Theor Chem Acc (1998) 99:215–219
DOI 10.1007/s0021498000m5

Theoretical
Chemistry Accounts
© Springer-Verlag 1998

*Regular article*

# A new technique for the early detection of stiffness in coupled differential equations and application to standard Runge-Kutta algorithms

**Morris Bader**

Department of Chemistry, Moravian College, Bethlehem, PA 18018, USA

Received: 18 September 1997 / Accepted: 13 February 1998 / Published online: 17 June 1998

**Abstract.** Higher-order Runge-Kutta (RK) algorithms employing local truncation error (LTE) estimates have had very limited success in solving stiff differential equations. These LTEs do not recognize stiffness until the region of instability has been crossed after which no correction is possible. A new technique has been designed, using the local stiffness function (LSF), which can detect stiffness very early before instability occurs. The LSF is a normalized dimensionless ratio which is essentially based on the product of the step size and the geometric mean of all the slopes. It is exceedingly sensitive to the onset of stiffness. Together, the LSF and the LTE form a complementary pair which can cooperate to help solve some mildly stiff equations which were previously intractable to RK algorithms alone. Examples are given of implementation and LSF performance.

**Key words:** Runge-Kutta – Stiff differential equations – Numeric integration

## 1 Introduction

Runge-Kutta (RK) algorithms have always been considered superb tools for the numerical integration of ordinary differential equations (ODEs). Their limited ability to handle stiff ODEs has enabled other methods, such as Gear's predictor-corrector [1] and Milne's backward differences [2], to supersede them. However, the fact that RKs are self-starting, easy to program, and show extreme accuracy and versatility in non-stiff problems has led to their continuous analysis and use in mathematical research. One of the most exciting developments in RK usage has been the discovery that by judicious rearrangement of interim values of the RK predictor one can obtain a second predictor of one order less. The two equations are generally referred to as an RK pair. Fehlberg [3] was among the first to suggest on theoretical grounds that the difference between the two predictors would be directly proportional to the local truncation error (LTE). The unusual success of the Fehlberg approach was addressed in the popular text by Forsythe [4], and cited as the "state of the art" of RK code. The LTE is then used as a test to see whether a step has been successful, and if not, the step size is reduced (usually halved) until the LTE passes the tolerance requirement. The beauty of the RK pair is that it requires no extra function evaluations, which is the most time consuming aspect of all ODE solvers. This breakthrough initiated a search for RK algorithms of higher and higher order and better error estimates.

Shampine [5] was convinced that a high-order RK pair would be capable of solving stiff ODEs. This gave Prince and Dormand [6–8] and Dormand et al. [8] the motivation to compute some RK pairs of fairly high seventh- and eighth-order magnitude. With so many new algorithms available it was necessary to systematically quantify some of these results. Some experimental tests were devised [9] to determine the actual "working order" of the algorithm, which gave its intrinsic accuracy, and then to examine the integrity of the LTE predictor. In an RK algorithm the LTE is proportional to $h^r$, where $h$ is the step-size and $r$ is the order of the algorithm. The order is considered to be the number of terms of a Taylor series approximated, or the number of function evaluations required to take one step. However this is often misleading. By taking a known problem and calculating the LTE exactly for different step sizes, one can determine the experimental value of $r$, the "working order" of the algorithm. This number is quite important since halving the step size reduces the LTE by a factor of $2^r$. It soon became apparent that the RK pair worked better as a team, not converging or diverging as the problem evolved, otherwise the most accurate RK algorithm would still perform poorly overall. The best RK pair by all statistical measures appeared to be the Butcher algorithm [10] and an error estimate devised by this author which is shown in Table 1. The RK-Butcher algorithm is nominally considered sixth order since it requires six function evaluations, but in actual practice the "working order" is closer to five but still exceeds all the other algorithms examined including the RK-Fehlberg.

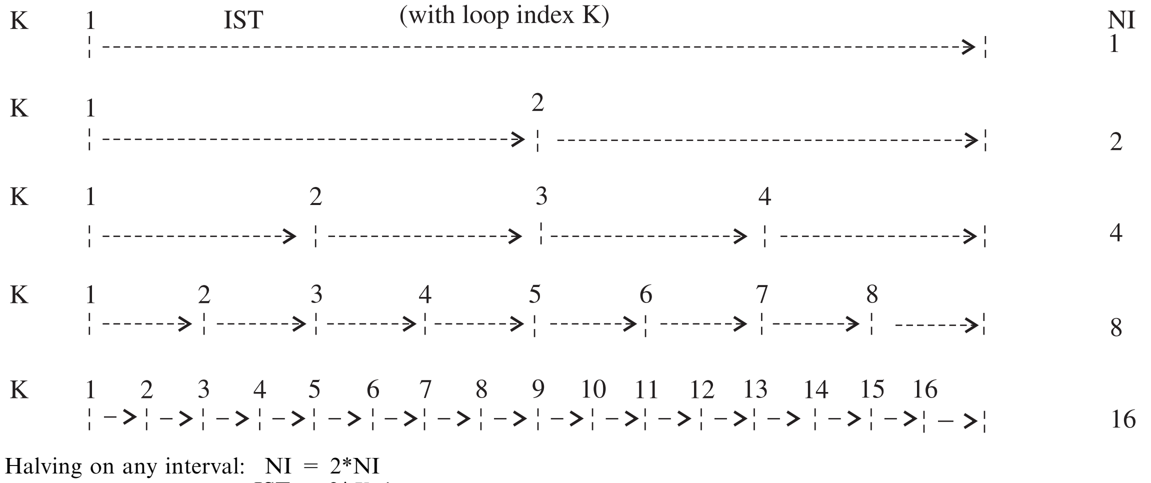
MIDAS [11, 12] is a FORTRAN computer program which was intended to be the very best ODE solver using a standard RK algorithm. It has been constantly refined to reflect every nuance of computer logic and designed to give the very best results possible. It is presently armed with the Butcher pair as the sole integrator. The construction of MIDAS follows that of DSS/2 developed by Schiesser [13] in which the specific problem is entered as a FORTRAN subroutine. Techniques were developed to take advantage of the extreme accuracy of the RK pair to select the correct step size immediately without constant halving, and to double the step size when conditions permit. MIDAS incorporates the novel "railroad" step-size control (Table 2) which guarantees that every subinterval will end on a print boundary exactly, and that no subinterval has an unusual length. For RKs the latter is exceedingly important since step size is a factor in overall accuracy. In MIDAS, step size is totally flexible and completely under program control. Table 2 shows that halving the step size is always allowed, while doubling the step size is allowed only on odd intervals. A counter allows doubling only on completion of NI/2 successful steps in the non-stiff mode, where NI is the number of subintervals in one print interval (PI). This railroad mechanism eliminates many thousands of FORTRAN statements to determine whether the step has crossed the PI boundary. Equally important is the fact that regardless of the number of subintervals which may be in the thousands, at the end of each step the PI is incremented by one PI unit and not by the sum of the subinterval steps which reduces the enormous problem of accumulated round-off error. In addition, since most of the step size bookkeeping is in the integer mode, it has made MIDAS one of the fastest and most accurate of all standard RK codes. The most surprising result occurred when it was discovered that MIDAS could solve the Edelson flare simulation [14] with ease at the time this problem was considered a very difficult benchmark test of all ODE solvers.

One of the difficulties of all RK algorithms is that they tend to explode when tackling stiff ODEs. There are many theoretical definitions of stiffness [5]. For our purposes we will consider two categories. The first occurs when the curvature of the problem becomes extreme. The error term of the RK pair follows the same path as the main predictor and reports an acceptable error while the RK pair are both far from the true value. This condition worsens while the user is totally unaware that the algorithm is reporting garbage. This problem often occurs in coupled ODEs where the equations have drastically different time constants. The step size cannot satisfy all the equations simultaneously. Such problems are exceedingly difficult for any type of algorithm. The Edelson flare problem falls in this category. The second type of stiffness is extremely devious. All ODEs have a general solution and an infinite number of particular solutions depending on the initial conditions. When small round-off and truncation errors begin to accumulate they act as if the initial conditions have been changed and now parasitic terms, particularly exponentials often appear as if by magic. Again, the problem will explode for no apparent reason. While time constant problems are usually evident beforehand, it is often

**Table 1.** RK-Butcher algorithm in equation form

$$
\begin{aligned}
&\mathrm{K1} = hF(X_n,\ Y_n)\\
&\mathrm{K2} = hF(X_n + h/4,\ Y_n + \mathrm{K1}/4)\\
&\mathrm{K3} = hF(X_n + h/4,\ Y_n + \mathrm{K1}/8 + \mathrm{K2}/8)\\
&\mathrm{K4} = hF(X_n + h/2,\ Y_n - \mathrm{K2}/2 + \mathrm{K3})\\
&\mathrm{K5} = hF(X_n + 3h/4,\ Y_n + 3\mathrm{K1}/16 + 9\mathrm{K4}/16)\\
&\mathrm{K6} = hF(X_n + h,\ Y_n - 3\mathrm{K1}/7 + 2\mathrm{K2}/7\\
&\qquad + 12\mathrm{K3}/7 - 12\mathrm{K4}/7 + 8\mathrm{K5}/7)\\
&Y_{n+1} = Y_n + (7\mathrm{K1} + 32\mathrm{K3} + 12\mathrm{K4} + 32\mathrm{K5} + 7\mathrm{K6})/90\\
&Y^*_{n+1} = Y_n + (\mathrm{K1} + 4\mathrm{K4} + \mathrm{K6})/6\\
&\mathrm{LTE} = Y_{n+1} - Y^*_{n+1}
\end{aligned}
$$

**Table 2.** Railroad mechanism for step size control


K 1 IST (with loop index K) NI
1
K 1 2
2
K 1 2 3 4
4
K 1 2 3 4 5 6 7 8
8
K 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
16

Halving on any interval: NI = 2*NI
IST = 2*$K$–1
Doubling only on an odd interval: NI = NI/2
IST = ($K$+1)/2
Step size: $h$ = PI/NI
Loop: DO ( ) $K$ = IST, NI ($K$ = step index)

impossible to anticipate hidden parasitic terms. Some harmless looking equations are often killer problems for RK algorithms in general. This disability arising from stiffness is what keeps many researchers looking for a still higher-order RK pair that may be immune from this disease.

In this paper we present a new approach, that of detecting the onset of stiffness very early while the algorithm still has enough stability to deal with it. The stiffness detector is a type of error function which is independent of the RK algorithm itself and does nothing else but put up a warning flag that stiffness has been detected.

## 2 Method

We assume that when a problem is first initialized and the derivatives evaluated for the first time, the problem is not yet stiff. Generally the first few steps of any problem are not at all stiff. This fact is utilized by most of the stiff ODE solvers which use some type of simple RK to calculate the initial values to get the system started. If we allow the PI to represent the size of the first RK step, $DY$ to represent the derivatives, and $i$ the index over all equations, we can then write a function:

$$\mathrm{HSTD} = \mathrm{PI} * \mathrm{SQRT}\left[\sum (DY_i)^2)\right] \ .$$

HSTD can be called a ‘‘characteristic length’’ and represents the largest allowable step size for an acceptable solution of any problem stiff or not. The user has the obligation to select a value for PI which is appropriate to the problem and not excessive. The program will print out all values at each PI boundary. HSTD is only calculated once at the very beginning of the problem and maintains a constant value throughout. On the chance that HSTD is zero (very unlikely), it can be set to one with no change of effect. As soon as the RK begins to advance we calculate:

$$\mathrm{LSF} = h * \mathrm{SQRT}\left[\sum (DY_i)^2)\right] \Big/ \mathrm{HSTD} \ ,$$

where $h$ is the present step size and LSF is the local stiffness function, which is a dimensionless ratio. Essentially, the LSF warns us when the step size $h$ is perceived to be too large for the system of equations, hence the designation of HSTD as a characteristic length.

The LSF function is based closely on the work of Shampine and Gordon [15] who stipulated that the step size and curvature are linked together by the stability of the problem, with the proviso that the step size and curvature be inversely related. In effect, we took Shampine's verbal conjecture and transformed it into the closest quantitative mathematical function we could devise. The assumption is that if we start in a region of stability the LSF will warn us very early if we are leaving that zone. As we shall see, the LSF handles that job remarkably well. The form of the function guarantees that the LSF will be a real (not integer) positive number and that it will increase as the problem increases in stiffness. The LSF function is normalized so that it becomes a relative value which makes it general enough to be applicable to any problem of any number of equations. The LSF is very sensitive to curvature since it is the geometric mean of all the slopes and influenced mainly by the steepest values. It is important to note that the LSF is not an error estimate in the usual sense, it detects stiffness only and is used to keep the problem in the stable region where the RK pair is still functional.

**Table 3.** Empirical interpretation of LSF

| LSF range | Interpretation |
|---|---|
| 0–1 | Problem is not stiff, RK pair acceptable |
| 1–5 | Slightly stiff, results still acceptable |
| > 5 | Results may be unreliable |

We can examine the following empirical heuristic Table 3.

With this information it is possible to ward off the oncoming stiffness, but drastic methods must be taken early if the problem is to be solved. The simplest approach is to reduce the step size precipitously without interfering with the railroad mechanism. This can be achieved in an indirect manner. Let TOL be the user specified error (absolute or relative) and LTE the estimate of the RK pair. We can then use the following action to force the algorithm into the stiff mode:

**Table 4.** Use of LSF to affect toletrance

| LSF range | Action |
|---|---|
| 0.5–1.0 | TOL = TOL ∗ 1.0D-7 |
| 1.0–5.0 | TOL = TOL ∗ 1.0D-2 |
| > 5.0 | TOL = TOL ∗ 1.0D-2 |

The point of Table 4 is that TOL still plays a strong role in determining overall accuracy. TOL is decreased enormously which in turn decreases step size. Once the stiffness is abated, TOL can be reset to its initial value and the problem reset to the non-stiff mode. One of the most important constructs of the MIDAS code is the overall feedback between the LTE estimate, step size, TOL, and the LSF function. Setting TOL to a small value pushes the algorithm severely as seen below. The following piece of FORTRAN code determines the step size both in the stiff and non-stiff modes. This code brings the step size to the correct value without constant halving and testing. It also greatly reduces step size oscillation. Very little of the prior non-stiff program code need be altered to include the LSF function. It should be very easy to insert in any code without rewriting the entire program. Note that the following code still implements the railroad mechanism of Table 2. Here PI is divided into NI subintervals by the code below, and step size $h$ = PI/NI. Note also that integer arithmetic is used wherever possible to speed processing.

```
c...Implementation of the railroad mechanism
c...This code determines number of subintervals NI
c...which determines step-size; h = PI/NI
c...IST and K are loop indices
c...NADJ is a temporary variable
c...Compare this code with diagram in Table 2
      if(LTE .gt. TOL)then
        nadj = 1
        if(TOL.gt. 0.0)nadj = 1 + dlog10(LTE/TOL)
        nadj = 2**nadj
        ni = ni*nadj
        ist = 1 + nadj*(k-1)
      endif
```

When the LTE exceeds the allowed value of TOL the step size is reduced, the problem is restarted from the last successful subinterval, and the LSF retested to see whether this new step size is acceptable. As seen above, the use of the LSF function, which affects only TOL, does not invalidate the use of the RK pair as they mesh together very well. Later on, when the LSF is brought to a small value the non-stiff mode can be re-invoked. With these refinements, and by use of both the LSF and the RK pair, it has been possible for MIDAS to solve a few stiff problems that were previously intractable by standard RK algorithms.

## 3 Problems

### *3.1 Example 1*

The first problem [2] has been selected to show how a simple ODE can turn into a stiff one without any

**Table 5.** Variation of LSF in stiff and non-stiff modes

| Run | C | Maximum LSF | |
|---|---|---|---|
| | | Stiff mode OFF | Stiff mode ON |
| 1 | 0.0 | 0.00 | 0.00 |
| 2 | 0.2 | 35.85 | 0.78 |
| 3 | 0.4 | 97.27 | 0.92 |
| 4 | 0.6 | 220.11 | 1.40 |
| 5 | 0.8 | 588.62 | 3.76 |

warning. This is a single ODE that is used to demonstrate the RK code in the stiff and non-stiff modes.

The problem is: $dY/dX = X + Y$.

Let us choose $Y(0)$ to be $C - 1$ with $C$ in the range of $0.0 < C < 0.8$.

When $C$ is 0.0 the solution is $Y = -(X + 1)$, a simple linear equation. However, as soon as $C$ takes any other value, even infinitesimally different from 0.0, the solution abruptly changes to $Y = C * \exp(X) - (X + 1)$ and a hidden exponential suddenly comes to life. So, if one chooses $C = 0$, and round-off and truncation errors occur, it may appear as if the initial conditions were changed and an unanticipated exponential suddenly appears. Since one does not usually know the analytic solution in advance, such terms are usually referred to as "parasitic". We will run this problem five times varying $C$ by an increment of 0.2 from 0.0 to 0.8, and let $X$ run from 0.0 to 5.0 with a step of 1.0. As this is only a slightly stiff problem the output is not necessary since MIDAS prints the analytic solution exactly within the user specified value of TOL. The purpose of this exercise is to show the maximum values of the LSF in the non-stiff and stiff modes. Table 5 shows how the LSF follows the curvature exactly.

Although the LSF in this problem is rather large and shows considerable stiffness, the stability of MIDAS in the non-stiff mode can still overcome these difficulties. In the non-stiff mode the LSF function is ignored. As shown in Figs. 1 and 2, both sets of LSF values follow approximately the shape of the exponential which is the basic cause of the stiffness. When the parameter $C$ is zero

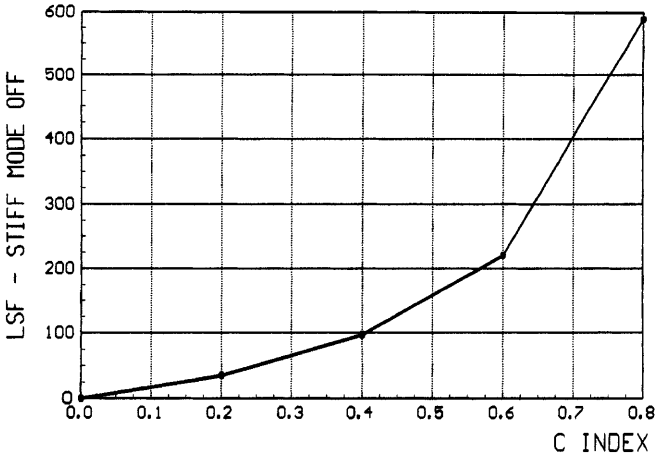


**Fig. 1.** Plot of the local stiffness function vs. the variable coefficient c with normal step size control (stiff mode off)

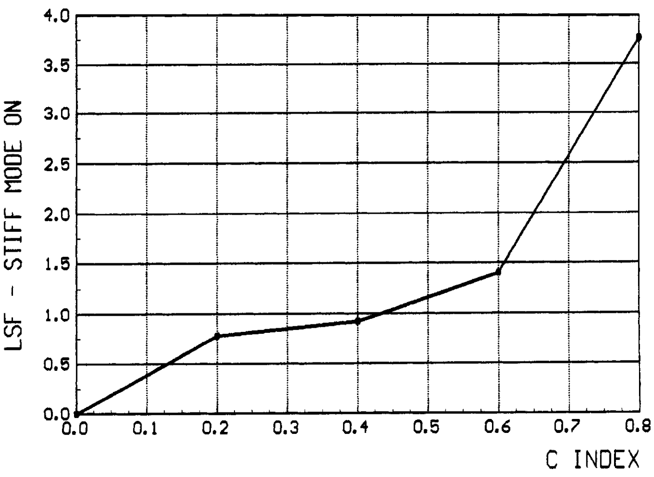


**Fig. 2.** Plot of the local stiffness function vs. the variable coefficient c with automatic rapid decrease in step size (stiff mode on)

the equation is a straight line and the LSF maximum over the entire problem shows a correct zero value. Clearly, this problem demonstrates that the LSF function and the MIDAS code performed as well as expectation would permit. The only downside to this entire procedure is the cost in derivative evaluations. In the non-stiff mode the number of derivative calls summed over all five runs was 1,118, while in the stiff mode it took 63,050. While this is a significant increase, the time factor was hardly noticeable on a 66 MHz PC. It took only a second or so for the stiff problem to complete.

## *3.2 Example 2*

The second problem is an unusually deceptive pair of stiff ODEs which look harmless enough at first glance, but carry an enormous parasitic term ready to explode. This exercise shows the great power and versatility of the LSF function in a truly difficult problem.

$$DY/DX = Z \ ,$$

$$DZ/DX = 3 - 2X^2 + 2Y - Z \ ,$$

with $0 < X$ (step 1) $< 50$. The analytical solution is rather simple:

$$Y(\text{anal}) = X(X + 1) \quad Y(o) = 0 \quad TOL = +/- 0.0001$$

$$Z(\text{anal}) = 2X + 1 \quad Z(o) = 1 \ .$$

The general solution is:

$$Y(\text{anal}) = C * \exp(X) + X(X + 1)$$

and

$$Z(\text{anal}) = C * \exp(X) + 2X + 1 \ .$$

In this problem $DZ/DX$ has the constant value 2.00 throughout the entire range of $X$, and any slight deviation from that is strong evidence that the problem is about to explode. The sums and differences in the $DZ/DX$ equation cause round-off error to accumulate so rapidly that the constant $C$ in the general solution comes

**Table 6.** Output for pair of stiff ODEs: [a]
Run no. 1. Initial value independent variable = 0.000E + 00; final value independent variable = 5.000E + 01. No. of equations = 2; print interval = 1.000E + 00; type of error... abs; error criterion = 1.000E-04; maximum interval ratio = 513

| $X$ | $Y$ | $Y$-Anal | $Z$ | $Z$-Anal | $DY$ | $DZ$ | LSF |
|---|---|---|---|---|---|---|---|
| 0.0 | 0.000 | 0.000 | 1.000 | 1.000 | 1.000 | 2.000 | 0.000 |
| 10.0 | 110.000 | 110.000 | 21.000 | 21.000 | 21.000 | 2.000 | 0.134 |
| 20.0 | 420.000 | 420.000 | 41.000 | 41.000 | 41.000 | 2.000 | 0.136 |
| 30.0 | 930.000 | 930.000 | 61.000 | 61.000 | 61.000 | 2.000 | 0.413 |
| 40.0 | 1640.000 | 1640.000 | 81.000 | 81.000 | 81.000 | 2.000 | 0.276 |
| 50.0 | 2550.000 | 2550.000 | 101.000 | 101.000 | 101.000 | 2.000 | 0.346 |

[a]Summary stiff option.....ON; maximum stiffness = 0.622; no. of derv. calls = 103317; end of run (1)

alive causing the two exponentials in $Y$ and $Z$ to accelerate each other. One can attribute the difficulty mainly to round-off and not algorithmic truncation error since one can immediately see a difference in results if the constants 2 and 3 in the $DZ$ equation are set to single or double precision. In Table 6 the output has been abbreviated to every tenth point. Note that in the stiff mode the maximum LSF over the entire output can be held to 0.622 which allows for the exact analytical results up to $X = 50$. Some years ago this problem was used to test various RK algorithms on a large CDC 6600 computer with 29 significant digits which practically eliminates round-off error. Still, very few algorithms could get past $X = 10$ before blowing up. Again in this work, the only quantity which shows any extreme value is the number of function calls, in this case over 103,000. Although it is true that so many thousands of small steps accumulate round-off and truncation error so rapidly that they defeat most RK algorithms, in this case it does not happen. A probable explanation is that no matter how many steps are taken, the railroad mechanism updates the independent variable (in this case $X$) by one PI unit at the end of each successful step, keeping the round-off error in $X$ constant regardless of the number of subintervals. Since both parasitic terms are independent of $Y$, of the form $\exp(X)$, it takes much longer before the problem explodes. One may consider this particular example a severe test of all ODE solvers and RK algorithms especially.

## 4 Conclusions

In summary, the LSF appears to be an efficient and useful concept. For example, it helps explain why the Edelson flare problem was relatively easy to solve in the non-stiff mode since the maximum LSF value is 0.997 over the entire range. Thus the problem is not stiff at all to the RK-Butcher algorithm. It is our strong belief that the inclusion of the LSF function can elevate the performance of RK algorithms in general and improve any other procedures which require advance warning of the onset of stiffness.

*Acknowledgements.* The author would like to offer sincere thanks and appreciation to Dr. William E. Schiesser, McAnn Professor of Chemical Engineering, Lehigh University, Bethlehem, PA, for his constant encouragement and advice which made this work possible. He would also like to thank his son William A. Bader M.S.E.E., Senior Analyst, Software Consulting Services, Bath, PA, and recipient of the Mathematics Medal, Lehigh University, 1986, for his perceptive discussions on the LSF function.

## References

1. Gear CW (1971) Commun ACM 14:176–180
2. Milne EM (1970) Numerical solution of differential equations. Dover, New York, pp 49–51
3. Fehlberg E (1970) Computing 6:61–71
4. Forsythe GE (1977) Computer methods for mathematical computations. Prentice-Hall, Englewood Cliffs, pp 121–133
5. Shampine LF (1977) ACM Trans Math Software 3:44–53
6. Prince PJ, Dormand JR (1981) J Comp Appl Math 7:67–75
7. Dormand JR, Prince PJ (1984) IMA J Numer Anal 4:169–184
8. Dormand JR, El-Mikkawy MEA, Prince PJ (1987) IMA J Numer Anal 7:423–430
9. Bader M (1987) Comput Chem 11:121–124
10. Butcher JC (1964) J Aust Math Soc 4:179
11. Bader M (1976) MIDAS. A new tool for the study of ordinary differential equations. Scientific programmers, Bethlehem, Pa.
12. Bader M (1988) Conference on the Numerical Solution of Initial Value Problems for Ordinary Differential Equations. University of Toronto, Toronto, Canada
13. Schiesser WE (1976) DSS/2. Differential systems simulator, version 2, Lehigh University, Bethlehem, Pa.
14. Edelson D (1975) J Chem Educ 52:642–644
15. Shampine LF, Gordon MK (1975) Computer solution of ordinary differential equations. Freeman, San Francisco, Calif., pp 71–74