Pergamon

Computers Math. Applic. Vol. 31, No. 6, pp. 1–11, 1996
Copyright©1996 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0898-1221/96 $15.00 + 0.00

S0898-1221(96)00001-6

Error Estimation and Step Size Control for Delay Differential Equation Solvers Based on Continuously Embedded Runge-Kutta-Sarafyan Methods

S. P. CORWIN AND S. THOMPSON
Department of Mathematics and Statistics, Radford University
Radford, VA 24142, U.S.A.

Dedicated with warmth and respect to Professor Diran Sarafyan

(Received May 1995; accepted September 1995)

Abstract—This paper considers the use of continuously embedded Runge-Kutta-Sarafyan methods for the solution of delay differential equations. It discusses simple ways to improve the error estimation and step size selection strategies for delay solvers based on Sarafyan methods. Numerical results are given which demonstrate the manner in which these estimates improve the accuracy of the solvers in a natural way.

Keywords—Algorithms, Delay differential equations, Derivative jump discontinuities, Functional differential equations, Interpolation for Runge-Kutta formulas, Retarded arguments, Retarded differential equations, Runge-Kutta formulas, Time delays.

INTRODUCTION

Continuously embedded Runge-Kutta-Sarafyan methods [1] may be used for the solution of delay differential equations in which the derivative at a given time depends on the solution at previous times. Neves and Thompson [2,3] describe the implementation and usage of a delay solver DRKLAG based on a (4,5) pair of Sarafyan methods. The problems for which DRKLAG is designed take the form

$$\frac{dy(t)}{dt} = f(t,\, y(t),\, y(t,\, \beta(t,\, y(t))))$$

or, more generally,

$$\frac{dy(t)}{dt} = f\,(t,\, y(t),\, y(t,\, \beta(t,\, y(t))),\, y'(t,\, \beta(t,\, y(t))))$$

for $t \in [a, b]$ and

$$\text{with } y(t) \equiv \phi(t) \text{ for } t \leq a, \text{ and } \beta(t,\, y(t)) \leq t \text{ for all } t.$$

(In the above, it is assumed that the integration proceeds from left to right. Obvious modifications to the notation apply if this is not the case.)

The author is grateful to an anonymous referee for a careful reading of the original version of this paper and for several constructive suggestions.

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

For the purposes of the study described in this paper, two additional experimental versions of DRKLAG were developed based on the method pairs described below. In each solver, the higher order method is used to advance the integration, and the lower order method is used for the purposes of error estimation and step size selection. Local extrapolation is used to treat the error estimate as one for the higher order method.

The purpose of the paper is to demonstrate the effectiveness of using alternate types of error control, rather than the standard difference of the embedded methods at integration grid points, for delay equations. We will present and discuss typical results which demonstrate the effect of each of three error estimates for the three solvers. We argue that more conservative forms of error estimation are appropriate for codes intended to solve delay problems. We present numerical results for the three delay codes to support this argument.

This paper is devoted to a discussion of alternate forms of error control for delay equations. It does not include a general discussion of the solution of delay equations. Readers interested in such a discussion are referred to [4]. An outstanding bibliography on the numerical solution of delay differential equations may be found in [5]. This paper also does consider realistic applications of delay equations. Readers interested in discussions of such applications are referred to [6,7]. Use of the DRKLAG software for the solution of complicated models arising in realistic problems is addressed in considerable detail elsewhere [8,9].

SARAFYAN METHODS

The following coefficient tableau contains the coefficients a_i and b_{ij} which determine the calculation of the Runge-Kutta derivative approximations for the first two pairs of methods. If the integration step size is denoted by h, these derivative approximations are defined by

$$k_0 = h\, f\,(t_n, y_n)$$

and

$$k_i = h\, f\left(t_n + a_i h, y_n + \sum_{j=0}^{i-1} b_{ij} k_j\right)$$

for $i > 0$. The coefficient tableau for the methods used in the first two pairs of methods [1] is given in Tableau 1. The polynomial coefficients for the methods used in the first pair are

$$\begin{aligned}
\Omega_1 &= k_0 \\
\Omega_2 &= \frac{-25k_0 + 48k_2 - 36k_3 + 16k_4 - 84k_5 + 81k_6}{6} \\
\Omega_3 &= \frac{70k_0 - 208k_2 + 228k_3 - 112k_4 + 490k_5 - 468k_6}{9} \\
\Omega_4 &= \frac{-40k_0 + 144k_2 - 192k_3 + 112k_4 - 399k_5 + 375k_6}{6} \\
\Omega_5 &= \frac{8\,(4k_0 - 16k_2 + 24k_3 - 16k_4 + 49k_5 - 45k_6)}{15} \\
\omega_1 &= k_0 \\
\omega_2 &= \frac{-127k_0 + 144k_2 + 36k_3 - 80k_4 + 27k_6}{42} \\
\omega_3 &= \frac{2\,(5k_0 - 8k_2 - 2k_3 + 8k_4 - 3k_6)}{3} \\
\omega_4 &= \frac{2\,(-13k_0 + 24k_2 + 6k_3 - 32k_4 + 15k_6)}{21}.
\end{aligned}$$

Tableau 1.

i	a_i	b_{ij}, $j = 0, \ldots, i-1$					
1	$\frac{1}{6}$	$\frac{1}{6}$					
2	$\frac{1}{4}$	$\frac{1}{16}$	$\frac{3}{16}$				
3	$\frac{1}{2}$	$\frac{1}{4}$	$-\frac{3}{4}$	$\frac{4}{4}$			
4	$\frac{3}{4}$	$\frac{3}{16}$	0	0	$\frac{9}{16}$		
5	1	$-\frac{4}{7}$	$\frac{3}{7}$	$\frac{12}{7}$	$-\frac{12}{7}$	$\frac{8}{7}$	
6	1	$\frac{7}{90}$	0	$\frac{32}{90}$	$\frac{12}{90}$	$\frac{32}{90}$	$\frac{7}{90}$

Let $c = (t - t_n)/h$. The two methods used in the first pair [1,3,10] are defined by

$$y_{4,7,1}(t_n + ch) = y_n + \sum_{i=1}^{5} \Omega_i c^i$$

$$y_{4,7,2}(t_n + ch) = y_n + \sum_{i=1}^{4} \omega_i c^i.$$

It is this first pair of methods which is implemented in the DRKLAG solver. Although each is a 7-stage method, the pair constitutes effectively a 6-stage pair since the final derivative evaluation for any step is used as the first derivative evaluation for the next step. For $c = 1$, $y_{4,7,1}$ is a fifth order C^1 method; and $y_{4,7,2}$ is a fourth order method. For $c \neq 1$, both methods are fourth order. For $c = 1$, the methods become

$$y_{4,7,1}(t_n + h) = y_n + \frac{7k_0 + 32k_2 + 12k_3 + 32k_4 + 7k_5}{90}$$

$$y_{4,7,2}(t_n + h) = y_n + \frac{3k_0 + 16k_2 + 4k_3 + 16k_4 + 3k_6}{42}.$$

Their difference for $c = 1$ is given by $E_{1,a} = E_1$ where

$$E_1 = (y_{4,7,1} - y_{4,7,2})(t_n + h) = \frac{4k_0 - 16k_2 + 24k_3 - 16k_4 + 49k_5 - 45k_6}{630}.$$

The second pair [1] uses a method $y_{4,7,3}$ which is similar to but more accurate than $y_{4,7,2}$. (The program in [11] may be used to see that the local truncation error coefficients for this method are smaller than the corresponding ones for $y_{4,7,2}$ by factors of about 5.) $y_{4,7,3}$ is the method denoted by y_T in [1, equations (32a)–(32e)]. The polynomial coefficients for this method are given by

$$T_1 = k_0$$

$$T_2 = \frac{-161k_0 + 176k_2 + 60k_3 - 112k_4 + 28k_5 + 9k_6}{54}$$

$$T_3 = \frac{718k_0 - 1072k_2 - 492k_3 + 1328k_4 - 392k_5 - 90k_6}{225}$$

$$T_4 = \frac{-68k_0 + 112k_2 + 72k_3 - 208k_4 + 77k_5 + 15k_6}{60}.$$

The method $y_{4,7,3}$ is defined by

$$y_{4,7,3}(t_n + ch) = y_n + \sum_{i=1}^{4} T_i c^i.$$

For each value of c, $y_{4,7,3}$ is a fourth order method. For $c = 1$, the method becomes

$$y_{4,7,3}(t_n + h) = y_n + \frac{206k_0 + 976k_2 + 336k_3 + 976k_4 + 161k_5 + 45k_6}{2700}.$$

The second pair of methods considered in this paper consists of $y_{4,7,1}$ and $y_{4,7,3}$. Their difference for $c = 1$ is given by $E_{2,a} = E_2$ where

$$E_2 = (y_{4,7,1} - y_{4,7,2})(t_n + h) = \frac{4k_0 - 16k_2 + 24k_3 - 16k_4 + 49k_5 - 45k_6}{2700}.$$

Each of the first two pairs is a continuously embedded (4,5) pair. The third pair [10,12,13] is a continuously embedded (4,6) pair. The coefficient tableau for this pair [12] is given in Tableau 2.

Tableau 2.

i	a_i	b_{ij}, $j = 0, \ldots, i-1$						
1	$\frac{1}{9}$	$\frac{1}{9}$						
2	$\frac{1}{6}$	$\frac{1}{24}$	$\frac{3}{24}$					
3	$\frac{1}{3}$	$\frac{1}{6}$	$-\frac{3}{6}$	$\frac{4}{6}$				
4	$\frac{1}{2}$	$\frac{1}{8}$	0	0	$\frac{3}{8}$			
5	$\frac{2}{3}$	$\frac{17}{9}$	$-\frac{63}{9}$	$\frac{51}{9}$	0	$\frac{1}{9}$		
6	$\frac{5}{6}$	$-\frac{22}{24}$	$\frac{33}{24}$	$\frac{30}{34}$	$-\frac{58}{24}$	$\frac{30}{24}$	$\frac{3}{24}$	
7	1	$\frac{281}{82}$	$-\frac{243}{82}$	$-\frac{522}{82}$	$\frac{876}{82}$	$-\frac{346}{82}$	$-\frac{36}{82}$	$\frac{72}{82}$

The polynomial coefficients for the methods in this pair are

$$A_1 = k_0$$
$$A_2 = \frac{-67056k_0 + 110124k_2 - 48717k_3 - 1408k_4 + 6624k_5 + 2196k_6 - 1763k_7}{11788}$$
$$A_3 = \frac{247660k_0 - 626292k_2 + 468639k_3 - 34376k_4 - 54594k_5 - 16740k_6 + 15703k_7}{17682}$$
$$A_4 = \frac{3(-120655k_0 + 369216k_2 - 354531k_3 + 68336k_4 + 39843k_5 + 11280k_6 - 13489k_7)}{23576}$$
$$A_5 = \frac{9(9961k_0 - 33804k_2 + 37287k_3 - 10328k_4 - 4113k_5 - 684k_6 + 1681k_7)}{14735}$$
$$B_1 = k_0$$
$$B_2 = \frac{-57501k_0 + 76743k_2 - 31810k_4 + 5715k_5 + 11691k_6 - 4838k_7}{11200}$$
$$B_3 = \frac{25081k_0 - 46683k_2 + 37210k_4 - 6615k_5 - 15471k_6 + 6478k_7}{2400}$$
$$B_4 = \frac{3(-69443k_0 + 147849k_2 - 156830k_4 + 33645k_5 + 80613k_6 - 35834k_7)}{22400}$$
$$B_5 = \frac{9(673k_0 - 1539k_2 + 1930k_4 - 495k_5 - 1143k_6 + 574k_7)}{2000}.$$

The methods [10,12] are defined by

$$y_{6,8,1}(t_n + ch) = y_n + \sum_{i=1}^{5} A_i c^i$$
$$y_{4,8,1}(t_n + ch) = y_n + \sum_{i=1}^{5} B_i c^i.$$

Each of the methods is an 8-stage method. For $c = 1$, $y_{6,8,1}$ is a sixth order method; and $y_{4,8,1}$ is a fourth order method. For other values of c, the two methods are fourth order.

For $c = 1$, the methods become

$$y_{6,8,1}(t_n + h) = y_n + \frac{41k_0 + 216k_2 + 27k_3 + 272k_4 + 27k_5 + 216k_6 + 41k_7}{840}$$

$$y_{4,8,1}(t_n + h) = y_n + \frac{284069k_0 + 1765233k_2 + 2202290k_4 + 207765k_5 + 1599021k_6 + 325622k_7}{6384000}.$$

Their difference for $c = 1$ is given by $E_{3,a} = E_3$ where

$$\begin{aligned} E_3 &= (y_{6,8,1} - y_{4,8,1})(t_n + h) \\ &= \frac{3}{112000}(161k_0 - 723k_2 + 1200k_3 - 790k_4 - 15k_5 + 249k_6 - 82k_7). \end{aligned}$$

Table 1 summarizes the methods used in the three pairs. Throughout this paper, we will refer to the corresponding versions of DRKLAG as DELAY1, DELAY2, and DELAY3. The intent of this paper is not to access the relative merits of the methods used or compare the performance of these solvers. Rather, it is to illustrate how the choice of different error estimates can affect the performance of a given delay solver.

Table 1. Runge-Kutta-Sarafyan delay solvers.

Code	Primary Method	Subsidiary Method	Method Orders ($c = 1$)	Interpolant Orders ($c \neq 1$)
DELAY1	$y_{4,7,1}$	$y_{4,7,2}$	5, 4	4, 4
DELAY2	$y_{4,7,1}$	$y_{4,7,3}$	5, 4	4, 4
DELAY3	$y_{6,8,1}$	$y_{4,8,1}$	6, 4	4, 4

ERROR ESTIMATES

It was observed in [3] that the difference of the solution polynomials for the Sarafyan methods used in DRKLAG factors into a convenient form. For each of the above three pairs of methods, a similar factorization is possible. We summarize the results in the following theorem which may be proved using an argument like that sketched in [3].

THEOREM. *If the step size is h, then for any $c = (t - t_n)/h$,*

$$\begin{aligned} (y_{4,7,1} - y_{4,7,2})(t) &= P_1(c)\, E_1, \\ (y_{4,7,1} - y_{4,7,3})(t) &= P_2(c)\, E_2, \text{ and} \\ (y_{6,8,1} - y_{4,8,1})(t) &= P_3(c)\, E_3, \end{aligned}$$

where the polynomials $P_i(c)$ are given by

$$\begin{aligned} P_1(c) &= 336c^5 - 855c^4 + 700c^3 - 180c^2, \\ P_2(c) &= 1440c^5 - 3735c^4 + 3096c^3 - 800c^2, \text{ and} \\ P_3(c) &= \frac{298296c^5 - 590885c^4 + 347140c^3 - 54130c^2}{421}. \end{aligned}$$

Also,

$$\begin{aligned} (y_{4,7,1} - y_{4,7,2})'(t) &= \frac{1}{h} p_1(c)\, E_1, \\ (y_{4,7,1} - y_{4,7,3})'(t) &= \frac{1}{h} p_2(c)\, E_2, \text{ and} \\ (y_{6,8,1} - y_{4,8,1})'(t) &= \frac{1}{h} p_3(c)\, E_3, \end{aligned}$$

where $p_i(c) = P_i'(c)$.

Note that the polynomials $P_i(c)$ are the same for each integration step and they do not depend on $\{k_i\}$. They may be thought of as amplification factors by which errors at integration grid points are magnified at nongrid points. They suggest simple forms of error estimation for the methods in question when applied to delay equations. For example, the maximum magnitude of $P_1(c)$ for $0 \leq c \leq 1$ is $|P_1(2/7)| = 8224/2401$ (note: [3] contains an incorrect statement regarding this), or approximately 3.4, since

$$P_1(c) = 1680 \int_0^c s\left(s - \frac{2}{7}\right)\left(s - \frac{3}{4}\right)(s-1)\,ds.$$

Rather than use E_1 to estimate the error for a given step, we may use

$$E_{1,b} = \frac{E_1}{P_1(2/7)}.$$

This amounts to requesting about an additional half digit of accuracy at each step. Intuitively, by requiring this additional accuracy, we are more confident that the off-grid solution values will be accurate within the requested error tolerance when they are used later to interpolate the solution.

A more conservative estimate is given by

$$E_{1,c} = \frac{E_1}{\int_0^1 |s(s-2/7)(s-3/4)(s-1)|\,ds} = 1680\,\frac{E_1}{2P_1(3/4) - 2P_1(2/7) - 1}.$$

The denominator in the $E_{1,c}$ estimate is about 12.4. (The corresponding factors for $E_{2,c}$ and $E_{3,c}$ are approximately 24.6 and 62.6, respectively.) Thus the difference of the two methods and hence the accuracy of the Runge-Kutta polynomial interpolant can be better controlled by requesting about an extra digit of accuracy at each step. Of course, the standard estimate $E_{1,a}$ could simply be used with a smaller error tolerance; but in the tests we have performed, the above two estimates, particularly the second, consistently do a better job of actually delivering the accuracy requested of the solver. We favor using the alternate error estimates directly because they increase our confidence in the accuracy of the off-grid solution values which are used later and because they, at the same time, arise in a natural way from the continuously embedded methods being used. Estimates similar to $E_{1,b}$, and $E_{1,c}$, and which will we denote by $E_{2,b}$, $E_{2,c}$, $E_{3,b}$, and $E_{3,c}$, are obtained easily for use with the other two pairs of methods based on the polynomials P_2 and P_3.

Although we will not pursue the matter in this paper, we note as a matter of interest that estimates based on the maximum magnitudes or integrals of P_i on the interval [0,2] are also incorporated in the solvers. Such estimates allow the accurate solution of so-called vanishing and nearly vanishing delay problems. (By using a step size for which the solution polynomial is accurate for twice this value, the solution polynomials may be extrapolated when necessary near points at which a delay vanishes.) The interested reader is referred to [14] which contains an excellent discussion of the issues and hard numerical realities associated with the solution of vanishing delay (and other) problems.

DERIVATIVE DISCONTINUITIES

A second nemesis for any solver for delay equations is the manner in which derivative jump discontinuities occur and propagate [3]. If the initial function is not compatible with the differential equation (that is, there is a derivative discontinuity of some order at $t = a$), when the delay function $\beta(t, y(t))$ later "crosses" $t = a$, the potential for a derivative discontinuity in the solution at that point exists. In general, such crossings occur at zeroes of odd multiplicity of the functions

$$\beta(t, y(t)) - Y = 0,$$

where Y is either $t = a$, or any subsequent point of derivative discontinuity. (The odd multiplicity guarantees that the delay function actually crosses the previous jump point.) The resulting tree of points with derivative discontinuities thus propagates from the initial discontinuity at $t = a$.

The present solvers contain provisions for automatically locating points of discontinuity using root finding. (They use root finding similar to that used in well-known ode solvers to locate the zeroes, and they augment the system of root functions each time a discontinuity point is found.) We will present results in the next section which illustrate the effect of having the codes locate points at which the potential for a discontinuity exists and include them as integration mesh points, and also which illustrate the effect on the above error control strategies of having the codes ignore such points.

We note that preliminary results suggest that the use of error estimates similar to those above but based on $P_i'(c)$ rather than $P_i(c)$ may provide a less expensive alternative to root finding for some problems. (For each of the three pairs, the difference of the derivatives of the methods is equal to $(1/h)P_i'(c)\,E_{i,a}$.) Such estimates are very similar to the error per step estimates used in some codes; and they effectively permit automatic switching between the usual error estimates and per step estimates near discontinuities. Results pertaining to this issue will be presented elsewhere.

EXAMPLES AND NUMERICAL RESULTS

As a first test to demonstrate the effect of using the above error estimates, we used each of the three codes to solve each of the thirty problems in the well-known DETEST set of nonstiff test problems for odes [15]. (This is not a purely academic exercise: delay codes should do a reasonable job solving odes since the need to solve systems of odes without delays arises frequently in connection with the solution of delay equations; see [8] for an example.) Each problem was solved for each of the four error tolerances 10^{-4}, 10^{-6}, 10^{-8}, and 10^{-10}, for a total of 120 cases. In each case, equal values were used for ϵ_a and ϵ_r, the absolute and relative error tolerances, respectively.

DRKLAG is modeled very closely after a well-known Runge-Kutta ode solver DDERKF [16] whenever possible. It uses mixed absolute-relative error tests and controls the integration step size depending on the magnitude of

$$\frac{|\text{estimated error}|}{\epsilon_a + \epsilon_r|\text{computed solution}|}.$$

The codes attempt to achieve a value of 0.5 for this quantity at each integration step.

Table 2 contains a summary of the results. For each of the three pairs of methods and each of the forms of error control described above, Table 2 contains the average error overrun (AVGERO) at the final integration time and the average number of derivative evaluations (AVGNFE) required for the 120 cases. By error overrun is meant the quantity

$$\frac{|\text{actual error}|}{\epsilon_a + \epsilon_r|\text{exact solution}|}.$$

In practice, it is difficult to predict the magnitude of this overrun since it depends on the stability of the differential equation; but in principle, a value of 0.5 is optimal since that is what the codes aim for at each step. The results illustrate the improvement in the delay codes' ability to control the error if either of the two alternate forms of error estimation are used. The results for the third form are particularly encouraging. Of course, nothing is free; and the codes do, in fact, have to work harder to achieve the additional accuracy. However, the resulting loss in efficiency is justified by the additional accuracy. As a crude indication of efficiency, we note that

Table 2. DETEST test results.

	AVGERO	AVGNFE
DELAY1		
$E_{1,a}$	51.6	1608
$E_{1,b}$	14.0	1985
$E_{1,c}$	3.4	2486
DELAY2		
$E_{2,a}$	98.1	1253
$E_{2,b}$	7.5	2024
$E_{2,c}$	4.0	2548
DELAY3		
$E_{3,a}$	72.2	1348
$E_{3,b}$	11.4	1767
$E_{3,c}$	2.8	2260

AVGNFE for the ode solver in [16] for this test is 1328 (although AVGERO is considerably larger than for any of the delay solver options).

As a second test, we considered the following problem.

EXAMPLE 1. [3]

$$\begin{aligned} \frac{dy(t)}{dt} &= \frac{1}{t}\, y(t)\, y(\ln(y(t))) && \text{for } t \geq 1 \\ y(t) &= 1 && \text{for } 0 \leq t \leq 1. \end{aligned}$$

The solution for this problem has derivative jump discontinuities at the points $t = 0$, $t = e$, $t = e^2$, and $t = e_3$ (orders 1, 2, and 3, respectively). The exact solution for $t \leq e_3$ is given by

$$y(t) = \begin{cases} t, & \text{if } 1 \leq t \leq e, \\ \exp\left(\dfrac{t}{e}\right), & \text{if } e \leq t \leq e^2, \\ \left(\dfrac{e}{3 - \ln(t)}\right)^e, & \text{if } e^2 \leq t \leq e_3, \end{cases}$$

where $e_3 = \exp(3 - \exp(1 - e))$.

Tables 3 and 4 contain the results for the three delay solvers and several error tolerances. The tables contain the maximum error overrun (MAXERO) during the integration and the number of derivative evaluations (NFE) required to solve the problem for each error tolerance. The results in Table 3 were obtained by using the codes' root finding option to automatically locate the points of derivative discontinuity. Table 4 contains the results obtained when the discontinuities were ignored. In both cases, the codes perform satisfactorily; but the results in the first case are clearly better. They demonstrate the potential problems associated with forcing a solver to fend for itself in the presence of discontinuities. However, in both cases, the integration overruns generally are better for the alternate error estimates.

As a third test, we considered the following problem.

EXAMPLE 2. [17]

$$\begin{aligned} \frac{dy(t)}{dt} &= y(t) + y(t-1) - \frac{1}{4}y'(t-1) && \text{for } 0 \leq t \leq 2 \\ y(t) &= -t && \text{for } t \leq 0. \end{aligned}$$

Table 3. Example 1 results (discontinuities located).

	10^{-4}		10^{-6}		10^{-8}		10^{-10}	
	MAXERO	NFE	MAXERO	NFE	MAXERO	NFE	MAXERO	NFE
DELAY1								
$E_{1,a}$	3.0	262	8.4	286	6.5	646	6.5	1558
$E_{1,b}$	1.6	286	2.5	346	1.9	826	1.9	1978
$E_{1,c}$	.45	328	.52	436	.53	1042	.53	2542
DELAY2								
$E_{2,a}$	9.1	250	4.0	226	2.8	502	27.9	1174
$E_{2,b}$	1.5	280	1.9	358	1.7	832	1.7	2008
$E_{2,c}$	.30	328	.44	460	.45	1072	.53	2626
DELAY3								
$E_{3,a}$	4.0	283	21.6	203	25.7	468	2.0	1259
$E_{3,b}$	1.0	314	3.0	275	3.6	636	1.0	1720
$E_{3,c}$	.20	331	.41	355	.71	844	.59	2204

Table 4. Example 1 results (discontinuities ignored).

	10^{-4}		10^{-6}		10^{-8}		10^{-10}	
	MAXERO	NFE	MAXERO	NFE	MAXERO	NFE	MAXERO	NFE
DELAY1								
$E_{1,a}$	15.6	136	629.5	313	360.5	664	527.6	1639
$E_{1,b}$	1.3	142	422.0	367	96.5	844	45.7	2029
$E_{1,c}$	.59	202	.98	541	46.2	1102	7.4	2617
DELAY2								
$E_{2,a}$	634.3	100	1388.8	229	408.9	514	25.9	1279
$E_{2,b}$	18.9	148	25.4	391	12.9	904	13.7	2095
$E_{2,c}$	12.3	208	6.6	535	7.5	1150	4.5	2743
DELAY3								
$E_{3,a}$	190.3	110	295.2	238	431.6	566	217.3	1373
$E_{3,b}$	364.8	134	48.5	369	25.8	735	8.1	1793
$E_{3,c}$	28.0	216	7.8	472	7.0	943	5.5	2254

The solution for this problem has first derivative jump discontinuities at $t = 1$ and $t = 2$. The exact solution for $t \leq 2$ is given by

$$y(t) = \begin{cases} -\frac{1}{4} + t + \frac{1}{4}e^t, & \text{if } 0 \leq t \leq 1 \\ \frac{1}{2} - t + \frac{1}{4}e^{t-1} + \frac{3}{16}\,t\,e^{t-1} & \text{if } 1 \leq t \leq 2. \end{cases}$$

This problem was solved using the delay codes for several error tolerances. The results obtained are summarized in Table 5. Once again, the improved performance of the codes using the alternate error estimates is evident.

We have performed similar experiments for numerous other problems from various test sets. Comparable improvements were obtained in all cases. Consequently, we recommend that solvers such as DRKLAG employ the more conservative error estimates, particularly the third (which is now the default in DRKLAG).

SUMMARY

This paper considered the use of Runge-Kutta-Sarafyan methods for the solution of delay equations. One widely used code and two experimental variants of it based on Sarafyan methods

Table 5. Example 2 results.

	10^{-4}		10^{-6}		10^{-8}		10^{-10}	
	MAXERO	NFE	MAXERO	NFE	MAXERO	NFE	MAXERO	NFE
DELAY1								
$E_{1,a}$	41.7	80	17.1	176	16.7	266	16.1	482
$E_{1,b}$	5.4	122	4.8	188	4.8	308	5.0	572
$E_{1,c}$	1.3	146	1.2	218	1.4	362	1.3	710
DELAY2								
$E_{2,a}$	74.9	70	73.9	146	71.9	224	66.7	398
$E_{2,b}$	27.6	68	4.1	194	4.4	260	4.4	584
$E_{2,c}$	1.1	146	1.1	218	1.2	368	1.1	722
DELAY3								
$E_{3,a}$	64.0	84	13.5	209	14.5	287	13.6	453
$E_{3,b}$	55.8	77	2.7	232	2.5	334	2.7	542
$E_{3,c}$	.54	172	.56	263	.60	382	.56	668

were discussed. Numerical results were presented which demonstrate that by controlling the local error using more conservative, but natural, error estimates, the accuracy of each of the codes can be significantly improved. The DRKLAG solver along with various test programs including those used to obtain the results given in this paper are available from the author. Maple programs which may be used to verify the estimates given in the theorem in this paper or modified for similar methods are also available from the author.

REFERENCES

1. D. Sarafyan, Approximate solution of ordinary differential equations and their systems through discrete and continuous embedded Runge-Kutta formulae and upgrading of their order, *Computers Math. Applic.* **28** (10–12), 353–384 (1994).
2. K.W. Neves and S. Thompson, DRKLAG: Solution of systems of functional differential equations with state dependent delays, TR-92-003, Computer Science Department Technical Report Series, Radford University, Radford, VA, (1992).
3. K.W. Neves and S. Thompson, Software for the numerical solution of systems of functional differential equations with state dependent delays, *J. Appl. Num. Math.* **9**, 385–401 (1992).
4. R.E. Bellman and K.L. Cooke, *Differential-Difference Equations*, Math. in Sci. and Eng. **6**, Academic Press, (1963).
5. C.T.H. Baker, C.A.H. Paul and D.R. Willé, A bibliography on the numerical solution of delay differential equations, Numerical Analysis Report No. 269, Department of Mathematics, University of Manchester, Manchester, England, (1995).
6. N. MacDonald, *Biological Delay Systems: Linear Stability Theory*, Cambridge University Press, Cambridge, (1989).
7. J.D. Murray, *Mathematical Biology*, 2nd edition, Springer-Verlag, New York, (1993).
8. S.P. Corwin, D. Sarafyan and S. Thompson, Solution of systems of delay differential equations with state dependent delays using the DRAKE/DRKLAG solvers (to be submitted).
9. S. Thompson, Use of a delay differential equation solver for the solution of a complex biological control model arising in the study of the human respiratory system, (in preparation).
10. D. Sarafyan, (private communication).
11. M.E. Hosea, Rapid calculation of Runge-Kutta truncation error coefficients, Technical Report #92-7, Mathematics Department, Southern Methodist University, Dallas, TX, (1992).
12. D. Sarafyan, Effective and efficient numerical solution of ordinary differential equations, In *Proceedings of SHARE XXXVIII*, San Francisco, CA, (1972).
13. S. Thompson, Implementation and evaluation of several continuously imbedded methods of Sarafyan, Technical Report ORNL/TM-10434, Oak Ridge National Laboratory, Oak Ridge, TN, (1987).
14. C.T.H. Baker, C.A.H. Paul and D.R. Willé, Issues in the numerical solution of evolutionary delay differential equations, Numerical Analysis Report No. 248, Department of Mathematics, University of Manchester, Manchester, England, (1994).
15. W.H. Enright and J.D. Pryce, Two FORTRAN packages for assessing initial value methods, *ACM Transactions on Math. Software* **13** (1), 1–27 (1987).

16. L.F. Shampine and H.A. Watts, Software for ordinary differential equations, In *Sources and Development of Mathematical Software*, (Edited by W.R. Cowell), pp. 112–133, Prentice Hall, (1984).
17. F. Kappel and K. Kunisch, Spline approximations for neutral functional differential equations, *SIAM J. Numer. Anal.* **18**, 1058–1080 (1981).