simplicity, the sequence of type T will be abbreviated as seq(T), while that with dimensions $[l_1, ..., l_n]$ as seq(T, $l_1, ..., l_n$).

5 Module Decomposition

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

Level 1	**Level 2**
Hardware-Hiding Module	Infrastructure (hardware-related part, external)
Behaviour-Hiding Module	Data definition module Conversion module Transformation module Basic operations module Advanced operations module
Software Decision Module	Linear solver (external or partially external) Integral (external or partially external) Infrastructure (software-related part, external)

Table 1: Module Hierarchy

6 MIS of Infrastructure

6.1 Module

Infrastructure.

6.2 Uses

None.

6.3 Syntax

6.4 Exported Data Type

FLOAT a floating point data type used in the library.

sequence the abstract sequence data type.

7 MIS of Data Definition

7.1 Module

Data Definition.

7.2 Uses

Infrastructure.

7.3 Syntax

7.3.1 Exported Constants

None.

7.3.2 Exported Data Type

There is one exported data type, `CFST`, which is the type of a CFS object. (abstract, with one type template `FLOAT`, indicating floating point types)

The structure of `CFST` is

- n: integer
- ω: `FLOAT`
- A: seq(`FLOAT`)

- B: seq(`FLOAT`)

The mathematical notation is

$$\text{CFST} := \text{tuple of } (n : \mathbb{N}, \omega : \text{FLOAT}, A : \text{TA}, B : \text{TB}) \tag{1}$$

where TA and TB are

$$\begin{aligned} \text{TA} :=& seq(\texttt{FLOAT}, n) \\ \text{TB} :=& seq(\texttt{FLOAT}, n+1) \end{aligned} \tag{2}$$

During implementation, it is recommended that the sequence type used above are random accessible.

7.3.3 Exported Access Programs

The exported access programs for this module are mainly getters and setters. For simplicity, we use the following getter and setter rules.

For each variable X, the getters and setters are

Name	In	Out	Exceptions
`getX`		`X`: type of X	None.
`setX`	`X`: type of X		None.

Furthermore, we design getters and setters for each element of A and B. For each X being A or B, the syntax is

Name	In	Out	Exceptions
`getXi`		`X`: type of X	OOR.
`setXi`	`X`: type of X		OOR.

7.4 Semantics

7.4.1 Assumptions

7.4.2 Access Routine Semantics

This part follows the most common rules of getters and setters. For simplicity, we do not elaborate on them.

8 MIS of Linear Solver

8.1 Module

Linear Solver.

8.2 Uses

Infrastructure.

8.3 Syntax

8.3.1 Export Access Programs

Name	In	Out	Exceptions
LinSolve	m: $\mathbb{N}$ A: seq(FLOAT, m, m) b: seq(FLOAT, m)	x: seq(FLOAT, m)	Solution non-exist Solution not unique

8.4 Semantics

8.4.1 Assumptions

8.4.2 Access Routine Semantics

This part follows the most common semantics of linear solvers, and for simplicity, we do not elaborate on it.

9 MIS of Integral

9.1 Module

Integral.

9.2 Uses

Infrastructure.

9.3 Syntax

9.3.1 Export Access Programs

Name	In	Out	Exceptions
Integral	f: FLOAT→FLOAT a, b: FLOAT	res: FLOAT	Integral non-exist or not computable

9.4 Semantics

9.4.1 Access Routine Semantics

`Integral(f, a, b)`:

- output: $\int_a^b f(t)\mathrm{d}t$
- exception: Evident

10 MIS of Conversion

10.1 Module

Conversion.

10.2 Uses

Data Definition.

10.3 Syntax

10.3.1 Export Access Programs

With `O` being short for `OUT_OF_BOUNDARY` and `M` being short for `MISMATCHED_SIZE`,

Name	In	Out	Exceptions
`ConvertFrom`	$n \in \mathbb{N}$ ω: `FLOAT` A: sequence of `FLOAT` B: sequence of `FLOAT`	`CFST CFS`	`O`: $\omega \leq 0$ `M`: Mismatch between n, (size of A -1), and size of B
`ConvertTo`	`CFST CFS` user-acquired space for outputs	Same as inputs of `ConvertFrom`	None

10.4 Semantics

10.5 Access Routine Semantics

ConvertFrom(n, ω, A, B):

- output: CFS.n, CFS.ω, CFS.A, CFS.$B := n, \omega, A, B$

- exception: exc := ($\omega \leq 0 \Rightarrow O||A| \neq n+1 \Rightarrow M||B| \neq n \Rightarrow M$)

The semantics for `ConvertTo` is straightforward, and we do not elaborate on it.

11 MIS for Transformation

11.1 Module

Transformation.

11.2 Uses

Data Definition, Integration.

11.3 Syntax

11.3.1 Export Access Programs

Name	In	Out	Exceptions
`TransformTo`	$f \in \{\mathbb{R} \to \mathbb{R}\}$ $n \in \mathbb{N}$ $\omega \in \mathbb{R}^+$	`CFST CFS`	None.
`FunctionValue`	`CFST CFS` $t \in \mathbb{R}$	$V \in \mathbb{R}$	None.

11.4 Semantics

11.5 Access Program Semantics

`TransformTo`(f, n, ω):

- output: CFS.n, CFS.ω, CFS.A, CFS.$B := n, \omega, A, B$, where A_i and B_i are computed in accordance with T1 of the SRS.

`FunctionValue`(CFS, t):

- output: $V := \sum_{i=0}^{\text{CFS}.n} \text{CFS}.A(i)\cos(i\omega t) + \sum_{i=1}^{\text{CFS}.n} \text{CFS}.B(i)\sin(i\omega t)$

12 MIS of Basic Operation

12.1 Module

Basic Operations.

12.2 Uses

Data Definition, Linear Solver

12.3 Syntax

12.3.1 Export Access Programs

Name	In	Out	Exceptions
`CFSMatch`	`CFST CFS1, CFST CFS2`	`Bool res`	None.
`Addition`	`CFST CFS1, CFST CFS2`	`CFST CFSres`	mismatch of n and ω
`Subtraction`	`CFST CFS1, CFST CFS2`	`CFST CFSres`	Same as `Addition`
`Multplication`	`CFST CFS1, CFST CFS2`	`CFST CFSres`	Same as `Addition`
`Divison`	`CFST CFS1, CFST CFS2`	`CFST CFSres`	Same as `Addition`, plus any exception raised by linear solver and reclassified in `Division`.
`Amplitude`	`CFST CFS1`	`FLOAT amp`	None.

12.4 Semantics

12.4.1 Access Program Semantics

`CFSMatch(CFST CFS1, CFST CFS2)`:

- output: $(\text{CFS1}.n = \text{CFS2}.n) \wedge (\text{CFS1}.\omega = \text{CFS2}.\omega) \Rightarrow \text{TRUE}|\text{TRUE} \Rightarrow \text{FALSE}$

The semantics of `Addition`, `Subtraction`, and `Multiplication` are similar in structure, and the only difference is the calculation of the A and B variables shown below, which is consistent with the corresponding theories introduced in SRS.

As an example, we give the semantics of the `Addition` function.

`Addition(CFST CFS1, CFST CFS2)`:

- output: $A_i := \text{CFS1.getAi}(i) + \text{CFS2.getAi}(i)$, $B_i := \text{CFS1.getBi}(i) + \text{CFS2.getBi}(i)$, $A := [A_0, ..., A_{\text{CFS1.n}}]$, $B := [B_0, ..., B_{\text{CFS1.n}}]$, $\text{CFSres}.n, \text{CFSres}.\omega, \text{CFSres}.A, \text{CFSres}.B := \text{CFS1}.n, \text{CFS1}.\omega, A, B$
- exception: $\text{exc} := (\text{CFSMatch}(\text{CFS1}, \text{CFS2}) = \text{FALSE} \Rightarrow \text{MISMATCH\_CFS})$

As for `Division`, the difference is much significant. The A and B is computed as follows

$$\begin{aligned} x :=& \text{LinSolve}(M, y), \\ A :=& x[0, \text{CFS1}.n], \\ B :=& x[\text{CFS1}.n + 1, |x| - 1] \end{aligned}$$

where M and y are constructed in accordance with the theories for division in SRS.
`Amplitude(CFS CFS1)`:

- output: $amp := \sqrt{\text{CFS1.getAi}(0)^2 + (1/2) * \sum_{i=1}^{\text{CFS1}.n}(\text{CFS1.getAi}(i)^2 + \text{CFS1.getBi}(i)^2)}$

13 MIS for Advanced Operation

13.1 Module

Advanced Operation.

13.2 Uses

Basic Operation.

13.3 Syntax

In the following section, `TST` is a function type $\mathbb{Z}^* \to$ FLOAT, and for any object `TS` of this type associated with a mathematical function, `TS(i)` gives the `i`-th Taylor coefficient of this mathematical function.

13.3.1 Export Access Programs

Name	In	Out	Exceptions
`ToleratedEquality`	`CFST CFS1, CFST CFS2, FLOAT tol`	`Bool res`	Same as `Addition`
`Power`	`CFST CFS`, $m \in Z^*$	`CFS CFSres`	None
`Function`	`CFST CFS, TST TS`	`CFST, CFSres`	None.

13.4 Semantics

13.4.1 Access Routine Semantics

`ToleratedEquality(CFST CFS1, CFST CFS2, FLOAT tol)`:

- output: $res :=$ (Amplitude(Subtraction($CFS1, CFS2$)) $\leq tol \Rightarrow$ TRUE|TRUE $\Rightarrow$ FALSE)

- exception: exc := (CFSMatch(CFS1, CFS2) = FALSE $\Rightarrow$ MISMATCH_CFS)

`Power(CFST CFS,` $m \in Z^*$`):`

- output: $A :=< 1, 0, ..., 0 >$, $|A| = (n + 1)$; $B :=< 0, 0, ..., 0]$, $|B| = n$
  CFSzero.n, CFSzero.ω, CFSzero.A, CFSzero.B $= n, \omega, A, B$ CFSres := ($m = 0 \Rightarrow$ CFSzero|TRUE $\Rightarrow$ Multiplication(CFS, Power(CFS, $m - 1$)))

`Function(CFST CFS, TST TS):`

- output: CFSres := $\sum_{i=0}^{\text{CFS}.n}(1/i!)\text{TS}(i)\text{Power}(\text{CFS}, i)$

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.

14 Appendix

[Extra information if required —SS]