Computing breaking points in implicit delay differential equations

Guglielmi, Nicola; Hairer, Ernst

How to cite

GUGLIELMI, Nicola, HAIRER, Ernst. Computing breaking points in implicit delay differential equations. In: Advances in computational mathematics, 2008, vol. 29, n° 3, p. 229–247. doi: 10.1007/s10444-007-9044-5

This publication URL: https://archive-ouverte.unige.ch//unige:5206
Publication DOI: 10.1007/s10444-007-9044-5

Computing breaking points in implicit delay differential equations [1,2]

Nicola Guglielmi
Dip. di Matematica Pura e Applicata, Università dell'Aquila,
via Vetoio (Coppito), I-67010 L'Aquila, Italy,
e-mail: guglielm@univaq.it
and
Ernst Hairer
Dept. de Mathématiques, Université de Genève,
CH-1211 Genève 24, Switzerland,
e-mail: Ernst.Hairer@math.unige.ch

Abstract

Systems of implicit delay differential equations, including state-dependent problems, neutral and differential-algebraic equations, singularly perturbed problems, and small or vanishing delays are considered. The numerical integration of such problems is very sensitive to jump discontinuities in the solution or in its derivatives (so-called breaking points).

In this article we discuss a new strategy - peculiar to implicit schemes - that allows codes to detect automatically and then to compute very accurately those breaking points which have to be inserted into the mesh to guarantee the required accuracy. In particular for state-dependent delays, where breaking points are not known in advance, this treatment leads to a significant improvement in accuracy. As a theoretical result we obtain a general convergence theorem which was missing in the literature (see [BZ03]). Furthermore, as a useful by-product, we design strategies that are able to detect points of non-uniqueness or non-existence of the solution so that the code can terminate when such a situation occurs. A new version of the code RADAR5 together with drivers for some real-life problems is available on the homepages of the authors.

Subject Classifications: AMS(MOS): 65L20; CR: G1.9.
Keywords: Implicit delay differential equations, Runge–Kutta methods, Radau IIA methods, breaking points, non-existence and non-uniqueness of solutions, error control, numerical well-posedness.

[1] Supported by the Italian M.I.U.R. and G.N.C.S.
[2] Supported by the Swiss National Science Foundation, project # 200020-101647.

1 Introduction

The time evolution of physical systems, whose rate of change also depends on the configuration at previous time instances, is often modelled in terms of delay differential equations. Their solution has a much more complicated dynamics than that for ordinary differential equations (lack of smoothness in the solution, possibility of non-uniqueness or non-existence), and it is rarely possible to find an analytic expression for the solution (see for example [BC63]). This motivates the need for reliable and efficient numerical integrators for such problems. For a comprehensive introduction to the numerical literature on this subject we refer the reader to the recent books [BZ03], [Bru05] and to the review-paper [BPW95].

In this article we consider the application of stiffly accurate collocation methods based on Radau nodes [HNW93] to systems of delay differential equations of the general (implicit) form

$$\begin{aligned} M\,y'(t) &= f\Big(t, y(t), y\big(\alpha(t, y(t))\big)\Big) \\ y(t_0) &= y_0, \qquad y(t) = g(t) \quad \text{for} \quad t < t_0, \end{aligned} \tag{1.1}$$

where, for ease of presentation, we consider only the case of a single deviating argument. The $d \times d$ matrix M in (1.1) is assumed to be constant but possibly singular, and $\alpha(t, y(t)) \le t$ for all $t \ge t_0$. This includes small and vanishing delays.

As discussed in [GH01], the formulation (1.1) covers various classes of problems. Singularly perturbed problems are included by putting $M = \mathrm{diag}(I, \varepsilon I)$ with a small parameter $\varepsilon > 0$. Introducing $z(t) = y'(t)$ as new variable (actually only for those components of the derivative that are present in the right-hand side function) neutral delay differential equations can be written in the form (1.1) (see Sections 5.3 and 5.4). It is worthwhile to remark that there are codes which solve neutral problems without formulating them as implicit systems (e.g. DDE-STRIDE by Baker, Butcher and Paul [But92] and SNDDELM by Jackiewicz and Lo [JL91]).

Since we allow M to be singular, (1.1) covers a variety of differential-algebraic delay equations. It should be mentioned that our code can solve efficiently index 1 problems. We expect that it can be adjusted to solve also index 2 and index 3 problems in Hessenberg form, as it is the case in the absence of delay terms.

The discussion in this paper is organized as follows. We first emphasize differences in the solution of delay equations compared to ordinary differential equations (Section 2). We explain how Runge–Kutta methods can be extended to problems of the form (1.1), and we briefly discuss the appearance of breaking points and situations where the solution can bifurcate or cease to exist. The new algorithms for the detection and computation of breaking points are presented in Section 3. A few theoretical results (Section 4) and numerical experiments (Section 5) with the new code conclude the discussion.

2 Differences to Ordinary Differential Equations

This section is devoted to discussing how implicit Runge–Kutta methods can be extended to treat problems of the form (1.1). Furthermore, some typical features of delay differential equations (not present in ordinary differential equations) are discussed.

2.1 Numerical Methods for Delay Equations

Consider an s-stage implicit Runge–Kutta method with abscissæ $\{c_i\}$, weights $\{b_i\}$ and coefficients $\{a_{ij}\}$. We assume that the method is stiffly accurate so that $b_i = a_{si}$ for $i = 1, \ldots, s$. Using the step size $h_n = t_{n+1} - t_n$ and an approximation y_n to the solution at time t_n, one step of the method is given by

$$M\left(Y_i^{(n)} - y_n\right) = h_n \sum_{j=1}^{s} a_{ij} f\left(t_n + c_j h, Y_j^{(n)}, Z_j^{(n)}\right), \qquad i = 1, \ldots, s \tag{2.1}$$

where $y_{n+1} = Y_s^{(n)}$ approximates the solution at time t_{n+1}. For ordinary differential equations, where f only depends on t, y and not on z, this represents a nonlinear system for the internal stage values $Y_i^{(n)}$. For delay equations, we still have to define $Z_j^{(n)}$ which has to be an approximation to $y\big(\alpha(t, y(t))\big)$ at $t = t_n + c_j h$. For this we let $\alpha_j^{(n)} := \alpha(t_n + c_j\, h_n, Y_j^{(n)})$ and define

$$Z_j^{(n)} = \begin{cases} g\left(\alpha_j^{(n)}\right) & \text{if } \alpha_j^{(n)} < t_0 \\ u_m\left(\alpha_j^{(n)}\right) & \text{if } t_m \le \alpha_j^{(n)} < t_{m+1} \end{cases} \tag{2.2}$$

for some $0 \le m \le n$. Here, $u_m(t)$ is the continuous output computed at the m-th step (which is available for $t_m \le t \le t_{m+1}$),

$$u_m(t_m + \vartheta h_m) = \sum_{i=0}^{s} \mathcal{L}_i(\vartheta) Y_i^{(m)} \qquad \vartheta \in [0, 1],$$

where $Y_0^{(m)} := y_m$, h_m is the step size used at the m-th step, $\mathcal{L}_i(\vartheta)$ is the Lagrange polynomial of degree s satisfying $\mathcal{L}_i(c_j) = \delta_{ij}$ with δ_{ij} the Kronecker delta symbol (here we add $c_0 = 0$ to the abscissas of the method).

In the first interval after points of discontinuity (which are either given in advance by the user or which are dedected as breaking points during the computation) the dense output polynomial can optionally be replaced by

$$v_m(t_m + \vartheta h_m) = \sum_{i=1}^{s} \mathcal{L}_i(\vartheta) Y_i^{(m)} \qquad \vartheta \in [0, 1],$$

which interpolates the internal stage values but not y_m (see Section 2 of [GH01]). The use of this option is important in the presence of a jump discontinuity in the solution. This permits to have also

a discontinuity in the dense output approximation of the solution. In general, one has $v_m(t_m) \neq y_m = u_{m-1}(t_m)$ whereas always $u_m(t_m) = y_m = u_{m-1}(t_m)$. The accuracy is thus significantly improved. This is discussed and illustrated in [GH01].

Note that in the case where $\alpha_j^{(n)} \in [t_n, t_{n+1}]$, i.e., where the delay is smaller than the current step size, $u_m\big(\alpha_j^{(n)}\big)$ and $v_m\big(\alpha_j^{(n)}\big)$ are not known explicitly but only implicitly (through the stage values $Y_1^{(n)}, \cdots, Y_s^{(n)}$ which are being computed).

This integration scheme, with the 3-stage Radau IIA method as basic integrator, has been realized in the code RADAR5. It is described in detail in [GH01]. The aim of the present article is to improve this code by including a careful treatment of breaking points.

2.2 Breaking Points

A serious difficulty, when integrating delay differential equations, is the possible loss of regularity of the solution even in the presence of smooth functions $f(t, y, z)$, $g(t)$, and $\alpha(t, y)$ in the problem (1.1).

If $y_0 \neq g(t_0)$, the solution is obviously discontinuous at t_0. However, even when $y_0 = g(t_0)$ there is no reason why the (right-hand) derivative $y'(t_0)$ which is given by the delay differential equation should be equal to the (left-hand) derivative $g'(t_0)$. This irregularity at t_0 may propagate along the integration interval by means of the deviating argument $\alpha(t, y(t))$. Evidently, as soon as $\alpha(\xi, y(\xi)) = t_0$ for some $\xi > t_0$, due to the fact that some derivative of $y(t)$ has a jump discontinuity at t_0, the function $f\big(t, y(t), y\big(\alpha(t, y(t))\big)\big)$ is not smooth at ξ. In general, this creates a further jump discontinuity in some derivative of the solution $y(t)$ at ξ, and so on. Such discontinuity points are referred in the literature as *breaking points* (see for example [BZ03]).

In the case of a constant delay $\alpha(t, y) = t - \tau$, the breaking points are $\xi_k = t_0 + k\tau$ for $k = 1, 2, \ldots$. If the matrix M in (1.1) is invertible, a jump discontinuity at t_0 leads in general to a jump discontinuity at ξ_1 in the first derivative of $y(t)$; this then causes a jump discontinuity at ξ_2 in the second derivative of $y(t)$, and so on. Only a few of these breaking points are significant for a numerical integration, because discontinuities in a sufficiently high derivative of the solution are not recognized by the numerical method. If the matrix M is singular instead, e.g., neutral equations or simply $0 = y(t) - \gamma y(t - \tau)$, the jump discontinuities may appear in the same derivative of the solution at all ξ_k. Depending on the value of γ, the size of the jump increases or decreases.

In the more general case, where α depends on t (but still not on the solution $y(t)$), the breaking points can be computed in advance by solving first the scalar equation $\alpha(\xi) = t_0$ for ξ, then for every solution ξ_k the scalar equation $\alpha(\eta) = \xi_k$, etc. This may be useful, if these scalar equations do not have too many solutions. For an efficient computation, computed breaking points can be inserted in advance into the mesh of integration. But in the general (so-called state-dependent) case such an a priori computation is not possible.

If the breaking points are not included in the mesh and a variable step size integration is used,

the step sizes may be severely restricted near the low order jump discontinuities. It is the aim of this paper to design an algorithm that allows one to compute automatically the breaking points and to include them in the mesh of integration. In this way, not only step rejections will be avoided, but also the accuracy of the approximation will be improved.

2.3 Non-Existence and Non-Uniqueness of Solutions

> "A discontinuity in any of the elements ... may turn out to the cause of branching or termination of the solution." [EN73]

If the deviating argument $\alpha(t, y)$ is state dependent, it is not always possible to extend the solution beyond breaking points and it may happen that the solution bifurcates into two solutions. Let us explain the ideas of these phenomena. For a series of interesting examples and for a rigorous discussion of this topic we refer to [EN73].

Assume that the solution $y(t)$ of (1.1) has a jump discontinuity at ζ (which typically is equal to t_0) and that the solution is well-defined and unique up to a breaking point $\xi > \zeta$ satisfying $\alpha(\xi, y(\xi)) = \zeta$. Further assume the existence of two smooth functions $\eta^+(t), \eta^-(t)$, defined on a neighbourhood of ζ, such that

$$\begin{aligned} \eta^+(t) &= y(t) \qquad \text{for} \quad t > \zeta \\ \eta^-(t) &= y(t) \qquad \text{for} \quad t < \zeta. \end{aligned} \tag{2.3}$$

For these two functions we consider the ordinary differential-algebraic equations

$$\begin{aligned} M z'(t) &= f\big(t, z(t), \eta^+\big(\alpha(t, z(t))\big)\big) \\ M z'(t) &= f\big(t, z(t), \eta^-\big(\alpha(t, z(t))\big)\big) \end{aligned} \tag{2.4}$$

with initial values $z(\xi) = Py(\xi)$, where P is a suitable projector if $y(\xi)$ is not yet consistent with the differential-algebraic equation (for invertible M, no projection is needed). They have unique solutions on a non-empty interval $(\xi, \xi + \varepsilon)$, and we denote them by $z^+(t)$ and $z^-(t)$, respectively. Neglecting situations where $\alpha(t, z^\pm(t)) = \zeta$ we have that (for $t > \xi$ sufficiently close to ξ)

$$z^+(t) \ \text{ is a solution of (1.1)} \qquad \text{iff} \qquad \alpha(t, z^+(t)) > \zeta \tag{2.5}$$

$$z^-(t) \ \text{ is a solution of (1.1)} \qquad \text{iff} \qquad \alpha(t, z^-(t)) < \zeta. \tag{2.6}$$

It can happen that both conditions, (2.5) and (2.6), are satisfied (bifurcation of the solution), that none of the conditions is satisfied (the solution ceases to exist), or that exactly one of the conditions is satisfied.

Example 2.1 *Following an example of [EN73] we consider the system*

$$\begin{aligned} \dot{y}_1(t) &= y_2(t) & y_1(t) &= 1 - t \qquad \textit{for} \ \ t \le 0 \\ 0 &= y_2(t) + y_2\big(y_1(t) - 2\big) & y_2(t) &= -1 \qquad \textit{for} \ \ t \le 0. \end{aligned} \tag{2.7}$$

It has the solution $y_1(t) = 1 + t$, $y_2(t) = 1$ *on the interval* $(0, 1)$, *and a breaking point at* $\xi = 1$ *which is created by the discontinuity at* $\zeta = 0$. *In a neighbourhood of* $\zeta = 0$ *we consider the smooth functions (cf. (2.3))*

$$\eta_2^+(t) = 1, \qquad \eta_2^-(t) = -1,$$

and the corresponding differential-algebraic equations which are obtained from (2.7) by replacing the term $y_2\big(y_1(t)-2\big)$ *with* $\eta_2^+\big(y_1(t)-2\big)$ *and* $\eta_2^-\big(y_1(t)-2\big)$, *respectively (cf. (2.4)). These problems have the solution*

$$z_1^+(t) = 3 - t, \quad z_2^+(t) = -1, \qquad z_1^-(t) = 1 + t, \quad z_2^-(t) = 1$$

close to $\xi = 1$. *Denoting* $\alpha(t, y) := y_1 - 2$ *we have*

$$\begin{aligned} \alpha(t, z^+(t)) &= 1 - t < 0 = \zeta \qquad \textit{for } \ t > 1 \\ \alpha(t, z^-(t)) &= -1 + t > 0 = \zeta \qquad \textit{for } \ t > 1, \end{aligned}$$

so that neither condition (2.5) nor condition (2.6) is satisfied. This proves that the solution of (2.7) terminates at $t = 1$.

3 Discussion of the New Algorithms

The efficiency of a code for solving delay and neutral differential equations can be improved significantly if breaking points are treated carefully. For constant delays and for delays $\alpha(t)$ not depending on the solution, all breaking points can be computed in advance and inserted into the mesh of integration. The insertion of all breaking points may lead to undesirable step size restrictions.

Let us mention here the interesting work by Willé and Baker [WB92] on the propagation of discontinuities in a system of delay differential equations. By means of an oriented graph, called dependency network, it is possible to model how discontinuities propagate in the system components in relation to the structure of the system. In this way it is possible to control the derivative order of a jump discontinuity and to detect the involved components. Differently from [WB92], in the present paper we do not compute and classify a priori breaking points, which would be further complicated by the presence of the mass matrix M, but we try to detect automatically, by error control mechanisms, those which are important in terms of the numerical integration.

The strategies of this section are mainly intended for state dependent delays, but they are also of interest for simpler situations, because only those breaking points will be computed that are relevant for obtaining a certain accuracy. Nevertheless, we remark that in the code RADAR5 it is also possible to choose the option to check for the presence of a breaking point at every step (also at accepted steps).

3.1 Detection of Breaking Points

To compute the set B of breaking points we start with $B = \{t_0\}$ and include possible discontinuities of the initial function $g(t)$ in (1.1) and of the right-hand side $f(t, \ldots)$ (see the example of Section 5.2). The problem is to find the zeros of the function

$$d_\zeta(t) = \alpha\big(t, u(t)\big) - \zeta, \tag{3.1}$$

where $\zeta \in B$ is a previous breaking point and $u(t)$ is a suitable approximation to the solution.

A naïve approach would be to check in every accepted step whether at least one of the functions $d_\zeta(t)$ changes sign. The breaking point can then be localized and added to the set B. Such a strategy is often expensive because step rejections, that usually occur when approaching a breaking point, are not avoided. Moreover, it does not tell us whether the localized breaking point is relevant for the actual computation. We propose the following strategy.

Algorithm 3.1 *Suppose that the problem is integrated successfully until t_n and a step size h_n is proposed for the next step. We expect a breaking point in the interval $[t_n, t_n + h_n]$ if the following two conditions occur:*

1. *the step is rejected, i.e., the iterative solver for the nonlinear system (2.1) fails to converge, or the local error estimate is not small enough,*
2. *there exists a previous breaking point ζ such that $d_\zeta(t) = \alpha(t, u_{n-1}(t)) - \zeta$ changes sign on $[t_n, t_n + h_n]$, where $u_{n-1}(t)$ is the continuous output polynomial of the preceding step.*

We emphasize that the search for breaking points is only activated in the case of a step rejection (an idea also used by [EH97a] for explicit Runge–Kutta methods). Since we are concerned with implicit Runge–Kutta methods this is a safe procedure (double check in convergence of the Newton iterations and in the error estimation) which is confirmed on a series of test problems. Moreover, this avoids the computation of those breaking points which do not affect the accuracy of the solution.

If one wants to be more cautious, one can check the condition (2) of Algorithm 3.1 also when the convergence of the iterative solver is slow or when the error estimate suddenly increases. This can be monitored by the user by setting a control parameter, and one can even force the code to check this condition in every step.

Remark 3.2 In the general case of multiple (say k) delays,

$$M\,y'(t) \quad = \quad f\Big(t, y(t), y\big(\alpha_1(t, y(t)), \ldots, \alpha_k(t, y(t))\big)\Big)$$

the algorithm is applied to all deviating arguments, one after the other, until a breaking point is possibly found. In such a case the code will attempt to compute it accurately, as explained in the

next section. If the deviating argument which changes sign is the i-th (with $1 \le i \le k$), then α_i will substitute α in equation (3.3) with ζ being the pertinent ancestor.

It is possible - in the presence of two (or more) breaking points in the interval $[t_n, t_n + h_n]$ - that the code does not immediately find the first breaking point. This will, however, not affect the correct behavior of the code because, in such a case, the new attempted step will be rejected and the first breaking point will be found by the subsequent application of Algorithms 3.1 and 3.3.

3.2 Computation of Breaking Points

Once a breaking point is detected by the Algorithm 3.1, i.e., a previous breaking point ζ is found such that a zero of $d_\zeta(t)$ is expected in the interval $[t_n, t_n + h_n]$, we are interested in its accurate computation. The idea is to consider the step size h_n as a variable in the Runge–Kutta system (2.1) and to determine it in such a way that $t_n + h_n$ exactly hits the breaking point. For this we consider the system

$$M\left(Y_i^{(n)} - y_n\right) = h_n \sum_{j=1}^{s} a_{ij} f\left(t_n + c_j h_n, Y_j^{(n)}, Z_j^{(n)}\right), \qquad i = 1, \dots, s \tag{3.2}$$

$$\alpha\big(t_n + h_n, u_n(t_n + h_n)\big) = \zeta \tag{3.3}$$

for the unknowns $Y_1^{(n)}, \dots, Y_s^{(n)}$, and h_n. Here, $u_n(t)$ is the dense output of the current step and depends on the stage values $Y_i^{(n)}$. For given h_n the system (3.2) is usually solved by simplified Newton iterations that exploit the structure of the system (see [GH01]). With the aim of not destroying this structure, we solve the system (3.2)-(3.3) with a *splitting* idea. Starting with $h_n = h_n^{[0]} := \xi^{(0)} - t_n$, where $\xi^{(0)}$ is an approximation to the breaking point (obtained from Algorithm 3.1), and with some approximation to the stage values, we apply the following algorithm.

Algorithm 3.3 *Consider the nonlinear system (3.2)-(3.3). To find its solution we proceed for* $k = 0, 1, \dots$ *until convergence as follows:*

1. *apply a simplified Newton iteration with respect to the variables* $Y_1, \dots, Y_s$ *and with fixed* $h_n = h_n^{[k]}$*; this yields a dense output approximation* $u_n^{[k]}(t_n + \vartheta h_n^{[k]})$ *for* $\vartheta \in [0, 1]$*;*
2. *with* $u_n(t)$ *replaced by* $u_n^{[k]}(t)$ *compute* $h_n^{[k+1]}$ *with a root finding algorithm applied to* (3.3)*;*

We terminate the iteration with a simplified Newton step.

The convergence of this algorithm can be very slow due to the following reason: since a breaking point ξ is expected in the current interval, some of the arguments in the computation of $Z_j^{(n)} = u_m(\alpha_j^{(n)})$ might be to the left of ζ and some to the right. The discontinuity of the dense output at ζ may prevent fast convergence or even lead to divergence. Only in the limit, all arguments will be at one side. To avoid this difficulty, we propose to use the extrapolated dense output also for

the arguments beyond ζ. With this modification (which is implemented in our code) the rate of convergence is linear but very fast as the following lemma shows.

Lemma 3.4 *Assume that, in addition to the smoothness of the functions in (1.1), also the mapping* $(t_n + c_j h_n, Y_j^{(n)}) \mapsto Z_j^{(n)}$ *is smooth. Furthermore, let* $u_n(h_n, t)$ *be the dense output solution with local error* $\mathcal{O}(h_n^{\sigma+1})$*, obtained from the solution of (3.2), and denote by* $\eta = \eta(h_n)$ *the solution of* $\alpha\big(t_n + \eta, u_n(h_n, t_n + \eta)\big) = \zeta$*, which we assume to be simple for all sufficiently small* h_n*. Then,*

$$|\eta(h_n) - \xi| \le \mathit{Const} \cdot h_n^{\sigma} \cdot |h_n - \xi|,$$

where $t_n + \xi$ *is the searched numerical breaking point, i.e.,* $\eta(\xi) = \xi$*.*

Proof.

We let $g(h_n, \eta) := \alpha\big(t_n + \eta, u_n(h_n, t_n + \eta)\big) - \zeta$, and we notice that $g(h_n, \eta(h_n)) = g(\xi, \xi) = 0$. Taylor expansion shows that

$$0 = g_{h_n}(\xi, \xi)\big(h_n - \xi\big) + g_{\eta}(\xi, \xi)\big(\eta(h_n) - \xi\big) + \text{higher order terms.} \tag{3.4}$$

By the nondegeneracy assumption of the breaking point, $g_\eta(\xi, \xi) \neq 0$ holds true. Hence, only the dependence on h_n of $g(h_n, \eta)$ and thus of $u_n(h_n, t_n + \eta)$ has to be studied for fixed η. By definition of the method, $u_n(h_n, t)$ is a polynomial of degree s interpolating y_n and $Y_1^{(n)}, \dots, Y_s^{(n)}$, which approximate the local solution $u_n(t)$ at $t_n + c_i h_n$ with an error of size $\mathcal{O}(h_n^{\sigma+1})$. As in standard convergence proofs for collocation methods (e.g., [HNW93, Sect. II.7]) we have (for $\sigma \le s$)

$$u_n(h_n, t) = u_n(t) + h_n^{\sigma+1}\, R\Big(h_n, \frac{t - t_n}{h_n}\Big),$$

where $R(h, \tau)$ is a smooth function of both variables. By the mean value theorem this implies $u_n(h_n, t) - u_n(\xi, t) = \mathcal{O}\big(h_n^{\sigma}(h_n - \xi)\big)$ for $t \in [t_n, t_n + h_n]$. Consequently, we also have $g(h_n, \eta) - g(\xi, \eta) = \mathcal{O}\big(h_n^{\sigma}(h_n - \xi)\big)$ which proves the statement by (3.4).

■

Remark 3.5 If Algorithm 3.1 wrongly predicts a breaking point in the current interval, then either the iteration of Algorithm 3.3 diverges or it converges to a step size that is larger than the original h_n (i.e., the breaking point lies outside the corresponding time-interval). In the first case, the step will be rejected and repeated with a reduced step size (according to classical step size strategies). In the second case, it depends on the local error estimate whether the step will be accepted or not.

The splitting technique of Algorithm 3.3 yields the breaking points with small overhead. Due to the fast convergence, only a few iterations are needed, and the main amount of work (simplified Newton iterations) is similar to that what has to be done anyway (that is the possible computation of the entries of the Jacobian of f and the factorization of the matrix involved in the simplified Newton iteration). The algorithm for computing the breaking point is thus the following:

Algorithm 3.6 *Suppose that a breaking point has been detected by Algorithm 3.1. We apply the splitting technique of Algorithm 3.3 to iteratively solve the augmented system (3.2)-(3.3). Then we proceed as follows:*

- *if the iterations converge and if the error control mechanism allows one to accept the step, we insert t_n+h_n into the set of breaking points B, and we continue the integration at $t_{n+1} = t_n+h_n$;*
- *otherwise, we reduce the step size according to classical criteria (e.g., [HNW93, Section II.3]), and we repeat the integration at t_n.*

Let us shortly compare our algorithm to previous strategies reported in the literature. The algorithm of [FN84] checks in *every* successful step if one of the functions $d_\zeta(t) = \alpha\big(t, u_n(t)\big) - \zeta$ changes sign. If this happens, the breaking point is computed as a zero of $d_\zeta(t)$. Notice that the use of $u_n(t)$ is dangerous, because on the current integration interval the solution is not smooth so that $u_n(t)$ can be a bad approximation. This algorithm is modified by [Hau97], where it is proposed to use $u_{n-1}(t)$ of the preceding step instead of $u_n(t)$. An improvement is reported, but using $u_{n-1}(t)$ outside the interval $[t_{n-1}, t_n]$ can also lead to large errors. The work of [EH97a] also uses this kind of 'extrapolation' to cross breaking points.

In contrast to most of the previous strategies, our approach computes only those breaking points that are relevant for the required accuracy (which saves unnecessary work and allows one to treat efficiently the situation where breaking points accumulate, e.g., the example of Section 5.2), and the breaking points are computed to high accuracy as will be explained in Theorem 4.1 below. For constant delays or for deviating arguments of the form $\alpha(t)$, the exact breaking points are found.

3.3 Detection of Non-Existence and Non-Uniqueness of the Solution

When the analytic solution stops to exist (cf. Section 2.3), a code which is not able to detect this occurrence typically terminates the integration after the step size has been reduced to some minimal value (at the price of a large number of rejected steps). If the solution bifurcates into two branches, the code typically follows one of them. Our algorithm for computing accurately breaking points allows us to detect points of non-existence and non-uniqueness of the solution. This is based on the discussion of Section 2.3.

Let $\xi = t_n$ and $\zeta = t_m$ be a numerical breaking point and its (generating) ancestor. Then, the polynomial functions $\eta_h^+(t) := v_m(t)$ and $\eta_h^-(t) := u_{m-1}(t)$ are well-defined in a neighbourhood of $\zeta = t_m$ and satisfy the discrete analogue of (2.3). With a small (with respect to h_{n-1}) step size $\varepsilon > 0$ we then compute one step of some numerical method applied to the differential-algebraic systems (2.4), where $\eta^+(s)$ and $\eta^-(s)$ are replaced by $\eta_h^+(s)$ and $\eta_h^-(s)$, respectively. The numerical results y_n^+ and y_n^- are then used to check the conditions in (2.5)-(2.6). Consequently, if both conditions

$$\alpha(t_n + \varepsilon, y_n^+) > t_m \qquad \text{and} \qquad \alpha(t_n + \varepsilon, y_n^-) < t_m \tag{3.5}$$

are satisfied, the solution bifurcates into two solutions at t_n; if none of them is satisfied, the solution stops at t_n. For the special case of ordinary delay differential equations ($M = I$, identity matrix), one can use the explicit Euler method to obtain

$$y_n^+ = y_n + \varepsilon f\big(t_n, y_n, v_m(t_m)\big), \qquad y_n^- = y_n + \varepsilon f\big(t_n, y_n, u_{m-1}(t_m)\big).$$

An even more interesting situation are neutral delay equations (see the example of Section 5.4) of the form

$$y'(t) = f\big(t, y(t), y\big(\alpha(t, y(t))\big), y'\big(\alpha(t, y(t))\big)\big). \tag{3.6}$$

In view of an application of our code, we write it in the from (1.1) by introducing a new variable $z(t) = y'(t)$ (it suffices to introduce such a variable only for those components which appear as derivatives in the right-hand side of (3.6)),

$$\begin{aligned} y'(t) &= f\big(t, y(t), y\big(\alpha(t, y(t))\big), z\big(\alpha(t, y(t))\big)\big) \\ 0 &= f\big(t, y(t), y\big(\alpha(t, y(t))\big), z\big(\alpha(t, y(t))\big)\big) - z(t). \end{aligned} \tag{3.7}$$

In this case we can check termination or bifurcation in a similar way using again (3.5). Denoting by $v_m^{[y]}(t)$, $v_m^{[z]}(t)$ and $u_{m-1}^{[y]}(t)$, $u_{m-1}^{[z]}(t)$ the dense output approximations to $y(t)$ and $z(t)$ of Section 2.1, we put this time

$$\begin{aligned} y_n^+ &= y_n + \varepsilon f\big(t_n, y_n, v_m^{[y]}(t_m), v_m^{[z]}(t_m)\big), \\ y_n^- &= y_n + \varepsilon f\big(t_n, y_n, u_{m-1}^{[y]}(t_m), u_{m-1}^{[z]}(t_m)\big). \end{aligned}$$

If exactly one of the conditions (3.5) is satisfied, so that the solution continues to exist, we suggest to go on with the integration with the right-hand limit of $z(t)$ at t_n: if $\alpha(t_n + \varepsilon, y_n^+)$ and $\alpha(t_n + \varepsilon, y_n^-)$ are both larger then t_m, it is given by $z_n^+ = f\big(t_n, y_n, v_m^{[y]}(t_m), v_m^{[z]}(t_m)\big)$, and if both expressions are smaller than t_m, it is $z_n^+ = f\big(t_n, y_n, u_{m-1}^{[y]}(t_m), u_{m-1}^{[z]}(t_m)\big)$. The use of these values improves the convergence of the Newton iterations for the first step after the breaking point.

In these two important situations, the non-existence and non-uniqueness check are activated in our code.

4 Theoretical Aspects

This section is devoted to highlighting the qualitative improvements implied by the algorithm for the accurate computation of breaking points.

4.1 Accuracy of the Computed Breaking Points

All algorithms for the computation of breaking points are based on a root finding method for the scalar function

$$d_{\zeta_h}(t) = \alpha\big(t, u_h(t)\big) - \zeta_h, \tag{4.1}$$

where ζ_h is some previously computed numerical breaking point, and $u_h(t)$ is an approximation to the exact solution of (1.1) on the interval $[t_n, t_{n+1}]$. For most integration methods, and in particular for collocation methods based on Radau nodes, it is observed that the error in the mesh points is usually much smaller than the error in the interior of the integration interval. It is therefore desirable that the zero of (4.1) coincides with the end of the integration interval. Algorithm 3.6 is designed to achieve this goal. We formalize this result in the following theorem.

Theorem 4.1 *Let $y(t)$ be the solution of (1.1), and let ζ and $t_n + \xi$ be exact breaking points of the problem such that $\alpha\big(t_n + \xi, y(t_n + \xi)\big) = \zeta$. Further, let ζ_h be an approximation to ζ obtained with sufficiently small step sizes, and let $t_n + \xi_h$ be the breaking point computed by the Algorithm 3.6, such that $t_n + \xi_h = t_{n+1}$. If*

$$\frac{d}{dt}\Big(\alpha\big(t, y(t)\big)\Big)\Big|_{t=t_n+\xi} \neq 0, \tag{4.2}$$

then we have

$$|\xi_h - \xi| \leq C\big(\|y_{n+1} - y(t_{n+1})\| + |\zeta_h - \zeta|\big).$$

Proof.

For given h_n, we denote the solution of the nonlinear system (3.2) by $Y_i^{(n)}(h_n)$. Since we consider only stiffly accurate Runge–Kutta methods, the numerical approximation at t_{n+1} is $y_{n+1}(h_n) := Y_s^{(n)}(h_n)$. The step size $h_n = \xi_h$ obtained from Algorithm 3.6 satisfies the equation (3.3) which can be written as $\alpha\big(t_n + \xi_h, y_{n+1}(h_n)\big) = \zeta_h$. With the function

$$F(\xi, e, \zeta) := \alpha\big(t_n + \xi, y(t_n + \xi) + e\big) - \zeta$$

we have $F(\xi_h, y_{n+1} - y(t_{n+1}), \zeta_h) = 0$ and $F(\xi, 0, \zeta) = 0$ for the exact breaking points. An application of the implicit function theorem thus proves the statement.

■

Often, the global error is expressed in terms of some power of the maximal step size h used during the integration. For the case $M = I$ (and for index 1 differential-algebraic problems) it is known that the global error of s-stage collocation methods is of size $\mathcal{O}(h^r)$, where $r = \min(p, s + 1)$ and p is the classical order of the method, e.g., $p = 2s - 1$ for the Radau IIA methods (cf. [HW96] and [BZ03]). The above theorem implies that the error in the computed breaking point is at most $\mathcal{O}(h^r)$.

Notice that for deviating arguments $\alpha(t)$ that do not depend on the solution, the accuracy of the computed breaking points only depends on the root finding algorithm for the equation $\alpha(\xi) = \zeta$ and not on the accuracy of the solution.

4.2 Considerations on the Global Error

For problems (1.1) with state dependent delays it is not possible to obtain reasonable error bounds for the global error $y_n - y(t_n)$. It may happen that t_n is a numerically computed breaking point, and the corresponding exact breaking point is slightly different. If at this point the solution has a jump discontinuity, the global error is there $\mathcal{O}(1)$. This is probably the reason why the convergence proofs of [BZ03] (e.g., that of Theorems 6.1.2 and 6.5.3) require that all (exact) breaking points up to a certain order are included in the mesh of integration.

It is possible to circumvent this difficulty by introducing a time transformation that is close to the identity. In the following we denote by $y_h(t)$ the numerical solution also if variable step sizes are employed. The value h represents the maximal step size.

Theorem 4.2 *Consider a smooth delay problem (1.1) which, on the considered interval, has only a finite number of breaking points and all of them satisfy (4.2). Furthermore, assume that the global error of the Runge–Kutta method is of size $\mathcal{O}(h^r)$ provided that all exact breaking points are included in the mesh.*

If, instead of the exact breaking points, those obtained by Algorithm 3.6 are used, then we have on bounded intervals

$$\|y_h(t) - y(\tau)\| \;=\; \mathcal{O}(h^r), \tag{4.3}$$

where the function $\tau = \tau(t)$ satisfies $\tau = t + \mathcal{O}(h^r)$.

We do not give a formal *proof* of this statement, because it requires to distinguish between the various problems that are covered in the formulation (1.1); namely, ordinary delay equations and delay differential-algebraic equations of various indices. We just outline the ideas of the proof which are common to all different types. As soon as one encounters the first breaking point larger than t_0, it may happen that the computed value ξ_h slightly differs from the exact breaking point ξ. By Theorem 4.1 we have the estimate $\xi_h - \xi = \mathcal{O}(h^r)$. Introducing a time transformation $t \leftrightarrow \tau$ which is $\mathcal{O}(h^r)$ close to the identity and which maps $[t_0, \xi_h]$ onto $[t_0, \xi]$ (e.g., a linear transformation), we obtain (4.3) for $t \le \xi_h$. The interval from ξ_h to the next computed breaking point has to be treated in the same way, and so on (this happens a finite number of times). Finally the standard convergence proof (propagation of local errors and accumulation of these errors) has to be applied; see for example Theorem 4.3.5 of [BZ03].

5 Numerical Experiments

With the new version of our code we have successfully and efficiently solved all problems of the test sets of [Pau94] and [EH97b], in particular those where breaking points arise. In this final section we

illustrate the behaviour and performance of the algorithm developed in this work at some selected typical examples.

5.1 Scalar State Dependent Delay Equation

We first consider Problem 1.3.10 of [Pau94]. It is a state dependent scalar problem with known solution on the considered interval. The equation is

$$y'(t) = y\big(y(t)\big) \qquad \text{for} \qquad t \geq 2 \tag{5.1}$$

with initial condition $y(t) = 0.5$ for $t < 2$, and $y(2) = 1$. The discontinuity of the solution at $\xi_0 = 2$ creates breaking points at $\xi_1 = 4$ and $\xi_2 = 4 + 2\ln 2 \approx 5.386$. The exact solution is $y(t) = t/2$ for $\xi_0 \leq t \leq \xi_1$, $y(t) = 2\exp(t/2 - 2)$ for $\xi_1 \leq t \leq \xi_2$, and $y(t) = 4 - 2\ln(1 + \xi_2 - t)$ for $\xi_2 \leq t \leq 5.5$.

Figure 1 illustrates the difference of a code that does not include breaking points in the mesh (code A, left picture) and one where the algorithms of the previous sections are activated (code B, right picture). To get the same accuracy of about five digits, we apply the code A with tolerance $Tol = 10^{-7}$, and the code B with $Tol = 10^{-4}$. As initial step size we take $h = 0.01$. As can be seen from the left picture of Fig. 1, a large number of steps are needed with code A to overcome breaking points, whereas with the code B the breaking points are hit exactly. For the integration over the interval $[2, 5.5]$, the code A requires 30 accepted steps and has 20 step rejections (15 on the left neighbourhood of the first breaking point); the code B gives the same accuracy with 7 accepted and 3 rejected steps. Two of them are necessary to detect the breaking points, and the other is due to error estimation. It is interesting to mention that for tolerances $Tol < 4 \cdot 10^{-1}$ all breaking points are captured with high accuracy.

Table 1 presents a comparison between the previous and the new version of our code for various

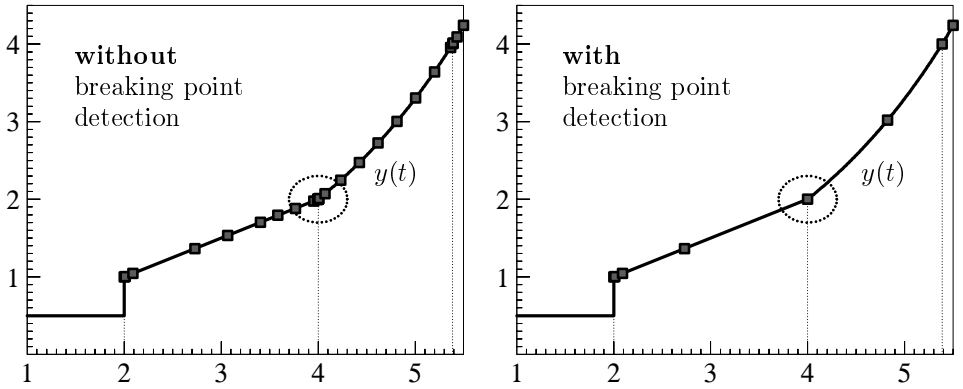

Figure 1: Numerical solution of problem (5.1) obtained by the code RADAR5 with breaking point computation (right) and without it (left).

	RADAR5 – old version				RADAR5 – new version			
Tol	feval	accept	reject	error	feval	accept	reject	error
10^{-3}	175	12	9	$2.8 \cdot 10^{-4}$	80	7	4	$1.6 \cdot 10^{-5}$
10^{-6}	355	24	17	$5.6 \cdot 10^{-6}$	120	13	4	$7.5 \cdot 10^{-9}$
10^{-9}	691	55	29	$2.2 \cdot 10^{-7}$	207	24	5	$9.5 \cdot 10^{-10}$
10^{-12}	1390	152	33	$3.4 \cdot 10^{-10}$	473	62	5	$8.8 \cdot 10^{-14}$

Table 1: Comparison between old and new version of RADAR5 for problem (5.1).

tolerances. The abbreviation 'feval' stands for the number of function evaluations, 'accept' and 'reject' for the numbers of accepted and rejected steps, and 'error' for the relative error at the endpoint of integration. In the comparison we choose $Atol = Rtol = Tol$ and take $h = 0.01$ as initial step size. The old version has many step rejections (increasing with the accuracy) and requires many accepted steps for overcoming the breaking points. For the new version the number of rejected steps is independent of the required accuracy, and a higher accuracy is obtained with much fewer function evaluations.

5.2 Real-Life System With State Dependent Delays

An interesting system of delay differential equations with state dependent delays is that of [Wal78], which has been used as test equation in [GH01]. We refer to the latter publication for the description of the system, the parameters, and the initial conditions. The only exception is that we use the function $f_2(y, w) = y + w$ instead of $f_2(y, w) = 10^{-12} + y + w$ (according to Waltman the term 10^{-12} is "an artifact" and was included only for numerical purposes). This is a demanding problem and many codes have severe difficulties in solving it.

The right-hand side of the differential equation has jump discontinuities at $t_0 = 35$ and $t_1 = 197$, but the solution is continuous and has jumps only in its derivatives. There are two state dependent delays $\alpha_1(t, y(t)) = y_5(t)$ and $\alpha_2(t, y(t)) = y_6(t)$. The first delay is constant on $[0, t_0]$ and then it monotonically approaches t (nearly vanishing delay). The second one is constant on the interval $[0, t_1]$, then it has an extremely steep slope and rapidly approaches t (see Fig. 3 of [GH01]). Breaking points for the solution are obviously t_0 and t_1 which are included in the mesh.

The new version of our code RADAR5 successfully solves this problem on the interval $[0, 300]$ for all tolerances. A comparison between the old version (which does not compute automatically breaking points) and the new one is given in Table 2. Note that for the highest tolerance the algorithm implemented in the previous version has a premature termination due to an excessive stepsize restriction.

	RADAR5 – old version		RADAR5 – new version	
Tol	feval	error	feval	error
10^{-3}	2800	0.778	2227	0.218
10^{-6}	4244	1.05e-2	3409	6.85e-4
10^{-9}	8537	2.48e-4	7939	3.32e-6
10^{-12}	—	—	22694	3.66e-8

Table 2: Comparison between old and new version of RADAR5 for the Waltman problem.

The discontinuity at $t_0 = 35$ generates breaking points

$$55.2132518, \qquad 69.2671816, \qquad 79.6396056, \qquad 87.5583983, \quad \ldots$$

with increasing degree of differentiability. Depending on the tolerance, only a few of them are computed by our code and inserted into the mesh.

Due to the nearly vanishing delays in the problem, there is a huge amount of breaking points beyond $t_1 = 197$. Using the option which searches for breaking points in every accepted step and applying the code with stringent tolerance 10^{-14}, it detects 45 breaking points in the tiny interval $(197, 197 + 10^{-5})$. We do not recommend the use of this option for practical simulations. Any code that tries to compute all breaking points will be inefficient for this problem, because it has to take too many steps.

5.3 Neutral System With Constant Delay

We next consider a neutral problem taken from [Kua91], see also Problem 2.4.3 of [Pau94] or Problem E2 of [EH97b], which here is written in the form (1.1) with singular $M = \mathrm{diag}(1, 1, 0)$ as

$$\begin{aligned} y_1'(t) &= y_1(t)\big(1 - y_1(t-\tau) - \rho y_3(t-\tau)\big) - y_2(t)F(y_1(t)) \\ y_2'(t) &= y_2(t)\big(F(y_1(t)) - \alpha\big) \\ 0 &= y_1(t)\big(1 - y_1(t-\tau) - \rho y_3(t-\tau)\big) - y_2(t)F(y_1(t)) - y_3(t). \end{aligned} \tag{5.2}$$

We have $F(y) = y^2/(y^2+1)$, $\alpha = 0.1$, $\rho = 2.9$, $\tau = 0.42$, and the initial functions are $y_1(t) = 0.33 - t/10$ and $y_2(t) = 2.22 + t/10$ for $t \le 0$. To see that this is a neutral problem, notice that $y_1'(t) = y_3(t)$ and eliminate the variable y_3 from the system. Equation (5.2) has constant delay and the breaking points are therefore known in advance as $\xi_k = k\tau$, $k = 1, 2, \ldots$. The size of the jumps in these breaking points can be seen in Fig. 2, where the derivative $y_3(t) = y_1'(t)$ is plotted on the intervals $[0, 6]$ and $[24, 30]$. It decreases slowly with time.

To test the code in the presence of jump discontinuities, we apply it without including the breaking points into the mesh. Over the integration interval $[0, 6]$ all breaking points are detected and correctly

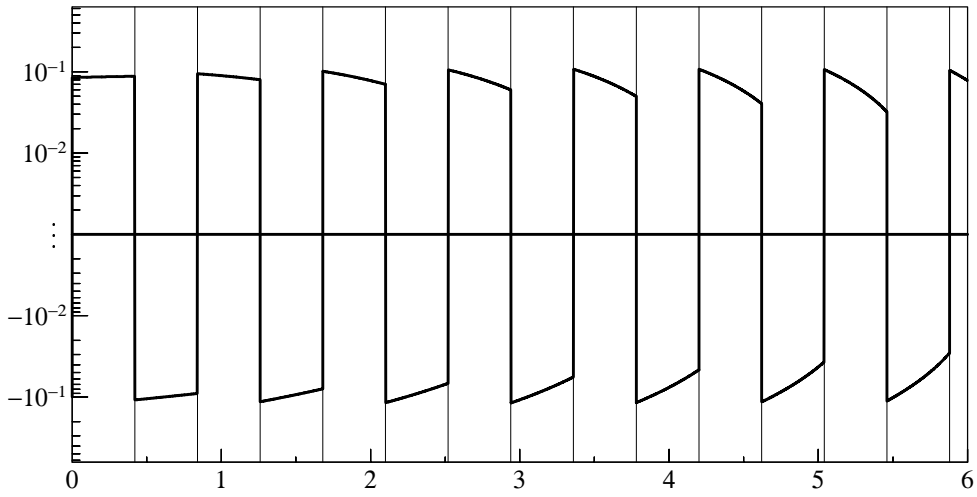

Figure 2: The component $y_3(t)$ of the solution of the differential-algebraic delay system (5.2). Vertical lines indicate the positions of the breaking points.

computed by the code, even for very rough tolerances like $Tol = 1$. On the longer interval $[0, 30]$, 35 out of 71 breaking points are found when the code is applied with $Tol = 10^{-1}$, 54 when it is applied with $Tol = 10^{-2}$, and all 71 breaking points are found for tolerances $Tol \leq 10^{-3}$.

It is important to mention that in the presence of jump discontinuities in the solution, care has to be taken with the computation of the right-sided limit of the solution at the breaking point. This can be done by a suitable projection of the left-sided limit onto the set of consistent initial values for the differential-algebraic equation. For the case of problem (5.2), one simply has to replace the third component by the value obtained from the algebraic relation in (5.2).

	RADAR5 – old version				RADAR5 – new version			
Tol	feval	accept	reject	error	feval	accept	reject	error
10^{-3}	1049	98	30	$4.0 \cdot 10^{-3}$	2644	186	138	$4.4 \cdot 10^{-5}$
10^{-6}	7411	676	292	$3.0 \cdot 10^{-5}$	4592	369	130	$1.2 \cdot 10^{-7}$
10^{-9}	16112	1505	513	$1.6 \cdot 10^{-7}$	10063	918	123	$1.0 \cdot 10^{-9}$
10^{-12}	32023	2977	752	$2.5 \cdot 10^{-7}$	27556	2665	121	$2.4 \cdot 10^{-11}$

Table 3: Comparison between old and new version of RADAR5 for problem (5.2).

Similar to Table 1, we present in Table 3 a comparison between the old and new versions of our code. The column 'error' gives the relative error of the Euclidean norm of the solution vector $(y_1(t), y_2(t))$ at the end point of integration $T = 30$. For both versions the accuracy of the derivative $y_1'(t)$ is typically less by two or three digits. With the exception of the "lucky run" of the old version

with $Tol = 10^{-3}$, we can draw the same conclusion as before. The number of step rejections and the total number of function evaluations are reduced, and the accuracy is significantly improved.

5.4 Neutral System With State Dependent Delay and Terminating Solution

Our final example is a modification by [EH97a] of a problem originally considered by [CG73],

$$y'(t) = \cos(t)\Big(1 + y\big(t\,y^2(t)\big)\Big) + c\,y(t)\,y'\big(ty^2(t)\big) + g(t) \tag{5.3}$$

with $g(t) = (1-c)\sin t\,\cos\big(t\,\sin^2 t\big) - \sin\big(t + t\,\sin^2 t\big)$ and initial value $y(0) = 0$. With this initial value, it has $y(t) = \sin t$ as exact solution and there are no discontinuities and no breaking points. We therefore modify it as follows: we put $g(t) = 0$ and $c = 0.6$ in the delay equation (5.3), and we consider it for $t > 0.25$. As initial functions we take

$$y(t) = -t/2, \quad y'(t) = -1/2 \quad \text{for} \quad t \le 0.25. \tag{5.4}$$

This is an explicit neutral problem with state dependent delay and very challenging for a numerical integrator. Figure 3 shows its solution together with its derivative $y'(t)$ and the retarded argument $\alpha(t, y(t)) = ty^2(t)$. It has breaking points $\{\xi_i\}$ at (rounded to nine digits)

1.16655385, 2.63630258, 3.26643820, 3.49770380 3.70029694, 3.96003956

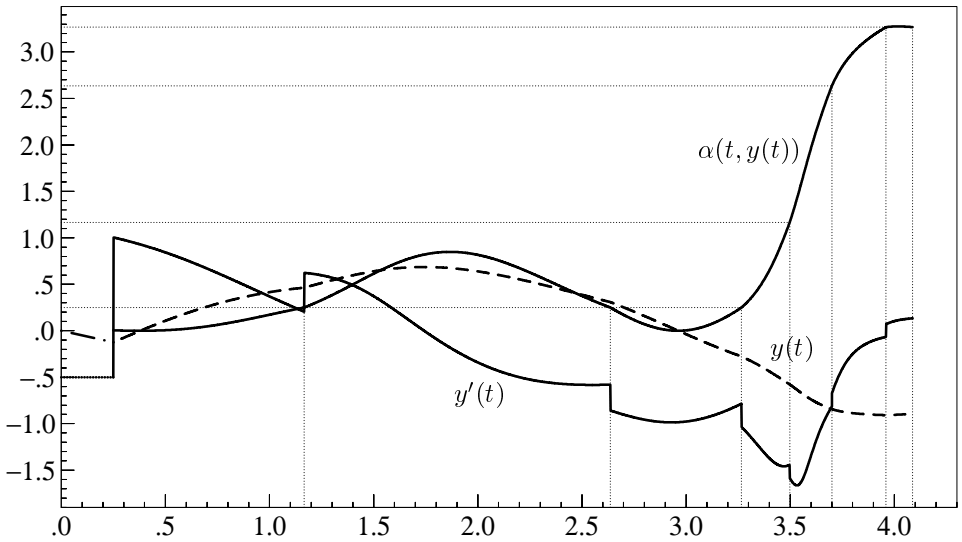

Figure 3: The solution $y(t)$ as broken line, its derivative $y'(t)$, and the retarded argument $\alpha(t, y(t)) = ty(t)^2$ of the neutral problem (5.3). Vertical and horzontal lines indicate the positions of the breaking points and the terminating point.

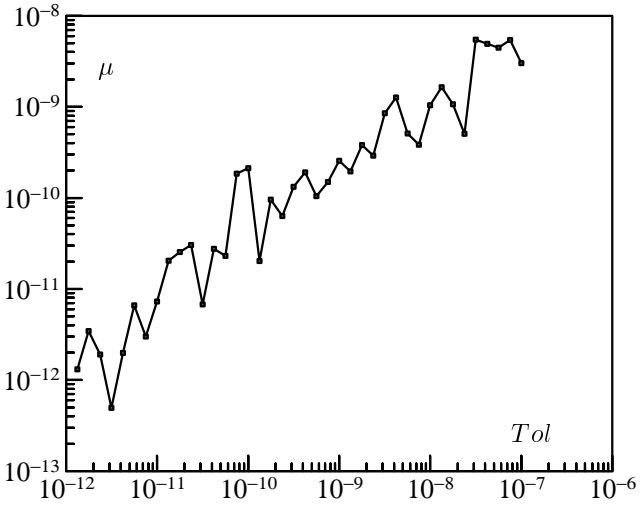

Figure 4: Mean accuracy of computed breaking points: in horizontal axis we represent the input tolerance and in the vertical axis we represent the quantity μ.

and the solution terminates at $\xi_7 \approx 4.09218182$. The position of these points is indicated by vertical lines in Fig. 3, and their ancestors can be found by following the horizontal lines passing through $(\xi, \alpha(\xi, y(\xi)))$.

The new version of our code successfully computes the solution. For tolerances smaller than 10^{-7} all breaking points are obtained with high accuracy and the termination point is detected correctly.

The diagram in Fig. 4 shows a measure (in logarithmic scale) of the accuracy of computed breaking points in terms of the input tolerance. In particular we draw the following quantity:

$$\mu = \frac{1}{6} \sum_{i=1}^{6} \frac{|\bar{\xi}_i - \xi_i|}{|\xi_i|}$$

where $\{\xi_i\}$ indicate exact breaking points (here we use instead highly accurate approximations) and $\{\bar{\xi}_i\}$ denote the breaking points computed by the code with the chosen input tolerance.

The observed behavior shows a very good correspondence between the required and the obtained accuracy.

5.5 Availability of the Software

A new release of the code is presently being distributed at the web-sites

http://univaq.it/~guglielm

http://www.unige.ch/~hairer/software.html

(where the first released version is actually available). It contains drivers for nine examples including problems simulating enzyme kinetics, acute hepatitis B virus infection, and the problems of the present paper.

Acknowledgments

We thank the referees for their comments and suggestions.

Most of the work described in this article has been developed during the stay of the first author at the University of Geneva. Nicola Guglielmi wishes to thank Ernst Hairer and Gerhard Wanner for making this visit possible.

References

[BC63] R. Bellman and K.L. Cooke. *Differential-difference equations*. Academic Press, New York, 1963.

[BPW95] C.T.H. Baker, C.A.H. Paul, and D.R. Willé. Issues in the numerical solution of evolutionary delay differential equations. *Adv. Comp. Math*, 3:171–196, 1995.

[Bru05] H. Brunner. *Collocation methods for Volterra integral and related functional differential equations*. Cambridge University Press, Cambridge, 2005.

[But92] J.C. Butcher. The adaptation of stride to delay differential equations. *Appl. Numer. Math.*, 9:415–425, 1992.

[BZ03] A. Bellen and M. Zennaro. *Numerical methods for delay differential equations*. Oxford University Press, Oxford, 2003.

[CG73] R. Castleton and L. Grimm. A first order method for differential equations of neutral type. *Math. Comp.*, 27:571–577, 1973.

[EH97a] W.H. Enright and H. Hayashi. A delay differential equation solver based on a continuous Runge-Kutta method with defect control. *Numer. Algorithms*, 16:349–364, 1997.

[EH97b] W.H. Enright and H. Hayashi. The evaluation of numerical software for delay differential equations. In *The quality of numerical software:assessment and enhancements*, pages 179–192. Chapman and Hall, London, 1997.

[EN73] L.E. El'sgol'ts and S.B. Norkin. *Introduction to the theory and application of differential equations with deviating arguments*. Academic Press, New York, 1973.

[FN84] A. Feldstein and K.W. Neves. High order methods for state-dependent delay differential equations with nonsmooth solutions. *SIAM J. Numer. Anal.*, 21:844–863, 1984.

[GH01] N. Guglielmi and E. Hairer. Implementing Radau-IIA methods for stiff delay differential equations. *Computing*, 67:1–12, 2001.

[Hau97] R. Hauber. Numerical treatment of retarded differential-algebraic equations by collocation methods. *Adv. Comput. Math.*, 7:573–592, 1997.

[HNW93] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1993.

[HW96] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*, volume 14 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1996.

[JL91] Z. Jackiewicz and E. Lo. The numerical solution of neutral functional differential equations by adams predictor-corrector methods. *Appl. Numer. Math.*, 8:477–491, 1991.

[Kua91] Y. Kuang. On neutral delay logistic Gause-type predator-prey systems. *Dynam. Stability Systems*, 6(2):173–189, 1991.

[Pau94] C.A.H. Paul. A test set of functional differential equations, 1994. Numerical Analysis Report 243, Univ. Manchester.

[Wal78] P. Waltman. A threshold model of antigen-stimulated antibody production. In *Theoretical immunology*, volume 8 of *Immunology Ser.*, pages 437–453. Dekker, New York, 1978.

[WB92] D.R. Willé and C.T.H. Baker. The tracking of derivative discontinuities in systems of delay differential equations. *Appl. Numer. Math.*, 9:209–222, 1992.