Numerical solution of Ordinary Differential equations.

Anne Kværnø

March 25, 2009

1 Some background on ODEs.

In this section some useful notation on ordinary differential equations will be presented. We will also give existence and uniqueness results, but without proofs.

A system of m first order ordinary differential equation is given by

$$y' = f(t, y) \tag{1}$$

or, written out, as

$$\begin{aligned} y_1' &= f_1(t, y_1, \cdots, y_m), \\ y_2' &= f_2(t, y_1, \cdots, y_m), \\ &\vdots \\ y_m' &= f_m(t, y_1, \cdots, y_m). \end{aligned}$$

This is an *initial value problem* (IVP) if the solution is given at some point t_0, thus

$$y_1(t_0) = y_{1,0},\ y(t_0) = y_{2,0},\ \cdots\ y_m(t_0) = y_{m,0}.$$

Example 1.1. *The following equation is an example of the Lotka-Volterra equation:*

$$\begin{aligned} y_1' &= y_1 - y_1 y_2, \\ y_2' &= y_1 y_2 - 2y_2. \end{aligned}$$

An ODE is called *autonomous* if f is not a function of t, but only of y. The Lotka-Volterra equation is an example of an autonomous ODE. A nonautonomous system can be made autonomous by a simple trick, just add the equation

$$y_{m+1}' = 1, \qquad y_{m+1}(t_0) = t_0,$$

and replace t with y_{m+1}. Also higher order ODE/IVPs

$$u^{(m)} = f(t, u, u', \cdots, u^{(m-1)}), \qquad u(t_0) = u_0,\ u'(t_0) = u_0',\ \cdots, u^{(m-1)}(t_0) = u_0^{(m-1)},$$

where $u^{(m)} = d^m u/dt^m$, can be written as a system of first order equations, again by a simple trick: Let

$$y_1 = u,\ y_2 = u',\ \cdots y_m = u^{(m-1)},$$

and we get the system

$$\begin{aligned} y_1' &= y_2, & y_1(t_0) &= u_0, \\ y_2' &= y_2, & y_2(t_0) &= u_0', \\ &\vdots & &\vdots \\ y_{m-1}' &= y_m, & y_{m-1}(t_0) &= u_0^{(m-2)}, \\ y_m' &= f(t, y_1, y_2, \cdots, y_m), & y_m(t_0) &= u_0^{(m-1)}. \end{aligned}$$

Example 1.2. *Van der Pol's equation is given by*

$$u'' + \mu(u^2 - 1)u' + u = 0.$$

Using $y_1 = u$ and $y_2 = u'$ this equation can be rewritten as

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= \mu(1 - y_1^2)y_2 - y_1. \end{aligned}$$

This problem was first introduced by Van der Pol in 1926 in the study of an electronic oscillator.

Before concluding this section, we present some existence and uniqueness results for solution of ODEs.

Definition 1.3. *A function $f : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$ satisfies the* Lipschitz condition *with respect to y on a domain $(a, b) \times D$ where $D \subset \mathbb{R}^m$ if there exist a constant L so that*

$$\|f(t, y) - f(t, \tilde{y})\| \leq L\|y - \tilde{y}\|, \qquad \text{for all} \quad t \in (a, b),\ y, \tilde{y} \in D.$$

The constant L is called the Lipschitz constant.

It is not hard to show that the function f satisfies the Lipschitz condition if $\partial f_i/\partial y_j,\ i, j = 1, \cdots, m$ are continuous and bounded on the domain.

Theorem 1.4. *Consider the initial value problem*

$$y' = f(t, y), \qquad y(t_0) = y_0. \tag{2}$$

If

1. *$f(t, y)$ is continuous in $(a, b) \times D$,*
2. *$f(t, y)$ satisfies the Lipschitz condition with respect to y in $(a, b) \times D$.*

with given initial values $t_0 \in (a, b)$ and $y_0 \in D$, then (2) has one and only one solution in $(a, b) \times D$.

2 Numerical solution of ODEs.

In this section we develop some simple methods for the solution of initial value problems. In both cases, let us assume that we somehow have found solutions $y_l \approx y(t_l)$, for $l = 0, 1, \cdots, n$, and we want to find an approximation $y_{n+1} \approx y(t_{n+1})$ where $t_{n+1} = t_n + h$, where h is the stepsize. Basically, there are two different classes of methods in practical use.

1. *Onestep methods.* Only y_n is used to find the approximation y_{n+1}. Onestep methods usually require more than one function evaluation pr. step. They can all be put in a general abstract form
$$y_{n+1} = y_n + h\Phi(t_n, y_n; h).$$
2. *Linear multistep methods:* y_{n+1} is approximated from $y_{n-k+1}, \cdots, y_n$.

2.1 Some examples of onestep methods.

Assume that t_n, y_n is known. The exact solution $y(t_{n+1})$ with $t_{n+1} = t_n + h$ of (1) passing through this point is given by

$$y(t_n + h) = y_n + \int_{t_n}^{t_{n+1}} y'(\tau)d\tau = y_n + \int_{t_n}^{t_{n+1}} f(\tau, y(\tau))d\tau. \tag{3}$$

The idea is to find approximations to the last integral. The simplest idea is to use $f(\tau, y(\tau)) \approx f(t_n, y_n)$, in which case we get the Euler method again:

$$y_{n+1} = y_n + hf(t_n, y_n).$$

The integral can also be approximated by the trapezoidal rule

$$\int_{t_n}^{t_{n+1}} f(\tau, y(\tau)) = \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y(t_{n+1})).$$

By replacing the unknown solution $y(t_{n+1})$ by y_{n+1} we get the *trapezoidal method*

$$y_{n+1} = y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right).$$

Here y_{n+1} is available by solving a (usually) nonlinear system of equations. Such methods are called implicit. To avoid this extra difficulty, we could replace y_{n+1} on the right hand side by the approximation from Eulers method, thus

$$\begin{aligned}\tilde{y}_{n+1} &= y_n + hf(t_n, y_n);\\ y_{n+1} &= y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, \tilde{y}_{n+1})\right).\end{aligned}$$

This method is called the *improved Euler method*. Similarly, we could have used the midpoint rule for the integral,

$$\int_{t_n}^{t_{n+1}} f(\tau, y(\tau)) = \left(f(t_n + \frac{h}{2}, y(t_n + \frac{h}{2})\right),$$

and replaced $y(t_n + \frac{h}{2})$ by one half Euler step. The result is the *modified Euler method*:

$$\tilde{y}_{n+\frac{1}{2}} = y_n + \frac{h}{2} f(t_n, y_n),$$
$$y_{n+1} = y_n + h f(t_n + \frac{h}{2}, \tilde{y}_{n+\frac{1}{2}}).$$

Do we gain anything by constructing these methods? Let us solve the problem

$$y' == 2ty, \qquad , y(t_0) = 1, \qquad 0 \leq t \leq 1$$

using improved/modified Euler with $h = 0.1$. For each step, also the global error $e_n = y(t_n) - y_n$ is computed. For comparison, also the result for the Euler method is included.

	Euler		improved Euler		modified Euler	
t_n	y_n	e_n	y_n	e_n	y_n	e_n
0.0	1.000000	0	1.000000	0	1.000000	0
0.1	1.000000	$-9.95 \cdot 10^{-3}$	0.990000	$4.98 \cdot 10^{-5}$	0.990000	$4.98 \cdot 10^{-5}$
0.2	0.980000	$-1.92 \cdot 10^{-2}$	0.960696	$9.34 \cdot 10^{-5}$	0.960597	$1.92 \cdot 10^{-4}$
0.3	0.940800	$-2.69 \cdot 10^{-2}$	0.913814	$1.17 \cdot 10^{-4}$	0.913528	$4.03 \cdot 10^{-4}$
0.4	0.884352	$-3.22 \cdot 10^{-2}$	0.852040	$1.04 \cdot 10^{-4}$	0.851499	$6.45 \cdot 10^{-4}$
0.5	0.813604	$-3.48 \cdot 10^{-2}$	0.778765	$3.60 \cdot 10^{-5}$	0.777930	$8.71 \cdot 10^{-4}$
0.6	0.732243	$-3.46 \cdot 10^{-2}$	0.697773	$-9.69 \cdot 10^{-5}$	0.696636	$1.04 \cdot 10^{-3}$
0.7	0.644374	$-3.17 \cdot 10^{-2}$	0.612924	$-2.98 \cdot 10^{-4}$	0.611507	$1.12 \cdot 10^{-3}$
0.8	0.554162	$-2.69 \cdot 10^{-2}$	0.527850	$-5.58 \cdot 10^{-4}$	0.526202	$1.09 \cdot 10^{-3}$
0.9	0.465496	$-2.06 \cdot 10^{-2}$	0.445717	$-8.59 \cdot 10^{-4}$	0.443904	$9.54 \cdot 10^{-4}$
1.0	0.381707	$-1.38 \cdot 10^{-2}$	0.369053	$-1.17 \cdot 10^{-3}$	0.367153	$7.27 \cdot 10^{-4}$

As we can see, there is a significant improvement in accuracy, compared with the Euler method.

3 Runge-Kutta methods

The Euler method, as well as the improved and modified Euler methods are all examples on *explicit Runge-Kutta methods* (ERK). Such schemes are given by

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + c_2 h, y_n + h a_{21} k_1), \\ k_3 &= f\big(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2)\big), \\ &\vdots \\ k_s &= f\big(t_n + c_s h, y_n + h \sum_{j=1}^{s-1} a_{sj} k_j\big), \\ y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i k_i, \end{aligned} \tag{4}$$

where c_i, a_{ij} and b_i are coefficients defining the method. We always require $c_i = \sum_{j=1}^{s} a_{ij}$. Here, s is the number of *stages*, or the number of function evaluations needed for each step.

The vectors k_i are called stage derivatives. The improved Euler method is then a two-stage RK-method, written as

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + h, y_n + hk_1), \\ y_{n+1} &= y_n + \frac{h}{2}(k_1 + k_2). \end{aligned}$$

Also implicit methods, like the trapezoidal rule,

$$y_{n+1} = y_n + \frac{h}{2}\big(f(t_n, y_n) + f(t_n + h, y_{n+1})\big)$$

can be written in a similar form,

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f\big(t_n + h, y_n + \frac{h}{2}(k_1 + k_2)\big), \\ y_{n+1} &= y_n + \frac{h}{2}(k_1 + k_2). \end{aligned}$$

But, contrary to what is the case for explicit methods, a nonlinear system of equations has to be solved to find k_2.

Definition 3.1. *An s-stage Runge-Kutta method is given by*

$$k_i = f\big(t_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij}k_j\big), \qquad i = 1, 2, \cdots, s,$$

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i.$$

The method is defined by its coefficients, which is given in a Butcher tableau

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & & & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}, \qquad \text{where } c_i = \sum_{i=1}^{s} a_{ij}, \quad i = 1, \cdots, s.$$

The method is explicit *if $a_{ij} = 0$ whenever $j \geq i$, otherwise* implicit.

Example 3.2. *The Butcher-tableaux for the methods presented so far are*

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \qquad \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \qquad \begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array} \qquad \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Euler *improved Euler* *modified Euler* *trapezoidal rule*

When the method is explicit, the zeros on and above the diagonal is usually ignored. We conclude this section by presenting the maybe most popular among the RK-methods over times, *The* 4th order Runge-Kutta method (Kutta – 1901):

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_1) \\ k_3 &= f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_2) \\ k_4 &= f(t_n + h, y_n + hk_3) \\ y_{n+1} &= y_n + \tfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \qquad \text{or} \qquad \begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} . \tag{5}$$

3.1 Order conditions for Runge-Kutta methods.

The following theorem is quite useful:

Theorem 3.3. *Let*

$$y' = f(t, y), \qquad y(t_0) = y_0, \qquad t_0 \le t \le t_{end}$$

be solved by a onestep method

$$y_{n+1} = y_n + h\Phi(t_n, y_n; h), \tag{6}$$

with stepsize $h = (t_{end} - t_0)/Nstep$. *If*

1. *the increment function* Φ *is Lipschitz in* y, *and*
2. *the local truncation error* $d_{n+1} = \mathcal{O}(h^{p+1})$,

then the method is of order p, *that is, the global error at* t_{end} *satisfies*

$$e_{Nstep} = y(t_{end}) - y_{Nstep} = \mathcal{O}(h^p).$$

A RK method is a onestep method with increment function $\Phi(t_n, y_n; h) = \sum_{i=1}^{s} b_i k_i$. It is possible to show that Φ is Lipschitz in y whenever f is Lipschitz and $h \le h_{max}$, where h_{max} is some predefined maximal stepsize. What remains is the order of the local truncation error. To find it, we take the Taylor-expansions of the exact and the numerical solutions and compare. The local truncation error is $\mathcal{O}(h^{p+1})$ if the two series matches for all terms corresponding to h^q with $q \le p$. In principle, this is trivial. In practise, it becomes extremely tedious (give it a try). It is (somewhat surprisingly) possible to express the series in terms of graphs, called *rooted trees*. However, in this note, we restrict ourself to write down the order conditions up

to order 4.

Order	Condition
1	$\sum b_i = 1$
2	$\sum b_i c_i = 1/2$
3	$\sum b_i c_i^2 = 1/3$
	$\sum b_i a_{ij} c_j = 1/6$
4	$\sum b_i c_i^3 = 1/4$
	$\sum b_i c_i a_{ij} c_j = 1/8$
	$\sum b_i a_{ij} c_j^2 = 1/12$
	$\sum b_i a_{ij} a_{jk} c_k = 1/24$