Module Interface Specification for CFS

Malavika Srinivasan

November 29, 2018

1 Revision History

Date	Version	Notes
Nov 10, 2018	1.0	First draft by Malavika
Nov 21, 2018	1.1	Corrections based on presentation by Malavika
Nov 24, 2018	1.2	MIS submission draft by Malavika

2 Symbols, Abbreviations and Acronyms

See CA Documentation at https://github.com/Malavika-Srinivasan/CAS741/blob/master/docs/SRS/CA.pdf

Contents

3 Introduction

The following document details the Module Interface Specifications for CFS (Curve Fitting Software). The goal of CFS is to find the parameters of the curve which is the best possible fit through the given set of 'n' data points, where $n >= 2$.

Complementary documents include the Commonality Analysis and Module Guide. The full documentation and implementation can be found at the repository `https://github.com/Malavika-Srinivasan/CAS741`.

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 CFS.

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 (-∞, ∞)
boolean	$\mathbb{B}$	a value from the set $\{True, False\}$

The specification of CFS 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, CFS 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

[Needs to be changed in MG based on Dr.Smith's suggestion —Malavika] The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Output
Software Decision Module	Sequence Services Interpolation Regression Plot

Table 1: Module Hierarchy

6 MIS of Input module

This module corresponds to R??, R?? and R?? in the CA document. The secrets of this module are how the data points are input and how they are verified. The load and verify secrets are isolated to their own access programs.

6.1 Module

input

6.2 Uses

Sequence services(10)

6.3 Syntax

Name	In	Out	Exceptions
loadInput	inFile : string	-	FileError
verifyInput	-	-	ValueError,TypeError
verifyDegree	degree:$\mathbb{N}$	$degVerify$:$\mathbb{B}$	ValueError

6.3.1 Exported Constants

NA

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
verifyDegree	degree:$\mathbb{N}$	$degVerify$:$\mathbb{B}$	ValueError

[I need the verifyDegree as an exported access program because I will be verifying the degree input from regression module if necessary. —Malavika]

6.4 Semantics

6.4.1 State Variables

From R?? in CA
t: $\mathbb{R}^n$
y: $\mathbb{R}^n$

6.4.2 Environment Variables

inFile: sequence of string [I am leaving it as sequence of strings because sometimes, you may want to append a file name to a path name. —Malavika] [Also, I am thinking if I really need a file input. I think this should be the user program's responsibility. My access programs do not have an interface to accept filename. They will accept only x and y arrays. This needs to be discussed. —Malavika]

6.4.3 Assumptions

- The loadInput will be called before calling the verifyInput method.
- Regression module will be called before calling verifyDegree method.
- The loadInput will be called before the values of any state variables will be accessed.

6.4.4 Access Routine Semantics

input.t:

- transition: NA
- output: $out := t$
- exception: NA

input.y:

- transition: NA
- output: $out := y$
- exception: NA

input.$degVerify$:

- transition: NA
- output: $out := degVerify$
- exception: NA

loadInput(s):

- transition: inFile is used to modify the state variables using the following procedural specification:
    1. Read data sequentially from inputFile to populate the state variables t and y.
    2. verifyInput()

- output: NA
- exception: exc := a file name s cannot be found $\vee$ the format of inFile is incorrect $\Rightarrow$ FileError.[I am yet to decide if the library will take a .txt and .csv or both. This specification will become formal once we decide the type of the input file. —Malavika]

verifyInput():

- transition:NA
- output: NA
- exception: exc :=

Name	Exception
$(\|t\| \leq 1 \vee \|y\| \leq 1)$	$\Rightarrow$ LengthError
$(\|t\| \neq \|y\|)$	$\Rightarrow$ LengthMismatchError
$(\forall x \in t \wedge x \notin \mathbb{R})$	$\Rightarrow$ TypeError
$(\forall x \in y \wedge x \notin \mathbb{R})$	$\Rightarrow$ TypeError

verifyDegree():

- transition:
  - $(deg \in \mathbb{N}) \Longrightarrow$degVerify$= True$
- output:
- exception: exc :=

Name	Exception
$(deg \notin \mathbb{N})$	$\Rightarrow$ ValueError

6.4.5 Local Functions

NA

7 MIS of Interpolation module

This module corresponds to R??, R??, R??, R?? and R?? in section ?? of CA document.

7.1 Module

interp

7.2 Uses

Input(6), Output(9), Sequence services(10)

7.3 Syntax

7.3.1 Exported Constants

NA

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
interpMonomial	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	-	-
interpLagrange	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	-	-
interpNewton	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	-	-
interpHermiteCubic	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	-	-
interpBSpline	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	-	-
evalMonomial	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	yNew:$\mathbb{R}^n$	-
evalLagrange	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	yNew:$\mathbb{R}^n$	-
evalNewton	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	yNew:$\mathbb{R}^n$	-
evalHermiteCubic	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	yNew:$\mathbb{R}^n$	-
evalBSpline	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	yNew:$\mathbb{R}^n$	-

7.4 Semantics

7.4.1 State Variables

- x:$\mathbb{R}^{mXn}$, where n is the number of data points and m is the number of sections.

- interval:$\mathbb{R}^m$, m is the number of sections

7.4.2 Environment Variables

NA

7.4.3 Assumptions

- evalMonomial() will be called after interpMonomial().
- evalLagrange() will be called after interpLagrange().
- evalNewton() will be called after interpNewton().
- evalHermiteCubic() will be called after interpHermiteCubic.
- evalBSpline() will be called after interpBSpline().

7.4.4 Access Routine Semantics

interp.x:

- transition: NA
- output: $out := x$
- exception: NA

interpMonomial(t,y):

- transition:
  - 1. x: $\mathbb{R}^{mXn} = 0.0$, this is initialization. This is necessary as all the methods will be using the same state variable.
    2. x: $\mathbb{R}^{mXn}$ obtained by solving for x using the equation below.

$$\begin{bmatrix} 1 & t_1 & t_1^2 & \dots & t_1^{n-1} \\ 1 & t_2 & t_2^2 & \dots & t_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & t_n & t_n^2 & \dots & t_n^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

  - $interval[0] := t[0]$, this is initialized to starting point for non piecewise functions.
- output: NA
- exception: NA

evalMonomial(t,y):

- transition: NA
- output: $out := yNew$
  $\forall t_i \in t$, yNew.append(y1) where y1 is obtained as shown below.

$$y1 = x_1 + x_2 t_i + x_3 t_i^2 + x_4 t_i^3 + ... x_n t_i^{n-1}$$

- exception: NA

interpLagrange(t,y):

- transition:
  - 1. x: $\mathbb{R}^{mXn} = 0.0$, this is initialization. This is necessary as all the methods will be using the same state variable.
    2. x: $\mathbb{R}^{mXn}$ obtained by solving for x using the equation below.

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

  - $interval[0] := t[0]$, this is initialized to starting point for non piecewise functions.
- output: NA
- exception: NA

evalLagrange(t,y):

- transition: NA
- output: $out := yNew$
  $\forall t_i \in t$, yNew.append(y_0) where y_0 is obtained as shown below.

$$y_0 = y_1 l_1(t) + y_2 l_2(t) + \dots y_n l_n(t).$$

  where $l_j(t)$ is given by,

$$l_j(t) = \frac{\Pi_{k=1,k\neq j}^n (t - t_k)}{\Pi_{k=1,k\neq j}^n (t_j - t_k)}$$

- exception: NA

interpNewton(t,y):

- transition:
  - 1. x: $\mathbb{R}^{mXn} = 0.0$, this is initialization. This is necessary as all the methods will be using the same state variable.
    2. x: $\mathbb{R}^{mXn}$ obtained by solving for x using the equation below.

$$\begin{bmatrix} \pi_0(t_0) & 0 & 0 & \dots & 0 \\ \pi_0(t_1) & \pi_1(t_1) & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ \pi_0(t_n) & \pi_1(t_n) & \pi_2(t_n) & \dots & \pi_n(t_n) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

    where,
    $i < j \implies \pi_j(t) = 0$, $i \geq j \implies \pi(t) = \Pi_{k=1}^{j-1}(t - t_k)$
  - $interval[0] := t[0]$, this is initialized to starting point for non piecewise functions.
- output: NA
- exception: NA

evalNewton(t,y):

- transition: NA
- output: $out := yNew$
  $\forall t_i \in t$, yNew.append(y_0) where y_0 is obtained as shown below.

$$y_0 = x_1 + x_2(t - t_1) + x_3(t - t_1)(t - t_2) + ...x_n(t - t_1)(t - t_2)...(t - t_{n-1})$$

- exception: NA

interpHermiteCubic(t,y):

- transition:
  - 1. x: $\mathbb{R}^{mXn} = 0.0$, this is initialization. This is necessary as all the methods will be using the same state variable.

2. x: $\mathbb{R}^{mXn}$ obtained by solving for x using the equation below.

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & t_n & t_n^2 & t_n^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 2t_n & 3t_n^2 \\ 0 & 0 & 2 & 6t_0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 2 & 6t_n \\ 0 & 0 & 0 & 6 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ x_{n(m+1)} \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_0^{(1)} \\ y_1^{(1)} \\ \vdots \\ y_n^{(1)} \\ y_1^{(2)} \\ \vdots \\ y_n^{(2)} \\ y_1^{(3)} \\ \vdots \\ y_n^{(3)} \end{bmatrix}$$

– interval: $\mathbb{R}^k = t[0]$ to $t[n-2]$, by default we make each point a breakpoint and hence there is a piecewise polynomial between $[t_{i-1}, t_i)$.

- output: NA.
- exception: NA

evalHermiteCubic(t,y):

- transition:
- output: $out := yNew$

  $\forall t_i \in t$, yNew.append(y_0) where y_0 is obtained as shown below.

$$y_0 = x_{k0} + x_{k1}t + x_{k2}t^2 + x_{k3}t^3 \quad for\ k = 0\ to\ n-2$$

- exception: NA

interpBSpline(t,y):

- transition:

  – 1. x: $\mathbb{R}^{mXn} = 0.0$, this is initialization. This is necessary as all the methods will be using the same state variable.

2. x: $\mathbb{R}^{mXn}$ obtained by using the formula below.
   For $k > 0$ to n, where n is the number of data points

$$x = v_i^k = \frac{t - t_i}{t_{i+k} - t_i}$$

   [BSpline gives the v values which will be used recursively in formula defined in IM?? of the CA document —Malavika]

- interval: $\mathbb{R}^k = t[0]$ to $t[n-2]$, by default we make each point a breakpoint and hence there is a piecewise polynomial between $[t_{i-1}, t_i)$.

- output: NA
- exception: NA

evalBSpline(t,y):

- transition:
- output: $out := yNew$

  $\forall t_i \in t$, yNew.append(y_0) where y_0 is obtained as shown below.

$$y_0 = B_i^k(t) = v_i^k(t)B_i^{k-1}(t) + (1 - v_{i+1}^k(t))B_{i+1}^{k-1}(t)$$

  for $k > 0$.
  [I want the notation of 'B' so that it is easy to understand the coefficients, thats why I have $y_0 = B...$ —Malavika]

- exception: NA

7.4.5 Local Functions

NA

8 MIS of Regression module

This module corresponds to R**??**, R**??** and R**??** in section **??** of CA document.

8.1 Module

reg

8.2 Uses

Input(6), Output(9), Sequence services(10)

8.3 Syntax

8.3.1 Exported Constants

NA

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
regNormalEq	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$, deg:$\mathbb{N}$	-	-
regAugSys	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$, deg:$\mathbb{N}$	-	-
regOrthogonalTn	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$, deg:$\mathbb{N}$	-	-
evalReg	t:$\mathbb{R}^n$	yNew:$\mathbb{R}^n$	-

8.4 Semantics

8.4.1 State Variables

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

8.4.2 Environment Variables

NA

8.4.3 Assumptions

- input.verifyDegree() will be called before any of the access programs will be used.
- evalReg() will be called after calling (regNormalEq() $\vee$ regAugSys() $\vee$ regOrthogonalTn()).

8.4.4 Access Routine Semantics

reg.x:

- transition: NA
- output: $out := x$
- exception: NA

regNormalEq(t,y,deg):

- transition:

- $degVerify$ = verifyDegree(deg)
- $(degVerify == True) \implies x^n = 0.0$ [This is initialization with maximum length possible. For a data of length n, you can maximum get n coefficients in the polynomial. —Malavika]
- $x^m = \mathbb{R}^m$, obtained by solving for x in the equation below.[m is the degree of the coefficients of the polynomial. —Malavika]

$$A^T Ax = A^T y$$

- output: NA
- exception: NA

regAugSys(t,y,deg):

- transition:
  - $degVerify$ = verifyDegree(deg)
  - $(degVerify == True) \implies x^n = 0.0$ [This is initialization with maximum length possible. For a data of length n, you can maximum get n coefficients in the polynomial. —Malavika]
  - $x^m = \mathbb{R}^m$, obtained by solving for x in the equation below.[m is the degree of the coefficients of the polynomial. —Malavika]

$$r + Ax = y$$

$$A^T r = 0$$

Where r is the residual vector.

- output: NA
- exception: NA

regOrthogonalTn(t,y,deg):

- transition:
  - $degVerify$ = verifyDegree(deg)

- $(degVerify == True) \implies x^n = 0.0$ [This is initialization with maximum length possible. For a data of length n, you can maximum get n coefficients in the polynomial. —Malavika]
- $x^m = \mathbb{R}^m$, obtained by solving for x in the equation below.[m is the degree of the coefficients of the polynomial. —Malavika]

$$Ax = Py$$

Where P is the orthogonality matrix.

- output: NA
- exception: NA

evalReg(t):

- transition: NA
- output: $out := yNew$, where $\forall\ t_i \in t$,

$$yNew.append(np.poly1d(x)(t_i))$$

- exception: NA

8.4.5 Local Functions

NA

9 MIS of Output module

This module corresponds to R?? in the CA document. The secrets of this module are how the output is given to the user program.

9.1 Module

output

9.2 Uses

Plot(11), Sequence services(10)

9.3 Syntax

Name	**In**	**Out**	**Exceptions**
plotDataFit	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$		-
coeffPlotScreen	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$,x:$\mathbb{R}^m$	x:$\mathbb{R}^m$, plot	-
coeffFile	x:$\mathbb{R}^m$,t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	x:$\mathbb{R}^m$	-

9.3.1 Exported Constants

NA

9.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
plotDataFit	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$		-
coeffPlotScreen	t:$\mathbb{R}^n$,y:$\mathbb{R}^n$,x:$\mathbb{R}^m$	x:$\mathbb{R}^m$, plot	-
coeffFile	x:$\mathbb{R}^m$,t:$\mathbb{R}^n$,y:$\mathbb{R}^n$	x:$\mathbb{R}^m$	-

9.4 Semantics

9.4.1 State Variables

NA

9.4.2 Environment Variables

OutputFile: Results.csv

9.4.3 Assumptions

- The interpolation or the regression module will be called before the output module.

9.4.4 Access Routine Semantics

plotDataFit(t,y):

- transition:
  - Call the appropriate evaluate method from the interpolation or regression module as shown below..
    1. $yNew =$ interp.evalMonomial(t,y) $\vee$ interp.evalLagrange(t,y) $\vee$ interp.evalNewton(t,y) $\vee$ interp.evalHermiteCubic(t,y) $\vee$ interp.evalBSpline(t,y) $\vee$ reg.evalReg(t)

2. plot(t, y) and plot(t,$yNew$)

- output: out:= plot
- exception: NA

coeffPlotScreen(t,y):

- transition :
  - plotDataFit(t,y)
- output: Out:= $interp.x \vee reg.x$, plot
- exception: NA

coeffFile():

- transition:
  - Create a csv file named 'Result'
  - write interp.x ∨ reg.x
- output: NA
- exception: NA

9.4.5 Local Functions

NA

10 MIS of Sequence Services

10.1 Module

seqSer [Provided by numpy Python —Malavika]

10.2 Uses

NA

10.3 Syntax

10.3.1 Exported Constants

NA

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
isAscending	$arr : \mathbb{R}^n$	$\mathbb{B}$	-
unique	$arr : \mathbb{R}^n$	$uniArr : \mathbb{R}^m$	-
array_equal	$arr1 : \mathbb{R}^n$,$arr2 : \mathbb{R}^n$	$\mathbb{B}$	-

10.4 Semantics

10.4.1 State Variables

NA

10.4.2 Environment Variables

NA

10.4.3 Assumptions

NA

10.4.4 Access Routine Semantics

unique(arr):

- transition: NA
- output: $out := uniArr : \mathbb{R}^m$ where $uniArr$ is given by,
  1. $arr[i] \neq arr[j], \forall i, j \in [(0 \text{ to } (|arr| - 1)] \wedge j \neq i \implies$
     $\forall i \in [0 \text{ to } |arr| - 1], uniArr[i] \coloneqq arr[i]$
- exception: NA

isAscending(arr)

- transition:NA
- output: $out := \neg\exists(i|i \in [0..|arr| - 2] : arr_{i+1} < arr_i)$
- exception: NA

array_equal($arr1$,$arr2$):

- transition: NA
- output: $out := (|arr1| == |arr2|) \wedge (\forall i \in [0 \text{ to } |arr1| - 1], arr1[i] == arr2[i]) \implies True$
- exception: NA

10.4.5 Local Functions

NA

10.4.6 Considerations

This module is provided by numpy in Python.

11 MIS of Plot Module

11.1 Module

plot

11.2 Uses

NA

11.3 Syntax

11.3.1 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
Plot	$X : \mathbb{R}^n, Y : \mathbb{R}^n$		SeqSizeMismatch

11.4 Semantics

11.4.1 State Variables

NA

11.4.2 Environment Variables

win: 2D sequence of pixels displayed on the screen

11.4.3 Assumptions

NA

11.4.4 Access Routine Semantics

Plot(X, Y)

- transition: modify win so that it displays an x-y graph of the data points showing X and the corresponding Y values. X is the independent variable and Y as the dependent variable.
- exception: $(|X| \neq |Y| \Rightarrow \text{SeqSizeMismatch})$

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.