4.2 Error control and stepsize selection.

A user of some numerical black box software will usually require one thing: The accuracy of the numerical solution should be within some user specified tolerance. To accomplish this we have to measure the error, and if the error is too large, it has to be reduced. For ordinary differential equations, this means to reduce the stepsize. On the other hand, we would like our algorithm to be as efficient as possible, that is, to use large stepsizes. This leaves us with two problems: How to measure the error, and how to get the right balance between accuracy and efficiency.

Local error estimate. As demonstrated in Figure 1, the global error $y(t_n) - y_n$ comes from two sources: the local truncation error and the propagation of errors produced in preceding steps. This makes it difficult (but not impossible) to measure the global error. Fortunately it is surprisingly easy to measure the *local error*, l_{n+1}, the error produced in one step when starting at (t_n, y_n), see Figure 2. Let $y(t; t_n, y_n)$ be the exact solution of the ODE through the point t_n, y_n. For a method of order p we get

$$l_{n+1} = y(t_n + h; t_n, y_n) - y_{n+1} = \Psi(t_n, y_n)h^{p+1} + \mathcal{O}(h^{p+2}),$$

where $\mathcal{O}(h^{p+1})$ refer to higher order terms [1] . The term $\Psi(t_n, y_n)h^{p+1}$ is called *the principal error term*, and we assume that this term is the dominating part of the error. This assumption is true if the stepsize h is sufficiently small. Taking a step from the same point t_n, y_n with a method of order $\hat{p} = p + 1$ gives a solution $\hat{y}_{n+1}$ with a local error satisfying

$$y(t_n + h; t_n, y_n) - \hat{y}_{n+1} = \mathcal{O}(h^{p+2}).$$

The *local error estimate* is given by

$$le_{n+1} = \hat{y}_{n+1} - y_{n+1} = \Psi(t_n, y_n)h^{p+1} + \mathcal{O}h^{p+2} \approx l_{n+1}.$$

Embedded Runge-Kutta pair Given a Runge-Kutta method of order p. To be able to measure the local error, we need a method of order $p + 1$ (or higher). But we do not want to spend more work (in terms of f-evaluations) than necessary. The solution is *embedded*

[1] Strictly speaking, the Landau-symbol $\mathcal{O}$ is defined by

$$f(x) = \mathcal{O}(g(x)) \quad \text{for } x \to x_0 \qquad \text{if} \qquad \lim_{x \to x_0} \frac{\|f(x)\|}{\|g(x)\|} < K < \infty$$

for some unspecified constant K. Thus $f(h) = \mathcal{O}(h^q)$ means that $\|f(h)\| \leq Kh^q$ when $h \to 0$, and refer to the remainder terms of a truncated series.

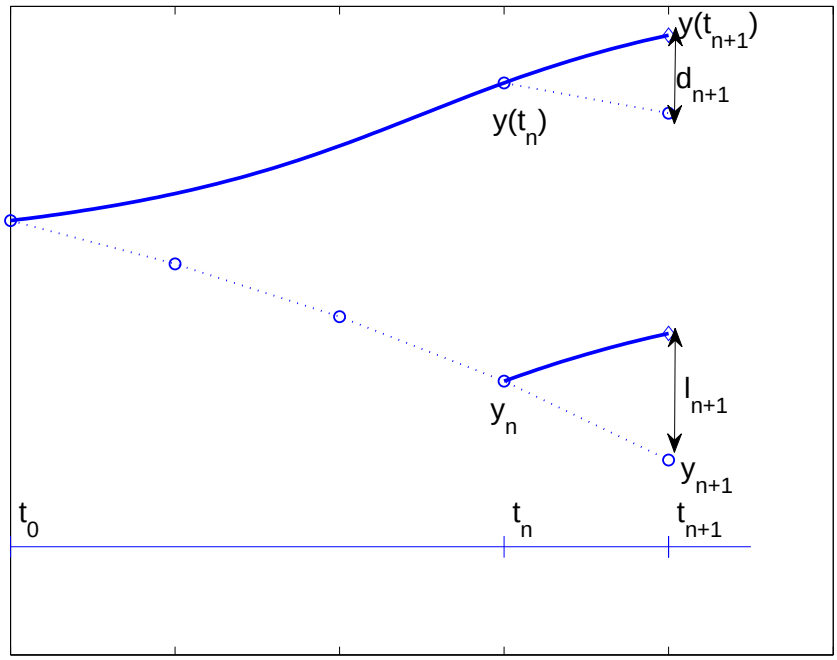


Figure 2: Lady Windermere's Fan

Runge-Kutta pairs, which, for explicit methods are given by

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} \\ \hline & b_1 & b_2 & \cdots & b_{s-1} \\ \hline & \hat{b}_1 & \hat{b}_2 & \cdots & \hat{b}_{s-1} & \hat{b}_s \end{array}$$

The method given by the b_i's is of order p, the error estimating method given by the $\hat{b}_i$'s is of order $p+1$. (Sometimes it is the other way round. The important thing is to have two methods of different order.) The local error estimate of y_{n+1} is then given by

$$le_{n+1} = \hat{y}_{n+1} - y_{n+1} = h\sum_{i=1}^{s}(\hat{b}_i - b_i)k_i.$$

Example 4.8. *A combination of the Euler method and improved Euler will result in the*

following pair

0		
1	1	
	1	
	$\frac{1}{2}$	$\frac{1}{2}$

so that

$$k_1 = f(t_n, y_n), \quad k_2 = f(t_n + h, y_n + hk_1), \quad y_{n+1} = y_n + hk_1, \quad l_{n+1} \approx le_{n+1} = \frac{h}{2}(-k_1 + k_2).$$

Example 4.9. *Assume that you have decided to use improved Euler, which is of order 2, as your advancing method, and you would like to find an error estimating method of order 3. There are no 2-stage order 3 ERKs, so you have to add one stage to your method. This gives a method like*

0			
1	1		
c_3	a_{31}	a_{32}	
	$\frac{1}{2}$	$\frac{1}{2}$	
	$\hat{b}_1$	$\hat{b}_2$	$\hat{b}_3$

where we require $c_3 = a_{31} + a_{32}$, which give us five free parameters. These have to satisfy all four order condition for an order 3 method. Using c_3 as a free parameter, we get the following class of 3th order methods:

$$b_1 = \frac{3c_3 - 1}{6c_3}, \quad b_2 = \frac{2 - 3c_3}{6(1 - c_3)}, \quad b_3 = \frac{1}{6c_3(1 - c_3)}, \quad a_{31} = c_3^2, \quad a_{31} = c_3 - c_3^2.$$

It is also possible to use the highest order method to advance the solution. In this case, we still measure the local error estimate of the lowest order order solution, but we get a more accurate numerical solution for free. This idea is called *local extrapolation*.

MATLAB has two integrators based on explicit Runge-Kutta schemes, `ODE23` which is based on an order 3/2 pair by Bogacki and Shampine, (a 3th order advancing and a 2nd order error estimating method), and `ODE45` based on an order 5/4 pair by Dormand and Prince. Both use local extrapolation.

Stepsize control Let the user specify a tolerance Tol, and a norm $\|\cdot\|$ in which the error is measured. Let us start with t_n, y_n, and do one step forward in time with a stepsize h_n, giving y_{n+1} and le_{n+1}. If $\|le_{n+1}\| \leq Tol$ the step is accepted, and we proceed till the next step, maybe with an increased stepsize. If $\|le_{n+1}\| > Tol$ the step is rejected and we try again with a smaller stepsize. In both cases, we would like to find a stepsize h_{new} which gives a local error estimate smaller than Tol, but at the same time as close to Tol as possible. To find the right stepsize, we make one assumption: The function $\Psi(t_n, y_n)$ of the principle error term do

not change much from one step to the next, thus $\|\Psi(t_n, y_n)\| \approx \|\Psi(t_{n+1}, y_{n+1})\| \approx C$. Then

$$\begin{aligned} \text{we have:} \qquad \|le_{n+1}\| &\approx C \cdot h_n^{p+1} \\ \text{we want:} \qquad Tol &\approx C \cdot h_{new}^{p+1} \end{aligned}$$

We get rid of the unknown C by dividing the two equations with each other, and h_{new} can be solved from

$$\frac{\|le_{n+1}\|}{Tol} \approx \left(\frac{h_n}{h_{new}}\right)^{p+1}.$$

Rejected steps are wasted work, and it should be avoided. Thus we choose the new stepsize somewhat conservative. The new stepsize is computed by

$$h_{new} = P \cdot \left(\frac{Tol}{\|le_{n+1}\|}\right)^{\frac{1}{p+1}} h_n. \tag{14}$$

where P is a *pessimist factor*, usually chosen somewhere in the interval [0.5,0.95]. In the discussion so far we have used the requirement $\|le_{n+1}\| \leq Tol$, that is *error pr. step* (EPS). This do not take into account the fact that the smaller the step is, the more steps you take, and the local errors from each step adds up. From this point of view, it would make sense to rather use the requirement $le\|_{n+1} \leq Tol \cdot h_n$, that is *error pr. unit step* (EPUS). The stepsize selection is then given by

$$h_{new} = P \cdot \left(\frac{Tol}{\|le_{n+1}\|}\right)^{\frac{1}{p}} h_n. \tag{15}$$

Careful analysis has proved that the local extrapolation together with EPS gives proportionality between the global error and the tolerance. The same is true for the use of the lower order method to advance the solution in combination with EPUS.

5 Stiff equations and linear stability

Example 5.1. *Given the ODE*

$$y' = -1000y, \qquad y(0) = 1.$$

with exact solution

$$y(t) = e^{-1000t}.$$

Thus $y(t) \to 0$ *as* $t \to \infty$. *The Euler method applied to this problem yields*

$$y_{n+1} = y_n - 1000hy_n = (1 - 1000h)y_n.$$

so that $y_n = (1 - 1000h)^n$. *This gives us two situations:*

$$\begin{aligned} &\textit{If} \quad |1 - 1000h| < 1 \quad \textit{then} \quad y_n \to 0 \ \textit{as} \ n \to \infty. \\ &\textit{If} \quad |1 - 1000h| > 1 \quad \textit{then} \quad |y_n| \to \infty \ \textit{as} \ n \to \infty \end{aligned}$$

Clearly, the second situation do not make sense at all, as the numerical solution is unstable even if the exact solution is stable. We have to choose a stepsize $h < 0.002$ *to get a stable numerical solution for this problem.*

To be more general: Consider a linear ODE

$$y' = My, \qquad y(0) = y_0, \tag{16}$$

where M is a constant, $m \times m$ matrix. We assume that M is diagonalizable, that is

$$V^{-1}MV = \Lambda$$

where

$$\Lambda = diag\{\lambda_1, \lambda_2, \cdots, \lambda_m\}, \qquad V = [v_1, v_2, \cdots, v_m],$$

where λ_i, $i = 1, \cdots, m$ are the eigenvalues of M and v_i are the corresponding eigenvectors. By premultiplying (16) with V^{-1}, we get

$$V^{-1}y' = V^{-1}MVV^{-1}y, \qquad V^{-1}y(t_0) = V^{-1}y_0$$

or, using $u = V^{-1}y$,

$$u' = \Lambda u, \qquad u(t_0) = V^{-1}y_0 = u_0.$$

The system is now decoupled, and can be written componentwise as

$$u_i' = \lambda_i u_i, \qquad u_i(0) = u_{i,0}, \qquad \lambda_i \in \mathbb{C}, \qquad i = 1, \cdots, m. \tag{17}$$

We have to accept the possibility of complex eigenvalues, however, as M is a real matrix, then complex eigenvalues appears in complex conjugate pairs. In the following, we will consider the situation when

$$\mathrm{Re}(\lambda_i) < 0 \ \text{ for } \ i = 1, \cdots, m, \qquad \text{thus} \qquad y(t) \to 0 \text{ as } t \to \infty. \tag{18}$$

Apply the Euler method to (16):

$$y_{n+1} = y_n + hMy_n.$$

We can do exactly the same linear transformations as above, so the system can be rewritten as

$$u_{i,n+1} = (1 + h\lambda_i)u_{i,n}, \qquad i = 1, \cdots, m.$$

For the numerical solution to be stable, we have to require

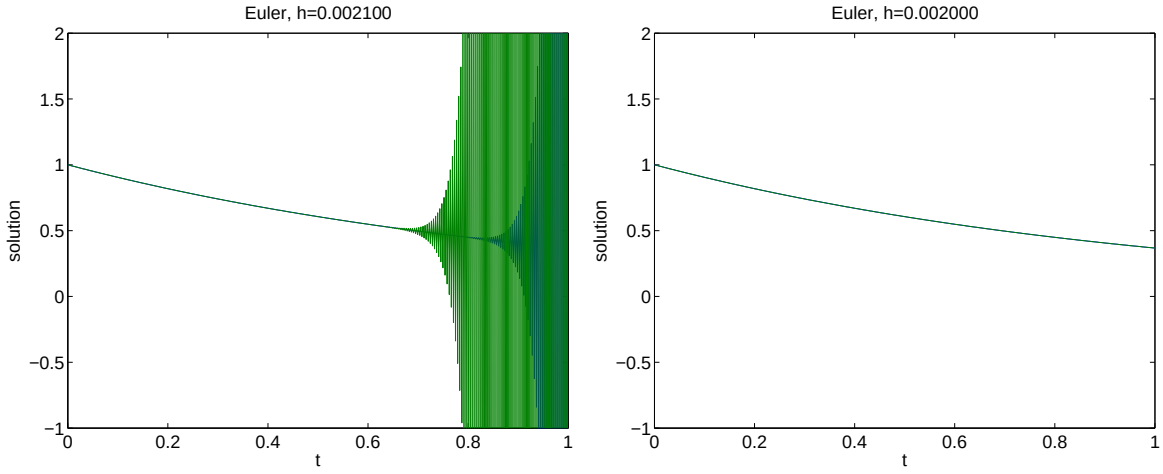
$$|1 + h\lambda_i| \leq 1, \qquad \text{for all the eigenvalues } \lambda_i. \tag{19}$$

(The case $|1+h\lambda_i h| = 1$ is included, as this is sufficient to prevent the solution from growing.)

Example 5.2. *Given*

$$y' = \begin{bmatrix} -2 & 1 \\ 998 & -999 \end{bmatrix} y, \qquad y(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

with exact solution $y_1(t) = y_2(t) = e^{-t}$. *The matrix has eigenvalues* -1 *and* -1000. *The initial values are chosen so that the fast decaying mode is missing in the exact solution. This problem is solved by Eulers method, with two almost equal stepsizes,* $h = 0.0021$ *and* $h = 0.002$. *The difference is striking, but completely in correspondence with* (19) *and the result of Example 5.1.*



The MATLAB-file `linsys` used to produce these plots are given on the web-page. Use it to do your own experiments.

Example 5.2 is a typical example of a stiff equation. The stepsize is restricted by a fast decaying component.

Example 5.3. *Let*

$$M = \begin{bmatrix} -2 & -2 \\ 1 & 0 \end{bmatrix} \qquad \textit{with eigenvalues} \qquad \lambda_{1,2} = -1 \pm i.$$

The requirement (19) *becomes*

$$|1 + h(-1 \pm i)| \leq 1 \quad \textit{or} \quad (1-h)^2 + h^2 \leq 1 \quad \textit{which is satisfied if and only if} \quad 0 \leq h \leq 1.$$

Stiffness occurs in situations with fast decaying solutions (transients) in combination with slow solutions. If you solve an ODE by an adaptive explicit scheme, and the stepsize becomes unreasonable small, stiffness is the most likely explanation. If the stepsizes in additions seems to be independent of your choice of tolerances, then you can be quite sure. The stepsize is restricted by stability related to the transients, and not by accuracy. The effect is demonstrated in the MATLAB-file `teststiffness` available on the web page. The *backward Euler* method is one way to overcome this problem:

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \tag{20}$$

or, applied to the problem of (17)

$$u_{i,n+1} = u_{i,n} + h\lambda u_{i,n+1}, \qquad \Rightarrow \qquad u_{i,n+1} = \frac{1}{1-h\lambda_i} u_{i,n+1}.$$

Since $|1/(1-h\lambda_i)| \leq 1$ whenever $\mathrm{Re}(\lambda_i) \leq 0$ there is no stepsize restriction caused by stability issues. In fact, $u_{i,n+1} \to 0$ as $\mathrm{Re}(h\lambda_i) \to -\infty$, so fast transients decay quickly, as they are supposed to do. But this nice behaviour is not for free: for a nonlinear ODE a nonlinear system of equations has to be solved for each step. We will return to this topic later.

Linear stability theory for Runge-Kutta methods.

Given the linear test equation

$$y' = \lambda y, \qquad \lambda \in \mathbb{C}. \tag{21}$$

Thus $\lambda = \alpha + i\beta$. The solution can be expressed by

$$y(t_n + h) = e^{\alpha h} e^{i\beta h} y(t_n).$$

Clearly, the solution is stable if $\alpha \leq 0$, that is $\lambda \in \mathbb{C}^-$. For the numerical solution we then require the stepsize h to be chosen so that

$$|y_{n+1}| \leq |y_n| \quad \text{whenever} \lambda \in \mathbb{C}^- \tag{22}$$

When a RK method is applied (21), we simply get

$$y_{n+1} = R(z) y_n, \qquad z = h\lambda$$

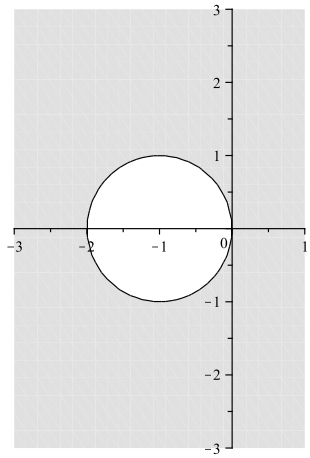
where R is a polynomial or a rational function. R is called *the stability function* of the RK method. The numerical solution is stable if $|R(z)| \leq 1$, otherwise it is unstable. This motivates the following definition of *the region of absolute stability* as

$$\mathcal{D} = \{z \in \mathbb{C} \ : \ |R(z)| \leq 1\}.$$

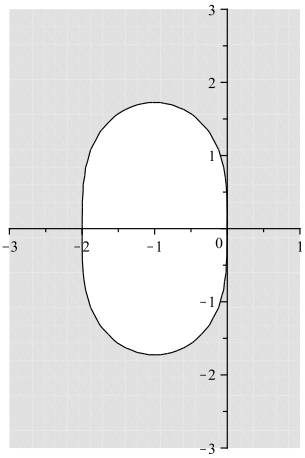
The condition (22) is satisfied for all $h > 0$ if

$$\mathbb{C}^- \in \mathcal{D},$$

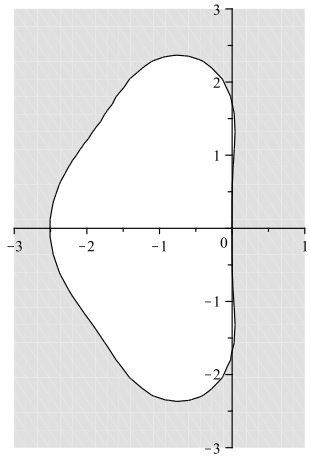
Methods satisfying this condition are called *A-stable*. The Backward Euler method (20) is an example of an A-stable method.



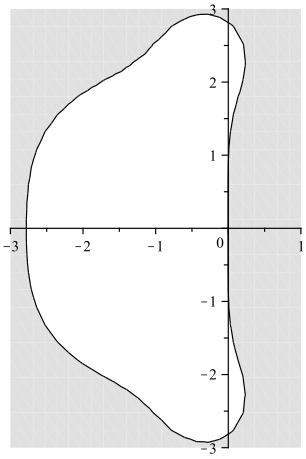
$p = s = 1$



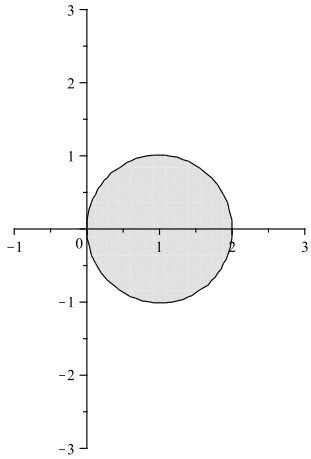
$p = s = 2$



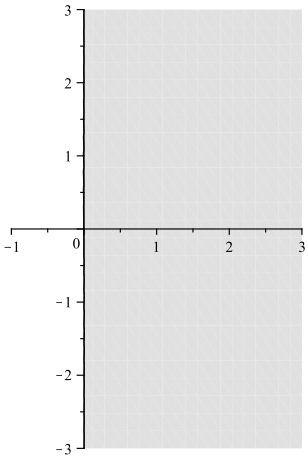
$p = s = 3$



$p = s = 4$



Backward Euler



Trapezoidal rule

Figure 3: Stability regions in $\mathbb{C}^-$: The first four are the stability regions for explicit RK methods of order $p = s$. The white regions are stable, the grey unstable.

Example 5.4. *A 2-stage ERK applied to* (21) *is given by:*

$$k_1 = \lambda y_n, \qquad k_2 = \lambda(y_n + ha_{21}\lambda y_n), \qquad y_{n+1} = y_n + h\lambda(b_1 + b_2)y_n + (h\lambda)^2 b_2 a_{21} y_n$$

If this method is of order 2, then $b_1 + b_2 = 1$ *and* $b_2 a_{21} = 1/2$*, so that*

$$R(z) = 1 + z + \frac{1}{2}z^2.$$

The stability function of an s-stage ERKs is a polynomial of degree s. As a consequence, no ERKs can be A-stable! If the order of the method is s, then

$$R(z) = \sum_{i=0}^{s} \frac{z^i}{i!}.$$

See Figure 3 for plots of the stability regions. But it has been proved that ERK with $p = s$ only exist for $s \leq 4$. To get an order 5 ERK, 6 stages are needed.

Example 5.5. *The trapezoidal rule (see section 3.1) applied to* (21) *gives*

$$y_{n+1} = y_n + \frac{h}{2}(\lambda y_n + \lambda y_{n+1}) \qquad \Rightarrow \qquad R(z) = \frac{1+z}{1-z}.$$

In this case $\mathcal{D} = \mathbb{C}^-$*, which is perfect.*

To summarise:

- For a given $\lambda \in \mathbb{C}^-$, choose a stepsize h so that $h\lambda \in \mathcal{D}$.
- If your problem is stiff, use an A-stable method.
- There are no A-stable explicit methods.