Numerical solution of ordinary differential equations

Anne Kværnø

Nov 10, 2022

The relevant python code can be found in `numode.py`.

1 Introduction

We will here discuss the numerical solution of systems of ordinary differential equations (ODEs).

Scalar ODEs. A scalar ODE is an equation of the form

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0,$$

where $y'(t) = \frac{dy}{dt}$. The *initial condition* $y(t_0) = y_0$ is required for a unique solution.

NB! It is common to use the term *initial value problem (IVP)* for an ODE for which the initial value $y(t_0) = y_0$ is given, and we only are interested in the solution for $t > t_0$. In this note, only initial value problems are considered.

Example 1: Consider the solution of the ODE

$$y'(t) = -2ty(t),$$

which can be written in the form $y'(t) = f(t, y(t))$ with

$$f(t, y) = -2ty.$$

The general solution is the function

$$y(t) = Ce^{-t^2},$$

where C is a constant that depends on the initial condition $y(t_0)$, which we can find by using the techniques from Mathematics 1: We write first $y' = dy/dt$ and then multiply formally with dt to obtain

$$dy = -2ty\, dt.$$

If y is equal to 0, we see that the solution of the equation is the constant zero function. Else we can divide by y and obtain

$$\frac{1}{y}\, dy = -2t\, dt.$$

Now we compute indefinite integrals of both sides, which yields

$$\log|y| = -t^2 + K.$$

Taking the exponential on both sides, we get

$$|y| = e^K e^{-t^2},$$

or

$$y = \pm e^K e^{-t^2}.$$

Setting $C = \pm e^K$, we finally obtain the solution

$$y(t) = Ce^{-t^2}$$

for some $C \in \mathbb{R}$ (the case $C = 0$ corresponds to the constant zero solution we have discussed previously). In order to find the constant C, we have to use the initial condition for $y(t_0)$. For instance, we obtain for $t_0 = 0$ and $y(0) = 1$ the solution

$$y(t) = e^{-t^2}.$$

Systems of ODEs. A system of m ODEs is given by

$$\begin{aligned} y_1' &= f_1(t, y_1, y_2, \ldots, y_m), & y_1(t_0) &= y_{1,0} \\ y_2' &= f_2(t, y_1, y_2, \ldots, y_m), & y_2(t_0) &= y_{2,0} \\ &\vdots & &\vdots \\ y_m' &= f_m(t, y_1, y_2, \ldots, y_m), & y_m(t_0) &= y_{m,0} \end{aligned}$$

or, more compactly, by

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)), \qquad \mathbf{y}(t_0) = \mathbf{y}_0$$

where we use boldface to denote vectors in $\mathbb{R}^m$.

$$\mathbf{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_m(t) \end{pmatrix}, \qquad \mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} f_1(t, y_1, y_2, \ldots, y_m), \\ f_2(t, y_1, y_2, \ldots, y_m), \\ \vdots \\ f_m(t, y_1, y_2, \ldots, y_m), \end{pmatrix}, \qquad \mathbf{y}_0 = \begin{pmatrix} y_{1,0} \\ y_{2,0} \\ \vdots \\ y_{m,0} \end{pmatrix},$$

Example 2: The Lotka-Volterra equation is a system of two ODEs describing the interaction between predators and prey over time. The system is given as

$$\begin{aligned} y_1'(t) &= \alpha y_1(t) - \beta y_1(t) y_2(t), \\ y_2'(t) &= \delta y_1(t) y_2(t) - \gamma y_2(t). \end{aligned}$$

Here t denotes time, $y_1(t)$ describes the population of the prey species, and $y_2(t)$ the population of predators. The positive parameters α, β, δ, and γ depend on the populations to be modelled. We can write this system in the form $\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t))$ with

$$\mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} \alpha y_1 - \beta y_1 y_2 \\ \delta y_1 y_2 - \gamma y_2 \end{pmatrix}.$$

Note that the parameter t here actually does not appear on the right hand side.

Autonomous ODEs. An ODE is called *autonomous* if $\mathbf{f}$ is not a function of t, but only of $\mathbf{y}$. 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}. Thus, the ODE from Example 1, with initial values $y(0) = 1$ can be written as

$$\begin{aligned} y_1' &= -2 y_2 y_1, & y_1(0) &= 1, \\ y_2' &= 1, & y_2(0) &= 0. \end{aligned}$$

Higher order ODEs. An initial value problem for an ODE of order m is given by

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

Here $u^{(1)} = u'$ and $u^{(m+1)} = \frac{du^{(m)}}{dt}$ for $m > 0$.

Such equations can be written as a system of first order ODEs by the following trick:

Let

$$y_1(t) = u(t), \quad y_2(t) = u'(t), \quad y_3(t) = u^{(2)}(t), \quad \ldots \quad , y_m(t) = u^{(m-1)}(t)$$

such that

$$\begin{aligned} y_1' &= y_2, & y_1(t_0) &= u_0, \\ y_2' &= y_3, & y_2(t_0) &= u_0', \\ &\vdots & &\vdots \\ y_{m-1}' &= y_m, & y_{m-1}(t_0) &= u_0^{(m-2)}, \\ y_m' &= f(t, y_1, y_2, \cdots, y_{m-1}, y_m), & y_m(t_0) &= u_0^{(m-1)}. \end{aligned}$$

This is a normal system of first order ODEs. The solution u of the original m-th order ODE is just the first component y_1 of the solution of this system.

Example 3: Van der Pol's equation is a second order differential equation, given by

$$u^{(2)} = \mu(1 - u^2)u' - u, \qquad u(0) = u_0, \quad u'(0) = u_0',$$

where $\mu > 0$ is some constant. Common choices for initial values are $u_0 = 2$ and $u_0' = 0$.

It can be rewritten as a system of first order ODEs:

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= u_0, \\ y_2' &= \mu(1 - y_1^2)y_2 - y_1, & y_2(0) &= u_0'. \end{aligned}$$

2 Numerical methods for solving ODEs

In this note, we will discuss some techniques for the numerial solution of ordinary differential equations. For simplicity or presentation, we will develop and discuss these methods mostly based on scalar ODEs. The same methods, however, are equally applicable for systems of equations.

All the methods that we will discuss are so-called *one-step methods*. Given the ODE and the initial values (t_0, y_0), we choose some step size h and let $t_1 = t_0 + h$. Based on this information, we calculate an approximation y_1 to $y(t_1)$. Then, we repeat this process starting from (t_1, y_1) in order to calculate an approximation y_2 of $y(t_2)$, where $t_2 = t_1 + h$. This process is repeated until some final point, here called t_{end} is reached.

In one-step methods, the approximation y_{k+1} of $y(t_{k+1})$ does not depend on the values of y_{k-1}, y_{k-2}, $\ldots$, y_0. The main alternative to this type of methods are *multi-step methods*, where the approximation y_{k+1} of $y(t_{k+1})$ takes into account those values as well.

It should be emphasized that this strategy only will find approximations to the exact solution in some discrete points t_n, $n = 0, 1, \ldots$.

3 Numerical methods for solving ODEs

3.1 Euler's method

Let us start with the simplest example, Euler's method, known from Mathematics 1.

Given a (scalar) IVP

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0,$$

choose some step size h. The trick is as follows:

Perform a Taylor expansion (*Preliminaries*, section 4) of the exact (but unknown) solution $y(t_0 + h)$ around t_0:

$$y(t_0 + h) = y(t_0) + hy'(t_0) + \frac{1}{2}h^2 y''(t_0) + \cdots .$$

Assume the step size h to be small, such that the solution is dominated by the first two terms. In that case, these can be used as the numerical approximation in the next step, as the ODE implies that

$$y'(t_0) = f(t_0, y(t_0)) = f(t_0, y_0).$$

From this we obtain the approximation

$$y(t_0 + h) \approx y(t_0) + hy'(t_0) = y_0 + hf(t_0, y_0),$$

giving

$$y_1 = y_0 + hf(t_0, y_0).$$

This extends trivially to a system of equations, and to a general n, such that the algorithms becomes:

Euler's method.

- Given a function $\mathbf{f}(t, \mathbf{y})$ and an initial value $(t_0, \mathbf{y}_0)$.
- Choose a step size h.
- For $i = 0, 1, 2, \ldots$
- $\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n)$,
  $t_{n+1} = t_n + h$.

An implementation of Euler's method can be found in `numode.py`.

Numerical example 1: Test the implementation of Euler's method on the problem

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

which has the analytic solution

$$y(t) = e^{-t^2}.$$

Try with different step sizes, for instance $h = 0.1$, $h = 0.05$ and $h = 0.01$. In each case, compare the numerical solution with the exact one.

The following script solves the equation numerically, and plot the exact and the numerical solution. We can also make a plot of the error: See the function `ode_example1()` in `numode.py`.

Numerical exercise 1: Repeat the example on a logistic equation, given by

$$y' = y(1 - y), \qquad y(0) = y_0,$$

on the interval $[0, 10]$. Use $y_0 = 0.1$ as initial value. For comparision, the exact solution is

$$y(t) = \frac{1}{1 - (1 - \frac{1}{y_0})e^{-t}}.$$

Solve the equation numerically by using different step sizes h, and try different initial values.

Numerical example 2: Solve the Lotka-Volterra equation

$$\begin{aligned} y_1'(t) &= \alpha y_1(t) - \beta y_1(t) y_2(t), & y_1(0) &= y_{1,0}, \\ y_2'(t) &= \delta y_1(t) y_2(t) - \gamma y_2(t), & y_2(0) &= y_{2,0}. \end{aligned}$$

In this example, use the parameters and initial values

$$\alpha = 2, \quad \beta = 1, \quad \delta = 0.5, \quad \gamma = 1, \qquad y_{1,0} = 2, \quad y_{2,0} = 0.5.$$

Solve the equation over the interval $[0, 20]$, and use $h = 0.02$. Try also other step sizes, e.g. $h = 0.1$ and $h = 0.002$.

NB! In this case, the exact solution is not known. What is known is that the solutions are periodic and positive. Is this the case for the numerical solutions as well? Check for different values of h.

See the function `num_example2()` in `numode.py`.

3.2 Heun's method

We will now discuss a first, improved, alternative to Euler's method:

Assume that we want to solve an

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)).$$

Its exact solution can be written in integral form as

$$\mathbf{y}(t_n + h) = \mathbf{y}(t_n) + \int_{t_n}^{t_n+h} \mathbf{y}'(t)\, dt = \mathbf{y}(t_n) + \int_{t_n}^{t_n+h} \mathbf{f}(t, \mathbf{y}(t)) dt.$$

We now replace the integral on the right hand side by a numerical approximation using the trapezoidal rule:

$$\mathbf{y}(t_n + h) \approx \mathbf{y}(t_n) + \frac{h}{2}\big(\mathbf{f}(t_n, \mathbf{y}(t_n)) + \mathbf{f}(t_{n+1}, \mathbf{y}(t_{n+1}))\big).$$

Then we replace $\mathbf{y}(t_n)$ and $\mathbf{y}(t_{n+1})$ by the approximations $\mathbf{y}_n$ and $\mathbf{y}_{n+1}$. The resulting method is the *trapezoidal rule for ODEs*, given by

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{2}\big(\mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1})\big).$$

This is an example of an *implicit* method. The numerical approximation $\mathbf{y}_{n+1}$ appears on both sides of this equation and is therefore only implicitly given: If $t_n, \mathbf{y}_n$ is known, a nonlinear equation has to be solved in order to find $\mathbf{y}_{n+1}$, and this has to be done for each step. There are important applications where this actually makes sense, which we will discuss in a later lecture in the context of *stiff ODEs*.

However, for the moment we want to avoid this additional complication and thus replace the $\mathbf{y}_{n+1}$ on the right hand side by some approximation. One natural possibility here is to apply one step of Euler's method. This results in Heun's method:

$$\begin{aligned} \mathbf{u}_{n+1} &= \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n), \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{2}\big(\mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{f}(t_{n+1}, \mathbf{u}_{n+1})\big). \end{aligned}$$

The method is commonly written in the form

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{f}(t_n, \mathbf{y}_n), \\ \mathbf{k}_2 &= \mathbf{f}(t_n + h, \mathbf{y}_n + h\mathbf{k}_1), \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2). \end{aligned}$$

We notice that only a few lines have to be changed: see the implementation in `numode.py`.

Numerical example 3: Let us compare the numerical solution from Euler's and Heun's methods on the scalar test problem

$$y' = -2ty, \qquad y(0) = 1$$

with the exact solution $y(t) = e^{-t^2}$ on the interval $[0, 1]$. Use $h = 0.1$ for Euler's method and $h = 0.2$ for Heun's method.

The main computational work in these methods lies in the evaluations of the function $\mathbf{f}$, especially in the case of large systems of ODEs. Euler's method requires one function evaluation in each iteration, whereas Heun's method requires two function evaluations: one for the computation of $\mathbf{k}_1$ and a second one for the computation of $\mathbf{k}_2$. This means that a single iteration of Heun's method is roughly double as computationally expensive as one iteration of Euler's method. If we want to have a fair comparison between the two methods, we should thus use half as many iterations for Heun's method as for Euler's method, which in turn means that the step length h for Heun's method should be double that of Euler's method. In this example, using Euler's method with $h = 0.1$ and Heun's method with $h = 0.2$ both leads to 10 function evaluations for the solution on the interval $[0, 1]$, and thus to a comparable total amount of computational work.

See the function `ode_example3()` in `numode.py`.

From the results we see that Heun's method is significantly more accurate than Euler's method, even when the number of function evaluations are the same. Further, we notice that the error in Heun's method is reduced by a factor of approximately $1/4$ whenever the step size is reduced by a factor $1/2$, indicating that the error satisfies $|y(t_{\text{end}}) - y_N| \approx Ch^2$, and the method is of order 2. In the theory part below, we will prove that this certainly is the case.

Numerical exercises:

1. Solve the Lotka-Volterra equation from Numerical example 2 by Euler's and Heun's methods, again using twice as many steps for Euler's method as for Heun's method.
   - Use $h = 0.01$ for Euler's method and $h = 0.02$ for Heun's method.
   - Use $h = 0.1$ for Euler's method and $h = 0.2$ for Heun's method.
2. Solve Van der Pol's equation by use of Heun's method and with Euler's method, and compare. Experiment with different choices of the step size h. Use $\mu = 2$, and solve the equation on the interval [0,20]. Experiment with different values of μ.
3. Implement the classical Runge–Kutta method (also known as RK4) and verify numerically that the order of the method is 4. The method is given by

$$\begin{aligned}
\mathbf{k}_1 &= \mathbf{f}(t_n, \mathbf{y}_n) \\
\mathbf{k}_2 &= \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_1\right) \\
\mathbf{k}_3 &= \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_2\right) \\
\mathbf{k}_4 &= \mathbf{f}(t_n + h, \mathbf{y}_n + h\mathbf{k}_3) \\
\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4).
\end{aligned}$$

3.3 Runge–Kutta methods

Euler's and Heun's method are both examples of a large class of methods, called *Runge-Kutta methods*.

Definition: Runge–Kutta methods.

An s-stage Runge-Kutta method is given by

$$\mathbf{k}_i = \mathbf{f}\big(t_n + c_i h, \mathbf{y}_n + h\sum_{j=1}^{s} a_{ij}\mathbf{k}_j\big), \qquad i = 1, 2, \cdots, s,$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\sum_{i=1}^{s} b_i \mathbf{k}_i.$$

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

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & & & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

with

$$c_i = \sum_{j=1}^{s} a_{ij}, \quad i = 1, \cdots, s.$$

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

The Butcher-tableaux for some methods are:

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

Euler

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

Heun

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

trapezoidal rule

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

RK4

See the list of Runge–Kutta methods on wikipedia for more examples.

4 Theory

4.1 Existence and uniqueness results

Let us first state the conditions for which the ODE has a unique solution. We will need the following definition (which will also be used later in this note):

Definition: Lipschitz condition.

A function $\mathbf{f}\colon \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$ satisfies a *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

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

The constant L is called *the Lipschitz constant*.

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

Theorem: Existence and uniqueness of a solution.

Consider the initial value problem

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \qquad \mathbf{y}(t_0) = \mathbf{y}_0.$$

If

- $\mathbf{f}(t, \mathbf{y})$ is continuous in $(a, b) \times D$,
- $\mathbf{f}(t, \mathbf{y})$ satisfies the Lipschitz condition with respect to $\mathbf{y}$ in $(a, b) \times D$,

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

In the following, we will assume that these conditions are satisfied.

4.2 Error analysis

When an ODE is solved by Euler's method over some interval $[t_0, t_{\text{end}}]$, how will the error at t_{end} (or some arbitrary point) depend on the number of steps? Or more specifically, choose the number of steps N, let the step size be $h = (t_{\text{end}} - t_0)/N$, such that $t_{\text{end}} = t_N$. What can we say about the error $e_N = y(t_{\text{end}}) - y_N$?

The error analysis will be done on a scalar equation, but it also holds for systems of equations, as described in the end of the section.

Local and global errors. In this discussion we have to consider two kinds of errors:

- *Local truncation error* d_{n+1}: This is the error made in one single step, starting from $(t_n, y(t_n))$.
- *Global error* e_n: This is the difference between the exact and the numerical solution after n steps, $e_n = y(t_n) - y_n$.

In the following, we will see how to express the local truncation error, and we will see how the global and the local errors are related. We will use all this to find an upper bound for the global error at the end point $t_N = t_{\text{end}}$. The technique described here is quite standard for this type of error analysis.

Let us start with the local truncation error. The Taylor expansion of the exact solution of the ODE, starting from $(t_n, y(t_n))$ is given by

$$y(t_n + h) = y(t_n) + hy'(t_n) + \frac{1}{2}y''(\xi), \qquad t_0 < \xi < t_0 + h.$$

and one step with Euler's method, starting from $(t_n, y(t_n))$ is

$$y(t_n + h) = y(t_n) + hf(t_n, y(t_n)).$$

where $y'(t_n) = f(t_n, y(t_n))$. Thus the local truncation error is

$$d_{n+1} = y(t_n + h) - \big(y(t_n) + hy'(t_n)\big) = \frac{1}{2}h^2 y''(\xi), \qquad \xi \in (t_n, t_n + h).$$

We then have the following two equations:

$$\begin{aligned} y(t_n + h) &= y(t_n) + hf(t_n, y(t_n)) + d_{n+1}, && \text{the equation above} \\ y_{n+1} &= y_n + hf(t_n, y_n), && \text{Euler's method.} \end{aligned}$$

We subtract the second from the first, use that $e_n = y(t_n) - y_n$, and the mean value theorem for functions [1] This yields the expression

$$e_{n+1} = e_n + h\big(f(t_n, y(t_n)) - f(t_n, y_n)\big) + d_{n+1} = e_n + hf_y(t_n, \eta)e_n + d_{n+1},$$

[1] The mean value theorem: There is a ν between y and z such that $g(y) - g(z) = g'(\nu)(y - z)$. In our case, use $g(y) = f(t, y)$ for a fixed t.

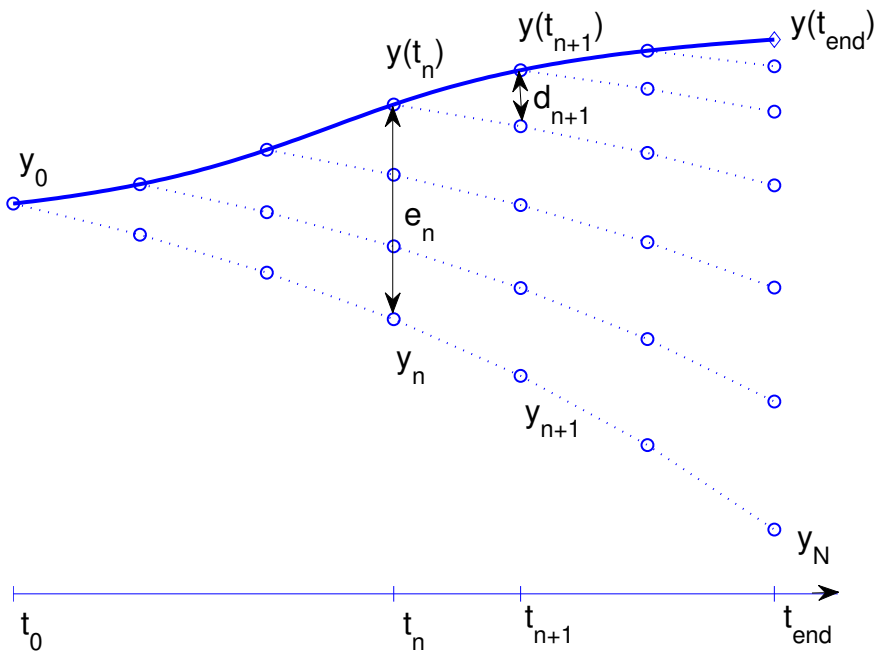

Figure 1: Lady Windermere's fan, named after a comedy play by Oscar Wilde. The figure describe the transport and the accumulation of the local truncation errors d_{n+1} into the global error at the end point, $e_N = y(t_{end}) - y_N$.

where $f_y = \frac{\partial f}{\partial y}$, and η is some value between y_n and $y(t_n)$. We now take the absolute value on each side, and apply the triangle inequality:

$$|e_{n+1}| = |e_n + hf_y(t_n, \eta)e_n + d_{n+1}| \leq |e_n| + h|f_y(t_n, \eta)||e_n| + |d_{n+1}|.$$

Assume now that there exist positive constants D and L satisfying

$$|f_y(t, y)| \leq L \qquad \text{and} \qquad |y''(t)| \leq 2D$$

for all values of t, y. Notice that L is a Lipschitz constant for f. From the inequality above we get that

$$|e_{n+1}| \leq (1 + hL)|e_n| + Dh^2.$$

Since $y_0 = y(t_0)$ we get $e_0 = 0$. The inequality above therefore results in the following estimates for the global errors:

$$\begin{aligned}
|e_1| &\leq Dh^2 \\
|e_2| &\leq (1 + hL)|e_1| + Dh^2 \leq \big((1 + hL) + 1\big)Dh^2 \\
|e_3| &\leq (1 + hL)|e_2| + Dh^2 \leq \big((1 + hL)^2 + (1 + hL) + 1\big)Dh^2 \\
&\vdots \\
|e_N| &\leq (1 + hL)|e_{N+1}| + Dh^2 \leq \sum_{n=0}^{N-1} (1 + hL)^n Dh^2
\end{aligned}$$

We will now apply two well known results:

- The sum of a truncated geometric series:

$$\sum_{n=0}^{N-1} r^n = \frac{r^N - 1}{r - 1} \text{ for } r \in \mathbb{R}.$$

- The series of the exponential:

$$e^x = 1 + x + \frac{1}{2}x^2 + \cdots = 1 + x + \sum_{n=2}^{\infty} \frac{x^n}{n!}$$

  which proves that $1 + x < e^x$ whenever $x > 0$.

Using these results, we obtain that

$$\sum_{n=0}^{N-1} (1 + hL)^n = \frac{(1 + hL)^N - 1}{(1 + hL) - 1} < \frac{(e^{hL})^N - 1}{hL} = \frac{e^{hLN} - 1}{hL} = \frac{e^{L(t_{\text{end}} - t_0)} - 1}{hL},$$

where the last equality holds because $(t_{\text{end}} - t_0) = hN$. Finally, we plug this into the inequality for $|e_N|$ above, and we see that we have proved the the following upper bound for the global:

$$|y(t_{\text{end}}) - y_N| = |e_N| \leq \frac{e^{L(t_{\text{end}} - t_0)} - 1}{L} Dh = Ch,$$

where the constant $C = \frac{e^{L(t_{\text{end}} - t_0)} - 1}{L} D$ depends on the length of the integration interval $t_{\text{end}} - t_0$, of certain properties of the equation (L and D), but *not* on the step size h.

The numerical solution converges to the exact solution since

$$\lim_{N \to \infty} |e_N| = 0.$$

If the step size is reduced by a factor of 0.5, so will the error. This is in agreement with the previous numerical result.

Remark: Following the proof of the error estimate for Euler's method, we see that a local truncation error of size h^2 leads to a final, global error of size h. Intuitively, this is because we need to take roughly $1/h$ steps in order to reach the point t_{end}, although a precise proof is quite a bit more complicated than that, as we have seen. However, what this also should mean is that a method with a truncation order of size h^{p+1} should lead to a global error of h^p. The results of the next section show that, under certain conditions, this is indeed the case.

4.3 A general convergence result

A one-step method applied to a system of ODEs $\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t))$ can be written in the generic form

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{\Phi}(t_n, \mathbf{y}_n; h),$$

where the *increment function* $\mathbf{\Phi}$ typically depends on the function $\mathbf{f}$ and some parameters defining the method.

For example, for Euler's method we have

$$\mathbf{\Phi}(t_n, \mathbf{y}_n; h) = \mathbf{f}(t_n, \mathbf{y}_n),$$

whereas for Heun's method

$$\mathbf{\Phi}(t_n, \mathbf{y}_n; h) = \frac{1}{2}\Big(\mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{f}\big(t_n + h, \mathbf{y}_n + hf(t_n, \mathbf{y}_n)\big)\Big).$$

All Runge–Kutta methods can in principle be written in this form as well (though in general for a rather complicated increment function $\mathbf{\Phi}$).

Definition: Order of a method.

A method is of order $p > 0$ if there is a constant $C > 0$ such that

$$\|\mathbf{e}_N\| = \|\mathbf{y}(t_{\text{end}}) - \mathbf{y}_N\| \leq Ch^p,$$

where N is the number of steps taken to reach t_{end} using the step size $h = (t_{\text{end}} - t_0)/N$.

The local truncation error $\mathbf{d}_{n+1}$ of such a method is

$$\mathbf{d}_{n+1} = \mathbf{y}(t_{n+1}) - (\mathbf{y}(t_n) + h\mathbf{\Phi}(t_n, \mathbf{y}(t_n); h))$$

Replace the absolute values in the above proof with norms (*Preliminaries*, section 1), and the argument above can be used to prove the following:

Theorem: Convergence of one-step methods.

Assume that there exist positive constants M and D such that the increment function satisfies

$$\|\mathbf{\Phi}(t, \mathbf{y}; h) - \mathbf{\Phi}(t, \mathbf{z}; h)\| \leq M\|\mathbf{y} - \mathbf{z}\| \tag{1}$$

and the local truncation error satisfies

$$\|\mathbf{y}(t+h) - (\mathbf{y}(t) + h\mathbf{\Phi}(t, \mathbf{y}(t), h))\| \leq Dh^{p+1}$$

for all t, $\mathbf{y}$ and $\mathbf{z}$ in a neighbourhood of the solution. In that case, the global error satisfies

$$\|\mathbf{e}_N\| = \|\mathbf{y}(t_{\text{end}}) - \mathbf{y}_N\| \leq Ch^p, \qquad \text{with } C = \frac{e^{M(t_{\text{end}}-t_0)} - 1}{M} D.$$

In the case where $\mathbf{\Phi}$ is a Runge–Kutta method, one can show that (1) is always satisfied for some constant M provided that the function $\mathbf{f}$ satisfies a Lipschitz condition.

Example: Let us consider Eulers method, $\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n)$. So in this case $\mathbf{\Phi}(t, \mathbf{y}) = \mathbf{f}(t_n, \mathbf{y}_n)$ and (1) is exactly the same as the Lipschitz condition for $\mathbf{f}$.

Order conditions for Runge–Kutta methods. It can be proved that a Runge–Kutta method is of order p with $p \leq 4$ if all the conditions up to and including p in the table below are satisfied:

p	conditions
1	$\sum_i b_i = 1$
2	$\sum_i b_i c_i = 1/2$
3	$\sum_i b_i c_i^2 = 1/3$
	$\sum_{i,j} b_i a_{ij} c_j = 1/6$
4	$\sum_i b_i c_i^3 = 1/4$
	$\sum_{i,j} b_i c_i a_{ij} c_j = 1/8$
	$\sum_{i,j} b_i a_{ij} c_j^2 = 1/12$
	$\sum_{i,j,k} b_i a_{ij} a_{jk} c_k = 1/24$

Here sums are taken over all the indices from 1 to s, with s being the number of stages of the method. It is possible to derive similar conditions also for higher orders p, but the conditions rapidly become more and more complex.

Recall also, that we have always assumed that Runge–Kutta methods satisfy the conditions $c_i = \sum_j a_{ij}$ for all $i = 1, \ldots, s$.

Example 6: Apply the conditions to Heun's method, for which $s = 2$ and the Butcher tableau is

$$\begin{array}{c|cc} c_1 & a_{11} & a_{12} \\ c_2 & a_{21} & a_{22} \\ \hline & b_1 & b_2 \end{array} = \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}.$$

The order conditions are:

$p = 1$	$b_1 + b_2 = \frac{1}{2} + \frac{1}{2} = 1$	OK
$p = 2$	$b_1 c_1 + b_2 c_2 = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 = \frac{1}{2}$	OK
$p = 3$	$b_1 c_1^2 + b_2 c_2^2 = \frac{1}{2} \cdot 0^2 + \frac{1}{2} \cdot 1^2 = \frac{1}{2} \neq \frac{1}{3}$	Not satisfied
	$b_1(a_{11}c_1 + a_{12}c_2) + b_2(a_{21}c_1 + a_{22}c_2) = \frac{1}{2}(0 \cdot 0 + 0 \cdot 1) + \frac{1}{2}(1 \cdot 0 + 0 \cdot 1)$	
	$= 0 \neq \frac{1}{6}$	Not satisfied

The method is of order 2.

Convergence properties of Heun's method (optional). This is included mostly to demonstrate that convergence analysis is not a trivial task.

To prove convergence and to find the order of a method two things are needed:

- the local truncation error, expressed as a power series in the step size h
- the condition $\|\mathbf{\Phi}(t, \mathbf{y}; h) - \mathbf{\Phi}(t, \mathbf{z}; h)\| \leq M\|\mathbf{y} - \mathbf{z}\|$

The local truncation error is found by comparing Taylor series expansions of the exact and the numerical solutions starting from the same point. In practice, this is not trivial. For simplicity, we will here do this only for a scalar equation $y'(t) = f(t, y(t))$. The result is valid for systems as well.

In the following, we will use the notation

$$f_t = \frac{\partial f}{\partial t}, \qquad f_y = \frac{\partial f}{\partial y}, \qquad f_{tt} = \frac{\partial^2 f}{\partial t^2} \qquad f_{ty} = \frac{\partial^2 f}{\partial t \partial y} \qquad \text{etc.}$$

Further, we will surpress the arguments of the function f and its derivatives. So f is to be understood as $f(t, y(t))$ although it is not explicitly written.

The Taylor expansion of the exact solution $y(t + h)$ is given by

$$y(t + h) = y(t) + hy'(t) + \frac{h^2}{2}y''(t) + \frac{h^3}{6}y'''(t) + \ldots.$$

Higher derivatives of $y(t)$ can be expressed in terms of the function f by using the chain rule and the product rule for differentiation:

$$\begin{aligned}
y'(t) &= f, \\
y''(t) &= f_t + f_y y' = f_t + f_y f, \\
y'''(t) &= f_{tt} + f_{ty}y' + f_{yt}f + f_{yy}y'f + f_y f_t + f_y f_y y' = f_{tt} + 2f_{ty}f + f_{yy}f^2 + f_y f_t + (f_y)^2 f.
\end{aligned}$$

We then find the series of the exact and the numerical solution around x_0, y_0 (any other point will do equally well). From the discussion above, the series for the exact solution becomes

$$y(t_0 + h) = y_0 + hf + \frac{h^2}{2}(f_t + f_y f) + \frac{h^3}{6}(f_{tt} + 2f_{ty}f + f_{yy}f^2 + f_y f_t + (f_y)^2 f) + \ldots,$$

where f and all its derivatives are evaluated in (t_0, y_0). For the numerical solution we get

$$\begin{aligned}
k_1 &= f(t_0, y_0) = f, \\
k_2 &= f(t_0 + h, y_0 + hk_1) \\
&= f + hf_t + f_y hk_1 + \frac{1}{2} f_{tt} h^2 + f_{ty} hhk_1 + \frac{1}{2} f_{yy} h^2 k_1^2 + \dots \\
&= f + h(f_t + f_y f) + \frac{h^2}{2}(f_{tt} + 2f_{ty} f + f_{yy} f^2) + \dots, \\
y_1 &= y_0 + \frac{h}{2}(k_1 + k_2) = y_0 + \frac{h}{2}\Big(f + f + h(f_t + f_y f) + \frac{h^2}{2}(f_{tt} + 2f_{ty} f + f_{yy} f^2) + \dots\Big) \\
&= y_0 + hf + \frac{h^2}{2}(f_t + f_y f) + \frac{h^3}{4}(f_{tt} + 2f_{ty} f + f_{yy} f^2) + \dots,
\end{aligned}$$

and the local truncation error will be

$$d_1 = y(t_0 + h) - y_1 = \frac{h^3}{12}(-f_{tt} - 2f_{ty} f - f_{yy} f^2 + 2f_y f_t + 2(f_y)^2 f) + \dots.$$

The first nonzero term in the local truncation error series is called *the principal error term*. For h sufficiently small this is the term dominating the error, and this fact will be used later.

Although the series has been developed around the initial point, series around $t_n, y(t_n)$ will give similar results, and it is possible to conclude that, given sufficient differentiability of f, there is a constant D such that

$$|d_n| \le Dh^3.$$

Further, we have to prove the condition on the increment function $\Phi(t, y)$. For f differentiable, there is for all y, z some ξ between t and y such that $f(t, y) - f(t, z) = f_y(t, \xi)(y - z)$. Let L be a constant such that $|f_y| \le L$, and for all t, y, z of interest we get

$$|f(t, y) - f(t, z)| \le L|y - z|.$$

The increment function for Heun's method is given by

$$\Phi(t, y) = \frac{1}{2}\big(f(t, y) + f(t + h, y + hf(t, y))\big).$$

By repeated use of the condition above and the triangle inequalitiy for absolute values we get

$$\begin{aligned}
|\Phi(t, y) - \Phi(t, z)| &= \frac{1}{2}|f(t, y) + f(t + h, y + hf(t, y)) - f(t, z) - hf(t + h, z + hf(t, z))| \\
&\le \frac{1}{2}\big(|f(t, y) - f(t, z)| + |f(t + h, y + hf(t, y)) - f(t + h, z + hf(t, z))|\big) \\
&\le \frac{1}{2}\big(L|y - z| + L|y + hf(t, y) - z - hf(t, z)|\big) \\
&\le \frac{1}{2}\big(2L|y - z| + hL^2|y - z|\big) \\
&= \Big(L + \frac{h}{2}L^2\Big)|y - z|.
\end{aligned}$$

Assuming that the step size h is bounded above by some H, we can conclude that

$$|\Phi(t, y) - \Phi(t, z)| \le M|y - z|, \qquad M = L + \frac{H}{2}L^2.$$

In conclusion: Heun's method is convergent of order 2.

5 Error estimates and stepsize selection

6 Stiff ODEs and linear stability analysis

7 A summary of some terms and definitions

There have been quite a few definitions and different error terms in this note. So let us list some of them (not exclusive):

Definitions:

$\mathbf{y}' = \mathbf{f}(x, \mathbf{y})$	the ODE
$\mathbf{y}(x\,;\, x^*, \mathbf{y}^*)$,	the exact solution of the ODE through $(x^*, \mathbf{y}^*)$
$\mathbf{y}(x) = \mathbf{y}(x\,;\, x_0, \mathbf{y}_0)$,	the exact solution of $\mathbf{y}' = \mathbf{f}(x, \mathbf{y})$, $\mathbf{y}(x_0) = \mathbf{y}_0$
$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{\Phi}(x_n, y_n; h)$,	one step of the method

Let $\mathbf{\Phi}$ represent a method of order p and $\hat{\mathbf{\Phi}}$ a method of order $p+1$.

Error concepts:

$\mathbf{d}_{n+1} = \mathbf{y}(x_n + h\,;\, x_n, \mathbf{y}(x_n)) - \Big(\mathbf{y}(x_n) + h\mathbf{\Phi}(x_n, \mathbf{y}(x_n); h)\Big)$,	the local truncation error	
$\mathbf{l}_{n+1} = \mathbf{y}(x_n + h\,;\, x_n, \mathbf{y}_n) - \Big(\mathbf{y}_n + h\mathbf{\Phi}(x_n, \mathbf{y}_n; h)\Big)$,	the local error	
$\mathbf{le}_{n+1} = h\Big(\hat{\mathbf{\Phi}}(x_n, \mathbf{y}_n; h) - \mathbf{\Phi}(x_n, \mathbf{y}_n; h)\Big)$,	the local error estimate,	$\mathbf{le}_{n+1} \approx \mathbf{l}_{n+1}$
$\mathbf{e}_n = \mathbf{y}(x_n) - \mathbf{y}_n$	the global error	