Abstract ODE Class

Saurav Samantaray

Department of Mathematics

Indian Institute of Technology Madras

October 27, 2024

Solving an Initial Value Problem

- Suppose we want to write an object-oriented program for calculating the numerical solution of initial value ordinary differential equations of the form

  $$\frac{dy}{dt} = f(t, y), \quad y(T_0) = y_0$$

  where $f(t, y)$ is a given function, and T_0, y_0 are given values.
- Many methods exist for calculating the numerical solution of equations such as these, for example, forward Euler method, various Runge–Kutta methods etc. .
- Suppose we want to calculate a numerical solution in the time interval $T_0 < t < T_1$ where T_1 is the final time.
- To solve this equation numerically, we require the user to specify an integration step size, which we denote by h.
- For the step size h we have we define the points $t_i, i = 0, 1, 2, \cdots, N$ by

  $$t_i = T_0 + ih$$

  where h is chosen so that $t_N = T_1$.
- The numerical solution at these points is denoted by $y_i, i = 0, 1, 2, \cdots, N$.

Numerical Methods

Forward Euler method

- set $y_0 = y_0$.
- For $i = 1, 2, \cdots, N$, y_i is given by

$$y_i = y_{i-1} + hf(t_{i-1}, y_{i-1}).$$

Fourth order Runge–Kutta method

-
- set $y_0 = y_0$.
- For $i = 1, 2, \cdots, N$, y_i is calculated using the following formulae:

$$\begin{aligned} k_1 &= hf(t_{i-1}, y_{i-1}), \\ k_2 &= hf\left(t_{i-1} + \frac{1}{2}h, y_{i-1} + \frac{1}{2}k_1\right), \\ k_3 &= hf\left(t_{i-1} + \frac{1}{2}h, y_{i-1} + \frac{1}{2}k_2\right), \\ k_4 &= hf\left(t_{i-1} + h, y_{i-1} + k_3\right), \\ y_i &= y_{i-1} + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned}$$

The Abstract Class Pattern

- One way of implementing these numerical methods would be to write a class called `AbstractOdeSolver` that has members that would be used by all of these numerical methods,
- such as variables representing the stepsize and initial conditions,
- a method that represents the function $f(t, y)$ on the right-hand side of the equation above,
- and a virtual method `SolveEquation` for implementing one of the numerical techniques described above.
- We would then implement each of the numerical methods using a class derived from]`AbstractOdeSolver`, and overriding the virtual function `SolveEquation`.
- The derived classes would then contain members that allow a specific numerical algorithm to be implemented, as well as the members of the base class `AbstractOdeSolver` that would be required by all of the numerical solvers.

The Abstract Class Pattern

- Using the class structure described above, the base class AbstractOdeSolver would not actually include a numerical method for calculating a numerical solution of a differential equation,
- So we would not want to ever create an instance of this class.
- It can be automatically enforced by making `AbstractOdeSolver` an abstract class.
- This is implemented by setting the virtual functions `SolveEquation` and `RightHandSide` to be pure virtual functions
- See ”AbstractOdeSolver.hpp”.
- We indicate that these functions are pure virtual functions by completing the declaration of these members with “= 0”.
- Should we mistakenly attempt to create an instance of the class `AbstractOdeSolver` we would get a compilation error.