Solving ODEs and DDEs with Residual Control

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

Abstract

We first consider the numerical integration of ordinary differential equations (ODEs) with Runge–Kutta methods that have continuous extensions. For some methods of this kind we develop robust and inexpensive estimates of both the local error and the size of the residual. We then develop an effective program, `ddesd`, to solve delay differential equations (DDEs) with time- and state-dependent delays. To get reliable results for these difficult problems, the code estimates and controls the size of the residual. The user interface of `ddesd` makes it easy to formulate and solve DDEs, even those with complications like event location and restarts.

1 Introduction

The first part of this paper considers the numerical solution of a first order system of ordinary differential equations (ODEs),

$$y'(t) = f(t, y(t))$$

on an interval $a \le t \le b$ with given initial value $y(a)$. Runge–Kutta (RK) methods start with $y_0 = y(a)$ and on reaching $y_n \approx y(t_n)$, take a step of size h_n in the direction of b to form an approximate solution at $t_{n+1} = t_n + h_n$. Popular codes estimate the error made in this step (the local error) and adjust the step size so as to satisfy a tolerance specified by the user. A Runge–Kutta formula provides an approximate solution only at t_{n+1}, but it can be supplemented with a continuous extension, an inexpensive approximation to $y(t)$ for $t_n \le t \le t_{n+1}$. This is important, even crucial, to applications like plotting smooth graphs, event location, and solving delay differential equations (DDEs). Control of the local error provides at most an indirect control of the error of the continuous extension. Enright [2] points out advantages of a direct control of the error of the continuous extension and suggests that this be done by estimating and controlling the size of its defect (residual). It is especially

attractive when solving DDEs: As we explain below, this application of RK methods requires an accurate numerical solution for all t. Also, discontinuities in low order derivatives of the solution are typical. Because of this it is much easier to estimate reliably the size of the residual than the local error. Higham [7, 8] develops two kinds of schemes based on explicit RK formulas for which an asymptotically correct estimate of the size of the defect can be obtained with a single evaluation of $f(t, y)$. Control of the size of the defect is natural in the sense of backward error analysis, but users are more accustomed to control of local error and in particular, find it easier to interpret error tolerances then. Here we connect the two ways of assessing error by establishing a simple relationship between the size of the residual and the local error for both kinds of schemes studied by Higham. Enright and Higham consider only explicit RK methods because implicit methods are generally thought to be inefficient for non-stiff ODEs. As we explain below, when solving DDEs with RK formulas, y_{n+1} is defined implicitly in important circumstances even when the formula is explicit. For this reason we consider implicit RK methods, too.

We use the methods studied in the first part of this paper to develop an effective way to solve a set of first order delay differential equations (DDEs)

$$y'(t) = f(t, y(t), y(d_1), \ldots, y(d_k))$$

on an interval $[a, b]$ with $y(t)$ given for $t \leq a$. It is assumed that all the delay functions $d_j = d_j(t, y(t))$ are such that $d_j \leq t$. Early work considered only functions of the form $d_j = t - \tau_j$ with constant lag τ_j. Codes for such problems typically use an explicit RK formula with continuous extension for the integration. By saving the information needed to evaluate the continuous extension on each $[t_m, t_{m+1}]$, an approximation is available for $y(t)$ at any t prior to the current point t_n. If the step size h_n is no bigger than the shortest lag, it is straightforward to compute y_{n+1} with an explicit RK formula because all the arguments $t-\tau_j$ of the formula are prior to t_n. In some respects the solutions of DDEs behave quite differently from the solutions of ODEs. Almost always there is a discontinuity in a low order derivative at the initial point and it propagates. Fortunately, for the class of DDEs studied in this paper, solutions smooth out as the integration proceeds. This is one reason why a step size appropriate to the solution might be much bigger than the shortest lag. If we try to use such a step size, some of the terms $y(t - \tau_j)$ will represent values of the solution in the span of the current step. These values are not yet known, so y_{n+1} is defined implicitly. Some codes restrict the step size so as to have an explicit recipe, but this is not an option when solving problems with more general delay functions. As illustrated by examples in §5, some DDEs have points t where $t - d_j(t, y(t))$ vanishes. A more fundamental difficulty is that when the delays are state-dependent, we must know the solution on $[t_n, t_{n+1}]$ in order to evaluate the delay functions $d_j(t, y(t))$ used in the RK formula to compute the solution on $[t_n, t_{n+1}]$.

At the present time no method for solving DDEs with time- and state-dependent delays is clearly best. A key issue is how to deal with the propagation

of discontinuities. Some of the most effective codes track discontinuities, but this is expensive and it is difficult to locate discontinuities accurately when the delay functions are state-dependent. Instead of tracking discontinuities, some codes rely upon standard algorithms for the estimation and control of the local error for ODEs to recognize and deal with them. Underlying these algorithms is an assumption that the solution is sufficiently smooth. This may not be the case when solving DDEs, so the reliability of results obtained in this way is questionable. We present here a MATLAB program, `ddesd`, for solving DDEs with time- and state-dependent delays that does not track discontinuities. To compute solutions reliably in the presence of discontinuities in low order derivatives of the solution, it estimates and controls the size of the defect. Specifically, an explicit Runge–Kutta formula and continuous extension developed in the first part of this paper is used to obtain an inexpensive estimate of the size of the defect that is meaningful even when the solution is not smooth. When the solution is sufficiently smooth, the scheme also provides a control of the local error. `ddesd` shares a powerful and convenient user interface with `dde23`. It has performed well on a large collection of DDEs taken from the literature.

2 Residual Control for ODEs

Suppose that the integration of a first order system of ODEs

$$y'(t) = f(t, y(t))$$

has reached t_n where we have $y_n \approx y(t_n)$. The local solution $u(t)$ is defined there by

$$u' = f(t, u), \quad u(t_n) = y_n$$

In taking a step of size h_n from t_n to obtain y_{n+1}, the local error of a Runge–Kutta method is

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

This error is estimated by taking the step with two formulas, one producing an approximation y_{n+1} with local error $O(h_n^{p+1})$ and another producing y^*_{n+1} with local error $O(h_n^{p+2})$. It is then immediate that

$$y^*_{n+1} - y_{n+1} = le_n + h.o.t.$$

Here "*h.o.t.*" is "higher order terms". At each step the codes adjust h_n so that a norm of the local error is no bigger than a tolerance τ. For efficiency, the codes try to use the largest step size that will deliver the specified accuracy. Although it is the error of the lower order formula that is estimated, popular codes advance the integration with the higher order result (local extrapolation) because it is believed to be more accurate.

Early codes reduced the step size so as to produce answers at specified output points, but later it was learned how to obtain continuous extensions. These are functions $p(t)$ computed using the stages formed when taking a step, and

perhaps some additional stages, that approximate $u(t)$ throughout $[t_n, t_{n+1}]$. The ones used in practice are polynomials, but other kinds of functions have been used for theoretical purposes. Continuous extensions are all but necessary for locating events and solving DDEs. Clearly it is of great practical importance that control of the error at the end of a step provide some control of the error in the span of the step. Early theoretical results of this kind are found in Stetter [18].

In the approach just outlined, we control the error made in stepping from t_n to t_{n+1} and supplement the formula with a continuous extension valid for all of $[t_n, t_{n+1}]$. Enright [2] suggests that we focus on the continuous extension and control its error directly. Polynomial continuous extensions on the various $[t_n, t_{n+1}]$ form in aggregate a piecewise polynomial function $S(t)$ that approximates $y(t)$ on all of $[a, b]$. Enright proposes that the size of the defect (residual),

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

be controlled. In a backward error analysis we regard the computed $S(t)$ 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, two effective codes for solving boundary value problems (BVPs), namely MIRKDC [5] and `bvp4c` [9], use it. It is used in the DDE solver DDVERK [3] and in the DDE solver `ddesd` that we present in this paper.

At each step we can evaluate the residual wherever we like and use these values to estimate the size of the residual over the span of a step. In his study of this task, Enright makes an important observation: If $p(t)$ is the continuous extension on $[t_n, t_{n+1}]$, then on this interval

$$r(t) = [p'(t) - u'(t)] + [f(t, u(t)) - f(t, p(t))]$$

For Lipschitzian f, the second term on the right is of the same order of accuracy as $u(t) - p(t)$. Generally $p(t)$ approximates $u(t)$ to higher order than $p'(t)$ approximates $u'(t)$. Assuming that this is so, Enright points out that

$$r(t) = -[u'(t) - p'(t)] + h.o.t.$$

This insight and the form he adopts for the continuous extension then provide guidance for estimation of a norm of the residual. Unfortunately, he does not obtain estimates that are asymptotically correct. Higham [7] shows how to accomplish this. His investigation is based on an approach [13, 6] that we took to constructing continuous extensions of RK methods. Here we study further consequences of the approach, so it is convenient to review it briefly.

In the course of taking a step, RK methods produce approximations to $y(t)$ at a number of points in $[t_n, t_n + h_n]$. Generally these approximations are of low order, but for some methods there are m distinct points $\{\xi_i\}$ where the approximations u_i satisfy

$$u_i - u(\xi_i) = O(h_n^{q+1}) \qquad i = 1, \ldots, m$$

In addition, the methods form $u_i' = f(\xi_i, u_i)$ for $i = 1, \ldots, r$ with $r \leq m$. For Lipschitzian f, these quantities satisfy

$$u_i' - u'(\xi_i) = O(h_n^{q+1}), \qquad i = 1, \ldots, r$$

The continuous extension $p(t)$ on $[t_n, t_{n+1}]$ is defined as the unique polynomial of degree less than or equal to $m+r-1$ that interpolates these approximations:

$$\begin{array}{ll} p(\xi_i) = u_i, & i = 1, \ldots, m \\ p'(\xi_i) = u_i', & i = 1, \ldots, r \end{array} \tag{2}$$

If we require that $\xi_1 = t_n$, $u_1 = y_n$, $\xi_r = t_{n+1}$, and $u_r = y_{n+1}$, these continuous extensions form a piecewise polynomial approximation $S(t)$ to $y(t)$ that is $C^1[a, b]$.

We analyzed the error of these continuous extensions by introducing the polynomial $Q(t)$ that interpolates the local solution:

$$\begin{array}{ll} Q(\xi_i) = u(\xi_i), & i = 1, \ldots, m \\ Q'(\xi_i) = u'(\xi_i), & i = 1, \ldots, r \end{array}$$

With this polynomial we write

$$u^{(k)}(t) - p^{(k)}(t) = \left[u^{(k)}(t) - Q^{(k)}(t)\right] + \left[Q^{(k)}(t) - p^{(k)}(t)\right] \tag{3}$$

It is shown in [13] that the first term, the "interpolation error" is $O(h_n^{m+r-k})$ and the second, the "data error", is $O(h_n^{q+r-k})$. In [13, 6] we studied two situations which allow us to come to a much better understanding of the behavior of the local error, namely when the interpolation error dominates and when the data error dominates. In [7] Higham studies residual control when the interpolation error dominates. Here we show that the schemes he investigates are even more attractive by relating their residual to their local error. Later [8] he considered methods for which the data error dominates. We also consider such methods and again show how to relate the residual to the local error.

2.1 Interpolation Error Dominates

In this section we suppose that the interpolation error dominates in (3) so that

$$u(t) - p(t) = O(h_n^{m+r})$$

and

$$r(t) = -\left[u'(t) - p'(t)\right] + h.o.t. = O(h_n^{m+r-1})$$

In these circumstances Shampine and Higham show that

$$\begin{aligned} u(t) - p(t) &= \delta(\sigma) \frac{u^{(m+r)}(t_n)}{(m+r)!} h_n^{m+r} + h.o.t. \\ u'(t) - p'(t) &= \delta'(\sigma) \frac{u^{(m+r)}(t_n)}{(m+r)!} h_n^{m+r-1} + h.o.t. \end{aligned}$$

The function $\delta(\sigma)$ here is a polynomial in a variable σ defined by $t = t_n + \sigma h_n$ and values σ_i defined by $\xi_i = t_n + \sigma_i h_n$, namely

$$\delta(\sigma) = \prod_{i=1}^{r} (\sigma - \sigma_i)^2 \prod_{i=r+1}^{m} (\sigma - \sigma_i)$$

These interpolants are interesting because to leading order, the way that the error behaves in the span of a step is known *a priori*. For a given interpolant, Higham suggests that we find points σ^* where the polynomial $\delta'(\sigma)$ has its maximum magnitude on $[0, 1]$. Correspondingly, we can work out

$$C_1 = \max_{0 \le \sigma \le 1} |\delta'(\sigma)| = |\delta'(\sigma^*)|$$

By evaluating the residual at one of these points, we obtain an asymptotically correct estimate of a weighted maximum norm of the residual:

$$\begin{aligned} \|r\| &= \max_{0 \le \sigma \le 1} \|r(t_{n+\sigma})\| \\ &= \max_{0 \le \sigma \le 1} \|u'(t_{n+\sigma}) - p'(t_{n+\sigma})\| + h.o.t. \\ &= |\delta'(\sigma^*)| \frac{\|u^{(m+r)}(t_n)\|}{(m+r)!} h_n^{m+r-1} + h.o.t. \\ &= \|r(t_{n+\sigma^*})\| + h.o.t. \end{aligned}$$

In this we use the standard notation $t_{n+\sigma} = t_n + \sigma h_n$. In contrast to other continuous extensions, a single, judiciously-chosen sample allows us to ascertain the behavior of the residual on the whole span of the step, at least to leading order.

Control of the residual is a robust and meaningful measure of the error of an integration, but there is no doubt that we would prefer a robust measure of the error in the solution itself. We now show that such a measure is to hand. For a given interpolant it is straightforward to work out

$$C_2 = \max_{0 \le \sigma \le 1} |\delta(\sigma)|$$

With this definition and the expression for the error in the solution we obtain

$$\begin{aligned} \|u - p\| &= \max_{0 \le \sigma \le 1} \|u(t_{n+\sigma}) - p(t_{n+\sigma})\| \\ &= C_2 \frac{\|u^{(m+r)}(t_n)\|}{(m+r)!} h_n^{m+r} + h.o.t. \end{aligned}$$

It is convenient to extend the concept of local error to the span of the step with the definition

$$Le_n(t_{n+\sigma}) = u(t_{n+\sigma}) - p(t_{n+\sigma})$$

Using the corresponding expression for the error in the first derivative of the solution, it is now easy to see that

$$\|Le_n\| = h_n \frac{C_2}{C_1} \|r(t_{n+\sigma^*})\| + h.o.t.$$

From this we see that we can obtain an asymptotically correct approximation of the maximum error in the span of a step by a single evaluation of the residual.

Conventionally codes control the difference between the local solution and a numerical approximation at the end of the step. With continuous extensions like these, we find that we can control this difference throughout the span of the step. From the point of view of backward error analysis, a numerical solution with a small residual is a good solution. Nevertheless, users are more familiar with control of the local error and so find it easier to appreciate the role of tolerances. We see now that with these continuous extensions, we can control a more familiar quantity and still control the residual. We also note that by construction, these methods have a maximum local error that is of lower order than the error in the value y_{n+1} used to advance the integration. Accordingly, controlling the maximum local error is somewhat like local extrapolation.

One of Higham's examples is a four-stage, fourth-order explicit RK formula with a cubic Hermite interpolant. For this interpolant, $m = 2 = r$ and $\xi_1 = t_n$, $\xi_2 = t_{n+1}$. For such a formula, $q = 4$ and the continuous extension is accurate to $O(h_n^4)$. By construction the residual vanishes at both ends of the step, but the expression

$$\delta'(\sigma) = 2\sigma(\sigma - 1)(2\sigma - 1)$$

shows that to leading order, it also vanishes at the middle of the step. The value of the residual at any other point can be used to obtain an asymptotically correct approximation of $\|r\|$. Higham observes that the maximum magnitude of this polynomial is attained at

$$\sigma_1^* = 1/2 - \sqrt{3}/6, \qquad \sigma_2^* = 1/2 + \sqrt{3}/6 \tag{4}$$

where $C_1 = \sqrt{3}/9$ and proposes that the residual be evaluated at one of these points. We add to this an easy calculation with $\delta(\sigma)$ that results in $C_2 = 1/16$. Any fourth order RK formula might be used with this continuous extension. Higham [7] gives some numerical results when the basic formula is the explicit four-stage formula known as the "3/8 formula". For `ddesd` we have preferred the "classic formula".

2.2 Data Error Dominates

In [6] we considered some formulas and continuous extensions for which it is possible to relate the local error at the end of the step to the local error throughout

the step. Here we connect this to the residual and use it to obtain an asymptotically correct estimate of the error. Because few of the well-known formulas have this desirable property, Higham [8] considers how to supplement a natural continuous extension so as to obtain another continuous extension for which an asymptotically correct estimate of the residual can be obtained with a single sample. As in §2.1, we show that the local error is related to the residual in a simple way. All the papers that we have cited assume that an explicit RK formula is used, but it is perfectly possible to use an implicit formula. As outlined in §1, implicit formulas are more attractive when solving non-stiff DDEs than ODEs. We have written versions of `ddesd` that use methods of the form studied in this section that are based on implicit RK formulas.

One of the examples of [6] supposes that a step is taken from t_n to t_{n+1} as two half steps with a formula of order 4. A number of authors have studied formulas that can be viewed as having this form plus some other stages that are used in a fifth order formula for estimating the local error. Here we simply assume that a formula of order 4 is chosen and discuss how to estimate the residual and the local error throughout the span of the step. We have experimented with this method when the fourth order formula is the implicit Simpson (3 point Lobatto) formula used by `bvp4c`.

A continuous extension for such a formula is obtained by interpolation with $m = 3 = r$ and $\xi_1 = t_n$, $\xi_2 = t_{n+1/2}$, $\xi_3 = t_{n+1}$. In [6] we showed that the quintic Hermite interpolant $p(t)$ approximates the local solution and its derivatives as

$$u^{(k)}(t) - p^{(k)}(t) = \left[A_2^{(k)}(t)/2 + A_3^{(k)}(t)\right] le_n + h.o.t. \tag{5}$$

The fundamental interpolating polynomials $A_2(t)$ and $A_3(t)$ are given in [13]. When $k = 0$ in (5),

$$u(t_n + \sigma h_n) - p(t_n + \sigma h_n) = -\sigma^2(-60\sigma^2 + 50\sigma - 15 + 24\sigma^3)\ le_n + h.o.t. \tag{6}$$

The polynomial multiplying le_n is plotted as Fig. 3.2 in [6]. As seen in the figure, the error increases smoothly from zero at t_n to its *maximum* of $\|le_n\|$ at t_{n+1}, i.e., $\|Le_n\| = \|le_n\| + h.o.t.$

In [13, 6] we made no use of the case $k = 1$ in (5), namely

$$u'(t) - p'(t) = 120\sigma(1-\sigma)(\sigma - 1/2)^2\ le_n/h_n + h.o.t. \tag{7}$$

However, we now appreciate that we can use this result to conclude that

$$r(t_{n+\sigma}) = -120\sigma(1-\sigma)(\sigma - 1/2)^2\ le_n/h_n + h.o.t.$$

An interesting fact about this continuous extension is that to leading order, its residual is of one sign. To leading order, the residual vanishes at $\{0, 1/2, 1\}$, but sampling the residual at any other point in the interval provides an asymptotically correct estimate of the local error. Just as Higham did for the example we studied in §2.1, we could find a point σ^* where the polynomial in this expression

has its maximum magnitude on $[0,1]$. With it we can obtain an asymptotically correct estimate of the size of the residual with a single sample:

$$\|r\| = \|r(t_{n+\sigma^*})\| + h.o.t.$$

After working out

$$C_3 = \max_{0\le\sigma\le 1} |-120\sigma(1-\sigma)(\sigma-1/2)^2|$$

we also have

$$\|le_n\| = h_n \|r(t_{n+\sigma^*})\|/C_3 + h.o.t. = \|Le_n\| + h.o.t.$$

Because the local error at the end of the step is the maximum local error in the span of the step, we obtain an asymptotically correct estimate of the maximum local error using just one sample of the residual. In contrast to the formulas of §2.1, the integration is advanced with a result of the same order of accuracy as the error that we control.

We illustrate Higham's approach [8] to robust defect control and the additional properties that we have observed with the two stage Radau IIA method

$$\begin{aligned} y_{n+1/3} &= y_n + h_n\left[\frac{5}{12}f_{n+1/3} - \frac{1}{12}f_{n+1}\right] \\ y_{n+1} &= y_n + h_n\left[\frac{3}{4}f_{n+1/3} + \frac{1}{4}f_{n+1}\right] \end{aligned}$$

This is a third order formula, so the cubic Hermite interpolant to value and slope at the two ends of the step is accurate to $O(h_n^4)$ throughout the span of the step. Unfortunately, it does not have the properties we want. To get a continuous extension that does, Higham uses $p(t)$ to construct some additional approximations to the first derivative of the local solution and interpolates them, too. For the present formula, we need only one such approximation, which we obtained for our experiments by first defining $y_{n+2/5} = p(t_{n+2/5})$ and then evaluating $f_{n+2/5}$. The continuous extension studied by Higham is the unique quartic polynomial $H(t)$ such that

$$H(t_n) = y_n, \qquad H(t_{n+1}) = y_{n+1}$$

and

$$H'(t_n) = f_n, \quad H'(t_{n+2/5}) = f_{n+2/5}, \quad H'(t_{n+1}) = f_{n+1}$$

For this interpolant his analysis shows that

$$u(t_{n+\sigma}) - H(t_{n+\sigma}) = -15\sigma^2(\sigma-2/3)(\sigma-6/5)\ le_n + h.o.t.$$

Inspection of the polynomial factor shows that it has its maximum magnitude of 1 at the end of the step, which is to say that $\|Le_n\| = \|le_n\| + h.o.t.$ This interpolant has the same order of accuracy as the cubic interpolant $p(t)$, but

with it, we know how the error is distributed throughout $[t_n, t_{n+1}]$. In particular, controlling $\|le_n\|$ controls the local error throughout the step. Higham goes on to derive an expression for the residual which in this instance is

$$r(t_{n+\sigma}) = -60\sigma(\sigma - 2/5)(\sigma - 1)\ le_n/h_n + h.o.t.$$

Higham was interested only in controlling the residual, but just as with our other examples, we can obtain an asymptotically correct estimate of the local error with one sample of the residual: We can find a point σ^* where the polynomial in this expression has its maximum magnitude on $[0, 1]$ and work out

$$C_4 = \max_{0\le\sigma\le1} |-60\sigma(\sigma - 2/5)(\sigma - 1)|$$

Then

$$\|le_n\| = h_n \|r(t_{n+\sigma^*})\|/C_4 + h.o.t. = \|Le_n\| + h.o.t.$$

3 Residual Control for DDEs

We now consider how to use the methods of §2 to solve DDEs of the form

$$y'(t) = f(t, y(t), y(d_1), \ldots, y(d_k)) \tag{8}$$

on an interval $[a, b]$ with $y(t)$ given for $t \le a$. We assume that all the delay functions $d_j = d_j(t, y(t))$ are such that $d_j \le t$. The DDE solvers ARCHI [12], DKLAG6 [1], and DDVERK [3] have much in common. They are all based on explicit RK formulas with continuous extensions. They allow time- and state-dependent delays. They provide for small and vanishing delays. In addition, they all accept problems of neutral type. These are problems of the form

$$y'(t) = f(t, y(t), y(d_1), \ldots, y(d_k), y'(d_{k+1}), \ldots, y'(d_m))$$

with $d_j < t$ for $j = k + 1, \ldots, m$. Neutral problems are much harder to solve because discontinuities in $y(t)$ do not smooth out as the integration advances. Our solver, `ddesd`, has all the properties mentioned for these solvers but one—it does *not* accept problems of neutral type. As we shall see, its algorithms take important advantage of this restriction.

Most DDEs have a discontinuity in the first derivative at the initial point because the derivative of the history is not equal to the value given by the differential equation. We do not allow neutral problems and we further restrict the class of problems that we do allow by assuming that this initial discontinuity is smoothed as the integration progresses. Moreover, we assume that any discontinuity in the first derivative that occurs after the start can be handled as the initial point of a restart. As we shall see in §4, the user interface of `ddesd` makes restarts convenient. With these assumptions, we have a solution that is C^1 for each integration and correspondingly, we insist that our numerical solution also be C^1.

We do not track discontinuities in `ddesd`. This is practical because of the restrictions we place on the class of problems accepted by the solver and algorithmic developments that we discuss below. The users of ARCHI and DKLAG6 have the option of not tracking discontinuities. If the option is specified, the solvers rely upon the usual error estimation and step size adjustment algorithms for ODEs to deal with them. In this it is assumed that estimating the local error by comparing the results of a pair of formulas will recognize a discontinuity of low order and lead to a step failure. Standard step size selection algorithms take this possibility into account and deal with it in a reasonably efficient way. DDVERK detects discontinuities by means of a residual error control. Suspected discontinuities of low order are located and special interpolants are used when stepping over a discontinuity. We go further and rely entirely on residual control. Our aim is to minimize the need for a smooth solution. As with conventional algorithms for local error control, our algorithms for error estimation and step size adjustment deal effectively with isolated discontinuities. We expect discontinuities to be isolated because we expect smoothing. DDVERK must give more attention to this matter because it allows neutral problems for which discontinuities are not smoothed as the integration progresses.

When solving DDEs, it is important that we control the error everywhere because the delay terms might require evaluating the numerical solution anywhere prior to the current t. Because we do not track discontinuities, it is especially important that we have a robust measure of the error. In `ddesd` we use the method studied in §2.1, the classic four-stage, fourth-order explicit RK formula with cubic Hermite interpolation as continuous extension. One of the things to be appreciated about solving DDEs is that when the solution is defined implicitly, evaluating the formula implicitly involves a good many stages. The implicit formulas of §2.2 are attractive because they involve relatively few stages for a given order. Later we discuss more fully the evaluation of the formulas and explain our choice.

For the formulas of §2 we are able to obtain an asymptotically correct estimate of $\|r\|$ and $\|Le_n\|$ with a single evaluation of the residual. Certainly the connection we established between the residual and the local error is important, but in the present context we emphasize the residual because it is well-defined no matter how smooth the solution of the DDE. All the formulas we have considered have

$$\|Le_n\| = h_n C\|r\| + h.o.t. = h_n C\|r(x_n + \sigma^* h_n)\| + h.o.t.$$

for a known constant C. DDVERK and the BVP solvers MIRKDC and `bvp4c` control $\|r\|$. Though a perfectly reasonable approach to error, it is less familiar than control of $\|le_n\|$. We attempt to get the advantages of both in `ddesd`. We control $h_n\|r\|$ at each step. If all is going well, this also controls $\|Le_n\|$ and it is easy to interpret the tolerances. If not, we still have a meaningful control of error. For the method of `ddesd` the constant C is rather small, namely $\sqrt{3}/144$, meaning that the solver is quite conservative when the asymptotic approximations are applicable.

By default Enright and Hayashi estimate the size of the residual in DDVERK by evaluating the residual at a single point. This point is judiciously chosen, but it does not result in an asymptotically correct estimate for all problems. There is an option for doubling the number of samples to get a more reliable, but still not asymptotically correct, estimate. There is also an option for using a more expensive continuous extension for which an asymptotically correct estimate can be computed with one sample of the residual. All the continuous extensions we have considered provide an asymptotically correct estimate of the size of the residual with a single sample. This is more favorable than the schemes of DDVERK, but we should be more cautious in `ddesd` because we do not track discontinuities. An obvious way to enhance the reliability of the estimate is to evaluate the residual at both the points (4) where the residual asymptotically has its maximum magnitude and estimate the maximum magnitude of each component of the residual by

$$r_{max} = \max(|r(t_n + \sigma_1^* h_n)|, |r(t_n + \sigma_2^* h_n)|) \tag{9}$$

To get a credible estimate of the size of the residual when asymptotic results are poorly applicable, we should represent the residual somehow over the whole span of the step. Kierzenka and Shampine [9] estimate an L_2 norm of the residual in `bvp4c` with a Lobatto quadrature formula. Sufficiently many nodes (samples of the residual) are used that the estimate is asymptotically correct. An integral measure of size and a moderately high order quadrature formula furnish a credible estimate when the residual is not very smooth or the step size is rather large. We have experimented with such an estimate in conjunction with the first implicit formula of §2.1. The circumstances are less favorable than those of `bvp4c` with the consequence that the estimate is relatively expensive in terms of evaluations of the residual. The second implicit formula of §2.1 has a lower order and a residual of one sign, making it possible to obtain an asymptotically correct estimate of an L_1 norm with fewer samples. Though interesting, the quadrature formula is a much less plausible estimate of an L_1 norm when the residual might change sign.

In `ddesd` we obtain a plausible estimate of the maximum of $|r(t_n + \sigma h_n)|$ on $[0, 1]$ by interpolating $r(t_n + \sigma h_n)$ for $0, \sigma_1^*, \sigma_2^*, 1$ with a cubic polynomial $q(\sigma)$ and finding its maximum magnitude. Because the residual vanishes at the end points of the interval, the quadratic $q'(\sigma)$ has only real roots. This makes it straightforward to compute the roots in $(0, 1)$ and evaluate $q(\sigma)$ there to obtain the maximum of $|q(\sigma)|$. The scheme is practical in MATLAB because the computation can be vectorized over the components of the residual. We experimented with it, but came to prefer something much simpler and more cautious. If $L_1(\sigma)$ and $L_2(\sigma)$ are the fundamental Lagrangian interpolating polynomials associated with σ_1^* and σ_2^*, respectively, then

$$|q(\sigma)| = |r(t_n + \sigma_1^* h_n) L_1(\sigma) + r(t_n + \sigma_2^* h_n) L_2(\sigma)| \le C_5 \, r_{max}$$

Here r_{max} is defined by (9) and

$$C_5 = \max_{0 \le \sigma \le 1} |L_1(\sigma)| + \max_{0 \le \sigma \le 1} |L_2(\sigma)| \approx 2.1342$$

At each step `ddesd` uses this bound in controlling the size of $h_n\|r\|$ in a weighted L_∞ norm. If the asymptotic results are applicable, the bound is about twice as big as the actual norm, but we regard this as an acceptable price for a plausible and inexpensive bound on the size of the residual when the asymptotic results are not applicable.

The fact that a formula might be implicit complicates greatly the solution of DDEs. It is usual to extrapolate the continuous extension on the preceding step to the current one to get a tentative solution there. With it the delayed terms can be evaluated and an improved solution computed. Each iteration of this kind is rather expensive in terms of function evaluations. That is one reason we preferred a formula of relatively few stages, and correspondingly low order, in `ddesd`. Another is that in the MATLAB problem solving environment it is usual to interpret solutions graphically and so compute solutions to only modest accuracy. The other codes that we have cited iterate until the implicit formula is evaluated to a specified tolerance. This is complicated by state-dependent delays and the potential for discontinuities in the span of the step. In `ddesd` we have preferred the simpler approach of a fixed number of corrections. A difficulty with a straightforward implementation of iteration is that stages are evaluated with a previous iterate, hence the residual may not be exactly zero at the end points. We insist on this so as to obtain a numerical solution $S(t) \in C^1[a, b]$.

After experimenting with IRKs we decided in favor of an explicit formula because most steps are not implicit. A step from t_n begins by extrapolating the solution from the preceding step to get $\mathcal{P}(t) \approx y(t)$. As each stage is formed, we must evaluate the delay functions for a value of t and $\mathcal{P}(t)$. Although we require that each $d_j(t, y(t)) \leq t$, it is possible that some $d_m(t, \mathcal{P}(t)) > t$. If this should happen, we reduce the value of d_m to t so as to avoid going into the future. If $d_j(t, \mathcal{P}(t)) \leq t_n$ for all $j = 1, \ldots, k$, we treat the step as explicit, i.e., we do not iterate. Because this decision is made using an approximate solution, it is possible that we treat a step as explicit when it should have been treated as implicit. If any $d_m(t, \mathcal{P}(t)) > t_n$, we treat the step as implicit and iterate exactly once.

We define the numerical solution $S(t)$ so that its residual is exactly zero at both ends of the step. We sample the residual at two other points to compute an asymptotically correct bound on its weighted maximum norm over the span of the step. A considerable virtue of our approach is that we compute a meaningful estimate of the size of the residual, no matter how $S(t)$ is obtained. This important point deserves further comment: When the delay functions are state-dependent, the locations of discontinuities depend on the approximate solution. It is usual that these functions are evaluated using a previous iterate, but we evaluate them using the accepted $S(t)$, hence estimate the size of the residual of $S(t)$ itself.

4 User Interface

The user interface for `dde23` is convenient and powerful, so we have provided `ddesd` with an interface that is nearly identical. Indeed, it is possible to replace `dde23` in a program by simply changing the name of the solver to `ddesd`. Details of the design of `dde23` are found in [16, 17], so here we discuss only a few issues special to `ddesd`.

`dde23` solves only problems with constant lags τ_j so that $d_j = t - \tau_j$ for $j = 1, \ldots, k$. A convenient way to describe such delays is to provide the τ_j in an array `lags`, and that is done in `dde23`. For the general delay functions $d_j(t, y(t))$ allowed by `ddesd` it is more natural to describe the delays by means of a function, and that is the usual input for `ddesd`. In MATLAB an input argument can have more than one data type. By exploiting this, `ddesd` allows users to input an array of lags just as for `dde23` and internally evaluates the corresponding delay functions.

The solvers return the solution in the form of a structure. In this structure is all the information needed to evaluate the solution through the last point reached in the integration. Fields of the structure contain the mesh selected by the solver and the solution on this mesh. If these answers are insufficient, an auxiliary function `deval` can be used to evaluate the solution anywhere in the interval of integration. As seen in Example 4.4 of Oberle and Pesch [11], it is sometimes necessary to compute the first derivative of the solution. Values of the first derivative at mesh points are available as a field in the solution structure. This is usually satisfactory for `dde23`, but `ddesd` takes relatively long steps, a fact that may be obvious in a phase plane plot. Accordingly, we have added to `deval` the option of evaluating the first derivative of the solution computed by `ddesd`. (Because the same interpolant is used by `dde23` and the BVP solver `bvp4c`, we made the option available for these solvers, too.)

`dde23` tracks discontinuities. To facilitate this, it has an option called `Jumps` that allows a user to specify points where it is known that the solution has a low order discontinuity. This is not a natural option for `ddesd`, so if a user sets it, the solver returns with a message saying that the problem should be solved by restarting at each of the points specified by the option. The design also provides for event location. Often such problems involve restarting the integration after locating an event. Let us, then, briefly review how restarts are handled in the design of `dde23`. In the first instance, the history can be specified as either a constant vector or as a function. Either way, this information is saved in the solution structure along with the numerical solution. All the user has to do on a restart is provide the solution structure as the history argument. This structure is extended on output to the last point reached in the current integration. Though the implementation is somewhat complicated as the solver sorts through all the possibilities, the design makes restarts very easy for a user.

Chapter 4 of the text [17] discusses the solution of DDEs with constant lags. It provides examples showing how to use `dde23` and the associated Instructor's Manual contains programs for solving the computational exercises. We have solved all these problems with `ddesd`. This was almost completely straightfor-

ward. Naturally the programs illustrating use of the `Jumps` option had to be recoded as explained above. A good many of the programs plot the solution at mesh points. Because `ddesd` may take longer steps than `dde23`, we sometimes had to use `deval` to get answers on a mesh fine enough to get a smooth graph. This collection of problems furnished a good test of `ddesd` for DDEs with constant lags. In particular, it tested the solver on problems that involve discontinuities, event location, and restarting.

Each step in `ddesd` is more expensive than in `dde23`, but the method is of higher order. Because of this it is not surprising that `dde23` is generally the more efficient at default tolerances, but `ddesd` is the more efficient at stringent tolerances. To enhance the reliability of the algorithm in `ddesd` for handling discontinuities, it has been made rather conservative when all is going well. Tracking discontinuities in `dde23` is both efficient and effective for this class of DDEs. It also provides a sounder theoretical basis for this solver. For DDEs with constant lags, our experience is that `dde23` is generally to be preferred over `ddesd`. It is useful to have an alternative to `dde23`, but that is not why we developed `ddesd`. Our goal was to develop a MATLAB program to solve time– and state–dependent DDEs. The next section reports some numerical experiments with such problems.

5 Numerical Experiments

We have solved many time– and state–dependent DDEs from the literature. The results were plausible in all cases and agreed with analytical solutions and reference values as well as might be expected for the tolerances specified. Here we report some details for a few of the test problems for DDE solvers assembled by Enright and Hayashi [4]. The set includes neutral problems which, of course, we could not solve with `ddesd`, leaving 10 problems in four classes for our experiments. Enright and Hayashi use a single error tolerance on the residual. For `ddesd` we vary the relative error tolerance `RelTol` and take the absolute error tolerance `AbsTol` equal to `RelTol` $\times 10^{-3}$. The default tolerances in `ddesd` then correspond to `RelTol` equal to 10^{-3}.

In our first experiment we tested whether the solver is doing what it is supposed to do, namely controlling the size of the residual. We solved each of the ten problems for relative error tolerances $10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$ and computed the maximum over all steps of the ratio of $h_n\|r\|$ to the tolerance. To compute this statistic we approximated $\|r\|$ using 20 equally spaced samples in $[t_n, t_n + h_n]$. `ddesd` kept the size of the residual well below the specified tolerance in every case. Indeed, the worst performance was seen in solving C2 for which the residual overruns were $0.82, 0.69, 0.85, 0.83$, respectively. For all other problems and tolerances, the maximum residual overrun was 0.62 and the median was 0.30. The results of this experiment inspire some confidence that when `ddesd` returns a numerical solution, the solution will satisfy the differential equations with a residual no larger than the specified tolerance.

Class A consists of two problems with constant lags, so they can be solved

	Ssteps	Fsteps	Fevals	Time
dde23	238	76	943	0.24
ddesd	286	62	2087	1.0

Table 1: Statistics for problem A1.

	Ssteps	Fsteps	Fevals	Time
dde23	208	62	811	0.19
ddesd	276	2	1669	0.89

Table 2: Statistics for problem A2.

with dde23 as well as ddesd. We solved them with both programs using default tolerances. Plotting the numerical solutions together showed that the solvers produced the same solutions to graphical accuracy. By setting the Stats option to on we obtained the results displayed in Table 1 and Table 2. In these tables Ssteps is the number of successful steps, Fsteps is the number of failed attempts to take a step, Fevals is the number of derivative evaluations, and Time is the run time of the second invocation of the program.

Some of the problems in the test set have analytical solutions. B2 consists of a single DDE,

$$y'(t) = g(y(t/2)) - y(t)$$

Here $g(s) = 1$ if $s < 0$ and $g(s) = -1$ if $s \geq 0$. A discontinuous function of the delayed term presents obvious difficulties for a solver. The DDE is to be integrated over $[0, 2\log(66)]$ with initial value $y(0) = 1$. There is no history function because the delay vanishes at $t = 0$, another difficulty for a solver. The solution given in [4] is piecewise smooth on the interval of integration, but there are two discontinuities of very low order. To exemplify their testing program, Enright and Hayashi [4] present results for DDVERK applied to the A1 problem. In Table 3 we present similar results for ddesd applied to the B2 problem. The results they report are not directly comparable to ours because the codes do not have the same goals. We report the overrun in the residual, Res0, as explained above. Using the analytical solution we similarly compute and report the overrun in the global error, Err0. These two quantities are the equivalents for ddesd of the quantities "MAX DEF" and "MAX GLB ERR" reported by Enright and Hayashi for DDVERK. As it happens, the values $0.62, 0.60$ for the residual overrun are the biggest of our first experiment if we exclude those of C2. It must be kept in mind that global error overruns depend not just on how well the residual is controlled, but also on the stability of the problem itself.

D1 is a system of two equations with a state-dependent delay. The DDEs

$$\begin{aligned} y_1'(t) &= y_2(t) \\ y_2'(t) &= -y_2(e^{(1-y_2(t))})\, y_2^2(t)\, e^{(1-y_2(t))} \end{aligned}$$

RelTol	Ssteps	Fsteps	Fevals	ResO	ErrO
10^{-3}	55	22	464	0.62	0.89
10^{-4}	83	27	663	0.60	0.90
10^{-5}	128	37	988	0.30	1.5
10^{-6}	201	43	1463	0.35	9.9

Table 3: Statistics for problem B2.

RelTol	Ssteps	Fsteps	Fevals	ResO	ErrO
10^{-3}	37	1	235	0.13	0.50
10^{-4}	52	0	357	0.22	1.1
10^{-5}	90	0	605	0.28	2.2
10^{-6}	158	0	1041	0.31	3.5

Table 4: Statistics for problem D1.

are to be solved on $[0.1, 5]$ with history given by the analytical solution $y_1(t) = \log(t), y_2(t) = 1/t$. In addition to the difficulty of a state-dependent delay, this delay vanishes in the course of the integration, specifically at $t = 1$. Table 4 shows how `ddesd` performed in solving D1 for a range of tolerances.

6 Conclusions

Delay differential equations with time- and state-dependent delays are difficult to solve numerically and at this time, no approach is obviously the best. We have developed robust and inexpensive estimates of both the local error and the size of the residual for some Runge–Kutta methods with continuous extensions. Using an explicit method of this kind, we have developed a program, `ddesd`, to solve DDEs that is based on control of the size of the residual. The user interface of `ddesd` makes it easy to formulate and solve DDEs, even those with complications like event location and restarts. The program has performed well on a wide selection of problems from the literature.

References

[1] S.P. Corwin, D. Sarafyan, and S. Thompson, DKLAG6: a Code Based on Continuously Imbedded Sixth Order Runge–Kutta Methods for the Solution of State Dependent Functional Differential Equations, Appl. Numer. Math. 24 (1997) 319–333.

[2] W.H. Enright, A New Error–Control for Initial Value Solvers, Appl. Math. Comput. 31 (1989) 588–599.

[3] W.H. Enright and H. Hayashi, A Delay Differential Equation Solver Based on a Continuous Runge–Kutta Method with Defect Control, Numer. Alg. 16 (1997) 349–364.

[4] W.H. Enright and H. Hayashi, The evaluation of numerical software for delay differential equations, in: R.F. Boisvert, ed., The Quality of Numerical Software: Assessment and Enhancement, Chapman & Hall, London, 1997, pp. 179–192.

[5] W.H. Enright and P.H. Muir, Runge–Kutta Software with Defect Control for Boundary Value ODEs, SIAM J. Sci. Comput. 17 (1996) 479–497.

[6] I. Gladwell, L.F. Shampine, L.S. Baca, and R.W. Brankin, Practical Aspects of Interpolation in Runge–Kutta Codes, SIAM J. Sci. Stat. Comput. 8 (1987) 322–341.

[7] D.J. Higham, Robust Defect Control with Runge–Kutta Schemes, SIAM J. Numer. Anal. 26 (1989) 1175–1183.

[8] D.J. Higham, Runge–Kutta Defect Control using Hermite–Birkhoff Interpolation, SIAM J. Sci. Stat. Comput. 12 (1991) 991–999.

[9] J. Kierzenka and L.F. Shampine, A BVP Solver Based on Residual Control and the MATLAB PSE, ACM Trans. Math. Softw., 27 (2001) 299–316.

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

[11] H.J. Oberle and H.J. Pesch, Numerical Treatment of Delay Differential Equations by Hermite Interpolation, Numer. Math. 37 (1981) 235–255.

[12] C.A.H. Paul, A user-guide to ARCHI, Numer. Anal. Rept. No. 283, Maths. Dept., Univ. of Manchester, UK, 1995.

[13] L.F. Shampine, Interpolation for Runge–Kutta Methods, SIAM J. Numer. Anal. 22 (1985) 1014–1027.

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

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

[16] L.F. Shampine and S. Thompson, Solving DDEs in MATLAB, Appl. Numer. Math. 37 (2001) 441–458.

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

[18] H.J. Stetter, Considerations concerning a theory for ODE solvers, in: Numerical Treatment of Differential Equations, R. Bulirsch, R.D. Grigorieff, and J. Schroder, eds., Lecture Notes in Mathematics No. 631, Springer, New York, 1978, pp. 188–200.