ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computational and Applied Mathematics 185 (2006) 261–277

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.com/locate/cam

Open issues in devising software for the numerical solution of implicit delay differential equations

Nicola Guglielmi[1]

Dipartimento di Matematica Pura e Applicata, Università dell'Aquila via Vetoio (Coppito), I-67010 L'Aquila, Italy

Received 31 March 2003

Abstract

We consider initial value problems for systems of implicit delay differential equations of the form

$$My'(t) = f(t, y(t), y(\alpha_1(t, y(t))), \ldots, y(\alpha_m(t, y(t)))),$$

where M is a constant square matrix (with arbitrary rank) and $\alpha_i(t, y(t)) \leqslant t$ for all t and i.

For a numerical treatment of this kind of problems, a software tool has been recently developed [6]; this code is called RADAR5 and is based on a suitable extension to delay equations of the 3-stage Radau IIA Runge–Kutta method.

The aim of this work is that of illustrating some important topics which are being investigated in order to increase the efficiency of the code. They are mainly relevant to

(i) the error control strategies in relation to derivative discontinuities arising in the solutions of delay equations;
(ii) the integration of problems with unbounded delays (like the pantograph equation);
(iii) the applications to problems with special structure (as those arising from spatial discretization of evolutions PDEs with delays).

Several numerical examples will also be shown in order to illustrate some of the topics discussed in the paper.

Keywords: RADAR5; Implicit delay differential equations; Radau method; Numerical code; Error control

E-mail address: guglielm@univaq.it.

[1] Supported by the Italian M.I.U.R.

0377-0427/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2005.03.010

1. Introduction

We consider systems of delay differential equations in the general (implicit) form:

$$My'(t) = f(t, y(t), y(\alpha(t, y(t)))), \quad t \in (t_0, T],$$
$$y(t_0) = y_0,$$
$$y(t) = g(t), \quad t < t_0, \tag{1}$$

where $T > t_0$ is a given constant, $y \in \mathbb{R}^d$ $(d \geqslant 1)$, f is a real-valued vector function, M is a constant (possibly singular) matrix, $\alpha(t, y(t))$ denotes a vector of m deviating arguments, $y(\alpha(t, y(t)))$ denotes the corresponding vector of retarded terms, i.e., $(y(\alpha_1(t, y(t))), \ldots, y(\alpha_m(t, y(t))))$ and $\alpha(t, y(t)) \leqslant t$ (to be intended componentwise). The value $g(t_0)$ may be different from y_0, allowing for a discontinuity at t_0. Very often we shall make reference to the so-called delays, that is $\tau_i(t, y(t)) := t - \alpha_i(t, y(t)),\ i = 1, \ldots, m$.

The considered class of problems includes retarded differential-algebraic systems, neutral delay differential equations of the form

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

singularly perturbed problems and problems with vanishing delays (see [6]).

The paper is organized as follows. In Section 2 we describe synthetically the integration technique; in Section 3 we deal with the error control strategy (illustrating it on a real-life neutral problem) and propose a novel technique for approximating jump discontinuities and in Section 4 we direct attention at problems with unbounded delays; finally in Section 5 we discuss issues related to the integration of evolution PDEs with memory effects.

2. The basic integration scheme

The integration scheme we consider is based on the s-stage Radau IIA method (in particular the code RADAR5 uses $s = 3$). For a detailed description we address the reader to [6] and—for technical details—to the documentation at the address http://www.unige.ch/math/folks/hairer/software.html

For simplicity we consider the case of a single deviating argument in (1) and focus our attention on the generic nth time step, from the grid-point t_n to the grid-point $t_{n+1} = t_n + h_n$. We use the following notation:

- $\{a_{ij}\}$ and $\{c_i\}$ $(i, j = 1, \ldots, s)$ are the coefficients and abscissae of the Radau IIA Runge–Kutta method;
- $f(t, y, z)$ denotes the right-hand-side function;
- $Y_\ell^{(n)}$ denotes the ℓth stage value computed at the (current) nth step;
- $\alpha_\ell := \alpha(t_n + c_\ell h_n, Y_\ell^{(n)})$;
- $u(t)$ denotes the continuous approximation to $y(t)$ computed during the integration process, that is

$$u(t) = \begin{cases} g(t) & \text{if } t \leqslant t_0, \\ u_m(t) & \text{if } t_m < t \leqslant t_{m+1}, \end{cases}$$

where $u_m(t)$ is step-by-step given by the collocation polynomial associated with the Radau method, that is (for $t_m \leqslant t \leqslant t_{m+1}$)

$$u_m(t_m + \vartheta h_m) = \ell_0(\vartheta) y_m + \sum_{i=1}^{s} \ell_i(\vartheta) Y_i^{(m)}, \quad \vartheta \in [0, 1], \tag{2}$$

where h_m is the stepsize used at the mth step, $\ell_i(\vartheta)$ is the polynomial of degree s satisfying $\ell_i(c_j) = \delta_{ij}$ (δ_{ij} being the Kronecker delta symbol), where we add $c_0 = 0$ to the abscissae $c_1, \ldots, c_s$ of the method.

The application of the Runge–Kutta method leads to the equations

$$0 = \underbrace{M(Y_i^{(n)} - y_n) - h_n \sum_{j=1}^{s} a_{ij} f(t_n + c_j h_n, Y_j^{(n)}, Z_j^{(n)})}_{F_i(t_n + c_i h_n, Y_1^{(n)}, \ldots, Y_s^{(n)}, Z_1^{(n)}, \ldots, Z_s^{(n)})}, \quad i = 1, \ldots, s, \tag{3}$$

$$Z_\ell^{(n)} = u(\alpha_\ell), \quad \ell = 1, \ldots, s$$

$$y_{n+1} = Y_s^{(n)} \quad \text{(the method is stiffly accurate).}$$

Observe that for $m = n$, as evidenced by (2), u_m identifies the continuous output to be computed in the current interval $[t_n, t_{n+1}]$ (which hence depends on the unknown stage values $Y_1^{(n)}, \ldots, Y_s^{(n)}$). This makes evident the *full* implicitness of the Runge–Kutta equations as soon as the delay vanishes or the code proceeds with a stepsize larger than the delay (which means that $\alpha_\ell > t_n$ for some ℓ).

2.1. Discontinuities in the solution

It is well known (see e.g., [1–3]) that jump discontinuities may occur in various orders of the derivative of the solution, independently of the regularity of the right-hand side. With respect to the case of *explicit* DDEs (that is when the matrix M is the identity), jump discontinuities do not necessarily smooth out for problems in the form (1). Consider for example the following equation (from [14]), having a constant delay $\tau = 1$,

$$y'(t) = y'(t-1), \quad t \geqslant 0,$$

$$y(0) = 0, \quad t = 0,$$

$$y(t) = (t+1)^5, \quad t < 0, \tag{4}$$

having analytical solution $y(t) = \lfloor t \rfloor + (t - \lfloor t \rfloor)^5$, $t \geqslant 0$ (here $\lfloor \cdot \rfloor$ denotes the integer part function). Hence the solution is only C_0-continuous and jump discontinuities in its first derivative occur at any integer point $\xi_i = i$, $i > 0$ (such points are usually called *breaking points* [1]). Observe that, by adding a new variable $v(t) = y'(t)$, the above equation can be written in form (1) as

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y'(t) \\ v'(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ v(t) - v(t-1) \end{pmatrix}. \tag{5}$$

If the points ξ_i are not included in the mesh either an order reduction may be detected (e.g., when using a constant stepsize which is not a sub-multiple of τ) or the stepsize may be severely restricted in a neighborhood of the jump discontinuities (when using variable stepsize).

3. Error estimation

In the case of a state-dependent delay or of a variable stepsize integration, the $\mathcal{O}(h^{s+2})$ contribution to the local error, due to the approximation of delayed variables, determines a global error $\mathcal{O}(h^{s+1})$. Hence the 3-stage Radau IIA method has classical order $p = 4$ (when applied to (1)).

Now we briefly outline the stepsize control mechanism implemented by the code RADAR5 (this means that we restrict our attention to the case $s = 3$). In ordinary differential equations the stepsize is chosen accordingly to a suitable error estimation at grid points. However, for problems of form (1), the control of the only local error at grid points could be very misleading and in many problems it turns out to be fundamental to estimate the error in the continuous numerical approximation to the solution.

The discrete advancing method is given by the tableau

$\frac{4-\sqrt{6}}{10}$	$\frac{88-7\sqrt{6}}{360}$	$\frac{296-169\sqrt{6}}{1800}$	$\frac{-2+3\sqrt{6}}{225}$
$\frac{4+\sqrt{6}}{10}$	$\frac{296+169\sqrt{6}}{1800}$	$\frac{88+7\sqrt{6}}{360}$	$\frac{-2-3\sqrt{6}}{225}$
1	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$
	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$

A classical error estimation at mesh points is obtained, as in the standard ODE framework, by embedding a method of lower order into the Radau method.

The embedded error-estimating method has classical order $p' = 3$ and is constructed by using the same stages $Y_i^{(n)}$, $i = 1, 2, 3$, plus the additional explicit stage $Y_0^{(n)}$ given by

$$Y_0^{(n)} = y_n \text{ to which corresponds } Z_0^{(n)} = u(\alpha_0) \text{ with } \alpha_0 = \alpha(t_n, y_n).$$

Therefore, denoting by $c_0 = 0$ the abscissa corresponding to this further stage, we get

$$M(\bar{y}_{n+1} - y_n) = h_n \sum_{i=0}^{3} \bar{b}_i f(t_n + c_i h_n, Y_i^{(n)}, Z_i^{(n)}),$$

where $\bar{y}_{n+1}$ denotes the numerical approximation to $y(t_{n+1})$ determined by the embedded method, whose weights are given by

$$\bar{b}_0 = \gamma_0, \quad \bar{b}_1 = \frac{16-\sqrt{6}}{36} - \frac{2+3\sqrt{6}}{6}\gamma_0, \quad \bar{b}_2 = \frac{16+\sqrt{6}}{36} - \frac{2-3\sqrt{6}}{6}\gamma_0, \quad \bar{b}_3 = \frac{1}{9} - \frac{1}{3}\gamma_0$$

with

$$\gamma_0 = \frac{6 - \sqrt[3]{9} + 3\sqrt[3]{3}}{30}$$

being the real eigenvalue of the coefficient matrix A of the method (see [9]).

Unfortunately the embedded method is not A-stable; hence as the stiffness of the problem and/or the stepsize h_n grow, the classical error estimate $\bar{\delta}_n = \bar{y}_{n+1} - y_{n+1}$ is unbounded and, consequently, it dramatically overestimates the true local error. Therefore, in order to overcome this inconvenience, the code utilizes the alternative quantity

$$\delta_n = P_n \bar{\delta}_n = \mathcal{O}(h_n^4) \quad \text{with } P_n = (M - h_n \gamma_0 G_n)^{-1} M, \tag{6}$$

or even $\delta_n = P_n^2 \bar{\delta}_n$, where the projection matrix P_n is constructed by using the Jacobian G_n of the function f (see Section 5); this is in complete analogy to what is done by the code RADAU5 (by Hairer and Wanner) for stiff ODEs; the only difference between the two cases is in the different structure of the Jacobian matrix G_n.

This technique, called "filtering" is essentially arbitrary; the problem is that the available error estimator $\bar{\delta}_n$ cannot be used in the stiff regime; this makes necessary to develop a new error estimator suitable for stiff computations. This is obtained in (6) by means of the projector P_n; the projector has several properties but some of them still need to be studied. As an example, if M is rank deficient, the behavior of the projector as $h \to 0$ deserves to be investigated. The approach of constructing a new error estimator for stiff computations has been considered by de Swart and Söderlind in [15]. Here the arbitrary "filtering" idea is abandoned in favor of a justified error estimator of proper classical order as $h \to 0$ and with a uniform upper bound when $h \to \infty$. It is a plan of the author to implement and include this estimator in a future version of RADAR5.

We remark that the LU-decomposition of the matrix $M - h_n \gamma_0 G_n$ in (6) is already available from the simplified Newton iteration which is used to solve the system (3) of the R–K equations and hence does not involve further computation (see [6]).

This alternative estimate of the local error has the advantage that, while asymptotically equivalent to the classical estimate for vanishing stiffness and stepsize h_n, it remains bounded (or even vanishes with the second choice) as the stiffness of the problem and/or the stepsize h_n grow.

We consider the following norm for an arbitrary (error) vector v_n,

$$\|v_n\| = \sqrt{\frac{1}{d} \sum_{i=1}^{d} \left(\frac{v_{n,i}}{s_i} \right)^2},$$

where $s_i = 1 + \rho |y_{n,i}|$ and ρ is the ratio $\mathrm{tol_r}/\mathrm{tol_a}$ between the relative ($\mathrm{tol_r}$) and absolute ($\mathrm{tol_a}$) input tolerances per step (which are used for the stepsize selection). Then we denote by σ_n the following measure of the error at grid points,

$$\sigma_n = \|\delta_n\|.$$

We shall call σ_n as the *discrete* component of the local error.

In the general case, the local order of the error-estimating method turns out to be 4, that is

$$\sigma_n = \mathcal{O}(h_n^4).$$

As we have mentioned, for delay equations, where the uniform accuracy of the numerical solution has also influence on the local error, it is necessary to control the error uniformly in time.

To do this we may consider in general also the polynomial

$$v_m(t_m + \vartheta h_m) = \sum_{i=1}^{s} \ell_i(\vartheta) Y_i^{(m)}, \quad \vartheta \in [0, 1], \tag{7}$$

of degree $s-1$, which interpolates the values $Y_i^{(m)}$ but not y_m (compare with (2)). In the considered case, that is $s = 3$, v_m is a parabola. It turns out that

$$\eta_n = \max_{\vartheta \in [0,1]} \|u_n(t_n + h_n\vartheta) - v_n(t_n + h_n\vartheta)\| = \|u_n(t_n) - v_n(t_n)\| = \mathcal{O}(h_n^3).$$

We use this quantity as an indicator for the uniform error and denote it as *continuous* component of the local error.

The estimate used for the stepsize control is finally given by

$$\mathrm{err}_n = \gamma_1 \sigma_n + \gamma_2 (\eta_n)^{4/3} = \mathcal{O}(h_n^4) \tag{8}$$

with the parameters $\gamma_1, \gamma_2 \geqslant 0$ possibly tuned by the user. This choice is the fruit of both theoretical and empirical analyses. The order of the estimation is 4 when the solution is smooth, and is obtained quite cheaply.

The criteria for accepting the current step and selecting the new one are based on the two input parameters $\mathrm{tol_r}$ and $\mathrm{tol_a}$, denoting the absolute and relative tolerances per step (actually in the code it is possible to define both tolerances componentwise but for simplicity we consider here the case where they are constant for all components).

If we set $\mathrm{tol_a} = \varepsilon$ and $\mathrm{tol_r} = 0$, the error test takes the form

$$\mathrm{err}_n \leqslant \varepsilon$$

and a possible prediction for the new stepsize h_{n+1} is given by the following classical formula:

$$\bar{h}_{n+1} = \max\left\{\omega_1, \min\left\{\omega_2, \varrho \sqrt[4]{\frac{\varepsilon}{\mathrm{err}_n}}\right\}\right\} h_n \tag{9}$$

with the parameters ω_1, ω_2 and ϱ possibly chosen by the user (default values are 0.2, 8.0 and 0.9, respectively). An alternative strategy used by the code is based on the Gustafsson predictive control (derived by the seminal works of Gustafsson et al. [8] and Gustafsson [7]). Such technique is also described in [9].

3.1. Behavior of the error estimator on a real-life neutral problem

In order to better illustrate the behavior of the error estimator (8), we consider its application to a real-life model. In particular we are interested in the relationship between the given input tolerance and the obtained accuracy without making use of any a priori knowledge of breaking points. We have chosen

Table 1
Computational results for problem (10)

ε	ST	RE	EM
10^{-2}	27	12	5.4610^{-3}
10^{-3}	55	39	1.7410^{-3}
10^{-4}	93	58	6.7310^{-3}
10^{-5}	144	92	1.5610^{-4}
10^{-7}	210	132	4.1910^{-6}
10^{-10}	362	181	4.3510^{-8}
10^{-12}	527	213	3.2710^{-9}

a system of neutral equations which models a predator–pray system due to Kuang [12]. It is given by the following two equations:

$$y_1'(t) = y_1(t)\left(1 - y_1(t-\tau) - \frac{29}{10}y_1'(t-\tau)\right) - \frac{y_1(t)^2 y_2(t)}{y_1(t)^2+1},$$
$$y_2'(t) = \left(\frac{y_1(t)^2}{y_1(t)^2+1} - \frac{1}{10}\right) y_2(t) \tag{10}$$

with $\tau = \frac{21}{50}$ and initial conditions $g_1(t) = \frac{33}{100} - t/10$ and $g_2(t) = \frac{111}{50} + t/10$ for $t \leqslant 0$. We transformed the problem into the form (1) and applied the code RADAR5 till the endpoint $T = 6$. We remark that, in order to set (10) into the form (1), it is necessary to pass to a higher dimension $d = 3$ (by introducing y_1' as a new variable).

In the experiments (summarized in Table 1), we do not insert the breaking points into the mesh and, for the error control, use (8) with parameters $\gamma_1 = \gamma_2 = 0.5$.

We obtained the following results, where ε indicates the given tolerance (here we assume that the absolute and relative tolerance are the same), ST the number of accepted steps, RE the number of rejected steps (by the error controller) and EM is a measure of the error at the final integration point (which is given by the Euclidean norm of the difference $(y_{1,N}, y_{2,N})^{\mathrm{T}} - (y_1(T), y_2(T))^{\mathrm{T}}$, where N denotes the number of executed steps, which means that $t_N = T$).

Table 1 shows a good convergence to zero of the error as the input tolerance ε decreases to zero.

We also observe that the ratio between accepted and rejected steps has a slight variation for tolerances $10^{-2} \leqslant \varepsilon \leqslant 10^{-10}$ (the ratio is always between 1.4 and 2). We remark that the large number of rejections is due to the non-regularity of the solution and in particular to the fact that breaking points are not directly approximated during the integration.

3.2. An effective comparison between error estimators

We consider now the scalar DDE (4) (which we have written in the equivalent form (5)), introduced in Section 1. Since the breaking points are known a priori, we could easily insert them in advance into the mesh; in such case the use of the sole discrete error estimate σ_n would turn out to be enough in order to get an accurate approximation of the solution. Nevertheless, since our aim is that of giving evidence to the

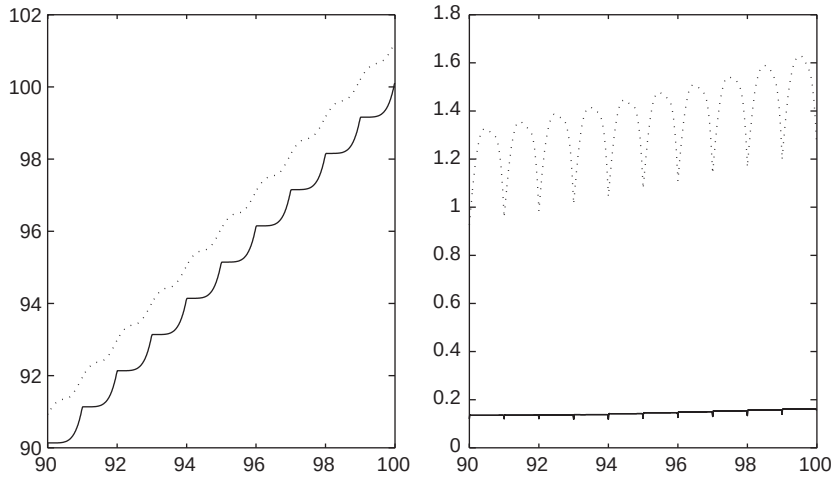

Fig. 1. In the *left side* are shown both the numerical approximations of the solution $y(t)$ of (4) obtained by using combined error control (continuous line) and sole discrete error control (dashed line), respectively. In the *right side* are shown the corresponding global errors, that is the error measured integrating (4) by making use of the combined error control (continuous line) and the error measured by using the sole discrete error control (dashed line).

importance of controlling the continuous error component in the general case where the breaking points are not available (this is certainly the case in state-dependent problems), we do not insert in advance any point into the mesh. We consider the integration of (5) in the time interval $t \in [0, 100]$ and use tolerances $\mathrm{tol_a} = \mathrm{tol_r} = 10^{-4}$ for both components y and v.

As a first experiment we integrate the equation by making use of the combined error control strategy given by (8). In such a way we obtain a good approximation of the solution y (as shown by the continuous line in Fig. 1). Such a control gives rise to a strong restriction of the stepsize in a neighborhood of the breaking points, due to the magnitude of the continuous error component. A possible way to obtain a more efficient integration could consist in a strategy attempting to tracking the discontinuities in parallel to the integration; this will be discussed in Section 3.3.

As a second experiment we make only use of the discrete error estimate σ_n. As shown in Fig. 1, the numerical approximation of the solution $y(t)$ (dashed line) appears to be quite inaccurate in the last part of the integration interval. This is due to the poor continuous approximation of the solution $(y(t), v(t))$ in a neighborhood of the breaking points. The effect of this is put in evidence by the high increase of the global error as the integration advances (see the right frame in Fig. 1).

3.3. A novel procedure for detecting and approximating breaking points

As we have seen, when breaking points are not known a priori, in order to get a precise integration of problem (1), it is necessary to determine them to a sufficient accuracy (which is related to the required input tolerances). Despite this is implicitly done by the error estimator (8), we are going to consider a more direct and efficient approach.

In the literature two different approaches have been developed for *explicit* DDEs (that is for the case when the matrix M is the identity); they are based, respectively, on

(a) discontinuity tracking, that is on a direct computation of discontinuities from the deviating arguments (e.g., by Manchester's group [16]);
(b) relying on uniform local error estimation based on the analysis of the so-called defect, that is on $y'(t) - f(t, y(t), y(\alpha(t, y(t))))$ (e.g., by Toronto's group et al. [5,4]).
Significant discontinuities are detected in this case by an analysis of the sequence of stepsizes and then located by a suitable analysis of the defect.

Both approaches present some advantages and difficulties. For the first approach, the construction of the network dependency graph (see [16]) is necessary and may be quite expensive. On the other hand, the second approach is computationally honerous especially in presence of jumps in the solution or its first derivative (which require a higher accuracy).

We propose here a novel approach which is related to some of the ideas proposed in [5] and is based on the ratio between the error estimates σ_n and η_n as a detection measure and on the extrapolated defect as localization function. We observe, in fact, that when a breaking point is internal to the considered time-step $[t_n, t_n + \bar{h}_n]$, the continuous error estimate η_n is much larger than the classical discrete estimate σ_n. The algorithm, which we denote with the JD acronym (standing for "jump discontinuity"), activates in case of a stepsize rejection and proceeds through the following phases (we focus attention on the nth time step).

Algorithm JD

1. Detection of a discontinuity by monitoring the ratio between the error indicators σ_n and μ_n; if $|\mu_n|/|\sigma_n| > \Theta$ (Θ being a given threshold, e.g., $\Theta = 10$), the interval $[t_n, t_n + \bar{h}_n]$ is *identified* to contain at least one breaking point. Otherwise the procedure exits.
2. Approximation of the jump discontinuity by means of the analysis of the *extrapolated* defect from the previous (accepted) mesh interval, that is

$$Mu'_{n-1}(s) - f(s, u_{n-1}(s), u(\alpha(s, u_{n-1}(s)))) \quad \text{for } s \in [t_n, t_n + \bar{h}_n]$$

(recall that $u(\cdot)$ denotes the computed piecewise polynomial continuous approximation to the solution available on the left of t_n). The predicted breaking point identifies with the nonregular point of the defect in the considered interval (see Fig. 2). Such approximation, say $\hat{\xi}_i$, may be obtained, e.g., by a bisection-like algorithm.
3. Insertion of the approximately computed breaking point into the mesh, which means setting $t_{n+1} = \hat{\xi}_i$.
4. Execution of a new step from t_n to t_{n+1}.

Let us consider again example (5) and focus attention on its numerical integration on the left side of the first breaking point $\xi_1 = 1$. Suppose first to make use of the combined error estimate (8) without providing any approximation of the breaking point during the advancing of the method (actual procedure adopted by the code) and then to make use of the algorithm JD in order to approximate the breaking points (novel procedure proposed).

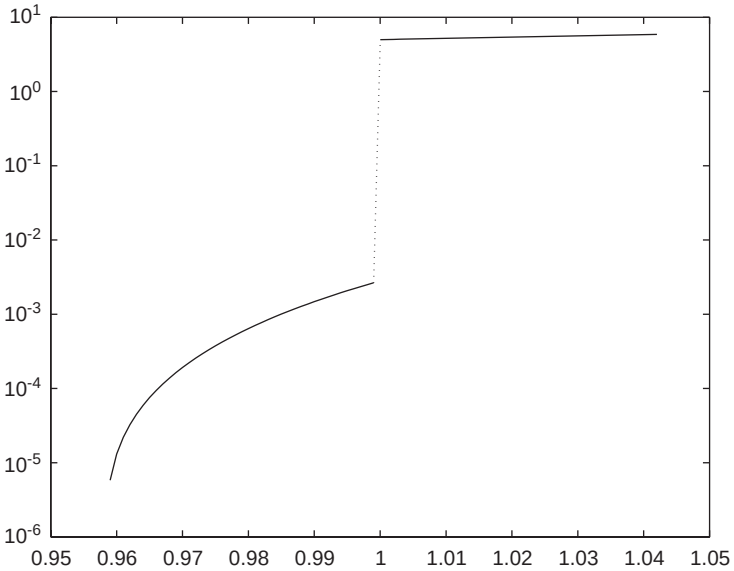

Fig. 2. Extrapolated defect (in logarithmic scale) in a neighborhood of $\xi_1 = 1$.

3.4. An experimental comparison

The following behavior is experimentally observed. Since the breaking point $\xi = 1$ is not inserted a priori into the mesh, at $t_n = 0.9580\ldots$ a stepsize $\bar{h}_n = 0.1335\ldots$ (s.t. $t_{n+1} > \xi$) is proposed by the *stepsize prediction* algorithm. Due to the high estimated error (8) the step is rejected. The measured ratio between the *continuous error estimator* and the *discrete error estimator* is $\mu_n/\sigma_n \approx 0.24 \cdot 10^3$; this *detects* the presence of a breaking point within the interval $[0.9580\ldots, 1.0915\ldots]$ (which is actually given by $\xi_1 = 1$). How to proceed?

Present procedure (*P*): Due to the large estimated error err_n a new, very small, stepsize (h_n=0.00292 ...) is predicted and finally accepted. Then the same situation is repeated for several times, that is an alternation of large rejected stepsizes and quite small accepted ones. After 13 steps (counting both accepted and rejected ones) the breaking point $\xi_1 = 1$ is overcome.

Novel procedure (*N*): If, instead, the algorithm JD is applied to approximate the breaking points, the behavior of the code is totally different. The extrapolated defect

$$Mu'_{n-1}(s) - f(s, u_{n-1}(s), u(\alpha(s, u_{n-1}(s)))) \quad \text{for } s \in [t_n, t_n + \bar{h}_n]$$

is processed in order to approximate the breaking point ξ_1. By means of some steps of a bisection-like method, an approximation (say $\hat{\xi}_1$) of the breaking point, $\hat{\xi}_1 \approx 0.9994\ldots$, is obtained (see Fig. 2) and the code sets $t_{n+1} = \hat{\xi}_1$; in this way the jump discontinuity is essentially reached in a further single step (having size $h_n = t_{n+1} - t_n \approx 0.0414\ldots$).

The advantage with respect to the actual procedure is in a significant reduction of the total number of steps (taking into account both accepted and rejected ones) and in a smaller global error, at the cost of a bisection-like algorithm for approximating the jump-discontinuities in the extrapolated defect. An efficient implementation of the proposed algorithm (JD) is under development and is planned to be

included in the next distribution of the code. In particular, the algorithm JD should have the aim to detect discontinuities in the solutions and presumably its first derivatives. For what concerns, instead, discontinuities in higher-order derivatives, the error control procedure (8) is expected to be efficient enough.

4. Problems with unbounded delays

In this section, we direct attention at problems of the form (1) characterized by the unboundedness of some of the delays.

4.1. The memory allocation problem

In the present implementation, after any accepted step, the coefficients of the collocation polynomials which provide the approximation to the solution in the current mesh interval are stored in a suitable *memory array*. Such information is stored in a memory segment of size $4 \cdot \hat{d} + 2$ double precision elements ($\hat{d}$ being the number of components of the solution—say y_{i_j}, $j = 1, \ldots, \hat{d}$, with $i_j \in \{1, \ldots, d\}$—presenting deviating arguments in the right-hand-side function f of (1)). In particular two cells store the left mesh-point t_n and the stepsize h_n and—for each component y_{i_j}—four cells contain the coefficients of the interpolating cubic polynomial (which is represented in the Newton form). We shall denote by $\mathscr{M}l$ the full size (that is the number of double precision elements) of the *memory array*.

The memorization process fills the array starting from the beginning and proceeds step by step towards its end. When the memory array is completely filled, the continuous output corresponding to the current step is stored at the beginning of the *memory array* (which can be interpreted as a circular stack). As a consequence the information relevant to the step which had been previously stored in the same initial segment of the memory array is lost; this means that the continuous approximation to the solution is usually available only on the right of a mesh point t_b, $b = \max(0, n - L)$ (L being the number of *storable steps*, that is $L = \mathscr{M}l/(4 \cdot \hat{d} + 2)$). Hence, typically, t_b increases as n increases.

If, during the execution of the nth step, any deviating argument falls on the right-hand side of t_0 and on the left-hand side of $t_b > t_0$ the integration process has to stop, since the required approximation to the solution in the past is not available.

Some remarks are timely.

(1) Since the *memory array* has an arbitrary but fixed length given by $\mathscr{M}l \propto L$, the approximation of the solution in the domain of integration (for a sufficiently large n) is only partially available, that is in the interval $[t_{n-L}, t_n]$.
(2) For a fixed L, the above interval has smaller length for increasing accuracies.
(3) No a priori information concerning the range of the deviating arguments is taken into account by the storage procedure.

Some improvements are actually being investigated and implemented. The first one is that of allowing *different storage* strategies; e.g., in the equation

$$y'(t) = f(t, y(t), y(qt)), \quad t \leqslant T, \quad (q < 1),$$

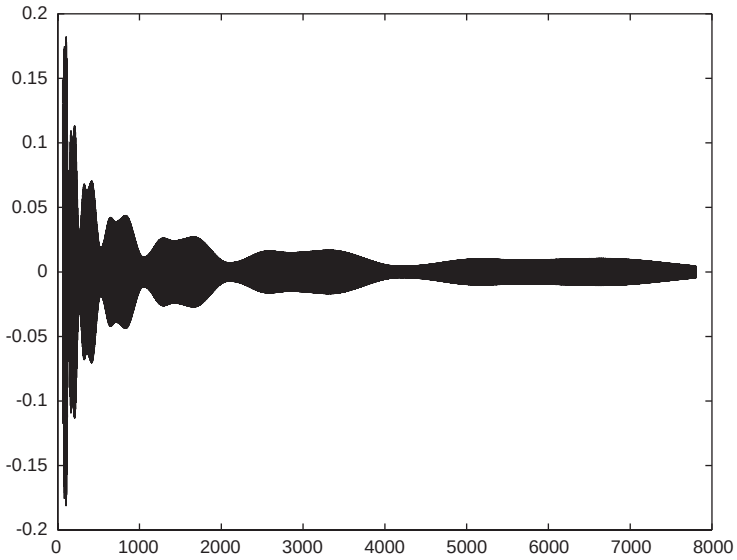

Fig. 3. Numerical approximation of the real part of the solution of (11) with $\varepsilon = 10^{-6}$.

the special form of the deviating argument $\alpha(t) = qt$ implies that the dense output is not needed if $t_n > qT$. This a priori information could of course be exploited by the code. A second, more theoretical, subject is that of investigating the effectiveness of suitable reformulations of the problem in hand, aiming to minimize the storage requirements.

4.2. An illustrative example

We present here an example which illustrates the potential difficulties lying in the numerical solution of an equation with an unbounded delay. Let us consider the following so-called pantograph equation (in complex form)

$$\begin{aligned} &y'(t) = (-1 + 10\mathrm{i})y(t) + 5\mathrm{e}^{\mathrm{i}t}y(t/2), \quad t > 0, \\ &y(0) = 1 - \mathrm{i}. \end{aligned} \tag{11}$$

The basic features of the considered equation are the following: on the one hand, the mentioned unboundedness from above of the delay $\tau(t) = t/2$, and on the other hand, the slow vanishing behavior of the solution, which may be accompanied by persisting oscillations (asymptotic stability of sub-exponential type is observed), as shown in Fig. 3. The combination of them makes the memory requirements very hard, as shown by the following illustrative experiments.

First observe that (11) may be rewritten as a two-dimensional system of the form (1) by simply separating the real and the imaginary part of the solution.

We used several relative tolerances $\mathrm{tol_r} = \varepsilon$ and corresponding absolute tolerances $\mathrm{tol_a} = 10^{-6}\varepsilon$ (uniformly for all components), and two different values L, namely $L = 10^4$ and $5 \cdot 10^4$ (we remark that the default value provided by the code is 10^4). The resulting length of the memory array is $\mathscr{M}l = 10L$ in this case.

Table 2
Statistics for the problem (11) with different tolerances

ε	$L = 10^4$		$L = 5 \times 10^4$	
	t_f	n.s.	t_f	n.s.
10^{-1}	$0.1250 \cdot 10^5$	19 829	$0.6327 \cdot 10^5$	100 013
10^{-2}	$0.7435 \cdot 10^4$	20 094	$0.3692 \cdot 10^5$	100 061
10^{-3}	$0.4609 \cdot 10^4$	20 099	$0.2289 \cdot 10^5$	99 921
10^{-4}	$0.3140 \cdot 10^4$	20 115	$0.1571 \cdot 10^5$	100 127
10^{-5}	$0.2232 \cdot 10^4$	20 178	$0.1116 \cdot 10^5$	100 191
10^{-6}	$0.1561 \cdot 10^4$	20 283	$0.7819 \cdot 10^4$	100 355
10^{-7}	$0.1027 \cdot 10^4$	20 351	$0.5138 \cdot 10^4$	100 211
10^{-8}	$0.5452 \cdot 10^3$	20 831	$0.2695 \cdot 10^4$	100 478
10^{-9}	$0.9440 \cdot 10^2$	20 241	$0.5340 \cdot 10^3$	103 614

In Table 2 we have reported, for each value of ε and of L, the maximum allowed number of steps (n.s.) and the corresponding final mesh-point t_f, where the integration stops because of the lack of storage capacity for the history of the numerical solution. It appears that, as the tolerance ε decreases, the memory required for storing the history necessary to reach the same final integration point becomes larger and larger. This example shows how the integration of an apparently simple problem as (11), may be very expensive. An optimal use of the memory is indeed very important.

5. On the application to evolution PDEs

As a final subject of this paper we discuss a very important task also for a general-purpose code like RADAR5; it consists in an efficient integration of problems with structure. Consider for example the so-called Hutchinson equation (see [11,13])

$$\frac{\partial}{\partial t}u(x,t) = \varepsilon \frac{\partial^2}{\partial x^2}u(x,t) + u(x,t)[1 - u(x,t-\tau)],$$

$$u(x,t) = \Psi(x,t) \quad t \in [-\tau, 0]; \quad x \in (0,1),$$

subject to homogeneous Dirichlet boundary conditions.

5.1. Semi-discretization in space

Using constant stepsize $\Delta x := 1/(d+1)$, and setting

$$y(t) = (u(\Delta x, t), u(2\Delta x, t), \ldots, u(1-\Delta x, t))^{\mathrm{T}},$$

we are lead to the d-dimensional system of DDEs

$$\frac{\mathrm{d}}{\mathrm{d}t}y(t) = \frac{\varepsilon}{\Delta x^2}\mathscr{D}y(t) + y(t) \circ (e - y(t-\tau)),$$

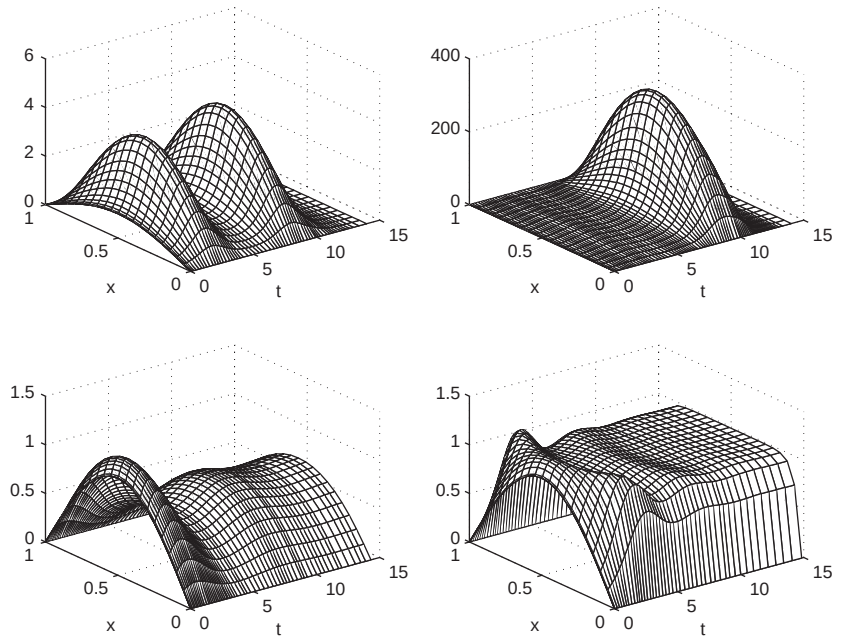

Fig. 4. Numerical approximation of the solution with $(\varepsilon, \tau) = (10, 0.1)$ (upper left), $(\varepsilon, \tau) = (10, 0.05)$ (upper right), $(\varepsilon, \tau) = (1.5, 0.05)$ (lower left) and $(\varepsilon, \tau) = (1, 0.001)$ (lower right).

$$y(t) = \Phi(t), \quad t \in [-\tau, 0] \tag{12}$$

with $\mathscr{D}$ the standard central difference operator, $e = [1, 1, \ldots, 1]^{\mathrm{T}}$ and $\circ$ denoting componentwise vector multiplication. The right-hand side has a very special structure and it is important to exploit it in the solution process of the Runge–Kutta equations.

The solution of the considered equation has a very interesting dynamics, which clearly depends on the choice of the diffusion parameter ε and of the delay τ. In Fig. 4 it is shown the behavior of the solution for different values of the coefficients (with respect to an initial function $\Psi(x, t) = \cos(t)\ \sin(\pi x)$). The solution presents different steady states (as is known); in particular, in the upper right figure, the solution increases to a very high value before collapsing to the zero steady-state; such phenomenon appears more evident if the diffusion coefficient is further decreased. An experimental numerical bifurcation analysis would be a very interesting subject, although being very computationally expensive. This makes evident the necessity of exploiting the structure of the system in order to decrease the number of floating point operations.

5.2. The nonlinear Runge–Kutta system in problems with structure

For general nonlinear differential equations (1), the system (3) of the Runge–Kutta equations has to be solved by an iterative method; usually—due to the stiffness of the problem—Newton's method is applied.

This method needs to solve for each iteration a linear system with matrix

$$\begin{pmatrix} \dfrac{\partial F_1}{\partial Y_1} & \cdots & \dfrac{\partial F_1}{\partial Y_s} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial F_s}{\partial Y_1} & \cdots & \dfrac{\partial F_s}{\partial Y_s} \end{pmatrix},$$

where, for sake of conciseness we omit the dependence of F_i $(i = 1, \ldots, s)$ on n and on its arguments (see (3)). An exact computation of the derivatives of the function $F_i(t_n + c_i h_n, Y_1, \ldots, Y_s, Z_1, \ldots, Z_s)$ would lead to the following expression:

$$\begin{aligned} \frac{\partial F_i}{\partial Y_k} = M\delta_{ik} - h_n a_{ik} \Bigg(& \frac{\partial f}{\partial z}(t_n + c_k h_n, Y_k, Z_k) u'(\alpha_k) \frac{\partial \alpha}{\partial y}(t_n + c_k h_n, Y_k) \\ & + \frac{\partial f}{\partial y}(t_n + c_k h_n, Y_k, Z_k) \Bigg) - h_n \sum_{j=1}^{s} a_{ij} \frac{\partial f}{\partial z}(t_n + c_j h_n, Y_j, Z_j) \frac{\partial Z_j}{\partial Y_k}, \end{aligned} \tag{13}$$

where δ_{ik} is the Kronecker delta symbol. Observe that $\partial Z_j / \partial Y_k = \mathscr{Z}_{jk} I_d$ (I_d being the $d \times d$ identity matrix); the scalar $\mathscr{Z}_{jk}$ is zero only if the deviating argument falls on the left side of t_n; more precisely, setting $\vartheta_j := (\alpha(t_n + c_j h_n, Y_j) - t_n)/h_n$, we have

$$\mathscr{Z}_{jk} = \begin{cases} \ell_k(\vartheta_j) & \text{if } \vartheta_j > 0, \\ 0 & \text{if } \vartheta_j \leqslant 0, \end{cases} \tag{14}$$

where $\ell_k(\cdot)$ still represents the polynomial of degree s satisfying $\ell_i(c_i) = 1$ and $\ell_i(c_j) = 0$ for $j \neq i$ (where we recall that $c_0 = 0$ and $c_1, \ldots, c_s$ are the abscissae of the method).

For an efficient computation, we simplify (13) by approximating all derivatives of the function f (that is $(\partial f/\partial y, z)(t_n + c_k h_n, Y_k, Z_k)$) by $(\partial f/\partial y, z)(t_n, y_n, z_n)$, where $z_n = u(\alpha_0)$ and $\alpha_0 = \alpha(t_n, y_n)$.

Then the simplified Newton iteration, in the equivalent variables $V_i := Y_i - y_n$, becomes

$$\begin{aligned} & \hat{J}\Delta V^{(r)} = F(V^{(r)}), \quad V^{(r+1)} = V^{(r)} + \Delta V^{(r)}, \quad r = 0, 1, \ldots \\ & \hat{J} = I \otimes M - h_n A \otimes \left(\frac{\partial f}{\partial y} + \frac{\partial f}{\partial z} \left(u'(\alpha_0) \frac{\partial \alpha}{\partial y} \right) \right) - h_n A \mathscr{Z} \otimes \frac{\partial f}{\partial z}, \end{aligned} \tag{15}$$

where

$$F(V^{(r)}) = (F_1(t_n + c_1 h_n, y_n + V_1^{(r)}, \ldots, y_n + V_s^{(r)}, Z_1^{(r)}, \ldots, Z_s^{(r)}), F_2(\cdots), \ldots, F_s(\cdots))^{\mathrm{T}},$$

$$Z_\ell^{(r)} = u(\alpha_\ell^{(r)}) \quad \text{with} \quad \alpha_\ell^{(r)} = \alpha(t_n + c_\ell h_n, y_n + V_\ell^{(r)}), \quad \ell = 1, \ldots, s$$

and $\partial f/\partial y$, $\partial f/\partial z$ denote the $d \times d$ matrices of the partial derivatives of f w.r.t. y and z variables, respectively, and $\partial \alpha/\partial y$ the $1 \times d$ matrix of the partial derivatives of α w.r.t. y, computed at (t_n, y_n, z_n), and $\mathscr{Z} = \{\mathscr{Z}_{jk}\}_{j,k=1}^{s}$.

In the above notation $V = (V_1, \ldots, V_s)^{\mathrm{T}} \in \mathbb{R}^{sd}$ and the apex r identifies the iteration number.

A further simplification considered in RADAR5 consists in possibly approximating the matrix $\mathscr{Z}$ as

$$\mathscr{Z} \approx \zeta I_s \quad \text{for a suitable } \zeta \in [0, 1],$$

(I_s being the $s \times s$ identity matrix) which determines a very special tensor structure of the approximated Jacobian $\hat{J}$. Then, following the ideas of Bickart and Butcher, the matrix $\hat{J}$ is premultiplied by $(h_n A)^{-1} \otimes I_d$. Successively, in order to simplify the structure of the system, the idea is to block-diagonalize A^{-1} (this is completely analogous to the ODE-case as shown e.g., in [10]). Indicating by T the transformation matrix, we have $T^{-1}A^{-1}T = D$; then, introducing the transformed variables $W := (T^{-1} \otimes I_d)\, V$, we obtain the equivalent Newton iteration

$$J\Delta W^{(r)} = \ldots, \quad W^{(r+1)} = W^{(r)} + \Delta W^{(r)}, \quad r = 0, 1, \ldots,$$
$$J = h_n^{-1} D \otimes M - I \otimes \left(\frac{\partial f}{\partial y} + \frac{\partial f}{\partial z} u'(\alpha_0) \frac{\partial \alpha}{\partial y} + \zeta \frac{\partial f}{\partial z} \right). \tag{16}$$

Since the obtained linear system has block-diagonal structure, the linear algebra is more efficient with respect to the original iteration (15). We remark that when $\mathscr{Z} \neq O$, a suitable choice of the parameter ζ is very important for the convergence of the simplified Newton iteration. An optimal choice (in terms of speed of convergence of the Newton iteration) of the parameter ζ is under investigation. We remark that in case of nonconvergence of the simplified Newton iteration, the code performs the original Newton iteration (15) without any transformation (in this case, in fact, the tensor-product structure which lead to the transformation (16) is lost).

5.3. Band structure of the Jacobian

A further advantage which can be possibly exploited by the linear solver is related to the band structure of the matrix

$$G = \frac{\partial f}{\partial y} + \frac{\partial f}{\partial z} u'(\alpha_0) \frac{\partial \alpha}{\partial y} + \zeta \frac{\partial f}{\partial z}. \tag{17}$$

This is the case, for example, of the considered semi-discretized Hutchinson equation, where

$$f(t, y, z) = \frac{\varepsilon}{\Delta x^2} \mathscr{D} y + y \circ (e - z)$$

and hence both

$$\frac{\partial f}{\partial y} = \frac{\varepsilon}{\Delta x^2} \mathscr{D} + \mathrm{diag}(e - z) \quad \text{and} \quad \frac{\partial f}{\partial z} = -\mathrm{diag}(y)$$

have band structure (here $\mathrm{diag}(v)$ denotes the diagonal matrix whose diagonal coincides with the vector v). In the actual release of RADAR5, contrary to the fact that the matrix $\partial f/\partial y$ may be stored by the user both in a full-matrix structure and in a band-matrix one, the code stores the matrix $\partial f/\partial z$ in a suitable one-dimensional array, using a sparse matrix format; this is due to the fact that in most cases such matrix does not need to be computed (recall that its use is necessary only when large stepsizes, with respect to the delays, are used). Consequently, concerning the matrix (17), the complete band-matrix case is not specifically treated at present. Allowing the linear algebra routines to exploit such possible *global* band structure of the system is certainly an important development which is planned to be added in the next release.

Acknowledgements

The issues discussed in this paper are the subject of a joint research with Ernst Hairer (Université de Genève). A new version of the code including modifications related to the topics considered here is distributed at the website http://www.unige.ch/math/folks/hairer/software.html (where the first released version is actually available).

The author wishes to thank Gustaf Söderlind for the careful reading of the manuscript and his very useful remarks.

References

[1] A. Bellen, M. Zennaro, Numerical Methods for Delay Differential Equations, Oxford University Press, Oxford, 2003.
[2] R. Bellman, K.L. Cooke, Differential-Difference Equations, Academic Press, New York, 1963.
[3] R. Driver, Ordinary and Delay Differential Equations, Springer, Berlin, 1977.
[4] W.H. Enright, H. Hayashi, A delay differential equation solver based on a continuous Runge–Kutta method with defect control, Numer. Algorithms 16 (1998) 349–364.
[5] W.H. Enright, K.R. Jackson, S.P. Nørsett, P.G. Thomsen, Effective solution of discontinuous IVPs using a Runge–Kutta formula pair with interpolants, Appl. Math. Comput. 27 (1988) 313–335.
[6] N. Guglielmi, E. Hairer, Implementing Radau IIA methods for stiff delay differential equations, Computing 67 (2001) 1–12.
[7] K. Gustafsson, Control theoretic techniques for stepsize selection in explicit Runge–Kutta methods, ACM Trans. Math. Software 17 (1991) 533–554.
[8] K. Gustafsson, M. Lundh, G. Söderlind, A PI stepsize control for the numerical solution of ordinary differential equations, BIT 28 (1988) 270–287.
[9] E. Hairer, G. Wanner, Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems, Springer, Berlin, 1996.
[10] E. Hairer, G. Wanner, Stiff differential equations solved by Radau methods, J. Comput. Appl. Math. 111 (1999) 93–111.
[11] D.J. Higham, T. Sardar, Existence and stability of fixed points for a discretized nonlinear reaction diffusion equation with delay, Appl. Numer. Math. 18 (1995) 155–173.
[12] Y. Kuang, On neutral delay logistic Gauss-type predator–pray system, Dynamic Stability Systems 6 (1991) 173–189.
[13] S. Luckhaus, Global boundedness for a delay differential equation, Trans. Amer. Math. Soc. 294 (1986) 767–774.
[14] C.A.H. Paul, A test set of functional differential equations, Technical report 243, University of Manchester, 1992.
[15] J.J.B. de Swart, G. Söderlind, On the construction of error estimators for implicit Runge–Kutta methods, J. Comput. Appl. Math. 86 (1997) 347–358.
[16] D.R. Willè, C.T.H. Baker, The tracking of derivative discontinuities in systems of delay differential equations, Appl. Numer. Math. 9 (1992) 209–222.