Remark on Algorithm 669

DESMOND J. HIGHAM
University of Toronto

Algorithm 669 [1, 2] is one of the few widely available, high-quality Runge–Kutta integrators with an efficient interpolation facility. The purpose of this note is to point out that the first derivative approximations computed from the interpolant can be made less sensitive to rounding errors.

In solving

$$dy/dx = f(x, y), \quad y(x_0) = y_0 \in \mathbf{R}^s,$$

the method in [1] advances from $y_n \approx y(x_n)$ to $y_{n+1} \approx y(x_n + h)$ and $y_{n+2} \approx y(x_n + 2h)$ by forming

$$y_{n+1} = y_n + h \sum_{\iota=1}^{6} b_{1\iota} k_\iota, \tag{1}$$

$$y_{n+2} = y_n + h \sum_{\iota=1}^{9} b_{2\iota} k_\iota. \tag{2}$$

Here the k_ι are values of the derivative function f. Approximations at intermediate points $x = x_n + \sigma h$, for $\sigma \in (0, 1) \cup (1, 2)$ are obtained using the polynomial

$$\begin{aligned} p(x_n + \sigma h) = y_n &+ \sigma h k_1 + \sigma^2 \left[h \sum_{\iota=1}^{6} b_{1\iota} k_\iota - h k_1 \right] \\ &+ \sigma^2 (\sigma - 1) \left[h k_7 - 2h \sum_{\iota=1}^{6} b_{1\iota} k_\iota + h k_1 \right] \\ &+ \tfrac{1}{4} \sigma^2 (\sigma - 1)^2 \left[h \sum_{\iota=1}^{9} b_{2\iota} k_\iota - 4 h k_7 - 2 h k_1 + 4h \sum_{\iota=1}^{6} b_{1\iota} k_\iota \right] \\ &+ \tfrac{1}{8} \sigma^2 (\sigma - 1)^2 (\sigma - 2) \left[2 h k_{10} + 8 h k_7 + 2 h k_1 - 6h \sum_{\iota=1}^{9} b_{2\iota} k_\iota \right]. \end{aligned} \tag{3}$$

Authors' address: Department of Computer Science, University of Toronto, Toronto, Ont., Canada, M5S 1A4

Table I. Errors in the Interpolant

	Unmodified		Modified	
h	E	E'	E	E'
1	3.63e0	−7.74e0	3.63e0	−7.74e0
1e-1	3.30e-6	2.09e-5	3.30e-6	2.07e-5
1e-2	1.98e-8	−2.46e-6	1.98e-8	6.36e-7
1e-3	−1.35e-8	−4.69e-5	−1.35e-8	5.39e-8

Note that $p(x)$ is the quintic Hermite polynomial interpolant to the solution data y_n, y_{n+1}, y_{n+2} and derivative data $k_1 := f(x_n, y_n)$, $k_7 := f(x_n + h, y_{n+1})$, $k_{10} := f(x_n + 2h, y_{n+2})$. The subroutine EXTRA that evaluates $p(x)$ and $p'(x)$ is passed the data y_n, y_{n+1}, y_{n+2}, k_1, k_7 and k_{10}. The increments $h\sum_{i=1}^{6} b_{1i}k_i$ and $h\sum_{i=1}^{9} b_{2i}k_i$ appearing in (3) are computed as $y_{n+1} - y_n$ and $y_{n+2} - y_n$ respectively (see (1) and (2)). Now standard error analysis shows that using $(y_n + h\sum_{i=1}^{6} b_{1i}k_i) - y_n$ to recover $h\sum_{i=1}^{6} b_{1i}k_i$ can generate rounding errors of order $y_n u$, where u represents the machine unit roundoff. This can be a relatively large error when h is small. The rounding error above is unlikely to pose a problem in the evaluation of $p(x)$ via (3), since here roundoff of order $y_n u$ is inherent. However, in the evaluation of the first derivative of the interpolant, $p'(x) := h^{-1}d/d\sigma(p(x_n + \sigma h))$, the rounding error arising from the formation of the increments makes a potentially significant order $h^{-1}y_n u$ contribution. Here the value we are attempting to compute is the order of f.

The difficulty can be overcome by computing the increments $\sum_{i=1}^{6} b_{1i}k_i$ and $\sum_{i=1}^{9} b_{2i}k_i$ directly, in the main integration routine. This can be done without an increase in storage; for example, the local arrays Y1 and F3 in the code are free to be used once y_{n+2} has been computed. The increments, rather than y_{n+1} and y_{n+2}, can then be passed to the interpolation routine EXTRA.

To illustrate the effect of this modification we give some results for the scalar test problem $y' = 4(2 - y)$, $y(0) = 1$. We forced the code to take a single step of size h and computed the errors $E = p(x_0 + .75h) - y(x_0 + .75h)$ and $E' = p'(x_0 + .75h) - y'(x_0 + .75h)$. (The value .75 was chosen arbitrarily.) The single precision version of the code was used ($u \approx 1.2 \times 10^{-7}$), with the errors computed in double precision. The results are recorded in Table I. The effect of the order $h^{-1}u$ rounding error in the unmodified evaluation of $p'(x)$ can be seen clearly.

It is worth stressing that the instability in the original implementation of $p'(x)$ is only likely to cause *severe* loss of significant figures in fairly unusual circumstances—typically when "hard" problems requiring $h \leq 10^{-2}$ are being solved to almost full machine accuracy.

A revised version of the algorithm now documents the changes described above. The changes can be implemented in a straightforward manner by deleting certain lines and 'uncommenting' others.

REFERENCES

1. CASH, J. R. A block 6(4) Runge–Kutta formula for nonstiff initial value problems. *ACM Trans. Math. Softw. 15*, 1 (1989) 15–28
2. CASH, J. R. Algorithm 669, BRKF45: A FORTRAN subroutine for solving first-order systems of nonstiff initial value problems for ordinary differential equations. *ACM Trans. Math. Softw. 15*, 1 (1989) 29–30.