A Variable Order Runge-Kutta Method for Initial Value Problems with Rapidly Varying Right-Hand Sides

J. R. CASH
Imperial College
and
ALAN H. KARP
IBM Scientific Center

Explicit Runge-Kutta methods (RKMs) are among the most popular classes of formulas for the approximate numerical integration of nonstiff, initial value problems. However, high-order Runge-Kutta methods require more function evaluations per integration step than, for example, Adams methods used in PECE mode, and so, with RKMs, it is especially important to avoid rejected steps. Steps are often rejected when certain derivatives of the solution are very large for part of the region of integration. This corresponds, for example, to regions where the solution has a sharp front or, in the limit, some derivative of the solution is discontinuous. In these circumstances the assumption that the local truncation error is changing slowly is invalid, and so any step-choosing algorithm is likely to produce an unacceptable step. In this paper we derive a family of explicit Runge-Kutta formulas. Each formula is very efficient for problems with smooth solutions as well as problems having rapidly varying solutions. Each member of this family consists of a fifth-order formula that contains imbedded formulas of all orders 1 through 4. By computing solutions at several different orders, it is possible to detect sharp fronts or discontinuities before all the function evaluations defining the full Runge-Kutta step have been computed. We can then either accept a lower order solution or abort the step, depending on which course of action seems appropriate. The efficiency of the new algorithm is demonstrated on the DETEST test set as well as on some difficult test problems with sharp fronts or discontinuities.

Categories and Subject Descriptors: G.1.7 [**Numerical Analysis**]; Ordinary Differential Equations

Additional Key Words and Phrases: Differential equations, Runge-Kutta methods, singularities

1. INTRODUCTION

Explicit Runge-Kutta methods (RKMs) are among the most popular classes of formulas for the numerical integration of the nonstiff initial value problem

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0. \tag{1}$$

Authors' addresses: J. R. Cash, Department of Mathematics, Imperial College, South Kensington, London SW7 2AZ, England; A. H. Karp, IBM Scientific Center, 1530 Page Mill Road, Palo Alto, CA 94304.

Because of their one-step nature, Runge-Kutta methods are self-starting, and can change the step size of integration as often and by as much as required. Efficient Adams formulas, such as the widely used code STEP [14], can also change step size at every step, but in order to reduce overhead and to enhance its theoretical support, it is strongly biased against doing so. RKMs also have the advantage that the theory supporting the interpolation in a variable step Adams code is less well developed than for Runge-Kutta methods.

One frequently quoted disadvantage of RKMs is that they require more function evaluations per step than do linear multistep methods used in PEC or PECE mode, and so may be uncompetitive with Adams methods if function evaluations are very expensive. However, one of the present authors (AHK) has been heavily involved in a spectral method for the solution of oil-reservoir problems. These investigations have shown that, despite the fact that function evaluations are extremely expensive, the great flexibility of step size allowed by RKMs often solves these problems somewhat more efficiently, and with considerably less storage space, than when using Adams methods.

This is due primarily to the erratic behavior of the solution, which typically has smooth regions coupled to very sharp fronts. An effective method for such problems needs to make frequent changes of step size. In solving these oil-reservoir problems, it was very noticeable that in regions where the solution trajectory was particularly rough, there were many rejected steps, which resulted in an expensive integration. In part, the present investigation arose from a desire to predict that a step will be rejected and to quit, or accept a lower order solution, before all the function evaluations required to complete a full Runge-Kutta step had been computed.

It is clear that, even though linear multistep methods use fewer function evaluations per step than Runge-Kutta methods, this advantage can be lost when the solution requires frequent changes in step size. Thus the claim that Runge-Kutta methods are generally less efficient than linear multistep methods when function evaluations are expensive—because they do more function evaluations per step—is not always valid. In practice it is often found that Runge-Kutta methods take larger steps, on average; and it is the distance divided by the cost that is important. However, when the step size is restricted by, for example, output requirements or nonsmooth behavior such as singularities, so that Runge-Kutta methods are unable to take a relatively large step, it is the absolute cost that is relevant.

For the oil-reservoir problems that one of us has solved, it is the flexibility of the step-size selection in the Runge-Kutta methods that is important. Typically the step size is cut by a large factor every 20 or so steps and by smaller amounts every 5 to 10 steps. While a Runge-Kutta method can decrease the step size a small amount, or increase the step size by a large amount at every step (typical codes allow a step increase by a factor of 5), linear multistep methods (as typified by [14]) can only halve the step size or double it every few steps. The ability of Runge-Kutta methods to change the step size by small amounts, as needed by the problem, in addition to increasing the step size rapidly when entering a smooth region, gives the Runge-Kutta methods a larger average step size than multistep methods on our reservoir problems.

Once we became interested in problems with solutions exhibiting very sharp fronts, it was natural for us to consider discontinuous initial value problems which are, in a sense, the limiting case. We use the term *discontinuous initial value problem* rather loosely to refer to a problem with a solution having a discontinuous derivative of some order. Such problems have recently become the focus of renewed attention [4, 7, 8], and there is considerable interest in solving them efficiently, reliably, and automatically. Although discontinuous IVPs are not our main topic of interest, we show in Section 4 that the algorithms developed in this paper do perform very efficiently on a large class of such problems.

A very widely used fixed-order Runge-Kutta code is RKF45 of Shampine and Watts [16]. The RKM on which this code is based uses a total of six function evaluations per step to give a main formula of order 5 and an imbedded formula of order 4. The difference between these two solutions gives a local error estimate in the fourth-order solution. If this error is less than a prescribed tolerance, it is normally the fifth-order solution that is accepted (i.e., local extrapolation is performed). This procedure is, of course, simply the well-known Fehlberg imbedding technique applied to a 5(4) formula.

An important extension to this idea was given by Bettis [1], and further developed by Verner [17]. Verner constructed special Runge-Kutta formulas, which he called CSIRK methods, that contain a complete set of imbedded RKMs. He gave families of order p that contain imbedded methods of all orders $1, 2, \ldots, p-1$ for $p = 5, 6, 7$, and 8. One possible application of these formulas is in the construction of a family of variable-order, imbedded Runge-Kutta methods with order $\leq p$. Shampine et al. [15] used a similar idea to choose between 4(3) and 8(7) explicit Runge-Kutta methods. They showed that there is often a big advantage in varying the order of Runge-Kutta codes. An important conclusion of Shampine et al. is that Runge-Kutta methods are very efficient if the order is properly matched to the accuracy required. Furthermore, they claim that if one could select at each step the best fixed-order Runge-Kutta code for that step, the resulting algorithm would compete with the best Adams codes in terms of derivative evaluations, and would possess several important advantages. We have found this to be especially true for problems having solutions with sharp fronts or discontinuities where a low-order solution often has very good accuracy while a higher order one has very poor accuracy.

Although there are some similarities between our approach and that of Shampine et al. [15], the two differ in several important respects. First, we implement a complete set of Runge-Kutta formulas, whereas they consider only 4(3) and 8(7) formulas. Second, we only accept a lower order solution that passes the error test if higher order solutions fail this test. This strategy is justified later in this paper. Finally, we are mainly interested in applying our formulas to problems having regimes where the solution is "rough" or discontinuous. In particular, our approach has proved to be very effective for integrating through points where certain low-order derivatives of the solution are either discontinuous or extremely large. For such problems, a low-order formula is normally more effective than a higher order one, and also has the possibility of giving a reasonable error estimate. Shampine et al. explicitly exclude this class of problem.

The variable order RKM which we describe reduces substantially the number of function evalutions for many of the problems we have solved and requires only a slight increase in overhead, which can normally be neglected if function evaluations are reasonably expensive. The basic formula we use is (6, 5) CSIRK, derived using the theory developed by Verner [17]. A six-stage CSIRK formula allows the construction of RKMs of all orders from 1 to 5, but as we explain in Section 3, we only allow the possibility of accepting orders 2, 3, or 5. At each step we use Fehlberg imbedding to compute an estimate of the error in lower order solutions. If the local error estimate in a low-order solution indicates that the step is likely to be rejected at order 5, we immediately reject the step and investigate the possibility of accepting a lower order solution.

This strategy has a number of advantages. If the solution has a sharp front or discontinuity, we frequently get the largest increase in step size in fewer than six function evaluations. More importantly, since we can often detect when the step needs to be cut, without computing all six function evaluations, the penalty for rejecting a step is greatly reduced. We have modified the code RKF45 of Shampine and Watts to incorporate these ideas. A listing of the code is available from one of the present authors (JRC) and from NETLIB.

2. A MODIFIED CSIRK METHOD

In this section we derive a special Runge-Kutta formula of the form

$$
\begin{array}{c|cccccc}
0 & 0 & & & & & \\
c_2 & a_{21} & 0 & & & & \\
c_3 & a_{31} & a_{32} & 0 & & & \\
c_4 & a_{41} & a_{42} & a_{43} & 0 & & \\
c_5 & a_{51} & a_{52} & a_{53} & a_{54} & 0 & \\
c_6 & a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & 0 \\
\hline
 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6
\end{array}
\tag{2}
$$

which has order 5, and also contains a complete family of imbedded Runge-Kutta formulas of orders 1, 2, 3, and 4. Runge-Kutta formulas with a complete family of imbedded methods, called CSIRK methods, have been studied in detail by Verner [17], who derived CSIRK formulas of order 5, 6, 7, and 8. Because of the special application we have in mind, we derive a different CSIRK formula from the one given by Verner. In what follows, we describe the design criteria for our formulas.

Our first concern is to derive formulas and error estimates that are of "good quality" for general initial value problems. With this in mind, we chose the coefficients of our CSIRK formulas so that their local truncation errors have the form suggested by Dormand and Prince [5] and Shampine [13]. It is useful to give a brief summary of these ideas here.

For discussion, we wish to compute a numerical solution of the initial value problem (1) using a Runge-Kutta formula of order p. We further assume that the numerical solution y_n at the step point x_n has already been computed and that our pth-order Runge-Kutta formula gives a solution y_{n+1} at the step point

$x_{n+1} = x_n + h$. The local solution $u(x)$ is defined to be the solution of

$$\frac{du}{dx} = f(x, u), \quad u(x_n) = y_n. \tag{3}$$

It is well known that

$$\begin{aligned} &u(x_n + h) - y_{n+1} \\ &\quad = h^{p+1} \sum_{j=1}^{\lambda_{p+1}} T_{p+1,j} D_{p+1,j} + h^{p+2} \sum_{j=1}^{\lambda_{p+2}} T_{p+2,j} D_{p+2,j} + O(h^{p+3}), \end{aligned} \tag{4}$$

where λ_{p+1} and λ_{p+2} are integers depending on p, the $D_{p+i,j}$ are elementary differentials depending on the problem, and the $T_{p+i,j}$ are error coefficients depending on the Runge-Kutta formula. It has been suggested by Dormand and Prince [5] that every elementary differential should contribute to both the local error and its estimate. We have constructed our formulas in line with this suggestion. From Eq. (4), we see that this requirement is $T_{p+1,j} \neq 0$ for all j.

The next design criterion of interest concerns the relative size of the local error in the imbedded formula compared with that of the higher order formula. The larger the error in the imbedded formula, the more accurate the local error estimate will tend to be. However, the step control procedure will be more conservative, resulting in smaller steps and a generally more expensive integration. This question has been investigated by Shampine [13], and we have derived our formulas with his suggestions in mind. For further comments on this subject and for additional references, see Cash [3].

There is a third important design criterion we need to take into account: the fact that we are particularly interested in nonsmooth solutions. We wish to choose the c_i so that they span the range [0, 1] with reasonable uniformity. To explain the need for this criterion, we describe our approach for detecting nonsmooth behavior, which can best be done by considering imbedded formulas of order 2 and 3. The second-order formula involves function evaluations at points x_n and $x_n + c_2 h$, while the third-order formula involves function evaluations at $x_n + c_i h$, $1 \leq i \leq 4$. Embedded in the second-order formula we have a formula of order 1, and an estimate of the error in the first- and second-order solutions can be obtained by imbedding in the usual way. If the order 1 and 2 solutions are sufficiently accurate (in a precisely defined sense, to be explained in Section 3), then we go on to compute the fourth-order solution with the expectation that the step length is sufficiently small to allow a highly accurate, high-order solution to be computed. If, however, the first-order solution has good accuracy while the second-order solution has very poor accuracy, then we would expect a "trouble spot" in the range $[x_n + c_2 h, x_n + c_4 h]$. If this were the case, we would accept the order 1 solution and reduce the step (as described in Section 3). Similarly, we have a well-defined course of action if the first-order solution has poor accuracy.

Given this strategy, it is clear that we would like the c_i to span the range [0, 1] and to be reasonably equally spaced, with one of the $c_i = 1$. This choice gives us a very good chance of detecting bad behavior in the right-hand side of Eq. (1), if it occurs. After having derived all the order relations for our CSIRK formulas, we attempted to choose the coefficients to satisfy all of these (somewhat impre-

cise) design criteria. The formula that we finally derived is:

$$
\begin{array}{c|cccccc}
0 & 0 & & & & & \\
\frac{1}{5} & \frac{1}{5} & 0 & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & 0 & & & \\
\frac{3}{5} & \frac{3}{10} & -\frac{9}{10} & \frac{6}{5} & 0 & & \\
1 & -\frac{11}{54} & \frac{5}{2} & -\frac{70}{27} & \frac{35}{27} & 0 & \\
\frac{7}{8} & \frac{1631}{55296} & \frac{175}{512} & \frac{575}{13824} & \frac{44275}{110592} & \frac{253}{4096} & 0 \\
\hline
 & \frac{37}{378} & 0 & \frac{250}{621} & \frac{125}{594} & 0 & \frac{512}{1771} \quad \text{Order 5} \\
 & \frac{2825}{27648} & 0 & \frac{18575}{48384} & \frac{13525}{55296} & \frac{277}{14336} & \frac{1}{4} \quad \text{Order 4} \\
 & \frac{19}{54} & 0 & -\frac{10}{27} & \frac{55}{54} & 0 & 0 \quad \text{Order 3} \\
 & -\frac{3}{2} & \frac{5}{2} & 0 & 0 & 0 & 0 \quad \text{Order 2} \\
 & 1 & 0 & 0 & 0 & 0 & 0 \quad \text{Order 1}
\end{array}
\tag{5}
$$

When used as a fixed-order method, we refer to Eq. (5) as RKFNC; when used as a variable order method, as VRKF.

It is interesting to examine the first few terms in the local error expansion of this formula and to compare them with the errors in certain other widely used 5(4) formulas. A reasonable way of measuring the local error in a pth-order formula, which has been used by several other authors, is to compute

$$\| T_{p+1} \|_2 \equiv \sqrt{\sum_{j=1}^{\lambda_{p+1}} T_{p+1,j}^2} \quad \text{and} \quad \| T_{p+2} \|_2 \equiv \sqrt{\sum_{j=1}^{\lambda_{p+2}} T_{p+2,j}^2}.$$

(See Eq. (4).) These relations give the 2-norm of the error coefficients of order $p + 1$ and $p + 2$ which are the first two nonvanishing terms in Eq. (4). For the formula described in this paper and two reference formulas, the details are given in Table I.

We note that, using these measures to quantify the local error, RKFNC has considerably smaller terms in its local error than either RKF45 or Verner's fifth-order CSIRK method. The hope is that on problems with reasonably smooth solutions, using RKFNC will result in increased efficiency. As we will see in Section 4, this hope is realized for the DETEST test set. Before giving any

Table I. Error Terms

Formula		Order	$\|T_5\|_2$	$\|T_6\|_2$	$\|T_7\|_2$
RKF45	Main	5	0	0.0034	0.0068
	Embedded	4	0.0018	0.0058	—
Verner 5(4)	Main	5	0	0.0037	0.0070
	Embedded	4	0.0012	0.0054	—
RKFNC	Main	5	0	0.0009	0.0013
	Embedded	4	0.0005	0.0011	—

numerical results, however, we first explain in detail our computational procedure for dealing with any regions where the solution of our problem exhibits particularly nonsmooth behavior.

3. A STRATEGY FOR DEALING WITH NONSMOOTH BEHAVIOR

The Runge-Kutta formula derived in the previous section has the special property that it contains imbedded solutions of all orders less than five. In addition, the formula has been designed so that the first five c_i values span the range [0, 1] with reasonable uniformity, so that we have a very good chance of spotting bad behavior in f if it occurs. Our aim is to derive an automatic strategy that allows us to quit early, i.e., before all six function evaluations have been computed on the current step, if we suspect trouble, and to accept a lower order solution if appropriate.

We assume that we have computed a numerical solution y_{n-1} at the step point x_{n-1} and that for the current step, from x_{n-1} to $x_n = x_{n-1} + h$, all six function evaluations are computed so that solutions of all orders from 1 to 5 are available. (We guarantee this situation for the first step with $n = 1$). We denote the imbedded solution of order i at x_n by $y_n^{(i)}$, $1 \le i \le 5$, and define

$$\mathrm{ERR}(n, i) = \| y_n^{(i+1)} - y_n^{(i)} \|^{1/(i+1)}, \qquad \text{for} \quad i \in 1, 2, 4. \tag{6}$$

We exclude the case $i = 3$ for two reasons. First, following the approach of Shampine et al. [15], we allow only a few different orders to be used, and we have chosen to allow orders 2, 3, or 5. Second, ERR(n, 3) is of no use in predicting when to quit early since all six k_i's are required before $y_n^{(4)}$ can be computed.

Suppose now that we were to accept the solution of order 5 at x_n. We wish to compute a suitable step length, $\bar{h}_4$, to be used in integrating from x_n to x_{n+1} using a 5(4) formula. A typical step-choosing strategy would compute $\bar{h}_4$ as

$$\bar{h}_4 = \frac{\mathrm{SF} \times h}{\mathrm{E}(n, 4)}, \qquad \text{where} \quad \mathrm{E}(n, 4) = \frac{\mathrm{ERR}(n, 4)}{\epsilon^{1/5}}. \tag{7}$$

Here ϵ is the local accuracy required (as specified by the user) and SF is a safety factor often taken to be 0.9. Similarly, if we were to accept either the second- or third-order solution at x_n, the steplengths $\bar{h}_1$, $\bar{h}_2$, respectively, that would be selected at the next step by our step-control algorithm would be

$$\bar{h}_i = \frac{\mathrm{SF} \times h}{\mathrm{E}(n, i)}, \qquad \text{where} \quad \mathrm{E}(n, i) = \frac{\mathrm{ERR}(n, i)}{\epsilon^{1/(i+1)}}, \quad i \in 1, 2. \tag{8}$$

From relations (7) and (8), we can compute what we call our quitting factors,

$$\mathrm{QUIT}(n, i) = \frac{\bar{h}_4}{\bar{h}_i} \equiv \frac{\mathrm{E}(n, i)}{\mathrm{E}(n, 4)}, \quad i \in 1, 2. \tag{9}$$

To explain how these quitting factors are used, it is convenient to assume that the fifth-order solution $y_n^{(5)}$ is accepted at x_n. We now consider what happens in integrating from x_n to x_{n+1}. The first stage in integrating forward from x_n is to compute two function evaluations k_1 and k_2, and use these to compute first- and second-order solutions, $y_{n+1}^{(1)}$ and $y_{n+1}^{(2)}$, at x_{n+1}. We can now compute

$$\mathrm{ERR}(n + 1, 1) \equiv \| y_{n+1}^{(2)} - y_{n+1}^{(1)} \|^{1/2}. \tag{10}$$

Use of this error estimate allows us to compute the step $\bar{h}_1$, which would be used for the next step if we were to accept the first-order solution:

$$\bar{h}_1 = \frac{\mathrm{SF} \times h}{\mathrm{E}(n + 1, 1)}, \tag{11}$$

where $\mathrm{E}(n + 1, 1)$ is defined as in Eq. (8).

We now have sufficient information to allow us to estimate the step $\tilde{h}_4$ which would be selected for the next step if all six function evaluations defining the current step were computed and a fifth-order solution, $y_{n+1}^{(5)}$, accepted. By definition $\tilde{h}_4 = \mathrm{QUIT}(n + 1, 1) \times \bar{h}_1$. We now assume that $\mathrm{QUIT}(n + 1, 1) \simeq \mathrm{QUIT}(n, 1)$, where $\mathrm{QUIT}(n, 1)$ is available from the previous step. Now we can make the approximation

$$\tilde{h}_4 = \mathrm{QUIT}(n, 1) \times \bar{h}_1 = \frac{\mathrm{QUIT}(n, 1) \times \mathrm{SF} \times h}{\mathrm{E}(n + 1, 1)}. \tag{12}$$

If $\tilde{h}_4 < \mathrm{SF} \times h$, that is,

$$\mathrm{QUIT}(n, 1) < \mathrm{E}(n + 1, 1), \tag{13}$$

then we expect that the fifth-order solution will be rejected, and we should abandon the current step. It is important to realize that this test can be performed after only two function evaluations have been computed. Although this approach forms the basis of our strategy, it is not quite the strategy that is used in practice, as we now explain.

An important part of our strategy is the assumption that the quit factors are changing slowly from step to step (as they normally will with smooth solutions), so a big change in the quit factor for a low-order solution signals trouble and causes us to abandon the current step. However, we must allow the quit factors to increase slowly from step to step, which means that the satisfaction of Eq. (13) is too severe a requirement to impose. We overcome this problem by introducing a *twiddle factor*, so that our code will quit early when

$$\mathrm{E}(n + 1, 1) > \mathrm{QUIT}(n, 1) \times \mathrm{TWIDDLE}(n, 1). \tag{14}$$

We experimented with several twiddle factors using only the information that they should be greater than unity and not "too big," i.e., probably less than 1.5. Our experiments showed clearly that the optimal value of these twiddle factors was problem-dependent, and so in our implementation we allowed them to be chosen dynamically. At the end of this section we list our complete strategy in step-by-step form. Before doing so, we explain what happens when a lower order solution is accepted. We first explain under what circumstances we accept solutions of various orders.

We assume that the current step is successful, i.e., $E(n, i) < 1$ for at least one $i \in [1, 2, 4]$. Then if

(i) $E(n, 1) < 1$, $E(n, 2) > 1$, $E(n, 4) > 1$, accept $y_{n+1}^{(2)}$
(ii) $E(n, 1) < 1$, $E(n, 2) < 1$, $E(n, 4) > 1$, accept $y_{n+1}^{(3)}$
(iii) $E(n, 1) < 1$, $E(n, 2) < 1$, $E(n, 4) < 1$, accept $y_{n+1}^{(5)}$.

Thus, we see that at each step we only accept the second-, third-, or fifth-order solutions. This strategy of allowing only a few different orders in a variable order Runge-Kutta code (i.e., not allowing first- or fourth-order solutions in our case) is in line with the results of Shampine et al. [15], who found this approach to be the most efficient.

Let us now suppose that case (i) holds, and we wish to accept the order 2 solution. The error estimate in this solution is based on the following information.

$$\begin{array}{c|cc} 0 & 0 & \\ \frac{1}{5} & \frac{1}{5} & 0 \\ \hline & & \end{array}$$

Since we have only used values of c_i up to 1/5, and we suspect there may be trouble ahead due to the unacceptability of higher order solutions, it would be dangerous to integrate past $x_n + h/5$ on the current step. Thus, using the above information, the solution we compute at $x_{n+1} = x_n + h/5$ is

$$y_{n+1} = y_n + \frac{h}{10}(k_1 + k_2). \tag{15}$$

The corresponding order 1 solution is

$$\bar{y}_{n+1} = y_n + \frac{h}{5} k_1$$

and the error estimate for this solution is

$$E_n^1(1/5) \equiv y_{n+1} - \bar{y}_{n+1} = \frac{h}{10}(k_2 - k_1).$$

If $\| E_n^1(1/5) \| > \epsilon$, then the step is abandoned and the integration is continued from (x_n, y_n) with a step $h/5$. Otherwise, the solution y_{n+1} is accepted, and the value $h/5$ is used for the next integration step starting from $(x_n + h/5, y_{n+1})$.

A similar strategy is used if the third-order solution, $y_{n+1}^{(3)}$, is accepted. The information on which this solution and its error estimate are based is as follows.

0	0				
$\frac{1}{5}$	$\frac{1}{5}$	0			
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$	0		
$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$	0	
	$\frac{19}{54}$	0	$-\frac{10}{27}$	$\frac{55}{54}$	Order 3
	$-\frac{3}{2}$	$\frac{5}{2}$			Order 2

Similar arguments to those used for the first-order solution can now be applied. Since the order 3 solution is acceptable, while the order 4 solution is not, we anticipate possible trouble in the range $[x_n + 3h/5, x_n + h]$. As a result, it is only safe to integrate forward a distance $3h/5$ during the current step. We therefore seek a third-order formula of the form

$$y_{n+3/5} = y_n + \frac{3}{5} h(b_1 k_1 + b_2 k_2 + b_3 k_3 + b_4 k_4)$$

which uses information that has already been computed to integrate forward a distance $3h/5$ rather than h. It is straightforward to show that the choice $b_1 = b_4 = 1/6$, $b_3 = 2/3$, $b_2 = 0$ satisfies this requirement. Our strategy is to accept the third-order solution with

$$y_{n+3/5} = y_n + h\left(\frac{1}{10} k_1 + \frac{2}{5} k_3 + \frac{1}{10} k_4\right) \tag{16}$$

as the solution at $x_n + 3h/5$, providing that it satisfies an appropriate error criterion. A second-order formula at $x_{n+3/5}$ is

$$\bar{y}_{n+3/5} = y_n + \frac{3}{5} h k_3.$$

An estimate of the error for this solution is

$$\mathrm{E}_n^2(3/5) \equiv y_{n+3/5} - \bar{y}_{n+3/5} = \frac{h}{10}(k_1 - 2k_3 + k_4).$$

If $\| \mathrm{E}_n^2(3/5) \| > \epsilon$, then we abandon the third-order solution and test the acceptability of a second-order solution at $x_n + h/5$, as previously described. Otherwise, the solution $y_{n+3/5}$ is accepted, and we continue the integration from $(x_n + 3h/5, y_{n+3/5})$ using a step $3h/5$.

We see from this approach that important use is made of what Shampine et al. refer to in [15] as "fall back" formulas. The idea is that, if an integration

over the whole step $[x_n, x_n + h]$ using the 5(4) pair fails because of something bad in the region $(x_n + h/5, x_n + h]$, then it is possible to use an independent, lower order solution that uses information taken only from the first part of the step. In our algorithm we have a "fall back" formula that uses information only in the region $[x_n, x_n + h/5]$ if the third-order solution appears bad, and a second "fall back" formula that uses information in the region $[x_n, x_n + 3h/5]$ if the fourth-order solution appears bad.

There are two different interpretations of what might go wrong in a step to cause a higher order solution to be rejected. It is convenient to discuss this point in terms of the lower order formulas. If the (1, 2) pair succeeds, but the (2, 3) pair does not, it may be that the (1, 2) pair has not sampled the function in the range $(x_n + h/5, x_n + h]$ where something bad happens that the third-order formula detects. Alternatively, it may be that the problem "looks" smooth to the second-order formula, but does *not* look smooth to the third-order formula. In the latter case, the third-order formula may be accurately reflecting the true behavior of the solution, and there may be nothing wrong with the last part of the step. In constructing our algorithm we have adopted the first point of view, i.e., that failure of a high-order solution, but success in a low-order solution, indicates bad behavior in the latter part of the step.

We now explain our complete algorithm in more detail. First, we initialize the constants.

(1) Set the twiddle factors.

These factors can be arbitrary, but they should be set just larger than unity. In our code we take TWIDDLE(0, 1) = 1.5 and TWIDDLE(0, 2) = 1.1. As explained earlier, these values will be changed automatically by the code.

(2) Set the quit factors.

These values should be set to be quite large initially to make the code take a full step at the start. In our code we use QUIT(0, 1) = QUIT(0, 2) = 100.

Next, we integrate one step at a time. Assume that an approximate solution y_{n-1} has been computed at the step point x_{n-1}.

(1) Compute the first two function evaluations in VRKF.
(2) From these evaluations obtain an error estimate ERR(n, 1) in the order 1 solution using Fehlberg imbedding with the order 1 and 2 solutions.
(3) Compute E(n, 1) as in (8).
(4) Check the error estimate.

IF E(n, 1) > TWIDDLE($n - 1$, 1) × QUIT($n - 1$, 1),
THEN abandon the step.
a. Put ESTTOL = E(n, 1)/QUIT($n - 1$, 1).
b. Choose the next step $\bar{h}$ as

$$\bar{h} = \max\left(\frac{1}{5}, \frac{SF}{\text{ESTTOL}}\right) \times h. \tag{17}$$

c. Go to step 1.
ENDIF

(5) Compute the third and fourth function evaluations in VRKF.
(6) Compute the order 3 solution.

(7) Compute $\mathrm{ERR}(n, 2) = \| y_n^{(3)} - y_n^{(2)} \|^{1/3}$ and hence $\mathrm{E}(n, 2)$ from (8).
(8) Check the error estimate.

IF $\mathrm{E}(n, 2) > \mathrm{TWIDDLE}(n-1, 2) \times \mathrm{QUIT}(n-1, 2)$,
THEN try a lower order solution.
IF $\mathrm{E}(n, 1) < 1$,
THEN check the error of the second order solution.
IF $\| \mathrm{E}_n^1(1/5) \| < \epsilon$,
THEN accept the second order solution computed from (15).
a. Cut the step from h to $h/5$.
b. Replace n by $n + 1$.
c. Go to step 1.
ELSE abandon the step.
a. Cut the step from h to $h/5$.
b. Go to step 1.
ENDIF
ELSE abandon the step.
a. Set $\mathrm{ESTTOL} = \mathrm{E}(n, 2)/\mathrm{QUIT}(n-1, 2)$.
b. Choose the step $\bar{h}$ as in (17).
c. Go to step 1.
ENDIF
ENDIF

(9) Compute the final two function evaluations in VRKF.
(10) Use all six function evaluations to compute the order 4 and 5 solutions.
(11) Compute $\mathrm{ERR}(n, 4) = \| y_n^{(5)} - y_n^{(4)} \|^{1/5}$ and hence $\mathrm{E}(n, 4)$ from (8).
(12) Check the error estimate.

IF $\mathrm{E}(n, 4) > 1$,
THEN readjust the twiddle factors.
IF $\mathrm{E}(n, i)/\mathrm{QUIT}(n-1, i) < \mathrm{TWIDDLE}(n-1, i), \quad i \in 1, 2$
THEN $\mathrm{TWIDDLE}(n, i) = \mathrm{MAX}(1.1, \mathrm{E}(n, i)/\mathrm{QUIT}(n-1, i))$.
ELSE $\mathrm{TWIDDLE}(n, i) = \mathrm{TWIDDLE}(n-1, i)$.
ENDIF
IF $\mathrm{E}(n, 2) < 1$,
THEN check the accuracy of the third order solution.
IF $\| \mathrm{E}_n^2(3/5) \| < \epsilon$,
THEN accept the order 3 solution computed from (16).
a. Cut the step from h to $3h/5$.
b. Replace n by $n + 1$.
c. Go to step 1.
ENDIF
ELSE try a lower order solution.
IF $\mathrm{E}(n, 1) < 1$,
THEN check the accuracy of the second order solution.
IF $\| \mathrm{E}_n^1(1/5) \| < \epsilon$,
THEN accept the order 2 solution computed from (15).
a. Cut the step from h to $h/5$.
b. Replace n by $n + 1$.
c. Go to step 1.

ELSE abandon the step.
a. Cut the step from h to $h/5$.
b. Go to step 1.
ENDIF
ELSE abandon the current step.
a. Set the step $\bar{h}$ as in (17) where ESTTOL $\equiv$ E$(n, 4)$.
b. Go to step 1.
ENDIF
ENDIF
ELSE accept the order 5 solution for $x_n = x_{n-1} + h$.
a. Choose the new step $\bar{h}$ as

$$\bar{h} = \min\left(5.0, \frac{SF}{\mathrm{E}(n, 4)}\right) \times h.$$

b. Update the quit factors. (We will give our strategy here and explain it later.) Define $Q_1 = \mathrm{E}(n, 1)/\mathrm{E}(n, 4)$ and $Q_2 = \mathrm{E}(n, 2)/\mathrm{E}(n, 4)$. Then for $j \in (1, 2)$,
IF $Q_j >$ QUIT$(n-1, j)$,
THEN $Q_j = \min(Q_j, 10 \times \mathrm{QUIT}(n-1, j))$.
ELSE $Q_j = \max(Q_j, \frac{2}{3} \times \mathrm{QUIT}(n-1, j))$.
ENDIF
QUIT$(n, j) = \max(1.0, \min(10000, Q_j))$.
c. Set TWIDDLE$(n, j) =$ TWIDDLE$(n-1, j)$.
ENDIF

The quit factors are not allowed to vary by arbitrary amounts at each step, since we do not want an isolated, very good solution or an isolated, very poor solution to have a big effect on the quit factors. We prefer to have the quit factors change slowly from step to step. A second reason for limiting the change in the quit factors comes from examining the error test

IF E$(n, j) >$ TWIDDLE$(n-1, j) \times$ QUIT$(n-1, j)$, **THEN** quit.

We see from this test that, if our quit factors are too small, then we are forced to quit early unnecessarily. Such a situation may occur when we move from a smooth region, where the quit factors will normally be large, into a rough region, where the quit factors would become very small if allowed to become so. If we then move back into a smooth region (which would typically happen after passing through a discontinuity), we may not be able to increase the step very quickly, due to the extremely small quit factors.

A final factor is that it is generally better to complete the step by computing all six function evaluations, and then to reject the solution, than to quit early when a full step would have been accepted. When a full step has been completed, we have a complete set of information on which to base our step-choosing strategy; when we quit early, we have much less information available for choosing h.

For all these reasons, it is much more desirable for the quit factors to be too large than too small. (However, for smooth problems the quit factors normally do an extremely good job of judging when to quit early.) In view of these observations, we limit the change in quit factors from step to step, but allow

Table II. Distance Forward per Function Evaluation

Order	Function evaluations (F)	Distance forward (D)	D/F
2	2	$h/5$	$h/10$
3	4	$3h/5$	$3h/20$
5	6	h	$h/6$

them to increase much more rapidly than they are allowed to decrease. In our program we limit the decrease to a factor of $\frac{2}{3}$ while allowing an increase of up to a factor of 10 per step.

Finally, in this section we wish to explain why we do not necessarily accept a lower order solution early and abandon the rest of the step if it passes the error test. A strategy of accepting early could well be used in a variable order code designed for smooth solutions, but we are particularly interested in problems where there is a distinct lack of smoothness. If we were to accept early and adopt the step-choosing strategy described earlier in this section, the distance travelled forward per function evaluation is as shown in Table II. We see that, on a step where solutions of all orders pass the error test, it is the order 5 solution that goes the farthest forward per function evaluation.

There is an obvious way to increase the distance travelled for the low-order solutions if we believe the solution to be smooth. Suppose the order 1 solution passes the error test. We accept this solution and compute

$$y_{n+1} = y_n + hf(x_n, y_n). \tag{18}$$

Accepting the solution y_{n+1} can be regarded as a bit suspect because we have only used information in the range $[x_n, x_n + h/5]$, and we have not sampled f at all in the range $(x_n + h/5, x_{n+1}]$. However, we go on to the next step and compute $f(x_{n+1}, y_{n+1})$. Using this extra function evaluation allows us to compute a second estimate of the error in y_{n+1} as

$$\begin{aligned} \mathrm{E} &\equiv y(x_{n+1}) - y_{n+1} \\ &\simeq \underbrace{\left\{ y_{n+1} - y_n - \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, y_{n+1})\right] \right\}}_{\text{Order 2 solution}} - \underbrace{\left\{ y_{n+1} - y_n - hf(x_n, y_n) \right\}}_{\text{Order 1 solution}} \\ &= \frac{h}{2}\left[f(x_n, y_n) - f(x_{n+1}, y_{n+1})\right]. \end{aligned}$$

This new error estimate uses information at x_n and x_{n+1}. If this estimate is less than the prescribed tolerance, then we continue with the next step (the cost of computing the additional error estimate will be negligible). Otherwise, we accept

$$y_{n+1} = y_n + \frac{h}{5} f(x_n, y_n).$$

The cost of having to reject the solution defined by (18) will be one function evaluation. We have not examined this approach because we are interested mainly in nonsmooth solutions.

4. NUMERICAL RESULTS

In this section we give some numerical results that were obtained by applying a code based on Eq. (5) and the ideas explained in Section 3 to some test problems. We consider first the performance of our code on some smooth problems. Then we look at how our code does with some rough problems.

The smooth problems chosen are the well-known DETEST test set which contains five classes of smooth problems. Table III compares the results of three codes, RKF45, RKFNC, and VRKF. RKFNC is identical to RKF45, except that it uses the coefficients from Eq. (5); VRKF uses the coefficients from Eq. (5) and the variable order scheme described in Section 3.

While RKF45 is no longer the "state of the art" 5(4) Runge-Kutta formula, it is very widely used (it appears in several subroutine libraries), and we feel it is a suitable code to compare to ours.

Although we have been very careful to derive our code with quality in mind, as explained in Section 2, we were very surprised at how well RKFNC performed on the DETEST test set compared to RKF45. As can be seen from Table III, there is a gain in efficiency of about 25% in terms of time and function evaluations. This leads us to believe that, for the DETEST test set at least, the code based on RKFNC is a considerable improvement over RKF45. We see no reason why this conclusion should not extend in general to problems with smooth solutions.

We also see that VRKF is somewhat better than RKF45 in terms of both function evaluations and time. However, VRKF is only superior to RKFNC in terms of function evaluations at low accuracy. The time required to check the errors at the intermediate steps makes VRKF less efficient than RKFNC, even at low orders, if function evaluations are fast. We believe this conclusion will hold for other problems, making RKFNC the method of choice for smooth problems.

We now consider the performance of our code on some problems with non-smooth solutions. Our primary interest is in problems having solutions with sharp fronts but which are not discontinuous. However, in our numerical experiments, we consider problems with discontinuities that are triggered by a condition on x. Problems with discontinuities triggered by y, or one of its derivatives, are rather more specialized. We consider them to be beyond the scope of this paper. Such problems are of considerable interest, and we hope to extend our algorithm to deal with these problems at a future time.

The class of discontinuous problems we are interested in are those where the discontinuities appear without warning. An example is where the right-hand side is supplied by some black box code that hides the switching from the user. Such problems are much harder than those for which we know the x-value where f changes, or we are given a switching function which, on reaching a certain known value, triggers the discontinuity. It is important to use information regarding the switching function, if it is known, and several authors have proposed algorithms for dealing with this problem [2, 6, 9, 11, 12].

The case of interest to us, where the switching function is not known, has been considered by Gear and Osterby [8] and by Enright et al. [7]. The approach adopted by Gear and Osterby attempts to identify the nature of the singularity.

Table III. Summary of Results on the 25 (unscaled) DETEST Test Set

$\log_{10}$TOL	Time	Function calls	Number of steps	Maximum local error	Fraction deceived	Fraction bad deceived
Results for RKF45						
−2.00	0.498	4123	548	50.869	0.224	0.038
−3.00	0.565	4587	637	9.300	0.148	0.006
−4.00	0.745	6344	884	3.442	0.070	0.000
−5.00	1.014	8935	1270	1.439	0.012	0.000
−6.00	1.453	12948	1893	6.938	0.005	0.001
−7.00	2.117	18741	2886	2.144	0.002	0.000
−8.00	3.136	27552	4477	1.491	0.001	0.000
−9.00	4.849	42648	7023	1.270	0.000	0.000
Overall	14.377	125878	19618	50.869	0.017	0.001
Results for RKFNC						
−2.00	0.408	3272	437	7.762	0.156	0.005
−3.00	0.478	3965	555	10.875	0.138	0.004
−4.00	0.626	5432	752	6.355	0.092	0.001
−5.00	0.828	7508	1068	5.285	0.050	0.001
−6.00	1.145	10710	1555	3.471	0.027	0.000
−7.00	1.628	15349	2349	1.485	0.013	0.000
−8.00	2.370	22419	3609	1.480	0.004	0.000
−9.00	3.601	34086	5636	1.560	0.001	0.000
Overall	11.083	102741	15961	10.875	0.024	0.000
Results for VRKF						
−2.00	0.495	3171	465	6.495	0.099	0.009
−3.00	0.591	3919	580	10.875	0.064	0.003
−4.00	0.760	5327	782	3.283	0.051	0.000
−5.00	1.029	7488	1115	2.196	0.025	0.000
−6.00	1.407	10688	1604	2.760	0.017	0.000
−7.00	2.033	15645	2392	1.783	0.008	0.000
−8.00	2.953	22940	3674	1.628	0.004	0.000
−9.00	4.457	34524	5697	1.209	0.000	0.000
Overall	13.726	103702	16309	10.875	0.014	0.000

It is not clear how their approach will perform when the function is rapidly varying but does not actually contain a discontinuity (see [8, p. 41]).

All of the algorithms that we have mentioned for discontinuities and rapidly varying functions are at a rather preliminary stage of development. Few numerical results have appeared in the literature. For this reason we will not attempt to compare different methods for dealing with discontinuities. Instead we compare the performance of our code with that of a standard 5(4) Runge-Kutta code used with Fehlberg imbedding and local extrapolation. Our aim is to demonstrate the superior performance of a modified 5(4) code compared to that of a conventional code of the same type.

In what follows we consider four test problems. The first two have sharp fronts but no discontinuities, while the second two have discontinuities triggered on x. As explained earlier, we assume that the analytic form of f is unknown to us, so that we are not able to use information concerning the exact location of any discontinuous points of f or its derivatives in deriving our algorithm.

Problem 1.

$$\begin{aligned} y' &= z, & y(0) &= 10 \\ z' &= z^2 - \frac{3}{(A + y^2)}, & z(0) &= 0 \end{aligned} \qquad 0 \le x \le 50.$$

This problem[1] does not have a known, analytic solution. However, it was found to be a very good model for testing ODE solvers for the oil-reservoir problems one of us was studying. The solution is smooth almost everywhere except near $x = 35$, where it develops a sharp front. We believe it to be a very interesting and illuminating test problem for codes designed to deal with nonsmooth solutions. The values of the parameter A, which we considered for this problem, are 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}. Errors are computed at the end of the range only ($x = 50$), where the solution was found using the code with a very small ε. Thus, the "exact" solution, used to compute the errors, itself has an error of about 10^{-9}.

Problem 2.

$$\begin{aligned} y' &= z, & y(-1) &= -1, \\ z' &= \frac{-(1 + \pi^2 A)\cos \pi x - \pi x \sin \pi x - xz + y}{A}, & z(-1) &= 0.0017, \end{aligned} \qquad -1 \le x \le 1.$$

The true solution to this problem from Hemker [10, p. 138] is

$$y(x) = \cos \pi x + x + \frac{\sqrt{2A}\, \exp(-x^2/(2A))\exp(1/(2A)) + \exp(1/(2A))\sqrt{\pi}\, erf\,(\pi/\sqrt{2A})}{\sqrt{2A} + \exp(1/(2A))\sqrt{\pi}\, erf\,(1/\sqrt{2A})}.$$

This problem has a smooth solution $x + \cos \pi x$ coupled to rapidly varying solutions at $x = \pm 1$. For a description of the behavior of the complementary functions, the reader is referred to [10, p. 16]. In Table V we give results for this problem with $A = 0.1$.

Problem 3.

$$y' = \begin{cases} 0 & x < 0 \\ x^A & x \ge 0 \end{cases} \qquad -1 \le x \le 1, \quad y(-1) = 0.$$

This linear problem from [15, p. 15] has a discontinuity in the $(A + 1)$th derivative at the point $x = 0$. In Table VI we give the results obtained for this problem in the cases $A = 0, 1, 2, 3$.

Problem 4.

$$y' = \begin{cases} 55 - 1.5y & \text{if } [x] \text{ even} \\ 55 - 0.5y & \text{if } [x] \text{ odd} \end{cases} \qquad 0 \le x \le 20, \quad y(0) = 110.$$

[1] This problem was taken from the literature, but the reference can no longer be found. We apologize to the author.

Table IV. Results for Problem 1

A	10^{-1}		10^{-2}		10^{-3}		10^{-4}		10^{-5}	
TOL	Evals	Error	Evals	Error	Evals	Error	Evals	Error	Evals	Error
Results for RKF45										
10^{-3}	276	0.24E-2	344	0.17E-2	417	0.18E-2	486	0.22E-2	566	0.23E-2
10^{-4}	390	0.12E-3	481	0.13E-3	567	0.15E-3	686	0.15E-3	795	0.18E-3
10^{-5}	578	0.71E-5	698	0.14E-4	846	0.10E-4	1016	0.11E-4	1177	0.12E-4
10^{-6}	783	0.48E-6	981	0.73E-7	1198	0.12E-6	1442	0.23E-7	1704	0.17E-6
10^{-7}	991	0.20E-8	1257	0.34E-7	1547	0.33E-7	1841	0.27E-7	2144	0.35E-7
10^{-8}	1546	0.85E-8	1887	0.10E-7	2259	0.91E-8	2661	0.10E-7	3111	0.10E-7
10^{-9}	2398	0.76E-9	2910	0.10E-8	3508	0.82E-9	4145	0.82E-9	4887	0.89E-9
Results for RKFNC										
10^{-3}	213	0.21E-3	283	0.92E-4	346	0.15E-3	420	0.19E-3	480	0.17E-3
10^{-4}	293	0.30E-4	358	0.21E-4	454	0.19E-4	519	0.39E-5	632	0.14E-4
10^{-5}	411	0.11E-5	538	0.29E-6	663	0.59E-7	799	0.22E-6	920	0.25E-6
10^{-6}	586	0.28E-7	734	0.13E-6	920	0.15E-6	1118	0.16E-6	1317	0.23E-6
10^{-7}	801	0.24E-7	1027	0.44E-7	1272	0.56E-7	1576	0.75E-7	1867	0.70E-7
10^{-8}	1185	0.56E-8	1446	0.90E-8	1715	0.11E-7	2011	0.12E-7	2364	0.12E-7
10^{-9}	1771	0.11E-8	2110	0.16E-8	2549	0.97E-9	3053	0.17E-8	3598	0.12E-8
Results for VRKF										
10^{-3}	211	0.26E-3	243	0.29E-3	297	0.27E-3	346	0.26E-3	396	0.25E-3
10^{-4}	281	0.21E-4	342	0.15E-4	405	0.16E-4	474	0.24E-4	549	0.18E-4
10^{-5}	430	0.20E-6	504	0.73E-6	596	0.54E-6	707	0.54E-6	823	0.10E-5
10^{-6}	599	0.50E-7	724	0.16E-6	857	0.15E-6	1049	0.21E-6	1255	0.31E-6
10^{-7}	800	0.34E-7	1042	0.46E-7	1303	0.50E-7	1605	0.54E-7	1902	0.60E-7
10^{-8}	1196	0.55E-8	1478	0.88E-8	1748	0.11E-7	2043	0.12E-7	2391	0.12E-7
10^{-9}	1788	0.10E-8	2124	0.15E-8	2573	0.16E-8	3087	0.15E-8	3623	0.16E-8

Evals: The number of function evaluations.
Error: The computed error at the endpoint.

Table V. Results for Problem 2

	Number of function evaluations			Modulus of error at endpoint		
TOL	RKF45	RKFNC	VRKF	RKF45	RKFNC	VRKF
10^{-3}	94	87	89	0.26E-0	0.48E-1	0.48E-1
10^{-4}	131	107	109	0.31E-1	0.28E-2	0.28E-2
10^{-5}	203	156	163	0.37E-2	0.20E-3	0.20E-3
10^{-6}	316	251	252	0.12E-3	0.14E-4	0.14E-4
10^{-7}	484	384	391	0.84E-5	0.98E-6	0.25E-6
10^{-8}	770	610	617	0.11E-5	0.50E-7	0.63E-7
10^{-9}	1191	952	964	0.10E-6	0.16E-8	0.15E-8

This problem is F2 of the DETEST test set. For this problem, the function f has a discontinuity wherever x is an integer. The results obtained are given in Table VII.

Before discussing our numerical results in detail, we briefly discuss the sort of numerical results we might expect to obtain. We first note that Runge-Kutta methods are one-step in nature. We therefore expect them to have less trouble with singularities than would multistep methods such as Adams

Table VI. Results for Problem 3

A	0		1		2		3	
TOL	Evals	Error	Evals	Error	Evals	Error	Evals	Error
Results for RKF45								
10^{-3}	67	0.72E-2	23	0.33E-2	18	0.84E-2	18	0.35E-2
10^{-4}	121	0.11E-3	40	0.16E-3	18	0.12E-4	24	0.12E-3
10^{-5}	155	0.24E-3	62	0.88E-4	46	0.18E-4	30	0.17E-4
10^{-6}	190	0.33E-4	79	0.34E-5	46	0.69E-6	52	0.36E-5
10^{-7}	246	0.20E-5	128	0.79E-6	64	0.29E-5	58	0.28E-7
10^{-8}	278	0.69E-7	155	0.18E-6	107	0.82E-7	81	0.34E-8
10^{-9}	327	0.45E-8	225	0.19E-7	112	0.94E-8	85	0.10E-9
Results for RKFNC								
10^{-3}	67	0.22E-2	29	0.11E-2	23	0.36E-3	18	0.68E-3
10^{-4}	90	0.92E-3	51	0.17E-3	24	0.12E-3	24	0.67E-4
10^{-5}	161	0.20E-3	56	0.12E-3	41	0.38E-4	24	0.11E-7
10^{-6}	174	0.11E-4	96	0.28E-6	52	0.11E-5	46	0.43E-6
10^{-7}	240	0.41E-7	107	0.20E-6	79	0.19E-6	58	0.99E-7
10^{-8}	254	0.19E-6	130	0.13E-7	97	0.12E-7	52	0.13E-9
10^{-9}	330	0.49E-8	153	0.37E-8	107	0.97E-7	69	0.14E-9
Results for VRKF								
10^{-3}	47	0.77E-2	19	0.31E-2	19	0.35E-3	19	0.17E-3
10^{-4}	67	0.15E-2	37	0.51E-3	25	0.21E-3	25	0.15E-3
10^{-5}	98	0.91E-4	54	0.16E-3	1ᵹ	0.72E-5	19	0.88E-4
10^{-6}	116	0.24E-4	67	0.85E-7	49	0.19E-5	55	0.42E-5
10^{-7}	126	0.55E-6	66	0.69E-7	64	0.76E-7	40	0.75E-6
10^{-8}	148	0.40E-6	68	0.12E-6	63	0.11E-8	61	0.89E-9
10^{-9}	159	0.13E-7	98	0.61E-9	73	0.22E-9	64	0.19E-10

Evals: The number of function evaluations.
Error: The computed error at the endpoint.

Table VII. Results for Problem 4

	Number of function evaluations			Modulus of the error at the endpoint		
TOL	RKF45	RKFNC	VRKF	RKF45	RKFNC	VRKF
10^{-3}	1013	1046	936	0.14E-1	0.50E-2	0.13E-1
10^{-4}	1704	1606	1213	0.16E-2	0.33E-3	0.60E-3
10^{-5}	2087	1983	1530	0.11E-3	0.10E-3	0.15E-3
10^{-6}	2601	2443	1918	0.58E-4	0.70E-5	0.87E-5
10^{-7}	3027	3011	2198	0.21E-5	0.61E-6	0.15E-5
10^{-8}	4176	3822	2853	0.12E-6	0.65E-8	0.86E-8
10^{-9}	5125	4640	3425	0.51E-7	0.16E-8	0.17E-7

predictor-corrector formulas, for example. Indeed, if the "natural" step sequence chosen by the Runge-Kutta code nearly hits the singularity, then the unmodified code may experience very little difficulty in passing through the singular point. In view of this fact, we expect the variable order code to generally perform better than RKF45. Of course, there may be cases when solving discontinuous problems where RKF45 performs particularly well, due to a fortunate choice of step-size sequence.

The situation is quite different for problems where f is rapidly varying but not discontinuous. In such cases the rough behavior in the solution will occur over a range of x rather than at a particular point. The unmodified code will not be able to step through the rough part of the solution trajectory without noticing it. For such problems, we would expect the modified code based on VRKF to be superior to the unmodified RKF45, especially for relaxed tolerances when the codes normally try to use a large step-length.

We first consider the results obtained for Problem 1. This problem has a sharp front near $x = 35$, which steepens as A decreases. It is very noticeable from the results that, as the front gets steeper, the steplength of integration has to be cut back more and more, resulting in many rejected steps. This is an example of a problem where there is not just one point where bad (singular) behavior occurs. Instead, the difficult region extends over a range of x.

As mentioned earlier, for such problems we expect the variable order code VRKF to perform better than either RKF45 or RKFNC. This is borne out by the results given in Table IV, which show that VRKF is generally 25%–35% more efficient than RKF45. A large part of this gain is due to the use of the new coefficients, as can be seen from the results of RKFNC. At a modest tolerance, VRKF needs about 10% fewer function evaluations than RKFNC. When TOL is less than 10^{-6}, RKFNC is more efficient than VRKF. This effect occurs because, at such small tolerances, the step-size is small compared to the width of the front. In other words, the function appears smooth at the resolution of the step-size. However, the difference is small, and VRKF can be used without great loss of efficiency, especially if the function evaluations are expensive.

Results for Problem 2 are rather harder to predict, since the problem has boundary layers at both ends of the range of integration. In fact, this equation was used as an experiment to see how the codes would perform on such a problem. As can be seen from Table V, RKFNC is about 20% more efficient in terms of function evaluations than RKF45 and achieves much better accuracy. Indeed, if we compare function evaluations against accuracy, we see that RKFNC needs about half the function evaluations of RKF45.

For this problem, VRKF and RKFNC need almost the same number of function evaluations. As before, this phenomenon is due to the rough part being confined to a relatively small section of the domain. Clearly, RKF45 and RKFNC do not have any trouble handling the roughness near the endpoints. This problem shows that RKFNC should be used for such problems, but that not much is lost by using VRKF.

Problem 3 is an example of a discontinuous initial value problem where we have just a single trouble spot, i.e., at $x = 0$. We expect that the fixed order methods would experience the greatest trouble in the case $A = 0$ when f is discontinuous. As A increases, the singularity becomes less severe, and we expect RKF45 and RKFNC to perform more efficiently. This expectation is borne out by the results of Table VI. We see that in the case $A = 0$, VRKF is considerably superior to RKFNC; in some cases, needing half as many function evaluations. As A increases the difference becomes less apparent.

Differences in implementation can have a major effect on the performance of a code on this problem, as discussed in some detail in [15]. An important point brought out by this problem is that a code must have some restriction on its

allowable step-size change if it is to perform efficiently. Different step-size restrictions can lead to dramatically different performance. However, since the three codes we are comparing have exactly the same limits imposed on their step-size change, we feel that our comparison is a fair one.

Finally, we consider Problem 4. Here we expect a large difference in the performance of the fixed-order and variable order codes, since there are twenty singularities in the range of integration instead of just one. The chances of a fixed-order code picking a natural step sequence, suitable for passing through all these singular points, is very small, so we do not expect RKF45 or RKFNC to have abnormally good behavior. We see from Table VII that these expectations are borne out in practice. VRKF is considerably more efficient than either RKFNC or RKF45, with the gain in efficiency generally being about 20% over RKFNC and 30% over RKF45.

5. CONCLUSIONS

In this paper we have developed a modified Runge-Kutta code that contains imbedded formulas of all orders. This code is suitable for dealing with initial value problems where the function f is changing rapidly in some part of the region of integration. We were careful to derive a "high quality" formula that would perform well on problems with smooth solutions. The results given in Table III for the DETEST test set indicate that this goal has been achieved. At least for these smooth problems, our code is superior to RKF45.

Problems with rough solutions are handled by noting that our Runge-Kutta code computes f at several, reasonably uniformly spaced points in $[x_n, x_{n+1}]$, and trouble spots can be recognized early by looking for nonsmooth behavior in f. This strategy allows us to quit early when a high-order solution may not be acceptable or to accept a low-order solution when it is appropriate to do so.

Our belief is that this approach for dealing with nonsmooth solutions is a very powerful and general one. We are at present seeking to extend this idea to the case where the switching function is driven by a condition on y rather than on x. However, we feel that, in common with other approaches for dealing with discontinuous problems, our approach is in an embryonic stage. More understanding, both computational and theoretical, of this problem class is needed. Despite this, we believe that the results we have presented are sufficiently good to show that this approach shows great promise and deserves further attention.

ACKNOWLEDGMENT

The authors are grateful to L. F. Shampine for many useful suggestions concerning this paper.

REFERENCES

1. BETTIS, D. G. Efficient embedded Runge-Kutta methods. In *Numerical Treatment of Differential Equations: Proceedings Oberwolfach, 1976, Lecture Notes in Mathematics, 631*. R. Bulirsch, R. D. Grigorieff, and J. Shröder, Eds., Springer, Berlin, 1978, 9–18.
2. CARVER, M. B. Efficient integration over discontinuities in ordinary differential equation simulation. *Math. Comput. Simul. 20,* 3 (1978), 190–196.
3. CASH, J. R. A block 6(4) Runge-Kutta formula for nonstiff initial value problems. *ACM Trans. Math. Softw. 15*, 1 (1989), 15–28.

4. CELLIER, F. E. Stiff computation: where to go? In *Stiff Computation*, R. C. Aiken, Ed., OUP, 1985, 386–392.
5. DORMAND, J. R., AND PRINCE, P. J. A family of imbedded Runge-Kutta formulae. *J. Comput. Appl. Math. 6* (1980), 19–26.
6. ELLISON, D. Efficient automatic integration of ordinary differential equations with discontinuities. *Math. Comput. Simul. 23*, 1 (1981), 12–20.
7. ENRIGHT, W. H., JACKSON, K. R., NORSETT, S. P., AND THOMSEN, P. G. Effective solution of discontinuous IVPs using a Runge-Kutta formula pair with interpolants. Numerical Analysis Rep. 113, Univ. of Manchester, Jan. 1986.
8. GEAR, C. W., AND ØSTERBY, O. Solving ordinary differential equations with discontinuities. *ACM Trans. Math. Softw. 10* (1984), 23–44.
9. HAY, J. L., CROSBIE, R. E., AND CHAPLIN, R. I. Integration routines for systems with discontinuities. *Comput. J. 17*, 3 (1974), 275–278.
10. HEMKER, P. W. A numerical study of stiff two-point boundary value problems. Mathematical Centre Tracts 80, Mathematisch Centrum, Amsterdam, 1977.
11. MANNSHARDT, R. One step methods of any order for ordinary differential equations with discontinuous right hand sides. *Numer. Math. 31*, 2 (1978), 131–152.
12. O'REGAN, P. G. Step size adjustment at discontinuities for fourth order Runge-Kutta methods. *Comput. J. 13*, 4 (1970), 401–404.
13. SHAMPINE, L. F. Some practical Runge-Kutta formulas. *Math. Comput. 46* (1986), 135.
14. SHAMPINE, L. F., AND GORDON, M. K. *Solution of Ordinary Differential Equations—The Initial Value Problem*. W. H. Freeman, San Francisco, Calif., 1975.
15. SHAMPINE, L. F., GORDON, M. K., AND WISNIEWSKI, J. A. Variable order Runge-Kutta codes. In *Computational Techniques for Ordinary Differential Equations Conference* (Univ. of Manchester, 1978), I Gladwell and D. K. Sayers, Eds. Academic Press, London, 1980, 83–101.
16. SHAMPINE, L. F., AND WATTS, H. A. Subroutine RKF45. In *Computer Methods for Mathematical Computations*. G. E. Forsythe, M. A. Malcolm, and C. B. Moler, Eds. Prentice-Hall, Englewood Cliffs, N.J., 1977, 135–147.
17. VERNER, J. H. Families of imbedded Runge-Kutta methods. *SIAM J. Numer. Anal. 16*, 5 (1979), 857–875.

Received November 1988; revised June 1989; accepted August 1989