Vectorial ODE System Solvers

Tianyang Dong, Jiahua Wu

December 14, 2018

1 Introduction

This project aims at solving vectorial linear ordinary differential equation system:

$$y'(t,y) = f(t,y) = Ay + g(t), t \in \mathbb{R}, y \in \mathbb{R}^n$$

using classic explicit solving methods: Euler Forward Method, Adams-Bashforth Method (order=1, 2, 3, 4) and Runge-Kutta fourth order Method. The methods are implemented in C++ using typical class structure and with the support of Googletest, test suites are used for validating the correctness of the program.

2 Program compilation and Typical Execution

The program is complied using cmake, in the folder cloned from c4science:

```
$ cmake .
$ make
```

Attention: When the build directory is modified, please change the paths in setting.dat accordingly.
An executable file will then be generated in ODE_Solver/. To run the program:

```
$ ./ODE_solver [setting-path(with respect to the executable)][solving
-method-name]
```

In our case, to run the program, in ODE_Solver/:

```
$ ./ODE_solver ./settings/setting.dat [solving-method-name]
//Valid solving method names: ForwardEuler, Adams_Bashforth,RKSystem4th.
```

With a valid input, the program runs the flow in 1.
To generate the documentation, in the file cloned from c4science:

```
$ doxygen Doxyfile
```

A folder called "Documentation", which contains two subfolders called "html" and "latex" , will be created. Please enter "html" and open the file "index.html" to have an overview of the project.

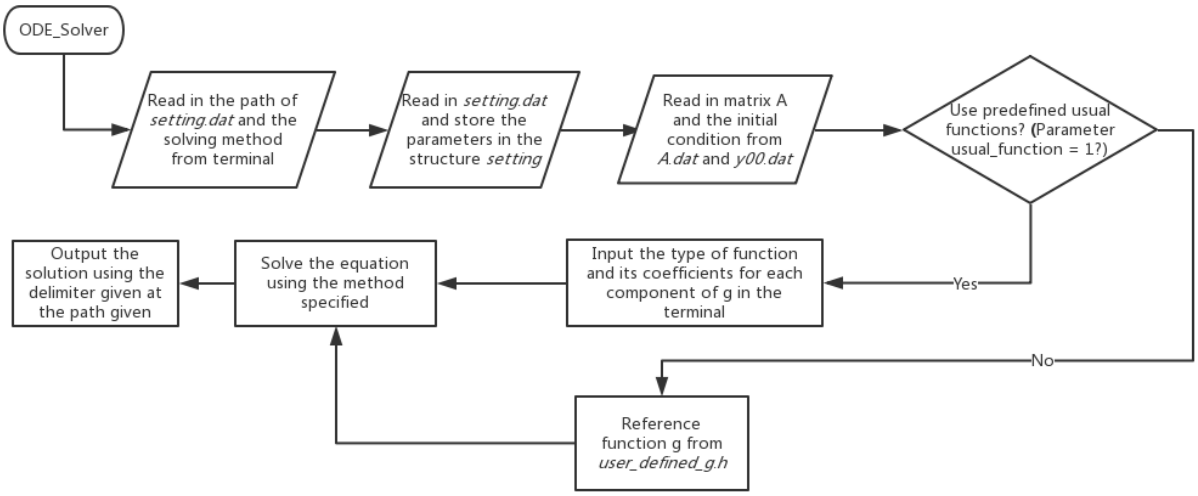

Figure 1: Flow Graph

3 Implementation

As for data structure, we employ std::vector to create matrix and vector, which, compared with static array, is more convenient to manipulate since its size can be dynamically changed. Additionally, a function pointer is used to store the function g. To facilitate further modification of data type, we create an alias Real for double through command typedef.
To compute the numerical solution of a linear ODE system, we need initial time t_0, end time t_n, initial condition y_{00}, and number of steps M. A is a $n \times n$ matrix, g is a $\mathbb{R} \to \mathbb{R}^n$ function.

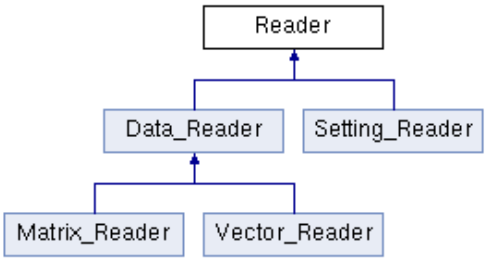

Figure 2: Class Hierarchy: Reader

Except the function g, all the other parameters are read from '.dat' files. Data_Reader and Setting_Reader are daughter classes of abstract class Reader. Another two classes Matrix_Reader and Vector_Reader derived from Data_Reader can read A and y_{00}, while Setting_Reader is responsible for reading t_0, t_n, M, the address of A, y_{00}, the address where the solution will be written in, and other relative information.
As can be seen from the graph 3, mother class ODE_System is used to store all those

variables that are read in by Data_Reader and its daughter classes. From ODE_System, three different classes are derived: ForwardEuler_System, Adams_Bashforth_System, and RKSystem_System. They corresponds to Forward Euler method, Adams Bashforth method and Runge-Kutta 4th order method.

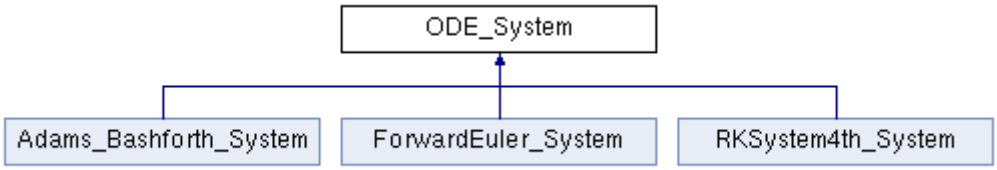

Figure 3: Class Hierarchy: ODE_Solver

4 Features

- Implement of Forward Euler method, four steps of Adams Bashforth method and Runge-Kutta 4th order method.
- Enable user to personalize various parameters including paths of prescribed data, output data path, initial time, end time, number of discretization step, etc.
- Three options of common elementary functions for g in the right hand side, handily used in terminal and possibility for user to define more complex function in 'settings/user_defined_g.h'.

5 Test

With the help of Googletest, several tests are implemented to validate the correctness of the program.
The test is conducted in three parts. The first part is about the input and output process, whether the data written in the folder settings/ can be read in by Setting_Reader, Matrix_Reader and Vector_Reader successfully. Besides, in case that the user wants to use his own function g, a test about the use of g in settings/user_defined_g.h is also carried out. We use macro TEST_P to generate a series of different parameters t, and validate the accuracy of $g(t)$. The solution written in solution/solution is read in again, so that we can test if the solution is correctly written.
The second part is about the usage of overloaded operators. Several simple examples are used to test if the operators (*, +, -) work or not.
In the third part we use three test suites. We use the implemented methods to compute the following example:
$y'(t) = f(t, y) = Ay + g(t)$

$$A = \begin{bmatrix} 2 & 1 & -2 \\ -1 & 0 & 0 \\ 1 & 1 & -1 \end{bmatrix}, g(t) = \begin{bmatrix} 2-t \\ 0 \\ 1-t \end{bmatrix}, \text{ initial condition } y_{00} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix},$$

initial time $t_0 = 0$, end time $t_n = 2$, number of steps $M = 100$.
The solution to this ODE system is $y(t) = e^x \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} + 2 \begin{bmatrix} \sin(x) \\ \cos(x) \\ \sin(x) \end{bmatrix} + \begin{bmatrix} \cos(t) \\ -\sin(t) \\ \cos(t) \end{bmatrix} + \begin{bmatrix} -1 \\ t \\ 0 \end{bmatrix}$
Eulertest, ABtest and RK4test are the test suites for the three solving methods. The computed solutions are compared with the accurate solution with a certain tolerance. To run the test, in the folder test/ :

```
$ cmake .
$ make
$ ./test
```

The results is as follows (part of the results are omitted):

```
[==========] Running 209 tests from 9 test cases.
[----------] Global test environment set-up.
[----------] Global test environment tear-down
[==========] 209 tests from 9 test cases ran. (56 ms total)
[  PASSED  ] 209 tests.
```

6 TODO

This project still has some limitations, however.

1. Our project now only supports three explicit solving methods. Implicit solving methods will be implemented in the future.

2. The project only supports some basic function types ($\sin, \cos, \exp$ and polynomials). If the user wants to use more complex forms of function g, he has to define his own function in a '.h' file and recomplie the whole program again. In future work, we will try to enable users to define complex functions from the terminal.

3. In this project we store the solution of every step, sometimes it is not necessary to do so because the user may only want the solution of the end time. The choice of solution store type may be offered, so that it is up to the user to decide how he wants the solution to be stored.