A Runge-Kutta-Fehlberg solver using traits and concepts (part I)

Matteo Caldana

13/04/2023

RKF solver

This exercise (a modified version of `Examples/src/RKFSolver`) is about a set of tools that implements embedded Runge-Kutta-Fehlberg (explicit) methods to solve non-linear scalar and vector Ordinary Differential Equations, based on the `Eigen` library.

Consider an ODE of the form

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t, y),$$

and consider the coefficients b_i, b_i^*, a_{ij}, c_i to be given in the form of a Butcher tableau. We recall that, at each time step t_n, an embedded RKF method of order p consists of the following steps.

1. Compute the high-order solution $y_{n+1} = y_n + h_n \sum_{i=1}^{p} b_i k_i$, where $k_i = f\left(t_n + c_i h_n, y_n + h_n \sum_{j=1}^{i-1} a_{ij} k_j\right)$, and $h_n = t_{n+1} - t_n$.
2. Compute the low-order solution $y_{n+1}^* = y_n + h_n \sum_{i=1}^{p} b_i^* k_i$.
3. Compute the error $\varepsilon_{n+1} = y_{n+1} - y_{n+1}^* = h_n \sum_{i=1}^{p} (b_i - b_i^*) k_i$.
4. Adapt the step size $h_{n+1} = \tau_n h_n$, where τ_n is a prescribed reduction (< 1)/expansion (> 1) factor depending on whether ε_{n+1} is larger or smaller than a prescribed tolerance.

Exercise: RKF solver

The code structure is the following:

- `ButcherRKF` contains the definition of the Butcher tableaux for some common RKF methods.
- `RKFTraits` defines the basic structure that enables to bind the type of the equation(s) to be solved (*i.e.* scalar or vector).
- `RKFResult` is a data structure containing the output of the RKF solver.
- `RKF` implements a generic RKF solver interface, filling a proper `RKFResult` object.

Starting from the provided solution sketch:

1. Implement the `concept`(s) defining a scalar and a vector in `RKFTraits.hpp`.
2. Implement the `RKF::RKFstep` method, performing just one timestep of the RKF method.
3. Implement the `RKF::solve()` time-advancing method, without error correction.
4. Use the just implemented solver to solve the model problem and the Van der Pol oscillator problem defined in the `main.cpp` file.
5. Include error correction into the solver and compare the results.

Additional work

Maybe you want to extend the class to include Diagonally Implicit Runge Kutta Methods (DIRK), where the A matrix of the Butcher array satisfies $A_{ij} = 0$ if $j < i$, but A_{ii} may be different from 0.

In this case, at each stage we need to solve a non-linear system. Try to investigate the changes needed to implement a DIRK starting from the already implemented RKF solution.

A possible implementation is in `Examples/src/RKFSolver/`.