# 1 Ordinary differential equations

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

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

with an initial condition

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

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

## 1.1 Runge-Kutta methods

Runge-Kutta methods are one-step methods for numerical integration of ODEs (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 $p$, or, 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 the most famous being the Runge-Kutta-Fehlberg fourth/fifth order method, *RKF45*, implemented in the renowned `rkf45.f` Fortran routine.

## 1.2 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.

### 1.2.1 A two-step method

Given two points, $(x_0, \mathbf{y}_0)$ and $(x_1, \mathbf{y}_1)$, the sought function $\mathbf{y}$ can be approximated in the vicinity of the point $x_1$ as

$$\bar{\mathbf{y}}(x) = \mathbf{y}_1 + \mathbf{y}_1' \cdot (x - x_1) + \mathbf{c} \cdot (x - x_1)^2, \tag{9}$$

where the coefficient $\mathbf{c}$ is found from the condition $\mathbf{y}(x_0) = \mathbf{y}_0$,

$$\mathbf{c} = \frac{\mathbf{y}_0 - \mathbf{y}_1 - \mathbf{y}_1' \cdot (x_0 - x_1)}{(x_0 - x_1)^2}. \tag{10}$$

The value of the function at the next point, $x_2$, can now be estimated as $\bar{\mathbf{y}}(x_2)$ from (9).

## 1.3 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 $\mathbf{y}(x_1)$,

$$\tilde{\mathbf{y}}_1 = \mathbf{y}_0 + \mathbf{f}(x_0, \mathbf{y}_0)\,, \tag{11}$$

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

$$\tilde{\bar{\mathbf{y}}}_1 = \mathbf{y}_0 + \frac{1}{2}\left(\mathbf{f}(x_0, \mathbf{y}_0) + \mathbf{f}(x_1, \tilde{\mathbf{y}}_1)\right) \tag{12}$$

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

$$\bar{\bar{\mathbf{y}}}(x) = \bar{\mathbf{y}}(x) + \mathbf{d} \cdot (x - x_1)^2(x - x_0). \tag{13}$$

The coefficient $\mathbf{d}$ can be found from the condition $\bar{\bar{\mathbf{y}}}'(x_2) = \bar{\mathbf{f}}_2$, where $\bar{\mathbf{f}}_2 = \mathbf{f}(x_2, \bar{\mathbf{y}}(x_2))$,

$$\mathbf{d} = \frac{\bar{\mathbf{f}}_2 - \mathbf{y}'_1 - 2\mathbf{c}\cdot(x_2 - x_1)}{2(x_2 - x_1)(x_2 - x_0) + (x_2 - x_1)^2}. \qquad (14)$$

Equation (13) gives a better estimate, $\mathbf{y}_2 = \bar{\bar{\mathbf{y}}}(x_2)$, of the function at the point $x_2$.

In this context the formula (9) is referred to as *predictor*, and (13) as *corrector*. The difference between the two gives an estimate of the error.

## 1.4 Step size control

### 1.4.1 Error estimate

The error $\delta y$ 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*),

$$\delta y \approx \frac{y_{\text{two-half-steps}} - y_{\text{full-step}}}{2^p - 1}, \qquad (15)$$

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

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 can serve as the estimate of the error.

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

```
function rkstep(f,x,y,h){
// Runge-Kutta midpoint step
var k0  = f(x,y)
var y12 = [y[i]+k0[i]*h/2 for(i in y)]
var k   = f(x+h/2,y12)
var y1  = [y[i]+k[i]*h for(i in y)]
var dy  = [(k[i]-k0[i])*h/2 for(i in y)]
return [y1, dy]
}
```

### 1.4.2 Adaptive step size control

The *tolerance* $\tau$ is the maximal accepted error on the given integration step consistent with the required absolute, $\delta$, and relative, $\epsilon$, accuracies to be achieved in the integration of an ODEs. Under assumption of random distribution of errors on the integration steps from $a$ to $b$ the tolerance, according to the central limit theorem, scales as a square root of the step-size $h$,

$$\tau = (\epsilon\|y\| + \delta)\sqrt{\frac{h}{b-a}}. \qquad (16)$$

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

$$h_{\text{next}} = h_{\text{previous}} \times \left(\frac{\text{tol}}{\text{err}}\right)^{\text{Power}} \times \text{Safety}, \qquad (17)$$

where Power $\approx 0.25$, Safety $\approx 0.95$.

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

```
function rkdrive(f,xlist,ylist,b,acc,eps,h)
{
// ODE driver: integrates y'=f(x,y) with
// absolute accuracy acc and relative
// accuracy eps until x=b with initial
// step h storing the results
// in arrays xlist and ylist
var norm = function(v)
  Math.sqrt(v.reduce(function(a,b)a+b*b,0))
var a = xlist[0]
var x = xlist[xlist.length-1];
var y = ylist[ylist.length-1];
if(x>=b) return;
if(x+h>b) h=b-x;
var [y1,dy]=rkstep(f, x, y, h);
var err=norm(dy);
var tol=(norm(y1)*eps+acc)*
  Math.sqrt(h/(b-a));
if(err>0)
  var new_h = h*Math.pow(tol/err,.25)*.95;
else
  var new_h = 2*h;
if(tol>err){ // accept step
  xlist.push(x+h);
  ylist.push(y1);
  rkdrive(f,xlist,ylist,b,acc,eps,new_h);}
else // reject step
  rkdrive(f,xlist,ylist,b,acc,eps,new_h);
}
```