II.4 Practical Error Estimation and Step Size Selection

> Ich glaube indessen, dass ein practischer Rechner sich meistens mit der geringeren Sicherheit begnügen wird, die er aus der Uebereinstimmung seiner Resultate für grössere und kleinere Schritte gewinnt.
> (C. Runge 1895)

Even the simplified error estimates of Section II.3, which are content with the leading error term, are of little practical interest, because they require the computation and majorization of several partial derivatives of high orders. But the main advantage of Runge-Kutta methods, compared with Taylor series, is precisely that the computation of derivatives should be no longer necessary. However, since practical error estimates are necessary (on the one hand to ensure that the step sizes h_i are chosen sufficiently small to yield the required precision of the computed results, and on the other hand to ensure that the step sizes are sufficiently large to avoid unnecessary computational work), we shall now discuss alternative methods for error estimates.

The oldest device, used by Runge in his numerical examples, is to repeat the computations with *halved* step sizes and to compare the results: those digits which haven't changed are assumed to be correct ("... woraus ich schliessen zu dürfen glaube ...").

Richardson Extrapolation

> ... its usefulness for practical computations can hardly be overestimated.
> (G. Birkhoff & G.C. Rota)

The idea of Richardson, announced in his classical paper Richardson (1910) which treats mainly partial differential equations, and explained in full detail in Richardson (1927), is to use more carefully the known behaviour of the error as a function of h.

Suppose that, with a given initial value (x_0, y_0) and step size h, we compute *two* steps, using a fixed Runge-Kutta method of order p, and obtain the numerical results y_1 and y_2. We then compute, starting from (x_0, y_0), *one big step* with step size $2h$ to obtain the solution w. The error of y_1 is known to be (Theorem 3.2)

$$e_1 = y(x_0+h) - y_1 = C \cdot h^{p+1} + \mathcal{O}(h^{p+2}) \tag{4.1}$$

where C contains the error coefficients of the method and the elementary differentials $F^J(t)(y_0)$ of order $p+1$. The error of y_2 is composed of two parts: the

transported error of the first step, which is

$$\Big(I + h\frac{\partial f}{\partial y} + \mathcal{O}(h^2)\Big)e_1,$$

and the local error of the second step, which is the same as (4.1), but with the elementary differentials evaluated at $y_1 = y_0 + \mathcal{O}(h)$. Thus we obtain

$$\begin{aligned} e_2 = y(x_0+2h) - y_2 &= \big(I + \mathcal{O}(h)\big)Ch^{p+1} + \big(C + \mathcal{O}(h)\big)h^{p+1} + \mathcal{O}(h^{p+2}) \\ &= 2Ch^{p+1} + \mathcal{O}(h^{p+2}). \end{aligned} \tag{4.2}$$

Similarly to (4.1), we have for the big step

$$y(x_0+2h) - w = C(2h)^{p+1} + \mathcal{O}(h^{p+2}). \tag{4.3}$$

Neglecting the terms $\mathcal{O}(h^{p+2})$, formulas (4.2) and (4.3) allow us to eliminate the unknown constant C and to "extrapolate" a better value $\widehat{y}_2$ for $y(x_0+2h)$, for which we obtain:

Theorem 4.1. *Suppose that y_2 is the numerical result of two steps with step size h of a Runge-Kutta method of order p, and w is the result of one big step with step size $2h$. Then the error of y_2 can be extrapolated as*

$$y(x_0+2h) - y_2 = \frac{y_2 - w}{2^p - 1} + \mathcal{O}(h^{p+2}) \tag{4.4}$$

and

$$\widehat{y}_2 = y_2 + \frac{y_2 - w}{2^p - 1} \tag{4.5}$$

is an approximation of order $p+1$ to $y(x_0+2h)$. □

Formula (4.4) is a very simple device to estimate the error of y_2 and formula (4.5) allows one to increase the precision by one additional order ("... The better theory of the following sections is complicated, and tends thereby to suggest that the practice may also be complicated; whereas it is really simple." Richardson).

Embedded Runge-Kutta Formulas

> Scraton is right in his criticism of Merson's process, although Merson did not claim as much for his process as some people expect.
> (R. England 1969)

The idea is, rather than using Richardson extrapolation, to construct Runge-Kutta formulas which themselves contain, besides the numerical approximation y_1, a second approximation $\widehat{y}_1$. The difference then yields an estimate of the local error for the less precise result and can be used for step size control (see below). Since it is at our disposal at every step, this gives more flexibility to the code and makes step rejections less expensive.

We consider two Runge-Kutta methods (one for y_1 and one for $\widehat{y}_1$) such that both use the *same* function values. We thus have to find a scheme of coefficients (see (1.8')),

$$\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{32} & a_{32} & & & \\
\vdots & \vdots & \ddots & & & \\
c_s & a_{s1} & a_{s2} & \dots & a_{s,s-1} & \\
\hline
 & b_1 & b_2 & \dots & b_{s-1} & b_s \\
\hline
 & \widehat{b}_1 & \widehat{b}_2 & \dots & \widehat{b}_{s-1} & \widehat{b}_s
\end{array} \tag{4.6}$$

such that

$$y_1 = y_0 + h(b_1 k_1 + \ldots + b_s k_s) \tag{4.7}$$

is of order p, and

$$\widehat{y}_1 = y_0 + h(\widehat{b}_1 k_1 + \ldots + \widehat{b}_s k_s) \tag{4.7'}$$

is of order $\widehat{p}$ (usually $\widehat{p} = p-1$ or $\widehat{p} = p+1$). The approximation y_1 is used to continue the integration.

From Theorem 2.13, we have to satisfy the conditions

$$\sum_{j=1}^{s} b_j \Phi_j(t) = \frac{1}{\gamma(t)} \qquad \text{for all trees of order} \le p\,, \tag{4.8}$$

$$\sum_{j=1}^{s} \widehat{b}_j \Phi_j(t) = \frac{1}{\gamma(t)} \qquad \text{for all trees of order} \le \widehat{p}\,. \tag{4.8'}$$

The first methods of this type were proposed by Merson (1957), Ceschino (1962), and Zonneveld (1963). Those of Merson and Zonneveld are given in Tables 4.1 and 4.2. Here, "name $p(\widehat{p})$" means that the order of y_1 is p and the order of the error estimator $\widehat{y}_1$ is $\widehat{p}$. Merson's $\widehat{y}_1$ is of order 5 only for *linear* equations with constant coefficients; for nonlinear problems it is of order 3. This method works quite well and has been used very often, especially by NAG users. Further embedded methods were then derived by Sarafyan (1966), England (1969), and Fehlberg (1964, 1968, 1969). Let us start with the construction of some low order embedded methods.

Methods of order 3(2). It is a simple task to construct embedded formulas of order 3(2) with $s = 3$ stages. Just take a 3-stage method of order 3 (Exercise II.1.4) and put $\widehat{b}_3 = 0$, $\widehat{b}_2 = 1/2c_2$, $\widehat{b}_1 = 1 - 1/2c_2$.

Table 4.1. Merson 4("5")

0					
$\frac{1}{3}$	$\frac{1}{3}$				
$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{6}$			
$\frac{1}{2}$	$\frac{1}{8}$	0	$\frac{3}{8}$		
1	$\frac{1}{2}$	0	$-\frac{3}{2}$	2	
y_1	$\frac{1}{6}$	0	0	$\frac{2}{3}$	$\frac{1}{6}$
$\widehat{y}_1$	$\frac{1}{10}$	0	$\frac{3}{10}$	$\frac{2}{5}$	$\frac{1}{5}$

Table 4.2. Zonneveld 4(3)

0					
$\frac{1}{2}$	$\frac{1}{2}$				
$\frac{1}{2}$	0	$\frac{1}{2}$			
1	0	0	1		
$\frac{3}{4}$	$\frac{5}{32}$	$\frac{7}{32}$	$\frac{13}{32}$	$-\frac{1}{32}$	
y_1	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$	
$\widehat{y}_1$	$-\frac{1}{2}$	$\frac{7}{3}$	$\frac{7}{3}$	$\frac{13}{6}$	$-\frac{16}{3}$

Methods of order 4(3). With $s = 4$ it is impossible to find a pair of order 4(3) (see Exercise 2). The idea is to add y_1 as 5th stage of the process (i.e., $a_{5i} = b_i$ for $i = 1, \dots, 4$) and to search for a third order method which uses all five function values. Whenever the step is accepted this represents no extra work, because $f(x_0 + h, y_1)$ has to be computed anyway for the following step. This idea is called FSAL (First Same As Last). Then the order conditions (4.8') with $\widehat{p} = 3$ represent 4 linear equations for the five unknowns $\widehat{b}_1, \dots, \widehat{b}_5$. One can arbitrarily fix $\widehat{b}_5 \neq 0$ and solve the system for the remaining parameters. With $\widehat{b}_5$ chosen such that $\widehat{b}_4 = 0$ the result is

$$\begin{aligned}
&\widehat{b}_1 = 2b_1 - 1/6, \qquad &&\widehat{b}_2 = 2(1-c_2)b_2, && \\
&\widehat{b}_3 = 2(1-c_3)b_3, \qquad &&\widehat{b}_4 = 0, \qquad &&\widehat{b}_5 = 1/6.
\end{aligned} \tag{4.9}$$

Automatic Step Size Control

> D'ordinaire, on se contente de multiplier ou de diviser par 2 la valeur du pas ...
> (Ceschino 1961)

We now want to write a code which automatically adjusts the step size in order to achieve a prescribed tolerance of the local error.

Whenever a starting step size h has been chosen, the program computes two approximations to the solution, y_1 and $\widehat{y}_1$. Then an estimate of the error for the less precise result is $y_1 - \widehat{y}_1$. We want this error to satisfy componentwise

$$|y_{1i} - \widehat{y}_{1i}| \le sc_i, \qquad sc_i = Atol_i + \max(|y_{0i}|, |y_{1i}|) \cdot Rtol_i \tag{4.10}$$

where $Atol_i$ and $Rtol_i$ are the desired tolerances prescribed by the user (relative errors are considered for $Atol_i = 0$, absolute errors for $Rtol_i = 0$; usually both tolerances are different from zero; they may depend on the component of the solution). As a measure of the error we take

$$err = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\Big(\frac{y_{1i}-\widehat{y}_{1i}}{sc_i}\Big)^2}; \qquad (4.11)$$

other norms, such as the max norm, are also of frequent use. Then err is compared to 1 in order to find an optimal step size. From the error behaviour $err \approx C\cdot h^{q+1}$ and from $1 \approx C\cdot h_{\rm opt}^{q+1}$ (where $q = \min(p,\widehat{p})$) the optimal step size is obtained as ("... le procédé connu", Ceschino 1961)

$$h_{\rm opt} = h\cdot(1/err)^{1/(q+1)}. \qquad (4.12)$$

Some care is now necessary for a good code: we multiply (4.12) by a safety factor *fac*, usually $fac = 0.8,\ 0.9,\ (0.25)^{1/(q+1)}$, or $(0.38)^{1/(q+1)}$, so that the error will be acceptable the next time with high probability. Further, h is not allowed to increase nor to decrease too fast. For example, we may put

$$h_{\rm new} = h\cdot\min\big(facmax, \max\big(facmin, fac\cdot(1/err)^{1/(q+1)}\big)\big) \qquad (4.13)$$

for the new step size. Then, if $err \le 1$, the computed step is *accepted* and the solution is advanced with y_1 and a new step is tried with $h_{\rm new}$ as step size. Else, the step is *rejected* and the computations are repeated with the new step size $h_{\rm new}$. The maximal step size increase *facmax*, usually chosen between 1.5 and 5, prevents the code from too large step increases and contributes to its safety. It is clear that, when chosen too small, it may also unnecessarily increase the computational work. It is also advisable to put $facmax = 1$ in the steps right after a step-rejection (Shampine & Watts 1979).

Whenever y_1 is of lower order than $\widehat{y}_1$, then the difference $y_1 - \widehat{y}_1$ is (at least asymptotically) an estimate of the local error and the above algorithm keeps this estimate below the given tolerance. But isn't it more natural to continue the integration with the higher order approximation? Then the concept of "error estimation" is abandoned and the difference $y_1 - \widehat{y}_1$ is only used for the purpose of step size selection. This is justified by the fact that, due to unknown stability and instability properties of the differential system, the local errors have in general very little in common with the global errors. The procedure of continuing the integration with the higher order result is called "local extrapolation".

A modification of the above procedure (PI step size control), which is particularly interesting when applied to mildly stiff problems, is described in Section IV.2 (Volume II).

Starting Step Size

> If anything has been made foolproof, a better fool will be developed. (Heard from Dr. Pirkl, Baden)

For many years, the starting step size had to be supplied to a code. Users were assumed to have a rough idea of a good step size from mathematical knowledge or previous experience. Anyhow, a bad starting choice for h was quickly repaired by the step size control. Nevertheless, when this happens too often and when the choices are too bad, much computing time can be wasted. Therefore, several people (e.g., Watts 1983, Hindmarsh 1980) developed ideas to let the computer do this choice. We take up an idea of Gladwell, Shampine & Brankin (1987) which is based on the hypothesis that

$$\text{local error} \approx Ch^{p+1}y^{(p+1)}(x_0).$$

Since $y^{(p+1)}(x_0)$ is unknown we shall replace it by approximations of the first and second derivative of the solution. The resulting algorithm is the following one:

a) Do one function evaluation $f(x_0,y_0)$ at the initial point. It is in any case needed for the first RK step. Then put $d_0 = \|y_0\|$ and $d_1 = \|f(x_0,y_0)\|$, where the norm is that of (4.11) with $sc_i = Atol_i + |y_{0i}|\cdot Rtol_i$.

b) As a first guess for the step size let

$$h_0 = 0.01\cdot(d_0/d_1)$$

so that the increment of an explicit Euler step is small compared to the size of the initial value. If either d_0 or d_1 is smaller than 10^{-5} we put $h_0 = 10^{-6}$.

c) Perform one explicit Euler step, $y_1 = y_0 + h_0 f(x_0,y_0)$, and compute $f(x_0+h_0,y_1)$.

d) Compute $d_2 = \|f(x_0+h_0,y_1) - f(x_0,y_0)\|/h_0$ as an estimate of the second derivative of the solution; the norm being the same as in (a).

e) Compute a step size h_1 from the relation

$$h_1^{p+1}\cdot\max(d_1,d_2) = 0.01.$$

If $\max(d_1,d_2) \le 10^{-15}$ we put $h_1 = \max(10^{-6}, h_0\cdot 10^{-3})$.

f) Finally we propose as starting step size

$$h = \min(100\cdot h_0, h_1). \qquad (4.14)$$

An algorithm like the one above, or a similar one, usually gives a good guess for the initial step size (or at least avoids a very bad choice). Sometimes, more information about h is known, e.g., from previous experience or computations of similar problems.

Numerical Experiments

As a representative of 4-stage 4th order methods we consider the "3/8 Rule" of Table 1.2. We equipped it with the embedded formula (4.9) of order 3.

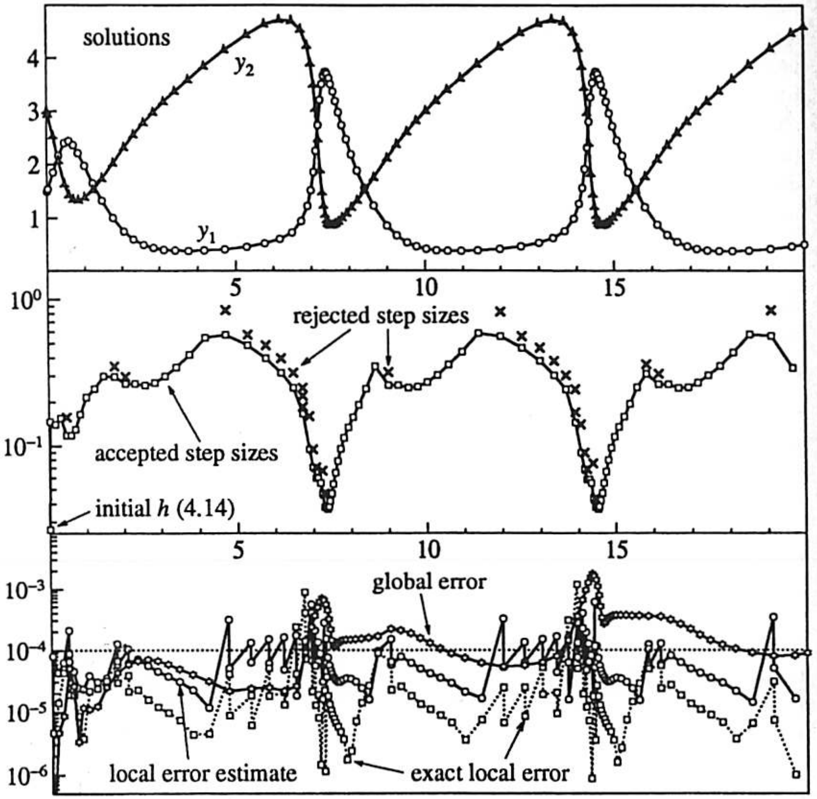

Fig. 4.1. Step size control, $Rtol = Atol = 10^{-4}$, 96 steps + 32 rejected

Step control mechanism. Fig. 4.1 presents the results of the step control mechanism (4.13) described above. As an example we choose the Brusselator (see Section I.16).

$$\begin{aligned} y_1' &= 1 + y_1^2 y_2 - 4y_1 \\ y_2' &= 3y_1 - y_1^2 y_2 \end{aligned} \tag{4.15}$$

with initial values $y_1(0) = 1.5$, $y_2(0) = 3$, integration interval $0 \le x \le 20$ and $Atol = Rtol = 10^{-4}$. The following results are plotted in this figure:

i) At the top, the solutions $y_1(x)$ and $y_2(x)$ with all accepted integration steps;

ii) then all step sizes used; the accepted ones are connected by a polygon; the rejected ones are indicated by ×;

iii) the third graph shows the local error estimate *err*, the exact local error and the global error; the desired tolerance is indicated by a broken horizontal line.

It can be seen that, due to the instabilities of the solutions with respect to the initial values, quite large global errors occur during the integration with small local tolerances everywhere. Further many step rejections can be observed in regions where the step size has to be decreased. This cannot easily be prevented, because right after an accepted step, the step size proposed by formula (4.13) is (apart from the safety factor) always increasing.

Numerical comparison. We are now curious to see the behaviour of the variable step size code, when compared to a fixed step size implementation. We applied both implementations to the Brusselator problem (4.15) with the initial values used there. The tolerances ($Atol = Rtol$) are chosen between 10^{-2} and 10^{-10} with ratio $\sqrt[3]{10}$. The results are then plotted in Fig. 4.2. There, the abscissa is the global error at the endpoint of integration (the "precision"), and the ordinate is the number of function evaluations (the "work"). We observe that for this problem the variable step size code is about twice as fast as the fixed step size code. There are, of course, problems (such as equation (0.1)) where variable step sizes are *much* more important than here.

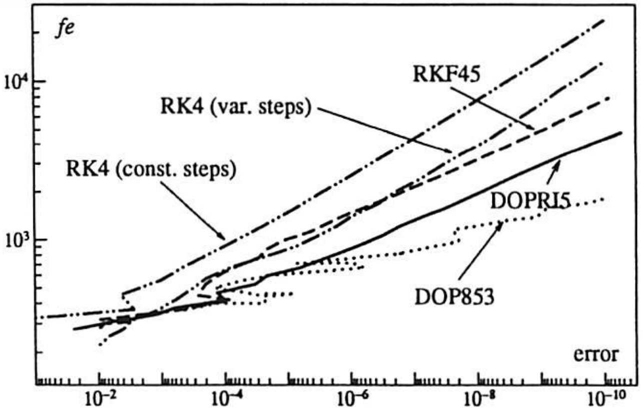

Fig. 4.2. Precision-Work diagram

In this comparison we have included some higher order methods, which will be dicussed in Section II.5. The code RKF45 (written by H.A. Watts and L.F. Shampine) is based on an embedded method of order 5(4) due to Fehlberg. The codes DOPRI5 (order 5(4)) and DOP853 (order 8(5,3)) are based on methods of

Dormand & Prince. They will be discussed in the following section. It can clearly be seen that higher order methods are, especially for higher precision, more efficient than lower order methods. We shall also understand why the 5th order method of Dormand & Prince is clearly superior to RKF45.

Exercises

1. Show that Runge's method (1.4) can be interpreted as two Euler steps (with step size $h/2$), followed by a Richardson extrapolation.

2. Prove that no 4-stage Runge-Kutta method of order 4 admits an embedded formula of order 3.

   Hint. Replace d_j by $\widehat{b}_j - b_j$ in the proof of Lemma 1.4 and deduce that $\widehat{b}_j = b_j$ for all j, which is a contradiction.

3. Show that the step size strategy (4.13) is invariant with respect to a rescaling of the independent variable. This means that it produces equivalent step size sequences when applied to the two problems

$$\begin{aligned} &y' = f(x,y), \qquad && y(0) = y_0, \qquad && y(x_{\text{end}}) = ? \\ &z' = \sigma \cdot f(\sigma t, z), && z(0) = y_0, && z(x_{\text{end}}/\sigma) = ? \end{aligned}$$

   with initial step sizes h_0 and h_0/σ, respectively.

   Remark. This is no longer the case if one replaces *err* in (4.13) by *err*$/h$ and q by $q-1$ ("error per unit step").

II.5 Explicit Runge-Kutta Methods of Higher Order

> Gehen wir endlich zu Näherungen von der fünften Ordnung über, so werden die Verhältnisse etwas andere. (W. Kutta 1901)

This section describes the construction of Runge-Kutta methods of higher orders, particularly of orders $p = 5$ and $p = 8$. As can be seen from Table 2.3, the complexity and number of the order conditions to be solved increases rapidly with p. An increasingly skilful use of simplifying assumptions will be the main tool for this task.

The Butcher Barriers

For methods of order 5 there are 17 order conditions to be satisfied (see Table 2.2). If we choose $s = 5$ we have 15 free parameters. Already Kutta raised the question whether there might nevertheless exist a solution ("Nun wäre es zwar möglich ..."), but he had no hope for this and turned straight away to the case $s = 6$ (see II.2, Exercise 5). Kutta's question remained open for more than 60 years and was answered around 1963 by three authors independently (Ceschino & Kuntzmann 1963, p. 89, Shanks 1966, Butcher 1964b, 1965b). Butcher's work is the farthest reaching and we shall mainly follow his ideas in the following:

Theorem 5.1. *For $p \geq 5$ no explicit Runge-Kutta method exists of order p with $s = p$ stages.*

Proof. We first treat the case $s = p = 5$: define the matrices U and V by

$$U = \begin{pmatrix} \sum_i b_i a_{i2} & \sum_i b_i a_{i3} & \sum_i b_i a_{i4} \\ \sum_i b_i a_{i2} c_2 & \sum_i b_i a_{i3} c_3 & \sum_i b_i a_{i4} c_4 \\ g_2 & g_3 & g_4 \end{pmatrix}, \quad V = \begin{pmatrix} c_2 & c_2^2 & \sum_j a_{2j} c_j - c_2^2/2 \\ c_3 & c_3^2 & \sum_j a_{3j} c_j - c_3^2/2 \\ c_4 & c_4^2 & \sum_j a_{4j} c_j - c_4^2/2 \end{pmatrix} \tag{5.1}$$

where

$$g_k = \sum_{i,j} b_i a_{ij} a_{jk} - \frac{1}{2} \sum_i b_i a_{ik} (1 - c_k). \tag{5.2}$$