Ordinary differential equations

Introduction

Many scientific problems can be formulated in terms of a system of *ordinary differential equations* (ODE),

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}) \ , \tag{1}$$

with an initial condition

$$\mathbf{y}(x_0) = \mathbf{y}_0 \ , \tag{2}$$

where $\mathbf{y}' \equiv d\mathbf{y}/dx$, and the boldface variables $\mathbf{y}$ and $\mathbf{f}(x, \mathbf{y})$ are generally understood as column-vectors.

Runge-Kutta methods

Runge-Kutta methods are one-step methods for numerical integration of ODE (1). The solution $\mathbf{y}$ is advanced from the point x_0 to $x_1 = x_0 + h$ using a one-step formula

$$\mathbf{y}_1 = \mathbf{y}_0 + h\mathbf{k}, \tag{3}$$

where $\mathbf{y}_1$ is the approximation to $\mathbf{y}(x_1)$, and $\mathbf{k}$ is a cleverly chosen (vector) constant. The Runge-Kutta methods are distinguished by their *order*: a method has order p if it can integrate exactly an ODE where the solution is a polynomyal of order p. In other words, if the error of the method is $O(h^{p+1})$ for small h.

The first order Runge-Kutta method is the Euler's method,

$$\mathbf{k} = \mathbf{f}(x_0, \mathbf{y}_0) \ . \tag{4}$$

Second order Runge-Kutta methods advance the solution by an auxiliary evaluation of the derivative, e.g. the *mid-point method*,

$$\begin{aligned} \mathbf{k}_0 &= \mathbf{f}(x_0, \mathbf{y}_0) \ , \\ \mathbf{k}_{1/2} &= \mathbf{f}(x_0 + \tfrac{1}{2}h, \mathbf{y}_0 + \tfrac{1}{2}h\mathbf{k}_0) \ , \\ \mathbf{k} &= \mathbf{k}_{1/2} \ , \end{aligned} \tag{5}$$

or the *two-point method*,

$$\begin{aligned} \mathbf{k}_0 &= \mathbf{f}(x_0, \mathbf{y}_0), \\ \mathbf{k}_1 &= \mathbf{f}(x_0 + h, \mathbf{y}_0 + h\mathbf{k}_0), \\ \mathbf{k} &= \frac{1}{2}(\mathbf{k}_0 + \mathbf{k}_1) \ . \end{aligned} \tag{6}$$

These two methods can be combined into a third order method,

$$\mathbf{k} = \frac{1}{6}\mathbf{k}_0 + \frac{4}{6}\mathbf{k}_{1/2} + \frac{1}{6}\mathbf{k}_1 \ . \tag{7}$$

The most commont is the fourth-order method, which is called *RK4* or simply *the Runge-Kutta method*,

$$\begin{aligned} \mathbf{k}_0 &= \mathbf{f}(x_0, \mathbf{y}_0) \ , \\ \mathbf{k}_1 &= \mathbf{f}(x_0 + \tfrac{1}{2}h, \mathbf{y}_0 + \tfrac{1}{2}h\mathbf{k}_0) \ , \\ \mathbf{k}_2 &= \mathbf{f}(x_0 + \tfrac{1}{2}h, \mathbf{y}_0 + \tfrac{1}{2}h\mathbf{k}_1) \ , \\ \mathbf{k}_3 &= \mathbf{f}(x_0 + h, \mathbf{y}_0 + h\mathbf{k}_2) \ , \\ \mathbf{k} &= \tfrac{1}{6}(\mathbf{k}_0 + 2\mathbf{k}_1 + 2\mathbf{k}_2 + \mathbf{k}_3) \ . \end{aligned} \tag{8}$$

Higher order Runge-Kutta methods have been devised, with the most famous being the Runge-Kutta-Fehlberg fourth/fifth order method, *RKF45*, implemented in the renowned `rkf45` Fortran routine.

Multistep methods

Multistep methods try to use the information about the function gathered at the previous steps. They are generally not *self-starting* as there are no previous points at the start of the integration. The first step must be a one-step method, for example a Runge-Kutta step.

A two-step method

Given the previous point, $(x_{-1}, \mathbf{y}_{-1})$, in addition to the current point $(x_0, \mathbf{y}_0)$, the sought function $\mathbf{y}$ can be approximated in the vicinity of the point x_0 as

$$\bar{\mathbf{y}}(x) = \mathbf{y}_0 + \mathbf{y}'_0 \cdot (x - x_0) + \mathbf{c} \cdot (x - x_0)^2, \tag{9}$$

where $\mathbf{y}'_0 = \mathbf{f}(x_0, \mathbf{y}_0)$ and the coefficient $\mathbf{c}$ is found from the condition $\mathbf{y}(x_{-1}) = \mathbf{y}_{-1}$,

$$\mathbf{c} = \frac{\mathbf{y}_{-1} - \mathbf{y}_0 + \mathbf{y}'_0 \cdot (x_0 - x_{-1})}{(x_0 - x_{-1})^2}. \tag{10}$$

The value of the function at the next point, x_1, can now be estimated as $\bar{\mathbf{y}}(x_1)$ from (9).

The error of this second-order two-step stepper can be estimated by a comparison with a first-order Euler's step, which is given by the linear part of (9). The difference $\|\mathbf{c}\|h^2$ between the two steppers can thus serve as the error estimate.

A three-step method

Including yet another point, $(x_{-2}, \mathbf{y}_{-2})$, allows to further increase the order of the approximationi (9),

$$\check{\mathbf{y}}(x) = \bar{\mathbf{y}}(x) + \mathbf{d} \cdot (x - x_0)^2 (x - x_{-1}) \,. \tag{11}$$

The coefficient $\mathbf{d}$ can be found from the condition $\check{\mathbf{y}}(x_{-2}) = \mathbf{y}_{-2}$, which gives

$$\mathbf{d} = \frac{\mathbf{y}_{-2} - \bar{\mathbf{y}}(x_{-2})}{(x_{-2} - x_0)^2 (x_{-2} - x_{-1})} \,. \tag{12}$$

The error estimate is now

$$\text{err} \approx \|\check{\mathbf{y}}(x + h) - \bar{\mathbf{y}}(x + h)\| \,. \tag{13}$$

Predictor-corrector methods

Predictor-corrector methods use extra iterations to improve the solution. For example, the two-point Runge-Kutta method (6) is as actually a predictor-corrector method, as it first calculates the *prediction* $\tilde{\mathbf{y}}_1$ for $\mathbf{y}(x_1)$,

$$\tilde{\mathbf{y}}_1 = \mathbf{y}_0 + h\mathbf{f}(x_0, \mathbf{y}_0) \,, \tag{14}$$

and then uses this prediction in a *correction* step,

$$\tilde{\tilde{\mathbf{y}}}_1 = \mathbf{y}_0 + h\frac{1}{2}\big(\mathbf{f}(x_0, \mathbf{y}_0) + \mathbf{f}(x_1, \tilde{\mathbf{y}}_1)\big) \tag{15}$$

Similarly, one can use the two-step approximation (9) as a predictor, and then improve it by one order with a correction step, namely

$$\bar{\tilde{\mathbf{y}}}(x) = \bar{\mathbf{y}}(x) + \mathbf{d} \cdot (x - x_0)^2 (x - x_{-1}). \tag{16}$$

The coefficient $\mathbf{d}$ can be found from the condition $\bar{\tilde{\mathbf{y}}}'(x_1) = \bar{\mathbf{f}}_1$, where $\bar{\mathbf{f}}_1 = \mathbf{f}(x_1, \bar{\mathbf{y}}(x_1))$,

$$\mathbf{d} = \frac{\bar{\mathbf{f}}_1 - \mathbf{y}'_0 - 2\mathbf{c} \cdot (x_1 - x_0)}{2(x_1 - x_0)(x_1 - x_{-1}) + (x_1 - x_0)^2}. \tag{17}$$

Equation (16) gives a better estimate, $\mathbf{y}_1 = \bar{\tilde{\mathbf{y}}}(x_1)$, of the function at the point x_1.

In this context the formula (9) serves as *predictor*, and (16) as *corrector*. The difference between the two gives an estimate of the error.

Step size control

Error estimate

The error δy of the integration step for a given method can be estimated e.g. by comparing the solutions for a full-step and two half-steps (the *Runge principle*),

$$\delta y \approx \frac{y_{\text{two\_half\_steps}} - y_{\text{full\_step}}}{2^p - 1}, \tag{18}$$

where p is the order of the algorithm used. It is better to pick formulas where the full-step and two half-step calculations share the evaluations of the function $\mathbf{f}(x, \mathbf{y})$.

Another possibility is to make the same step with two methods of different orders, the difference between the solutions providing an estimate of the error.

In a predictor-corrector method the correction itself can serve as the estimate of the error.

Table 1: Runge-Kutta mid-point stepper with error estimate.

```
#include<vector>
#include<functional>
#include<armadillo>
using namespace arma;

void rkstep12(
        const std::function<vec(double,vec)> & f,
        const double & x0, const vec & y0, const double & h,
        vec & y1, vec & dy)
{
        vec k0  = f(x0,y0);
        vec k12 = f(x0+h/2,y0+k0*h/2);
        y1 = y0 + k12*h;
        dy = (k0 -k12)*h/2;
}
```

Adaptive step size control

Let *tolerance* τ be the maximal accepted error consistent with the required absolute, δ, and relative, ϵ, accuracies to be achieved in the integration of an ODE,

$$\tau = \epsilon \|\mathbf{y}\| + \delta \,, \tag{19}$$

where $\|\mathbf{y}\|$ is the "norm" of the column-vector $\mathbf{y}$.

Suppose the inegration is done in n steps of size h_i such that $\sum_{i=1}^{n} h_i = b - a$. Under assumption that the errors at the integration steps are random and independent, the step tolerance τ_i for the step i has to scale as the square root of the step size,

$$\tau_i = \tau \sqrt{\frac{h_i}{b-a}}. \tag{20}$$

Then, if the error e_i on the step i is less than the step tolerance, $e_i \le \tau_i$, the total error E will be consistent with the total tolerance τ,

$$E \approx \sqrt{\sum_{i=1}^{n} e_i^2} \le \sqrt{\sum_{i=1}^{n} \tau_i^2} = \tau \sqrt{\sum_{i=1}^{n} \frac{h_i}{b-a}} = \tau \,. \tag{21}$$

In practice one uses the current values of the function $\mathbf{y}$ in the estimate of the tolerance,

$$\tau_i = (\epsilon \|\mathbf{y}_i\| + \delta) \sqrt{\frac{h_i}{b-a}} \tag{22}$$

The step is accepted if the error is smaller than tolerance. The next step-size can be estimated according to the empirical prescription

$$h_{\text{new}} = h_{\text{old}} \times \left(\frac{\tau}{e}\right)^{\text{Power}} \times \text{Safety}, \tag{23}$$

where Power ≈ 0.25, Safety ≈ 0.95. If the error e_i is larger than tolerance τ_i the step is rejected and a new step with the new step size (23) is attempted.

Table 2: An ODE driver with adaptive step size control.

```
#include <cmath>
#include <armadillo>
#include <vector>
using namespace arma;

void rkstep12(
   const std::function<vec(double,vec)> & f,
   const double & x0, const vec & y0, const double & h,
   vec & y1, vec & dy);

int rkdrive(
        const std::function<vec(double,vec)> & f,
        std::vector<double>&xlist, std::vector<vec>&ylist,
        double b, double h, double acc, double eps)
{
        int k=0, n=ylist[0].size();
        double a=xlist[0];
        vec dy(n),y1(n);
        while(xlist[k]<b){
                double x=xlist[k];
                vec    y=ylist[k];
                if(x+h>b) h=b-x;
                rkstep12(f,x,y,h,y1,dy);
                double err=norm(dy,2);
                double normy=norm(y1,2);
                double tol=(normy*eps+acc)*sqrt(h/(b-a));
                if(tol>err){ // accept step and go on
                        k++;
                        xlist.push_back(x+h);
                        ylist.push_back(y1);
                        }
                if(err>0) h = h*pow(tol/err,0.25)*0.95;
                else h = 2*h;
                }//end while
        return k;
}// end rkdrive
```