Implementing Radau IIA methods for stiff delay differential equations

N. Guglielmi[*], L'Aquila, and **E. Hairer**, Geneva

February 5, 2001

Abstract

This article discusses the numerical solution of a general class of delay differential equations, including stiff problems, differential-algebraic delay equations, and neutral problems. The delays can be state dependent, and they are allowed to become small and vanish during the integration. Difficulties encountered in the implementation of implicit Runge-Kutta methods are explained, and it is shown how they can be overcome. The performance of the resulting code – RADAR5 – is illustrated on several examples, and it is compared to existing programs.

AMS Subject Classifications: 65L06, 65Q05, 34K28

Keywords: Stiff delay differential equations, neutral problems, Runge-Kutta methods, implementation, step size control, numerical comparisons.

1 The Class of Considered Problems

We consider initial value problems of delay differential equations

$$\begin{aligned} M\,y'(t) &= f\Big(t, y(t), y(\alpha_1(t, y(t))), \ldots, y(\alpha_m(t, y(t)))\Big), \\ y(t_0) &= y_0, \qquad y(t) = g(t) \quad \text{for } t < t_0, \end{aligned} \tag{1}$$

[*]Partially supported by the Italian M.U.R.S.T. (project: "Numerical methods for evolutionary problems") and I.N.D.A.M.-G.N.I.M. (project: "Numerical methods for ordinary differential equations and applications").

where M is a constant $d \times d$ matrix and $\alpha_i(t, y(t)) \le t$ for all $t \ge t_0$ and for all i. The value $g(t_0)$ may be different from y_0, allowing for a discontinuity at t_0.

The presence of the matrix M in the problem formulation has several reasons. If a partial delay differential equation is discretized in space by finite elements, we obtain an equation of the form (1) where M is the mass matrix. A multiplication of the equation with M^{-1} would destroy the sparsity pattern of the problem and is not recommended.

Since we allow the matrix M to be singular, the above formulation includes all kinds of differential-algebraic delay equations. For $M = \mathrm{diag}(I, \varepsilon I)$ with a very small $\varepsilon > 0$, we get singularly perturbed problems, which form an important class of stiff problems. Moreover, neutral problems

$$y'(t) = f\Big(t, y(t), y(\alpha(t, y(t))), y'(t), y'(\alpha(t, y(t)))\Big)$$

can be written in the form (1), if we introduce a new variable $z(t) = y'(t)$ for the derivative. In fact, this problem becomes equivalent to

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y'(t) \\ z'(t) \end{pmatrix} = \begin{pmatrix} z(t) \\ z(t) - f\Big(t, y(t), y(\alpha(t, y(t))), z(t), z(\alpha(t, y(t)))\Big) \end{pmatrix}.$$

In an implementation, the special structure of the right-hand side can be exploited on the linear algebra level during the solution of arising nonlinear systems.

We are aware of the fact that, without any further assumptions, the problem (1) need not have a solution, but it is beyond the scope of this article to discuss questions on the existence and uniqueness of solutions. We simply assume throughout that a solution exists on the considered interval of integration (possibly with discontinuities).

The aim of this article is to show how implicit Runge-Kutta methods, in particular collocation methods based on Radau nodes, can be applied to solve problems of type (1). Many aspects of the implementation are a straightforward extension of ideas implemented in the code RADAU5 for ordinary differential equations, and described in Hairer & Wanner [8, Sect. IV.8]. Here we restrict our discussion to new difficulties that are due to the presence of small delays, and to large elements in the derivative of f with repect to the retarded arguments. Numerical experiments and comparisons with other approaches conclude this article.

2 Numerical Method

> A standard fourth-order Runge-Kutta . . . was used, although after the simulation was begun, it was apparent that a program suitable for "stiff" systems

would have been more convenient. (P. Waltman 1978 [11])

Our main interest is the numerical solution of stiff delay differential equations, so that explicit methods are excluded. Since collocation methods based on Radau nodes have been successfully applied to stiff ordinary differential equations (see the code RADAU5 of [8]), and since these methods have excellent stability properties also for delay equations (see for example Zennaro [13] and Guglielmi & Hairer [6]), it is quite natural to take them as a basis for a code solving stiff problems of the type (1).

For ease of presentation, we assume that only one lag term is present ($m = 1$) and that the problem is autonomous:

$$M\, y'(t) \;=\; f\Big(y(t), y(\alpha(t, y(t)))\Big), \tag{2}$$

where $\alpha(t, y(t)) \le t$, $y(t_0) = y_0$, and $y(t) = g(t)$ for $t < t_0$. The actual code handles multiple delays and is not restricted to autonomous systems.

Radau IIA methods are implicit Runge-Kutta methods, whose coefficient matrix $A = (a_{ij})$ is invertible, and whose weights satisfy $b_i = a_{si}$ ("stiff accuracy"). We denote $c_i = \sum_j a_{ij}$ (see [8] for more information on these methods). For an implementation we consider a grid $t_0 < t_1 < t_2 < \ldots$, and we denote the stepsize by $h_n = t_{n+1} - t_n$. An application of the Radau IIA methods to the problem (2) yields approximations $y_n \approx y(t_n)$ by solving the nonlinear system

$$M\Big(Y_i^{(n)} - y_n\Big) = h_n \sum_{j=1}^{s} a_{ij} f\Big(Y_j^{(n)}, Z_j^{(n)}\Big), \qquad y_{n+1} = Y_s^{(n)}, \tag{3}$$

where $Z_i^{(n)}$ is a suitable approximation to $y(\alpha_i^{(n)})$ with $\alpha_i^{(n)} = \alpha(t_n + c_i h_n, Y_i^{(n)})$. We put

$$Z_i^{(n)} = \begin{cases} g(\alpha_i^{(n)}) & \text{if } \alpha_i^{(n)} < t_0 \\ u_m(\alpha_i^{(n)}) & \text{if } \alpha_i^{(n)} \in [t_m, t_{m+1}], \end{cases} \tag{4}$$

where $u_m(t)$ is a polynomial approximation of the solution $y(t)$ on the interval $[t_m, t_{m+1}]$. A natural choice for u_m is the collocation polynomial, which is of degree s and passes through the values y_m, and $Y_i^{(m)}$ for $i = 1, \ldots, s$. Using the Lagrange interpolation formula it is seen to be of the form

$$u_m(t_m + \tau h_m) = \ell_0(\tau) y_m + \sum_{i=1}^{s} \ell_i(\tau) Y_i^{(m)}, \tag{5}$$

where $\ell_i(\tau)$ is the polynomial of degree s satisfying $\ell_i(c_i) = 1$ and $\ell_i(c_j) = 0$ for $j \neq i$ (here we add $c_0 = 0$ to the nodes $c_1, \ldots, c_s$ of the method).

Order of Convergence. If the matrix M is invertible, and if the delay is larger than the step size (i.e., $t - \alpha(t, y(t)) \geq h$), it follows from the standard theory for ordinary differential equations that the local error at grid points is $\mathcal{O}(h^{2s})$, and that the internal stages $Y_i^{(n)}$ approximate the local solution at $t_n + c_i h_n$ with an error of size $\mathcal{O}(h^{s+1})$. In the case of a state-dependent delay this gives an additional $\mathcal{O}(h^{s+2})$ contribution to the local error (due to the multiplication by h_n in (3)). Therefore, on bounded intervals, the global error is of size $\mathcal{O}(h^{s+1})$. For stiff problems, a further order reduction to $\mathcal{O}(h^s)$ is possible, which is in complete analogy to stiff ordinary differential equations (see e.g., [8, Chap. VI]).

If the delay is smaller than the step size, the theory for ordinary differential equations can no longer be applied, and a more involved analysis is necessary. Such a study is beyond the scope of this paper. However, we want to emphasize that a general purpose code for stiff delay equations should be able to allow for step sizes larger than the delay, because it is known that stiff solvers are efficient only if 'large' step sizes can be used. Difficulties in an implementation that arise due to the presence of large step sizes, are discussed in Sect. 3.

Discontinuities in the Solution. Since the considered class of problems contains differential-algebraic delay equations and neutral problems as special cases, discontinuities at t_0 need not be smoothed out. Consider for example the problem

$$0 \;=\; -y(t) + cy(t-1) \tag{6}$$

for $t > 0$ and $y(t) = g(t)$ for $t \leq 0$. On the interval $(k-1, k]$ the exact solution is $y(t) = c^k g(t-k)$. If $g(0) \neq cg(-1)$, the solution has a discontinuity of size $c^k(g(0) - cg(-1))$ at $t = k$ (see Fig. 1, left picture).

If we apply method (3) to the problem (6), we obtain the exact values $y_{n+1} = y(t_{n+1})$ and $Y_i^{(n)} = y(t_n + c_i h_n)$ for the numerical solution as well as for the internal stages, as long as $t_{n+1} \leq 1$. However, the collocation polynomial $u_m(t)$ of (5) is a very bad approximation of the solution on the very first interval $[t_0, t_1]$ (see Fig. 1, left). Therefore, in the first interval to the right of $t = 1$ the numerical solution y_n will be completely wrong, unless the collocation polynomial is only evaluated at the nodes $t_m + c_i h_m$. This is the case only in the unrealistic situation, where a constant step size $h = 1/k$ is used such that the delay is an integer multiple of the step size. Figure 1 (right) also illustrates that the same behaviour can be observed for stiff problems $\varepsilon y'(t) = -y(t) + cy(t-1)$, if $\varepsilon \to 0$.

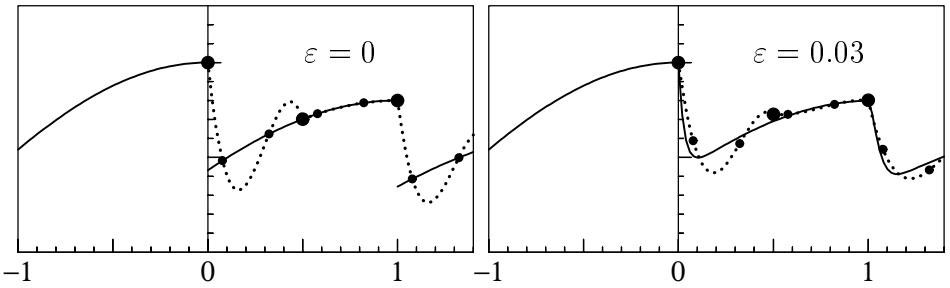

Figure 1: Solution of $\varepsilon y'(t) = -y(t) + 0.8y(t-1)$ with $y(t) = \cos t$ for $t \le 0$. The numerical solution of the 3-stage Radau IIA method ($h = 0.5$) is indicated by big bubbles, the internal stages by small ones; the dotted line is the collocation polynomial (5).

As a remedy for this difficulty we consider also the polynomial

$$v_m(t_m + \tau h_m) = \sum_{i=1}^{s} \ell_i(\tau) Y_i^{(m)} \tag{7}$$

of degree $s-1$, which interpolates the values $Y_i^{(m)}$ but not y_m. It can optionally be used in the first interval after points of discontinuity (which have to be given in advance by the user of the code). A careful estimation of the error in the continuous solution is therefore very important and will be the subject of Sect. 4.

3 Solving the Nonlinear Equations

An efficient solution of the nonlinear equations (3) is the most demanding part of an implementation of implicit Runge-Kutta methods. For stiff problems (or when M is singular), this system cannot be solved by fixed point iteration, and one is obliged to use some kind of simplified Newton iterations.

As common in the implementation of implicit Runge-Kutta methods, we premultiply the system (3) by $A^{-1} = (\omega_{ij})$ and so obtain the nonlinear system $F(Y) = 0$, where $Y = (Y_1, \ldots, Y_s)^T$ and the ith component of $F(Y)$ is

$$F_i(Y) \;=\; \sum_{j=1}^{s} \omega_{ij} M(Y_j - y_n) - h f(Y_i, Z_i) \tag{8}$$

(here, we suppress the subscript n when it does not give rise to confusion). In view of an application of simplified Newton iterations we compute

$$\frac{\partial F_i}{\partial Y_j} \;=\; \omega_{ij} M - h\delta_{ij}\Big(f_y(Y_i, Z_i) + f_z(Y_i, Z_i) u_m'(\alpha_i)\alpha_y(t_n + c_i h, Y_i)\Big), \tag{9}$$

where we have to add the term

$$-hf_z(Y_i, Z_i)\ell_j(\sigma_i) \qquad \text{if} \quad \sigma_i := (\alpha(t_n + c_i h, Y_i) - t_n)/h > 0.$$

For an efficient implementation, we replace the exact Jacobian of the nonlinear system with an approximation which, written in tensor notation, is given by

$$A^{-1} \otimes M - hI \otimes \Big(f_y + f_z u'_m(\alpha)\alpha_y\Big) - hL \otimes f_z. \tag{10}$$

The arguments of f_y, f_z, α, and α_y are choson independent of i. The $s \times s$ matrix L has elements given by

$$l_{ij} = \begin{cases} \ell_j(\sigma_i) & \text{if } \alpha(t_n + c_i h, Y_i) > t_n \\ 0 & \text{else.} \end{cases}$$

For an implementation we distinguish the two situations:

Step size is smaller than the delay or, more precisely, if $\sigma_i \le 0$ for $i = 1, \ldots, s$. For a constant delay (i.e., $\alpha(t, y) = t - \tau$), this happens if and only if $h \le \tau$ (because $c_i \le c_s = 1$). In this situation, the matrix L vanishes identically, and the matrix (10) has exactly the same structure as for ordinary differential equations $My' = f(y)$. Transforming the matrix A^{-1} to diagonal form, the linear system with matrix (10) can be solved efficiently as described in [8, Sect. IV.8]. The user of our code can either provide a subroutine with the analytic expression of $f_y(y, z) + f_z(y, z)y'(\alpha(t, y))\alpha_y(t, y)$ (where $z \approx y(\alpha(t, y(t)))$), or he can choose the option of computing it internally by numerical differentiation. Observe that the nasty second term of this matrix is only present for state-dependent delays.

Step size is larger than the delay. In this case, the matrix L in (10) is non-zero. Since, in general, the matrices A^{-1} and L are not simultaneously diagonalizable, the tensor product structure cannot easily be exploited for an efficient solution of the linear systems with matrix (10). However, when the delay is very small compared to the step size (i.e., $\alpha(t_n + c_i h, Y_i) \approx t_n + c_i h$), we get $\sigma_i \approx c_i$, and the matrix L becomes close to the identity. Consequently, the second and third terms in (10) can be considered together, and the idea of diagonalizing the matrix A^{-1} can again be applied.

In conclusion, we adopt the following strategy for approximating the Jacobian of the nonlinear Runge-Kutta equations: if $\alpha(t_n + c_i h, Y_i) \le t_n$ for at least one i, we let $L = 0$, otherwise we put $L = I$ (identity) in the Jacobian approximation of (10). In both cases standard techniques for stiff ordinary differential equations can

be applied, and the tensor product structure of the linear system can be exploited. In case of difficulties in the convergence of the simplified Newton iterations we use the correct L. This, however, requires the LR decomposition of the full matrix (10), which is about 5 times as expensive for the 3-stage method ($s = 3$).

4 Local Error Estimation and Step Size Control

Step size selection strategies for stiff ordinary differential equations are usually based on error estimations at grid points. For delay equations, where the accuracy of the dense output strongly influences the performance, such an approach is not sufficient. We shortly present the technique used in RADAU5 [8, Sect. IV.8], and we discuss a modification suitable for delay equations.

Standard error estimators for ordinary differential equations are based on embedded methods. Assuming for the moment M to be invertible, this leads to

$$\Delta y_n = hM^{-1} f(y_n, z_n) + \sum_{i=1}^{s} e_i (Y_i^{(n)} - y_n), \tag{11}$$

where the coefficients e_i are chosen such that $\Delta y_n = \mathcal{O}(h^{s+1})$ whenever the problem and the solution are smooth. For very stiff problems, the expression Δy_n largely overestimates the true local error, and as a remedy it is pre-multiplied once or twice with a kind of projection matrix:

$$P = (M - h\gamma(f_y + \ldots))^{-1} M, \tag{12}$$

where γ is a real eigenvalue of the Runge-Kutta matrix A. Whenever the tensor product structure in (10) is exploited, an LR decomposition of the matrix $M - h\gamma(f_y + \ldots)$ is already available from the simplified Newton iterations. We remark that the estimates $P\Delta y_n$ as well as $P^2\Delta y_n$ are meaningful also if the matrix M is singular.

Let us illustrate the disadvantage of using this error estimation for step size control. Consider the problem (6), which can be considered as the limit of $\varepsilon y'(t) = -y(t) + cy(t-1)$, where $\varepsilon \to 0$. Since $M = 0$, we get $P^2\Delta y_n = 0$. This is perfect as long as one is interested only in approximations at grid points, because there the local error is zero. However, for delay equations, where also the dense output is used for stepping forward, such an error estimation is not acceptable (see Fig. 1).

Estimation of the error in the dense output. In most steps we use the collocation polynomial $u_n(t)$ of (5) as dense output. In order to get an idea of its error, we

consider its difference to the lower degree polynomial $v_n(t)$ of (7), which is also a continuous approximation to the solution. The maximum difference is at the left endpoint t_n of the considered subinterval, so that

$$\mathit{err}_2 = v_n(t_n) - y_n \tag{13}$$

estimates this error.

Our code RADAR5 uses a combination of both error estimators in order to select the step sizes. This combination is based on heuristic arguments without deep theoretical foundations. Since it is still in an experimental stage, we do not present its details here.

5 Numerical Experiments

The code RADAR5 can handle delay equations of the form (1). It can be downloaded at the internet address

http://www.unige.ch/math/folks/hairer/software.html

Chemical Reaction with Delay. The Robertson problem [8, Sect. IV.10] is one of the most famous test problems for stiff solvers. We consider here the following modification:

$$\begin{aligned} y_1'(t) &= -0.04\, y_1(t) + 10^4\, y_2(t-\tau)\, y_3(t) \\ y_2'(t) &= 0.04\, y_1(t) - 10^4\, y_2(t-\tau)\, y_3(t) - 3\cdot 10^7\, y_2(t)^2 \\ y_3'(t) &= 3\cdot 10^7\, y_2(t)^2 \end{aligned} \tag{14}$$

with $\tau = 10^{-2}$ and initial values $y_1(0) = 1$, $y_2(0) = 0$, $y_3(0) = 0$, and $y_2(t) = 0$ for $t < 0$. This is a good test problem, if we consider it on a very large time interval, say on $[0, 10^{11}]$ (see Fig. 2). Since the exact solution tends to a steady state, an efficient code has to be able to increase the step size exponentially, so that in most steps the delay is much smaller than the step size.

Our code RADAR5 adapts the step size automatically to this situation. For example, with *Rtol* $= 10^{-6}$ and *Atol* $= 10^{-10}\cdot$*Rtol* the step size is about $h \approx 10^{-5}$ in the beginning and $h \approx 8\cdot 10^8$ at the end of the integration interval.

Artificial Problem with Vanishing Delay. Our second example, which is a modification by Enright & Hayashi [4] of a problem considered originally by Castleton

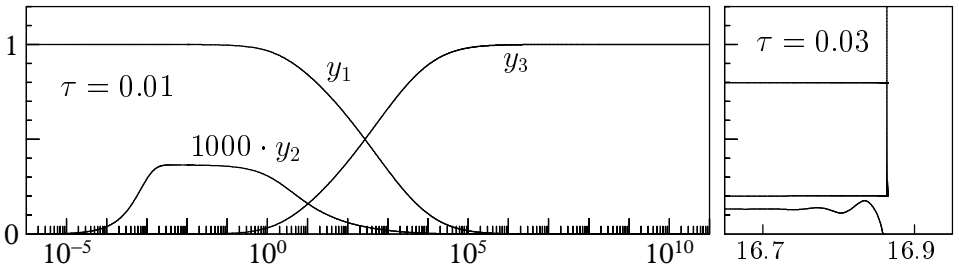

Figure 2: Solution of the modified Robertson problem (14) with $\tau = 0.01$. For the larger delay $\tau = 0.03$ (right picture) the component y_2 starts to oscillate, becomes negative, and explodes close to $x \approx 16.8$.

& Grimm [3], is

$$\begin{aligned} y'(t) &= \cos(t)\Big(1 + y(t\,y^2(t))\Big) + c\,y(t)\,y'(ty^2(t)) \\ &\quad + (1-c)\,\sin t\,\cos(t\,\sin^2 t) - \sin(t + t\,\sin^2 t) \end{aligned} \tag{15}$$

with initial value $y(0) = 0$. For every choice of the parameter c, it has $y(t) = \sin t$ as exact solution. It has a vanishing delay at $t = 0, \pi/2, 3\pi/2, \ldots$, and for $c = 1$ it has a singularity at $t = \pi/2$ (i.e., $y'(\pi/2)$ is not well defined by the equation (15). For this problem, the numerical solution of the nonlinear Runge-Kutta equations causes some difficulties.

We have rewritten the neutral equation (15) in the form (1) by introducing the new variable $z(t) = y'(t)$ as explained in Sect. 1, and we have applied our code RADAR5 with *Rtol* = *Atol* = 10^{-8}. Table 1 shows the results for different values of the parameter c. We display the total number of steps (accepted and rejected), the number of steps where difficulties appeared in solving the nonlinear system (so that the correct matrix L had to be used in (10)), and the global error at the endpoint of integration. We observe that such difficulties appear only for values close to ± 1. It is somewhat surprising that our code solves the problem correctly even for $c = 1$ (the singular case). We remark that the code of [4], which is based on fixed point iterations for the solution of nonlinear equations, solves this problem only for values of c in the region $-0.45 \le c \le 0.65$.

Threshold Model for Antibody Production. This is an interesting delay differential equation which models the antibody response to antigen challenge (Waltman [11]). The problem consists of six equations

$$y_1'(t) = -r y_1(t) y_2(t) - s y_1(t) y_4(t)$$

Table 1: Statistics for the problem (15) with $\mathit{Rtol} = \mathit{Atol} = 10^{-8}$

c	nr. of steps	nr. of full Jac	error at $t = \pi$
-1.0	127	12	$0.18 \cdot 10^{-8}$
-0.7	111	6	$0.42 \cdot 10^{-8}$
-0.3	99	0	$0.17 \cdot 10^{-9}$
0.0	91	0	$0.12 \cdot 10^{-8}$
0.3	120	2	$0.10 \cdot 10^{-8}$
0.7	144	14	$0.53 \cdot 10^{-9}$
1.0	164	19	$0.43 \cdot 10^{-7}$

$$
\begin{aligned}
y_2'(t) &= -r y_1(t) y_2(t) + \alpha r y_1(y_5(t)) y_2(y_5(t)) H(t - t_0) \\
y_3'(t) &= r y_1(t) y_2(t) \\
y_4'(t) &= -s y_1(t) y_4(t) - \gamma y_4(t) + \beta r y_1(y_6(t)) y_2(y_6(t)) H(t - t_1) \\
y_5'(t) &= H(t - t_0) f_1(y_1(t), y_2(t), y_3(t)) / f_1(y_1(y_5(t)), y_2(y_5(t)), y_3(y_5(t))) \\
y_6'(t) &= H(t - t_1) f_2(y_2(t), y_3(t)) / f_2(y_2(y_6(t)), y_3(y_6(t)))
\end{aligned}
\tag{16}
$$

where $\alpha = 1.8$, $\beta = 20$, $\gamma = 0.002$, $r = 5 \cdot 10^4$, $s = 10^5$, $t_0 = 35$, $t_1 = 197$, $H(x)$ is the Heavyside function ($H(x) = 0$ if $x < 0$ and $H(x) = 1$ if $x \geq 0$), $f_1(x, y, w) = xy + w$, and $f_2(y, w) = 10^{-12} + y + w$. The initial values and initial functions are given by $y_1(t) = 5 \cdot 10^{-6}$, $y_2(t) = 10^{-15}$, and $y_3(t) = y_4(t) = y_5(t) = y_6(t) = 0$ for $t \leq 0$.

This problem has several difficulties: the delay is state-dependent, it becomes very small and vanishes asymptotically (see the right picture of Fig. 3). The functions y_2 and y_4, y_6 are extremely steep at the values $t = 35$ and $t = 197$, respectively, and the problem is very stiff (as already observed in [11]). Indeed, the second equation of (16) is closely related to that considered in Fig. 1 with $\varepsilon = r^{-1}$, because $y_1(t)$ is nearly constant on a long interval (compare the second component of Fig. 3 with the right picture of Fig. 1). Since the size of the components is very small, one has to be careful with the scaling. The solution in Fig. 3 is obtained by RADAR5 with $\mathit{Rtol} = 10^{-9}$ and $\mathit{Atol} = 10^{-12} \cdot \mathit{Rtol}$ for the first four components, and $\mathit{Atol} = \mathit{Rtol}$ for the components y_5 and y_6. The discontinuities due to the Heavyside functions are harmless, if one declares the points t_0 and t_1 as gridpoints. None of the other codes, described in Sect. 6 below, can solve this problem.

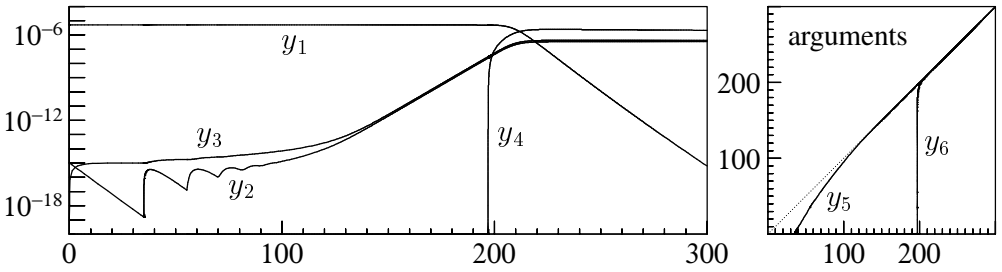

Figure 3: Solution components of the problem (16). The retarded arguments $y_5(t)$ and $y_6(t)$ are compared to t in the right picture.

Driver programs for all problems of this section are provided with the code RADAR5. We also included driver programs for further interesting delay differential equations: the hepatitis B virus infection model of [2], a chemical reaction (oregonator) proposed in [5], an enzyme kinetics problem of Okamoto & Hayashi [10] (see also [7, p. 348]), and a nonsmooth artificial problem.

6 Related Software and Conclusion

To our knowledge there is no code available that can solve general differential-algebraic delay equations of the form (1). However, there are several programs dealing with stiff delay equations or with problems admitting vanishing delays. We experimented with the following codes:

- DELH by Weiner & Strehmel [12]. This code is based on Rosenbrock type methods for the integration of partitioned stiff/nonstiff systems. It is restricted to one constant delay.
- DDE-STRIDE by Baker, Butcher & Paul [1], which is an adaptation of the code STRIDE to delay differential equations and neutral problems. This code can solve stiff problems with several state-dependent delays (including vanishing delays), but it is restricted to problems of the form (1) with $M = I$ (identity).
- DIFSUB-DDE by Bocharov, Marchuk & Romanyukha [2]. This is an extension of the code DIFSUB by Gear (based on BDF) to delay differential equations with constant delays.
- DDVERK by Enright & Hayashi [4]. This is a code for nonstiff delay equations with possible vanishing delays. It is based on explicit Runge-Kutta methods, and the nonlinear equations (arising for small delays) are

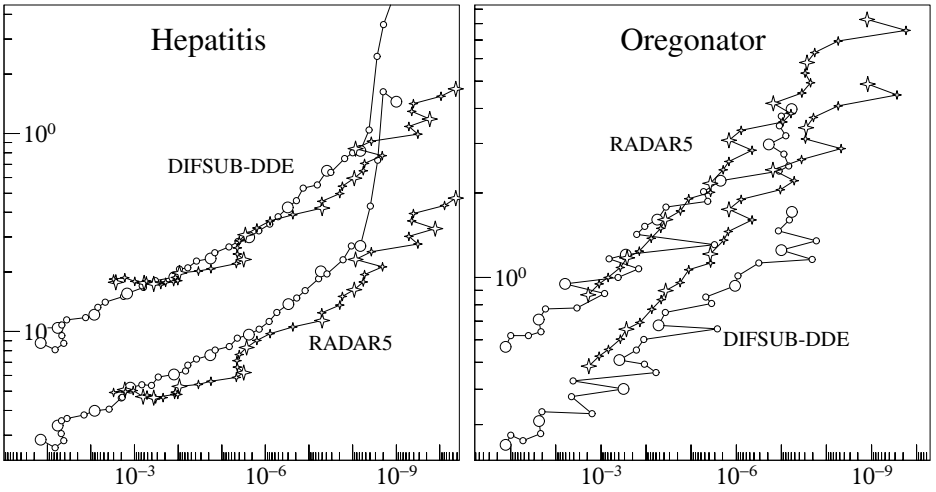

Figure 4: Work-precision diagram. We plot the cpu time on a SUN Ultra 10 work station (vertical axis) against the error of the solution at the end point of integration (horizontal axis). All runs are made with and without optimizing the compilation.

solved by fixed point iterations. This code is available from the netlib site at http://www.netlib.org/ode/ddverk.f

- SNDDELM by Jackiewicz & Lo [9]. This code solves nonstiff neutral delay equations with state-dependent (possibly vanishing) delays. It is based on Adams predictor-corrector methods and solves the problem (15) correctly for all $|c| \leq 1$.

We thank the authors of these codes for permitting us to use them for our experiments.

Since most of the codes for stiff delay equations (including ours) are still in an experimental stage, it is probably too early to make serious comparisons. Nevertheless we present some of our experiments in Fig. 4. We have applied RADAR5 and DIFSUB-DDE with many different tolerances (ranging from 10^{-1} for the Hepatitis problem and 10^{-4} for the Oregonator up to 10^{-11} for both problems). The results of RADAR5 are indicated by stars, those for DIFSUB-DDE by circles. Large symbols indicate tolerances with integer powers of 10. We have included the results obtained with and without the standard optimization option of the f90 compiler. Not only the accuracy of the numerical results are affected for some tolerances, but also the improvement (due to the optimization option) is seen to depend on the method and on the particular problem.

Hepatitis B Infection Model. This is a very stiff problem consisting of 10 equations with 5 constant delays. It is completely described in [2]. Here we consider the interval $[0, 130]$, and we use *Atol* $= 10^{-20} \cdot$ *Rtol*, because the components are very small. Both codes give satisfactory results, as can be seen in the left picture of Fig. 4.

Oregonator. This problem, taken from [5], consists of two equations and one constant delay. The equations are given by

$$y_1'(t) = 0.0804\, y_2(t) - 1.6 \cdot 10^9\, y_1(t)\, y_2(t-\tau) + 480\, y_1(t) - 8 \cdot 10^7\, y_1(t)^2$$
$$y_2'(t) = -0.0804\, y_2(t) - 1.6 \cdot 10^9\, y_1(t)\, y_2(t-\tau) + 480\, y_1(t),$$

the initial data are $y_1(t) = 10^{-10}$ and $y_2(t) = 10^{-5}$ for $t \leq 0$, and the constant delay is $\tau = 0.06$. We consider the integration interval $[0, 100.5]$ and *Atol* $= 10^{-9} \cdot$ *Rtol*. The results are shown in the right picture of Fig. 4.

Conclusion. We have presented a new code RADAR5, which can solve a large class of delay problems (including differential-algebraic delay equations, stiff problems, neutral delay equations, state-dependent delays, and problems with small or vanishing delays). Although the code is still under development, we have released a first version, which performs satisfactorily on all problems presented in this article. The code, together with seven driver programs, are available at the address 'http://www.unige.ch/math/folks/hairer' under item 'software'.

Acknowledgement. The work reported in this paper was developed during the stay of Nicola Guglielmi at the University of Geneva, in the academic year 1999/2000. This author wishes to thank Ernst Hairer and Gerhard Wanner for making this visit possible.

References

[1] C.T.H. Baker, J.C. Butcher, C.A.H. Paul, *Experience of STRIDE applied to delay differential equations*, Technical Report 208, Univ. Manchester, 1992.

[2] G.A. Bocharov, G.I. Marchuk, A.A. Romanyukha, *Numerical solution by LMMs of stiff delay differential systems modelling an immune response*, Numer. Math. 73, 131–148 (1996).

[3] R.N. Castleton, L.J. Grimm, *A first order method for differential equations of neutral type*, Math. Comp. 27, 571–577 (1973).

[4] W.H. Enright, H. Hayashi, *A delay differential equation solver based on a continuous Runge-Kutta method with defect control*, Numer. Algorithms 16, 349–364 (1998).

[5] I. Epstein, Y. Luo, *Differential delay equations in chemical kinetics. Nonlinear models: the cross-shaped phase diagram and the Oregonator*, J. Chemical Physics 95, 244–254 (1991).

[6] N. Guglielmi, E. Hairer, *Order stars and stability for delay differential equations*, Numer. Math. 83, 371–383 (1999).

[7] E. Hairer, S.P. Nørsett, G. Wanner, *Solving Ordinary Differential Equations I. Nonstiff Problems, 2nd edition*, Springer Series in Computational Mathematics **8**, Springer-Verlag Berlin, 1993.

[8] E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems, 2nd edition*, Springer Series in Computational Mathematics **14**, Springer-Verlag Berlin, 1996.

[9] Z. Jackiewicz, E. Lo, *The numerical solution of neutral functional-differential equations by Adams predictor-corrector methods*, Appl. Numer. Math. 8, 477–491 (1991).

[10] M. Okamoto, K. Hayashi, *Frequency conversion mechanism in enzymatic feedback systems*, J. Theor. Biol. 108, 529–537 (1984).

[11] P. Waltman, *A threshold model of antigen–stimulated antibody production*, Theoretical Immunology (Immunology Ser. 8), Dekker, New York, 437–453 (1978).

[12] R. Weiner, K. Strehmel, *A type insensitive code for delay differential equations basing on adaptive and explicit Runge-Kutta interpolation methods*, Computing 40, 255–265 (1988).

[13] M. Zennaro, *P-stability properties of Runge-Kutta methods for delay differential equations*, Numer. Math. 49, 305–318 (1986).

Nicola Guglielmi
Dip. di Matematica Pura e Applicata
Università dell'Aquila
via Vetoio (Coppito)
I-67010 L'Aquila
Italy
e-mail: guglielm@univaq.it

Ernst Hairer
Section de Mathématiques
Université de Genève

CH-1211 Genève 24
Switzerland
e-mail: Ernst.Hairer@math.unige.ch