Error Estimation and Control for ODEs

L.F. Shampine
Mathematics Department
Southern Methodist University
Dallas, TX 75275
lshampin@mail.smu.edu

February 3, 2004

Abstract

This article is about the numerical solution of initial value problems for systems of ordinary differential equations (ODEs). At first these problems were solved with a fixed method and constant step size, but nowadays the general-purpose codes vary the step size, and possibly the method, as the integration proceeds. Estimating and controlling some measure of error by variation of step size/method inspires some confidence in the numerical solution and makes possible the solution of hard problems. Common ways of doing this are explained briefly in the article.

Running Title: Error Estimation
Key Words: ODE, IVP, error estimation, error control

1 Introduction

We consider the numerical solution of initial value problems (IVPs) for ordinary differential equations (ODEs). It is convenient in both theory and practice to write the ODEs as a system of first order equations

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

For given initial value $y(a)$, we want to approximate $y(t)$ for $a \leq t \leq b$. All the popular codes for solving IVPs implement methods that start with $y_0 = y(a)$. On reaching $y_n \approx y(t_n)$, they take a step of size h_n in the direction of b to form an approximate solution at $t_{n+1} = t_n + h_n$. The early codes integrated with a constant step size specified by the user and this is still done for some purposes. Nowadays general-purpose codes estimate some kind of error and control it by adapting the step size, and perhaps the numerical method, to the solution. This article is about the estimation and control of error when solving IVPs, but at times it will be illuminating to contrast IVPs and boundary value problems (BVPs). For instance, the survey of Christiansen and Russell [3] shows that a good many definitions of error and ways of selecting the mesh points t_n are seen in programs that solve BVPs. Comparatively few are seen in programs that solve IVPs and in this article we discuss only the most common.

The most important reason for estimating error is to gain some confidence in the numerical solution. There is some cost associated with estimating error and adapting the step size/method, but generally this is a bargain because the IVP is solved more efficiently. Indeed, if a solution can change significantly on a scale that is very small compared to $b - a$, it may be impractical to solve the problem with a constant step size that is small enough to resolve the fastest changes in $y(t)$. And, if a constant step size is too big, the computation may become unstable. Controlling the error by adjusting the step size can often stabilize the integration.

We survey error estimation and control as seen in popular general-purpose codes for IVPs, but with a particular concern for issues pertinent to the time integration of partial differential equations. Accordingly, we emphasize modest accuracy and low to medium order methods. Addressing these issues is more difficult for higher order methods and this complicates the implementation of methods and the interpretation of results.

2 Preliminaries

There are two kinds of discrete-variable methods that are popular for solving IVPs. In taking a step of size h_n from an approximation $y_n \approx y(t_n)$, one-step methods such as Runge–Kutta (RK) form approximate solutions $y_{n,j} \approx y(\tilde{t}_{n,j})$ and evaluate f at arguments $\tilde{t}_{n,j}$ that are all in the span of the current step, i.e., $t_n \leq \tilde{t}_{n,j} \leq t_n + h_n$. Methods with memory like the Adams methods and the BDFs (backward differentiation formulas) make use of approximate solutions and values of f computed at mesh points $t_j < t_n$. The basic formulas produce an approximate solution only at t_{n+1}. They can be supplemented so as to obtain an approximate solution at any t in $[t_n, t_{n+1}]$. Suitable approximations underlie the very definitions of Adams formulas and BDFs, but they are the result of considerable research for one-step methods. For one-step methods, computing an approximation between mesh points is often called "dense output" and the scheme itself, a "continuous extension".

A general-purpose code that adapts the step size/method to the solution asks the user for a tolerance τ. At each step the code selects a step size/method so that some measure of error is smaller than τ. In order to solve the problem efficiently, the code tries to select a step size that is about as big as possible.

There are two major factors affecting the choice of step size. One is accuracy and the other is stability. Popular numerical methods produce results in qualitative agreement with the solution of the IVP only when step sizes/tolerances are sufficiently small. It is characteristic of a class of IVPs called "stiff" that the stability requirement is *much* more restrictive than the accuracy requirement. Because of this special methods are used for stiff IVPs. These methods are all implicit. It is much more expensive to evaluate an implicit formula when solving a stiff IVP, but the strong stability properties of some implicit formulas are so much better than explicit formulas that they are a great bargain.

3 True (Global) Error

The users of popular codes generally assume that the codes estimate and control the true (global) error

$$y(t_n) - y_n \tag{2}$$

They do *not*. In fact, it is unusual for a solver even to *estimate* the true error when solving IVPs. The solvers all integrate from a to b. Without solving a problem more than once, it is not possible to *control* the global error: The stability of the IVP itself governs the propagation of numerical errors. Accordingly, we do not know how the errors we make at each step, the local errors, will propagate. On reaching a point t_n, it is quite possible that we have already made errors that will prevent us from achieving a given global accuracy in the rest of the integration. Solving BVPs is different. The popular BVP solvers compute an approximate solution for the whole interval and then improve it until the specified global accuracy is achieved. In contrast to IVP solvers, it is common that these codes estimate and control the true error.

A natural way to assess the true error is to solve the problem a second time with a reduced step size or tolerance, as the case may be. If the second integration is more accurate, the error of the first result can be estimated by comparison. This is regarded as too expensive to do routinely when solving IVPs, but we recommend it for spot checks. Though plausible, this estimate of the true error can be inaccurate and unreliable because reducing the step size/tolerance may not result in a more accurate result. An example is given in [23] of a simple problem for which a production-grade code returned the *same machine numbers* as answers for absolute error tolerances $10^{-2}, 10^{-3}, \ldots, 10^{-7}$. Another simple problem is given there for which reducing the tolerance from 10^{-4} to 10^{-5} for another production-grade code *increased* the maximum error by a factor of 8. There are a number of reasons why this kind of thing happens: The numerical solution may not behave as expected because the step size/tolerance is too big. The same is true when the step size/tolerance is so small that the effects of finite precision arithmetic are visible. Other aspects of the integration may affect the choice of step size, hence the behavior of the numerical solution. One is providing for output. A more subtle reason is that step sizes are chosen conservatively so as to reduce the likelihood of a failed step. This means that some steps are unnecessarily small. When the problem is solved with a reduced tolerance, the solver may find that it is sometimes not necessary to reduce the step size of the first integration in order to pass the error test in the second integration. Some codes adapt the method to the solution. Reducing the tolerance can alter the methods selected and so affect substantially the behavior of the error. A regular behavior of the error when the tolerance is reduced is useful in assessing the true error, but it is also desirable in its own right. For this reason a number of authors, e.g. [40, 25, 2], have studied the issues and considered how to improve the tolerance proportionality of a solver.

The method of integrating twice can be made more reliable by coupling the two integrations. One such approach is to use the same method for both integrations and half the step size in the second integration. Using standard asymptotic results it is then not only possible to estimate the global error of the first integration, but also that of the more accurate second integration. Exploiting this, a code like GERK [34, 35] can compute a global error estimate at a modest additional cost. A variant [5, 6] of the approach is to develop formulas that not only approximate the solution, but also the global error. When the solution is moderately smooth, the approach works well, but it is difficult to get reliable results at both crude and stringent tolerances. This is quite serious because an important reason for wanting an estimate of the global error is to recognize an inadequate numerical solution, which is most likely at crude and stringent tolerances. A fundamental difficulty is that the approach depends on the solution being smooth throughout the interval of integration.

Schemes for the accurate estimation of global errors when solving IVPs are generally thought to be too expensive and of too limited applicability for production codes.

4 Error Estimation

After explaining in the last section what popular codes do not do, let us now consider what they actually do. Most codes estimate the local error, so we begin with it. We then consider estimation of the residual, a measure of error that shares some attributes of both local and global error.

4.1 Local Error

On reaching t_n, the local solution $u(t)$ is defined as the solution of

$$u' = f(t, u), \qquad u(t_n) = y_n \tag{3}$$

The local error of the step from t_n is

$$le_n = u(t_n + h_n) - y_{n+1} \tag{4}$$

It measures how well the method approximates the behavior of a solution of the ODE over one step. For a method of order p,

$$le_n = h_n^{p+1}\phi(t_n, y_n) + O(h_n^{p+2}) \tag{5}$$

Accordingly, we can control the local error by adjusting the step size. The global error is related to the local error by

$$ge_{n+1} = y(t_{n+1}) - y_{n+1} = [y(t_{n+1}) - u(t_{n+1})] + le_n$$

The first term on the far right is the difference at t_{n+1} of two solutions of the ODE that differ at t_n by

$$y(t_n) - u(t_n) = y(t_n) - y_n = ge_n$$

This term reflects the stability of the IVP itself, hence is beyond our control. A good way to think about the behavior of the true error is that it consists of a local error introduced in the present step plus the global error propagated from previous steps. If the IVP is unstable, this decomposition makes clear that small local errors are amplified in the numerical solution. On the other hand, we should not assume that the error increases throughout the integration. In any portion of an integration where the IVP is strongly stable, propagated errors are actually damped, i.e., $\|y(t_{n+1}) - u(t_{n+1})\| \ll \|ge_n\|$.

The usual way to estimate the local error can be described as taking each step with two formulas. There is a basic approximation y_{n+1} of order p and another approximation, y^*_{n+1}, of order $p^* > p$. Then

$$\begin{aligned} est_n = y^*_{n+1} - y_{n+1} &= [u(t_n + h_n) - y_{n+1}] - [u(t_n + h_n) - y^*_{n+1}] \\ &= le_n + O(h^{p+2}) \end{aligned} \tag{6}$$

We see that est_n is an asymptotically correct estimate of the local error of the lower order formula. Many codes control the size of this error, but advance the integration with the higher order result because it is believed to be more accurate. This is called local extrapolation [21].

4.2 Residual

Important applications require that we be able to approximate $y(t)$ at any point $a \le t \le b$. By forming a continuous extension that approximates $y(t)$ on each $[t_n, t_{n+1}]$, we obtain in aggregate an approximation $S(t)$ to $y(t)$ on all of $[a, b]$. Local error control provides at best an indirect control of the accuracy of $S(t)$. Enright [8] proposes controlling the size of the defect (residual) of $S(t)$,

$$r(t) = S'(t) - f(t, S(t))$$

In a backward error analysis we regard $S(t)$ as the exact solution of the ODE

$$S' = f(t, S) + r(t)$$

and we ask whether this ODE is close to the one given, i.e., whether the residual is small. This approach to the control of error has been fundamental in the solution of linear algebraic equations. Though it has not yet seen much use in solving initial value problems for ODEs, the BVP solvers MIRKDC [10] and `bvp4c` [19] use it. The DDE solvers DDVERK [9] and `ddesd` [28] also use it. Enright [7] is a nice survey of the solution of ODEs with control of the defect.

Residual control has a number of virtues. One is that it assesses the error throughout the interval of integration, not just the error made in advancing a step. It is hard to get a reliable assessment of error when the solution is not very smooth. It is also hard when the step sizes/tolerances are too big for asymptotic approximations to be valid or too small for the precision available. We can evaluate the residual wherever we like at a cost of an evaluation of $f(t, S(t))$. Clearly we can use this fact to estimate reliably the size of the residual even in the most difficult circumstances if we are willing to pay the price. Research in the area has concentrated on getting a reliable estimate inexpensively.

The schemes that Enright [8] proposes for estimating a norm of the residual are plausible, but they are not asymptotically correct. Higham [16] shows how to obtain an asymptotically correct estimate and in [17] he does the same for another kind of continuous extension. For both kinds of continuous extensions the behavior of the residual is known to leading order. Using this behavior it is possible to obtain an asymptotically correct estimate with a single sample. Kierzenka and Shampine [19] do something similar when solving BVPs, but they approximate the size of the residual in an integral norm. The samples are chosen so that the quadrature formula provides an asymptotically correct estimate, but the point is to obtain a credible estimate when the asymptotic results are of marginal applicability. In [28] Shampine extends the concept of local error from (4) to

$$Le_n = u(t_n + \sigma h_n) - S(t_n + \sigma h_n)$$

for $0 \leq \sigma \leq 1$. He then establishes a simple relationship between the size of the residual on the span of the step to the size of Le_n for both kinds of schemes investigated by Higham. For such formulas control of the residual provides a direct control of the local error throughout the span of a step, at least when all is going well.

5 Adjustment of Step Size

We estimate and control a measure of error mainly for two reasons. One is to have some confidence in the results. The other is to solve hard problems.

Though related, the tasks are distinct and sometimes we emphasize one more than the other. Estimation and control of error is not expensive, indeed, it may be essential for the efficient solution of an IVP, but in some contexts a constant step size is still popular. Even then a passive estimate of error is worth consideration.

5.1 Control of Local Error/Residual

One way to gain confidence in a solution is to estimate either the local error or the residual at each step and verify that it is smaller than a given tolerance. We can take advantage of this estimate to choose the largest step size that will result in an acceptable error. In this section we discuss adjustment of the step size in terms of local error because this is conventional and the adjustment is not essentially different when the size of the residual is controlled. We assume that in taking a step from t_n with a formula of order p , we compute an asymptotically correct estimate of the local error,

$$est_n = le_n + h.o.t. = h_n^{p+1}\phi(t_n, y_n) + O(h_n^{p+2})$$

Here $h.o.t.$ is "higher order terms". The user specifies a suitable norm and a tolerance τ. The result is to satisfy

$$\|est_n\| \leq \tau \tag{7}$$

This is called an error per step (EPS) criterion. If the estimated error does not pass this test, the step is rejected and repeated with a smaller step size. From the asymptotic behavior of the local error we see that if h_n is reduced to αh_n, then le_n is reduced to $\alpha^{p+1}\, le_n$. This tells us that the largest α for which the estimated local error will pass the error test is approximately

$$\alpha = \left(\frac{\tau}{\|est_n\|}\right)^{1/(p+1)} \tag{8}$$

If the step is a success and the error is unnecessarily small, we could increase the step size on the next step. We estimate the error of such a step of size $h_{n+1} = \alpha h_n$ to be

$$\begin{aligned} est_{n+1} &= h_{n+1}^{p+1}\,\phi(t_{n+1}, y_{n+1}) + h.o.t. \\ &= (\alpha h_n)^{p+1}\,\phi(t_n, y_n) + h.o.t. \\ &= \alpha^{p+1}\, est_n + h.o.t. \end{aligned}$$

From this we see that we can use the same recipe (8) for adjusting the step size after both successful and unsuccessful steps. There is considerable art to using this recipe in practice. For one thing, failed steps are expensive, so it is important to be conservative in the choice of step size. For another, very large changes of step size cannot be justified with these asymptotic arguments. And, the asymptotic approximations are not valid when the step size is too big or too small for the precision or the leading term in the expansion vanishes. More information about what is done in practice can be found in [36, 37, 30, 26].

The procedure outlined can be refined using some previously computed information to predict better the behavior of the local error. This is done in important early work by Zonneveld [42] that was significantly improved by Watts [41]. Gustafsson and coworkers [13, 14] took a different tack, regarding the selection of step size as a control problem. The schemes favored by Watts and Gustafsson *et alia* are quite similar, remarkably so in light of the very different approaches.

An alternative to (7) is to require that

$$\|est_n\| \leq h_n \, \tau \tag{9}$$

which is a criterion of error per unit step (EPUS). This criterion was in vogue at one time because if (9) holds at every step, it is easy to establish convergence and a bound on the true error that is proportional to the tolerance. This conclusion is not true for EPS. The most effective codes of the time used EPS, so what was missing in our understanding of this matter? Two issues are key: local extrapolation and an efficient step size.

Some of the effective codes that used EPS did local extrapolation (XEPS). It is not hard to show that this amounts to a generalized EPUS and then to prove that the convergence results for EPUS are true for this generalization. The local error estimator rests on the assumption that the higher order formula is more accurate. This suggests advancing the integration with the higher order formula, but that is not always a good idea. The most important difficulty with local extrapolation is that the stability of the integration then depends on the stability of the higher order formula. As an example, we observe that the BDFs are the most popular formulas for solving stiff IVPs. When the natural error estimator is used, local extrapolation results in a formula with unsatisfactory stability. That is why none of the widely-used BDF codes does local extrapolation.

If we assume that the step size is chosen to be about as big as possible, it is easy enough to analyze all the possibilities for one-step methods [24] and see that the error behaves in a regular way. There has been little success in analyzing the behavior of the error in the popular BDF and Adams codes that vary their order (method) as well as step size, but recently it has been shown [27] that XEPS control as implemented in ODE/STEP,INTRP [30] results in an error bound that is proportional to the tolerance.

An alternative way to view local extrapolation is to say that the step size is chosen to be efficient for a lower order method. In this view, the step size is smaller than it has to be. When the method used for error control is one order lower, this is not very important. Implicit RK methods can be an effective way to solve stiff IVPs, but it is hard to develop a pair of formulas with good properties that can be evaluated efficiently together. It is so hard that researchers have been willing to settle for a difference in orders greater than one. An important example is RADAU5 [15], which advances the integration with a formula of order 5, but selects a step size appropriate to a formula of order 3. The difference in order is uncomfortably large, but order 3 is the best that you can do with the information available in forming the higher order result. The formula of order 3 has good stability, but it is not nearly as good as that of the formula of order 5. RADAU5 compensates for this with the scheme proposed in [29] for filtering error estimates so as to match better the stability of the formula used to advance the integration of a stiff IVP. When there is a considerable discrepancy in the orders of the formula used for step size selection and the formula used to advance the integration, it is appropriate to take a different view of the whole process. We discuss this further in the next section.

5.2 Solve Hard Problems

Adapting the step size/method to the solution is not just a matter of improving the efficiency of the integration; it makes practical the solution of hard problems. If the solution changes at very different rates in portions of the integration, it may be impractical to solve the IVP with a constant step size small enough to resolve the fastest changes. A very old scheme for dealing with this is to adjust the step size so as to control the relative change in the numerical solution. This scheme has the virtues of simplicity and universal applicability. Certainly it is plausible to use a small step size where the solution is changing rapidly and a big one elsewhere, but it was a long time before the scheme was justified theoretically. In [38] it is analyzed within the framework of the conventional schemes outlined in §2. The great applicability of the scheme reflects its great disadvantage—because it treats all methods like a particular method of very low order, namely one, it is grossly inefficient for methods of even moderate order. Still, it does deal with sharp changes in the solution, even discontinuities, as illustrated by numerical examples in [38]. It also stabilizes the integration in a way that we take up now.

As noted in §2, the step size may have to be restricted if the numerical solution is to behave like $y(t)$. There is a theory of absolute stability that quantifies this restriction. The theory has certainly proved its worth in providing guidelines for practical computation, but it is rigorously applicable only to a small class of ODEs solved with constant step size. According to this theory, if a constant step size is too large, the computation can become unstable and the results “explode”. This never happens when using a production–grade code that adapts the step size to the solution. An explanation offered in [22] and subsequently refined in [39] is: When the step size is small enough to correspond to being inside the region of absolute stability, the numerical solution smoothes out. Standard local error estimators recognize that $y(t)$ is easy to approximate and cause the solver to increase the step size. Eventually the step size corresponds to being outside the stability region and the computation becomes unstable. Standard estimators recognize this instability as local errors that are unacceptably large and cause the solver to reduce the step size. Eventually the step size corresponds to being inside the stability region and the cycle repeats.

We should state explicitly that all stiff problems are hard in the sense discussed here. Stiffness is a vague concept, but there are three essential ingredients: The IVP must be stable. The ODEs must have solutions that can change on a time scale very short compared to $b - a$. The solution of interest may (and usually does) exhibit such a change, but it must be slowly varying over most of $[a, b]$. By definition the cost of solving a stiff IVP using a constant step size small enough to resolve the rapid changes that are possible is unacceptably high. Solving stiff problems is made practical by resorting to methods that can use a large step size where the solution is slowly varying. These methods are much more expensive per step than, say, explicit RK methods or Adams methods, but they can take steps that are so much bigger that the overall cost is quite acceptable.

Returning to the issue of finding pairs of implicit RK formulas for solving stiff IVPs, we appreciate now that a pair can be useful even if there is a considerable discrepancy in orders—the connection between local error and true error is somewhat weak at best and it is not at all unreasonable to regard step size adjustment as just a way of solving hard problems. Jay [18, pp. 438–439] adopts this point of view and describes a way of improving the qualitative behavior of a solver that is particularly valuable in these circumstances. For a pair of formulas of orders $p^* > p$, the step size h_n is selected so that the predicted error of size $O(h_n^{p+1})$ is approximately equal to the tolerance τ, which is to say that the step size is $O\left(\tau^{1/(p+1)}\right)$. With local extrapolation the actual error of the step is $O(h_n^{p^*+1})$, hence $O\left(\tau^{(p^*+1)/(p+1)}\right)$. When p^* is rather bigger than p, this error is much smaller than τ, which is to say that the step size is much smaller than necessary. The idea is to define an internal tolerance

$$\mu = \tau^{(p+1)/(p^*+1)}$$

and select h_n so that the error of the lower order formula is approximately equal to μ. This step size is $O(\mu^{1/(p+1)})$, hence the actual error is

$$O\left(\mu^{(p^*+1)/(p+1)}\right) = O(\tau)$$

In this way we can select a step size that has the right order for the formula used to advance the integration. And, the user finds that reducing the tolerance τ results in the local error being reduced proportionately. This argument assumes that local extrapolation is done, but an internal tolerance was similarly used in [25] to get tolerance proportionality when it is not. An internal tolerance is used to deal with the substantial discrepancy in orders in RADAU5 [15] that was mentioned earlier.

5.3 Passive Estimation of Error

As we have seen, there are important reasons for adapting the step size/method to the solution, but in some circumstances researchers prefer to use a constant step size and fixed method. It is still valuable to estimate error so as to have some confidence in the numerical solution. In particular, monitoring the error inspires some confidence that the step size is small enough both for resolving the behavior of the solution and for the numerical method to behave as expected. This is what we mean by "passive" estimation of error. It is not often done, so we want to make the point here that for some formulas it is not expensive. In this we emphasize low order schemes because they are of particular interest in contexts where a fixed step size is used.

The local error of the trapezoidal rule when the step size is a constant h has the form $-y^{(3)}(\xi)h^3/12$. It is easy enough to approximate the derivative using differences of the y_n or $f_n = f(t_n, y_n)$. These estimates are "free" because they are just linear combinations of quantities computed as the integration proceeds. An advantage of using the f_n is that fewer terms need be retained. This is quite satisfactory when solving non-stiff IVPs, but when solving stiff problems, errors in y_n can be significantly amplified in f_n, c.f. [26]. Accordingly, when solving stiff IVPs, standard local error estimators are based on approximating derivatives by differences of the y_n. For example, the local error of the trapezoidal rule is estimated this way in `ode23t` [32]. Similarly, the local error of any of the BDFs could be monitored passively using standard estimators like those of [12].

Many of the standard pairs of explicit RK formulas can be regarded as providing free estimates of a local error. For example, the seminal (4,5) pair of Fehlberg [11] was derived so as to provide an estimate of the error in the formula of order 4, but the many production codes such as RKF45 [36, 37] that use it do local extrapolation. The fifth order formula uses the minimal number of stages, namely 6. Consequently, if we do a constant step size integration with the minimal stage fifth order formula, we have available a free estimate of the local error of the companion fourth order formula. Another point is made by the BS(2,3) pair [1] because it is FSAL (First Same As Last): Dormand and Prince [4] note that the first stage of an explicit RK formula is always $f(t_n, y_n)$ and propose that this stage from the next step be used in the companion formula for error estimation in the current step. The stage is free only if the step is a success so that it can be reused. Most steps do succeed, so FSAL formulas are in wide use. They are attractive in the present situation because in a constant step size integration, all steps are accepted. Using the BS(2,3) pair with constant step size we integrate with a third order formula and obtain an estimate of the local error of a companion second order formula for “free”. Another low order example is the popular two-stage, second order RK method due to Heun. It is shown in [21] that after two steps of the same size, a linear combination of the stages provides an asymptotically correct estimate of the local error.

References

[1] Bogacki, P., and Shampine, L.F. (1989). A 3(2) pair of Runge-Kutta formulas. Appl. Math. Letters **2**, 1–9.

[2] Calvo, M.C., Higham, D.J., Montijano, J.M., and Rández, L. (1997). Stepsize selection for tolerance proportionality in explicit Runge–Kutta codes. Advances in Computational Math. **24**, 361–382.

[3] Christiansen, J., and Russell, R.D. (1978). Adaptive mesh selection strategies for solving boundary value problems. SIAM J. Numer. Anal. **15**, 59–80.

[4] Dormand, J.R., and Prince, P.J. (1980). A family of embedded Runge-Kutta formulae. J. Comput. Appl. Math. **6**, 19–26.

[5] Dormand, J.R., and Prince, P.J. (1984). Global error estimation with Runge-Kutta methods. IMA J. Numer. Anal. **4**, 169–184.

[6] Dormand, J.R., and Prince, P.J. (1985). Global error estimation with Runge-Kutta methods II. IMA J. Numer. Anal. **5**, 481–497.

[7] Enright, W.H. (2000). Continuous numerical methods for ODEs with defect control. J. Comp. Appl. Math. **125**, 159–170.

[8] Enright, W.H. (1989). A new error–control for initial value solvers. Appl. Math. Comput. **31**, 588–599.

[9] Enright, W.H., and Hayashi, H. (1997). A delay differential equation solver based on a continuous Runge–Kutta method with defect control. Numer. Alg. **16**, 349–364.

[10] Enright, W.H., and Muir, P.H. (1996). Runge–Kutta software with defect control for boundary value ODEs. SIAM J. Sci. Comput. **17**, 479–497.

[11] Fehlberg, E. (1970). Klassische Runge-Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme. Computing **6**, 61–71.

[12] Gear, C.W. (1971). *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ.

[13] Gustafsson, K. (1991). Control theoretic techniques for stepsize selection in explicit Runge-Kutta methods. ACM Trans. Math. Softw. **17**, 533–554.

[14] Gustafsson, K., Lundh, M., and Söderlind, G. (1988). A PI stepsize control for the numerical solution of ordinary differential equations. BIT **18**, 270–287.

[15] Hairer, E., and Wanner, G. (1991). *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*, Springer, Berlin.

[16] Higham, D.J. (1989). Robust defect control with Runge–Kutta schemes. SIAM J. Numer. Anal. **26**, 1175–1183.

[17] Higham, D.J. (1991). Runge–Kutta defect control using Hermite–Birkhoff interpolation. SIAM J. Sci. Stat. Comput. **12**, 991–999.

[18] Jay, L. (1998). Structure preservation for constrained dynamics with super partitioned additive Runge-Kutta methods. SIAM J. Sci. Comput. **20**, 416–446.

[19] Kierzenka, J., and Shampine, L.F. (2001). A BVP solver based on residual control and the MATLAB PSE. ACM Trans. Math. Softw. **27**, 299–316.

[20] MATLAB *6* (2000). The MathWorks, Inc., 3 Apple Hill Dr., Natick, MA 01760.

[21] Shampine, L.F. (1973). Local extrapolation in the solution of ordinary differential equations. Math. Comp. **27**, 91–97.

[22] Shampine, L.F. (1975). Stiffness and non-stiff differential equation solvers. In Collatz, L., et al. (eds.), *Numerische Behandlung von Differentialgleichungen*, ISNM 27, Birkhauser, Basel, pp. 287–301.

[23] Shampine, L.F. (1980). What everyone solving differential equations numerically should know. In Gladwell, I., and Sayers, D.K. (eds.), *Computational Techniques for Ordinary Differential Equations*, Academic, London, pp. 1–17.

[24] Shampine, L.F. (1985). The step sizes used by one-step codes for ODEs. Appl. Numer. Math. **1**, 95–106.

[25] Shampine, L.F. (1989). Tolerance proportionality in ODE codes. In Bellen, A., et al. (eds.), *Numerical Methods for Ordinary Differential Equations*, Lecture Notes in Math. No. 1386, Springer, Berlin, pp. 118–136.

[26] Shampine, L.F. (1994). Numerical Solution of Ordinary Differential Equations, Chapman & Hall, New York.

[27] Shampine, L.F. (2002). Variable order Adams codes. Comp. & Maths. with Applics. **44**, 749–761.

[28] L.F. Shampine, Solving ODEs and DDEs with residual control, http://faculty.smu.edu/lshampin/residuals.pdf

[29] Shampine, L.F., and Baca, L.S. (1984). Error estimators for stiff differential equations. J. Comp. Appl. Math. **11**, 197–207.

[30] Shampine, L.F., and Gordon, M.K. (1975). *Numerical Solution of Ordinary Differential Equations: the Initial Value Problem*, W. H. Freeman and Co., San Francisco.

[31] Shampine, L.F., Gladwell, I., and Thompson, S. (2003). *Solving ODEs with* MATLAB, Cambridge Univ. Press, New York.

[32] Shampine, L.F., and Reichelt, M.W. (1997). The MATLAB ODE suite. SIAM J. Sci. Comput. **18**, 1–22.

[33] Shampine, L.F., Reichelt, M.W., and Kierzenka, J.A. (1999). Solving index-1 DAEs in MATLAB and Simulink. SIAM Review **41**, 538–552.

[34] Shampine, L.F., and Watts, H.A. (1976). Global error estimation for ordinary differential equations. ACM Trans. Math. Softw. **2**, 172–186.

[35] Shampine, L.F., and Watts, H.A. (1976). Algorithm 504, GERK: global error estimation for ordinary differential equations. ACM Trans. Math. Softw. **2**, 200–203.

[36] Shampine, L.F., and Watts, H.A. (1977). The art of writing a Runge-Kutta code, Part I. In Rice, J.R. (ed.), *Mathematical Software III*, Academic, New York, pp. 257–275.

[37] Shampine, L.F., and Watts, H.A. (1979). The art of writing a Runge-Kutta code, II. Appl. Math. Comp. **5**, 93–121.

[38] Shampine, L.F., and Witt, A. (1995). A simple step size selection algorithm for ODE codes. J. Comp. Appl. Math. **58**, 345-354.

[39] Shampine, L.F., and Witt, A. (1995). Control of local error stabilizes integrations. J. Comp. Appl. Math. **62**, 333-351.

[40] Stetter, H.J. (1980). Tolerance proportionality in ODE-codes. In März, R. (ed.), *Seminarberichte No. 32*, Humboldt Univ., Berlin, pp. 109–123.

[41] Watts, H.A. (1984). Step size control in ordinary differential equation solvers. Trans. Soc. for Computer Simulation **1**, 15–25.

[42] Zonneveld, J.A. (1964). *Automatic Numerical Integration*, Mathematisch Centrum, Amsterdam.