Module Interface Specification for LODES (Library of ODE Solvers)

Paul Aoanan

December 17, 2017

1 Revision History

Date	Version	Notes
December 4, 2017	1.0	Initial draft.
December 17, 2017	1.1	Addressed comments and initial release.

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at the following Github link:
`https://github.com/aoananp/cas741/blob/master/Doc/SRS/CA.pdf`

[You don't really use the same symbols as your SRS. Instead, you seem to prefer symbols that are closer to the code. I think you might find the MIS easier to read if you used the SRS notation. For instance, x_0 is easier to read than x_0. If you want to use the code style symbols, you should show the mapping between the SRS symbols and the MIS symbols. —SS]

Contents

List of Tables

3 Introduction

The following document details the Module Interface Specifications for LODES, the Library of ODE Solvers.

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

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 LODES 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, LODES 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	External Interface Module Euler's Method Module Trapezoidal Method Module Heun's Method Module Runge-Kutta's Method Module
Software Decision Module	Equation String Parser Module Output Format Module

Table 1: Module Hierarchy

6 MIS of External Interface Module

This module is the interface exposed to the external world or driver program. It provides access to the library and returns the solution to the ODE IVP.

6.1 Module

lodes

6.2 Uses

EqParse (Section 7), Euler (Section 8), Trap (Section 9), Heun (Section 10), RK (Section 11), Output (Section 12)

6.3 Syntax

Name	**In**	**Out**	**Exceptions**
ODE_method	ODE_method $\in \{1, 2, 3, 4\}$	-	inputerror
ODE_eq	string	-	badODEEq
x_0	$\mathbb{R}$	-	badX0
y_0	$\mathbb{R}$	-	badY0
x_k	$\mathbb{R}$	-	badXK
h	h such that h $\in \mathbb{R}$ and h > 0	-	badH
plot	bool	-	-
displayResult	bool	-	-
x	-	[1 x n] $\in \mathbb{R}$	-
y	-	[1 x n] $\in \mathbb{R}$	-
success	-	BOOL	-

[Rather than ODE methods being selected by an integer from 1 to 4, why not define a new enumerated type with the possible values Euler, Trap, etc.? —SS] [Comment taken into account, but the design intent was to make it low-level in a sense that string manipulation shall be minimized. —PA]

6.4 Semantics

6.4.1 State Variables

none

6.4.2 Access Routine Semantics

lodes(ODE_method, ODE_eq, x_0, y_0, x_k, h, plot, displayResult): [displayResult is missing here —SS] [Thank you - added the variable. —PA]

- Pseudocode:

```
function f, bool eq_OK := EqParse(ODE_eq);
if NOT(eq_OK)
    return badODEEq := true, success := false;
if NOT(ISREAL(x_0))
    return badX0 := true, success := false;
if NOT(ISREAL(y_0))
    return badY0 := true, success := false;
if NOT(ISREAL(x_k))
    return badXK := true, success := false;
if NOT(ISREAL(h) and h > 0)
    return badH := true, success := false;
Select ODE_method:
    Case: 1
        x, y, success := euler(f, x_0, y_0, x_k, h);
    Case: 2
        x, y, success := trap(f, x_0, y_0, x_k, h);
    Case: 3
        x, y, success := heun(f x_0, y_0, x_k, h);
    Case: 4
        x, y, success := rk(f, x_0, y_0, x_k, h);
    Case: else
        return inputerror := true, success := false;
End Select

output(x, y, plot, displayResult);

return x, y, success;
```

7 MIS of the Equation String Parser

This module handles the implementation of the Equation String Parser module.

7.1 Module

eqParse

7.2 Uses

none

7.3 eqParse

Name	In	Out	Exceptions
ODE_eq	string	-	-
f	-	Machine-interpreted equation	-
eq_OK	-	bool	-

[You misinterpreted the MIS syntax section. The name is the name of the access program and the in and out are the types of the inputs and outputs. The name for your syntax section should be eqParse. —SS] [Thank you - modifications done in the document. —PA]

7.4 Semantics

7.4.1 State Variables

none

7.4.2 Access Routine Semantics

eqParse(ODE_eq):

- Pseudocode:

```
try
    f := parse(ODE_eq); %convert the ODE equation string to a machine-interpretable string
    return f, eq_OK := true;
catch
    return f := 0, eq_OK := false;
```

8 MIS of Euler's Method

This module handles the implementation of solving an ODE IVP using Euler's Method.

8.1 Module

euler

8.2 Uses

None applicable.

8.3 euler

Name	**In**	**Out**	**Exceptions**
f	Machine-interpreted equation	-	-
x_0	$\mathbb{R}$	-	-
y_0	$\mathbb{R}$	-	-
x_k	$\mathbb{R}$	-	-
h	h such that $h \in \mathbb{R}$ and $h > 0$	-	-
x	-	$\mathbb{R}^n$	-
y	-	$\mathbb{R}^n$	-
success	-	BOOL	-

[The notation for a sequence of reals is odd. It would be easier to understand if you just used $\mathbb{R}^n$ for the sequence of real numbers. —SS] [Thank you - it has been modified. —PA]

[If your constraint on h is violated, what happens? I would think that there would be an exception. —SS] [This constraint has been discussed in the Interface Module. No error handling is done in this module. —PA]

8.4 Semantics

8.4.1 State Variables

none

8.4.2 Access Routine Semantics

euler(f, x_0, y_0, x_k, h):

- Pseudocode:

```
success := false;
x(1) := x_0;
y(1) := y_0;
N := (x_0 - x_k) / h;
for n = 1 to N
        x(n+1) := x(n) + h;
        y(n+1) := y(n) + h * f(x(n), y(n));
end for
success := true;
return x, y, success;
```

9 MIS of Trapezoidal Method

This module handles the implementation of solving an ODE IVP using the Trapezoidal Method.

9.1 Module

trap

9.2 Uses

None applicable.

9.3 trap

Name	**In**	**Out**	**Exceptions**
f	Machine-interpreted equation	-	-
x_0	$\mathbb{R}$	-	-
y_0	$\mathbb{R}$	-	-
x_k	$\mathbb{R}$	-	-
h	h such that h $\in \mathbb{R}$ and h > 0	-	-
x	-	$\mathbb{R}^n$	-
y	-	$\mathbb{R}^n$	-
success	-	BOOL	-

9.4 Semantics

9.4.1 State Variables

none

9.4.2 Access Routine Semantics

euler(f, x_0, y_0, x_k, h):

- Pseudocode:

```
success := false;
double yNext = 0.0;
x(1) := x_0;
y(1) := y_0;
N := (x_0 - x_k) / h;
for n = 1 to N
        x(n+1) := x(n) + h;
        %Solve for yNext, then store in y(n+1)
        y(n+1) := solve(yNext := y(n) + (h/2) * f(x(n+1), yNext), yNext)
end for
success := true;
return x, y, success;
```

10 MIS of Heun Method

This module handles the implementation of solving an ODE IVP using Heun's Method.

10.1 Module

heun

10.2 Uses

None applicable.

10.3 heun

Name	**In**	**Out**	**Exceptions**
f	Machine-interpreted equation	-	-
x_0	$\mathbb{R}$	-	-
y_0	$\mathbb{R}$	-	-
x_k	$\mathbb{R}$	-	-
h	h such that $h \in \mathbb{R}$ and $h > 0$	-	-
x	-	$\mathbb{R}^n$	-
y	-	$\mathbb{R}^n$	-
success	-	BOOL	-

10.4 Semantics

10.4.1 State Variables

none

10.4.2 Access Routine Semantics

heun(f, x_0, y_0, x_k, h):

- Pseudocode:

```
success := false;
x(1) := x_0;
y(1) := y_0;
N := (x_0 - x_k) / h;
for n = 1 to N
        x(n+1) := x(n) + h;
        y(n+1) := y(n) + (h/2) * (f(x(n) + h, y(n) + h * f(x(n), y(n)));
end for
success := true;
return x, y, success;
```

11 MIS of Runge-Kutta 4 Method

This module handles the implementation of solving an ODE IVP using the Runge-Kutta 4 Method.

11.1 Module

rk

11.2 Uses

None applicable.

11.3 rk

Name	**In**	**Out**	**Exceptions**
f	Machine-interpreted equation	-	-
x_0	$\mathbb{R}$	-	-
y_0	$\mathbb{R}$	-	-
x_k	$\mathbb{R}$	-	-
h	h such that $h \in \mathbb{R}$ and $h > 0$	-	-
x	-	$\mathbb{R}^n$	-
y	-	$\mathbb{R}^n$	-
success	-	BOOL	-

11.4 Semantics

11.4.1 State Variables

none

11.4.2 Access Routine Semantics

rk(f, x_0, y_0, x_k, h):

- Pseudocode:

```
success := false;
x(1) := x_0;
y(1) := y_0;
double k1, k2, k3, k4;
N := (x_0 - x_k) / h;
for n = 1 to N
    x(n+1) := x(n) + h;
    k1 := f(x(n), y(n));
    k2 := f(x(n) + h/2, y(n) + h * (k1/2));
    k3 := f(x(n) + h/2, y(n) + h * (k2/2));
    k4 := f(x(n) + h, y(n) + h * k3);
    y(n+1) := y(n) + (h/6) * (k1 + 2*k2 + 2*k3 + k4);
end for
success := true;
return x, y, success;
```

12 MIS of the Output Module

This module handles the implementation of the Output module.

12.1 Module

output

12.2 Uses

Hardware Hiding (Section 13)

12.3 output

Name	In	Out	Exceptions
plot	BOOL	-	-
displayResult	BOOL	-	-
x	[1 x n]	-	-
y	[1 x n]	-	-

12.4 Semantics

12.4.1 State Variables

none

12.4.2 Access Routine Semantics

output(x, y, plot, displayResult):

- Pseudocode:
  ```
  if (plot)
          plot(x, y);
  if (displayResult)
          print(x);
          print(y);
  end if
  ```

13 MIS of the Hardware Hiding Module

This module handles the implementation of the Output module.

[Unless you are writing device drivers, you don't really need to specify the access programs for the hardware hiding module. As it is, your access programs are unclear. —SS]

13.1 Module

hardwarehiding

13.2 Uses

none

13.3 hardwarehiding

Name	**In**	**Out**	**Exceptions**
x	[1 x n]	-	-
y	[1 x n]	-	-

13.4 Semantics

13.4.1 State Variables

none

13.4.2 Access Routine Semantics

plot(x, y):

- Pseudocode:
  displayPlotToScreen(x, y); %display x vs. y graph on screen

display(x):

- Pseudocode:
  outputToScreen(x); %display array on screen

[I would have expected environment variables here to show the connection between your code and the file system and the screen. I would expect the output related environment variables to be in your output module. In this way you can hide the details of the hardware hiding module. —SS]

[The Hardware Hiding Module uses the same variables as the Output Module. The intent of this module is to bridge the gap between the OS and HW implementation with LODES. I am not familiar with the environment variables for MATLAB and MacOS (i.e. pixels, registry addresses, etc.) I will look into it further in the next revision of the MIS. —PA]

14 MG to MIS Traceability Matrix

MG Module	**MIS Module**
M1 - Hardware Hiding	Section 13 - Hardware Hiding
M2 - External Interface	Section 6 - External Interface
M3 - Equation String Parser	Section 7 - Equation String Parser
M4 - Output Format Module	Section 12 - Output
M5 - Euler's Method	Section 8 - Euler's Method
M6 - Trapezoidal Method	Section 9 - Trapezoidal Method
M7 - Heun's Method	Section 10 - Heun's Method
M8 - Runge-Kutta 4 Method	Section 11 - Runge-Kutta 4 Method

Table 2: Trace Between Module Guide and Module Interface Specification

[Nice to show this. —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`.

15 Appendix

Not applicable. [If it isn't applicable, you can comment out this section. —SS]