Implementation of Rosenbrock Methods

L. F. SHAMPINE
Sandia National Laboratories

Rosenbrock formulas have shown promise in research codes for the solution of initial value problems for stiff systems of ordinary differential equations (ODEs). To help assess their practical value, the author wrote an item of mathematical software based on one such formula. This required a variety of algorithmic and software developments Those of general interest are reported in this paper Among them is a way to select automatically, at every step, an explicit Runge-Kutta formula or a Rosenbrock formula according to the stiffness of the problem. Solving linear systems is important to methods for stiff ODEs and is rather special for Rosenbrock methods. A cheap, effective estimate of the condition of the linear systems is derived. Some numerical results are presented to illustrate the developments.

Categories and Subject Descriptors: G.1 7 [**Numerical Analysis**]: Ordinary Differential Equations—*initial value problems; single step methods; stiff equations*

General Terms Algorithms, Theory

Additional Key Words and Phrases: Rosenbrock Methods, FORTRAN Codes

1. INTRODUCTION

The most popular codes for the numerical solution of a stiff initial value problem for a system of ordinary differential equations (ODEs) are based on the backward differentiation formulas (BDF). There is a great need for a better understanding of many fundamental issues in both theoretical and practical terms. In addition the popular codes have certain weaknesses arising from both the formulas and their implementations. The situation has stimulated the investigation of many alternatives to the BDF. Because rather few have been developed so far as to result in items of mathematical software, it is difficult to evaluate the theoretical advances in the field.

In solving the system

$$y' = f(x, y), \qquad y(a) \text{ given}, \tag{1}$$

the implementations of the BDF employ the Jacobian matrix f_y in a simplified Newton iteration for the evaluation of the implicit formulas. This has suggested to many researchers the possibility of incorporating the Jacobian matrix directly into the formula. One line of development has been that of Rosenbrock formulas.

This work was performed at Sandia National Laboratories and was supported by the U.S. Department of Energy under Contract DE-AC04-76DP00789.

Author's address. Applied Mathematics Research Department, Sandia National Laboratories, Albuquerque, NM 87185.

1982 ACM 0098-3500/82/0600-0093 $00.00

For a differential equation in autonomous form, $y' = f(y)$, such methods have the form

$$(I - \gamma h f_y(y_0))k_i = hf\left(y_0 + \sum_{j=1}^{i-1} \alpha_{ij} k_j\right) + hf_y(y_0) \sum_{j=1}^{i-1} \gamma_{ij} k_j, \qquad i = 1, \ldots, s$$
$$y_1(x_0 + h) = y_0 + \sum_{i=1}^{s} C_i k_i. \tag{2}$$

Here the constants γ, α_{ij}, γ_{ij}, C_i define the formula. Each stage k_i is obtained by solving a system of linear equations with the same matrix. The linear combination of stages advances the solution y_0 at x_0 to y_1 at $x_0 + h = x_1$.

These formulas are not implicit in the sense that the BDF are and so avoid some implementation difficulties. It has proved possible to derive Rosenbrock formulas which in some respects have better stability than the higher order BDF. A price one pays for these and other advantages is that one must evaluate partial derivatives of f at every step. Ordinarily it is presumed that these partial derivatives are either clumsy or expensive to obtain, and for this reason the popular BDF codes try to evaluate f_y as infrequently as possible. This presumption is by no means always true, so Rosenbrock formulas should not be discarded for this reason alone. We shall restrict our attention in this paper to the class of problems for which the partial derivatives of f are convenient to obtain and are not a lot more expensive than the evaluation of f itself.

Recently Kaps and Rentrop [13] derived some Rosenbrock formulas with internal error estimators. This was a natural development in view of the history of explicit Runge-Kutta methods and was an important step in making the methods practical. The computational results they present suggest that Rosenbrock methods might be a practical alternative to the BDF. Their paper stimulated the author to develop a piece of mathematical software, DEGRK, based on a Rosenbrock formula. Here we report some of the algorithmic and software developments we considered necessary. Although these developments were realized in a particular code, most of the work is generally applicable to Rosenbrock methods.

At present, codes are clearly intended for stiff or nonstiff problems, but not both. Deciding the type of the problem is an impossible task for a user. This author considers the question of how to relieve the user of this decision to be the most pressing question in the area of ODE mathematical software. Within the class of problems we postulate here, the matter is relatively simple. We shall describe how to switch between an explicit Runge-Kutta formula pair and a Rosenbrock formula pair at any step reliably and economically. The implementation of DEGRK uses a Fehlberg F(4, 5) pair for the explicit Runge-Kutta formulas. If the problem is unequivocally nonstiff, the integration by DEGRK is nearly as efficient as that by RKF45 [21, 22], an effective code for nonstiff problems based on the F(4, 5) pair. The class of problems for which DEGRK is intended is easily recognized. In this class there is no particular reason for a user even to consider the issue of stiffness.

In this investigation we learned that virtually all of the published Rosenbrock methods have what we consider to be a serious defect for their use in productionquality codes. A variety of other one-step formulas suffer from the same defect. We have not seen this matter pointed out before, so we devote some space to it. It is the main reason we did not implement in DEGRK the formulas published by Kaps and Rentrop.

Rosenbrock methods solve linear systems which may become ill-conditioned. We shall present a practical and cheap approximation of the condition which may be of value for other methods as well.

2. GETTING PARTIAL DERIVATIVES

In the solution of (1) the Rosenbrock methods require evaluation of the partial derivatives f_y and f_x. Here we want to indicate some problems for which these partial derivatives are not inconvenient nor much more expensive to evaluate than f itself.

Perhaps the first observation ought to be that all the problems of the well-known test set [8] fall into this class. To be sure, many of the problems are artificial, but many are not. The supplementary test set of Enright and Hull [15, pp. 45-66] also falls into the class. Most of its problems arise from a description of chemical kinetics in a homogeneous solution reacting according to the mass action law. Chemistry problems of this kind typically fall in our class of interest.

Another class of problems which may well be suitable is the linear problem. The Jacobian f_y must be evaluated every time f is; so it cannot be expensive nor very inconvenient to provide it. The uncertainty lies in the f_x vector. Whether it is convenient and relatively inexpensive will depend on the problem.

In our experience and in reading the scientific literature, we have seen many individual problems which were in the class, and many which were not. One problem [18] which we use as a numerical example in Section 12 is

$$\frac{dx}{d\theta} = (1 + \xi)\left\{1 - (1 + N_f)x + \frac{N_f y}{y + K(1 - y)}\right\},$$

$$\frac{dy}{d\theta} = \frac{1 + \xi}{\xi} N_f \left\{x - \frac{y}{y + K(1 - y)}\right\}.$$

Here ξ, N_f, and K are (constant) parameters. This problem caught our eye because the chemical engineers were interested in a range of parameter values. For some values the problem is not stiff and for others, it is stiff. It illustrates the convenience of a code which does not ask the user to decide the type.

A very popular option in production codes for stiff problems is for the code to approximate the necessary Jacobians by numerical differentiation. This makes life easy for the user, but we do not think this option appropriate to Rosenbrock methods. One objection is fundamental. The Jacobian is merely an aid to the BDF codes—they will solve the ODE even if the approximation is terrible, albeit inefficiently. The Rosenbrock formulas are based on the partial derivatives and all statements about order and the like *depend* on an accurate Jacobian. Another objection is that approximating partial derivatives by numerical differentiation generally results in a rather expensive evaluation. As a general rule, we cannot expect the problem to be in our class of interest if f_y is approximated numerically.

3. THE FORM OF THE EQUATION

Theoretical treatments of Rosenbrock methods have taken the differential equation in autonomous form because it is convenient to avoid the special role of the independent variable. The research codes have followed the theory in this respect. Of course many problems do not arise in autonomous form, so users are expected to convert their problem.

We have chosen *not* to use the autonomous form for a number of reasons. One is the convenience of the software interface. The typical ODE solver accepts the form (1) so that users are accustomed to it. Conversion may be fairly described as a nuisance to the user and leads to questions about an appropriate error control for the x variable.

When using a Rosenbrock method, the linear systems to be solved constitute a significant fraction of the work. To reduce linear algebra costs ODE solvers provide options for various matrix structures. Conversion to autonomous form obviously affects the structure. We, for example, provide for banded Jacobians in DEGRK. This structure is lost on conversion. To retrieve it we would have to ask the user to recognize an unconventional structure for a problem in autonomous form, or to tell the code he actually started with a banded Jacobian and converted it. This kind of request is not likely to be popular with users.

The usual conversion to autonomous form adds an equation for x as a new dependent variable. The eigenvalues of the Jacobian of the augmented system are those of f_y plus an eigenvalue 0. This is not important, but *norms* may be more seriously affected. In the L_1 norm we use, $\|J\|_1 = \max(\|f_y\|_1, \|f_x\|_1)$. We use the norm of the Jacobian as a bound on the spectral radius to assure stability of the explicit Runge-Kutta formula, and for other purposes. Increasing the norm by conversion has a direct, harmful effect.

There are a couple of conceptual objections to the conversion. The typical BDF code, for example, accepts the form (1), and if the user provides analytical partial derivatives, he provides only f_y. The Rosenbrock methods require f_x *too*. This matter is concealed when all problems are accepted in autonomous form, but it is a distinction which could be important. Also, the conversion changes a linear to a nonlinear problem. It is interesting to note that the famous set of test problems [8] did precisely this with the Liniger-Willoughby problem D1. The conversion of linear problems obscures the fact that the Jacobian is immediately available in analytical form. It is not clear what algorithmic consequences might follow converting a linear to a nonlinear equation.

It is about as easy to implement the form (1) in a Rosenbrock code as the autonomous form. In many papers it has been considered obvious that one uses the autonomous form because of its elegance. For this reason we felt obliged to state a variety of arguments in support of our decision not to use it in DEGRK.

4. CONDITIONING

The Rosenbrock methods require the solution of linear systems involving matrices $E = I - h\gamma f_y$. This is also true of the typical implicit method for the solution of stiff ODEs, although it is done for a different purpose. It has been frequently commented that these matrices may be ill-conditioned, but we have not noticed any arguments to the effect that this must be so. We shall argue this here and devise a practical measure of the conditioning.

The situation is quite different in the cases of a Rosenbrock and, say, a BDF method. With the BDF and other implicit formulas, linear systems are solved to obtain successive iterates approximating the result defined implicitly. As described in [20, p. 109], this is normally arranged so that one solves for the change in the previous iterate. Ill-conditioning may slow down the overall iteration because some digits in the change are spoiled, but as long as a few leading digits are obtained correctly, the process "converges." With a Rosenbrock formula, the solutions enter directly (and indirectly through the function evaluations) into the solution value for the step. The situation for the first stage is especially clear. With such formulas, inaccurate solution of the linear system leads to inaccurate solution values. Normally one does not solve stiff ODEs to stringent (relative) accuracies, so with a reasonable computer word length, this is probably not very important in practice. However, in this respect Rosenbrock methods and methods like the BDF appear to differ fundamentally.

By definition $\mathrm{cond}(E) = \|E\|\|E^{-1}\|$. In general $\rho(M) \le \|M\|$, where $\rho(M)$ is the spectral radius of the matrix M. If λ is an eigenvalue of f_y, then $1 - h\gamma\lambda$ is an eigenvalue of E and its reciprocal an eigenvalue of E^{-1}. At this point we need to put in some information to the effect that the ODE problem is stiff. Stiffness is not a precisely defined concept. Nevertheless, many workers would be willing to accept a statement like the following. For a step size h yielding the required accuracy in the formula, the eigenvalues λ of the Jacobian f_y fall into two classes:

$$\begin{array}{ll} \mathrm{I} & |h\lambda| \ll 1, \\ \mathrm{II} & \mathrm{Re}(\lambda) \le 0. \end{array}$$

It is further assumed that neither class is empty, and that in class II there is an eigenvalue λ_j with $|h\lambda_j| \gg 1$. Notice that we do not take up the conditioning of a single equation.

The general result

$$\left| \frac{1}{1 - h\gamma\lambda} \right| \le 1 \qquad \text{if } \ \mathrm{Re}(\lambda) \le 0$$

tells us that no eigenvalue in class II causes $\rho(E^{-1})$ to be greater than 1. The assumption class I is not empty then implies that $\rho(E^{-1}) \doteq 1$. The assumption about class II says that

$$\rho(E) \doteq \max_k |\mathrm{h}\lambda_k| \ge |h\lambda_j| \gg 1.$$

From the general relation of spectral radius to norm, we now conclude

$$\mathrm{cond}(E) = \|E\|\|E^{-1}\| \ge \rho(E)\rho(E^{-1}) \gg 1.$$

Thus if the problem is stiff in the sense we have used, the matrix $E = I - h\gamma f_y$ *must* be ill-conditioned.

A problem is usually described as nonstiff if all eigenvalues of the Jacobian are in class I. This ignores the important role of the norm, and in these circumstances ill-conditioning is not precluded. If the stronger condition that $\|hf_y\|$ is rather less than 1 holds, it is easy to see that in this particular norm, $I - h\gamma f_y$ is not ill-conditioned.

Because conditioning directly affects Rosenbrock methods and because we have seen that ill-conditioning is to be expected, we considered how to get some idea of the conditioning. A scheme was devised [6] for LINPACK [7] which tries to compute a large lower bound for the condition of a factored matrix. A computable norm is chosen for $\|E\|$ which in LINPACK happens to be the same one we chose in DEGRK, namely, $\|E\|_1$. From $Ev = w$ one gets

$$\|E^{-1}\| \geq \|v\|/\|w\|.$$

The idea of [6] is to select a w judiciously so as to arrive at a large lower bound. We observed that there is a cheaper way to get a large lower bound in our context. It is perhaps a little clearer if we scale so that

$$E = f_y - \frac{1}{h\gamma} I.$$

Let λ_N be an eigenvalue of f_y of minimum modulus and let v be an associated eigenvector. Then

$$Ev = \left(\lambda_N - \frac{1}{h\gamma}\right) v$$

and as above

$$\|E^{-1}\| \geq \left| \frac{\gamma h}{1 - \gamma h \lambda_N} \right|$$

We shall approximate this lower bound by $h\gamma$. For stiff problems, $|h\lambda_N| \ll 1$, so this is a good approximation.

We could evaluate $\|E\|_1$ directly, but this does not seem worth the trouble. In general

$$\|E\| = \|f_y\| + \mathcal{O}\left(\frac{1}{h\gamma}\right).$$

In the particular norm we use, $\|E\|_1 \doteq \|f_y\|_1$ is an excellent approximation in the sense of relative error when $h\gamma\|f_y\| \gg 1$.

Finally then

$$\operatorname{cond}(E) \geq \frac{\gamma h \|E\|}{|1 - h\gamma\lambda_N|} \doteq \gamma h \|f_y\|,$$

where the approximation to the lower bound should be excellent if the ODE problem is stiff in the sense we have used.

The approximate lower bound for the condition is extremely convenient because all the pertinent quantities are computed (cheaply) for other purposes. For stiff problems it can be expected to provide a useful indication of conditioning. We have done a variety of experiments comparing the lower bound of LINPACK to our approximate lower bound. Experience with the LINPACK lower bound seems to show that it is comparable to the actual condition. The limited experiments we have done indicate that our cheap estimated lower bound is equally satisfactory in our very special circumstances.

Because of its generality, the LINPACK estimate is more expensive. It does a norm computation which we avoid by the approximation $\|E\| \doteq \|f_y\|$, available from other computations in DEGRK. It does two extra solutions of linear systems to form the estimate. The Rosenbrock procedure in DEGRK only does four solutions of linear systems in the step, so the LINPACK estimate represents a substantial extra expense. Because one advantage of the Rosenbrock methods may be their low overhead, the cheaper condition estimate is to be preferred here.

In DEGRK the question of ill-conditioning seems not to be serious. Because of the low-order formulas implemented and their less than optimal stability at infinity, severe ill-conditioning appears to be rare. In addition, the low order makes the code inappropriate for stringent tolerances. We have chosen to restrict the step size as necessary to ensure that

$$\| h\gamma f_y \| \leq 10^{10}$$

on a machine with about 14 decimal digits. Should such a restriction be imposed 10 times in a run, the integration is interrupted to warn the user of the situation and to inquire as to whether he wishes to continue.

5. FORMULA PAIRS IN DEGRK

In DEGRK we chose to implement a (4, 5) pair of formulas due to Fehlberg because they proved very satisfactory in other software, RKF45 [21, 22], we have written for nonstiff problems. Fehlberg intended that the integration be advanced using the fourth-order formula. In RKF45 we instead advanced the integration with the fifth-order formula, local extrapolation. The reasons given in [21, 22] for doing this remain valid in DEGRK, but in one respect the situation is quite different. The algorithm described in Section 8 for selecting methods guarantees that the step size used is stable for the F(4, 5) pair. Indeed, the conservative nature of the algorithm often means that when the F(4, 5) pair is used, the step size is well within the stability region. Thus the fact that the fifth-order formula is the more stable is not relevant in DEGRK. Furthermore, the constraint on the step size greatly increases the likelihood that the fifth-order formula is significantly more accurate than the fourth-order formula. As a result the local error estimate is more reliable and local extrapolation is more useful.

Kaps and Rentrop [13] have devised (3, 4) Rosenbrock formula pairs which are four-stage formulas involving three function evaluations and one partial derivatives evaluation per step. In their Proposition (3.19) they give a five-parameter family of formulas. In Proposition (3.20) they give a choice of parameters leaving one free parameter γ which results in a fourth-order formula satisfying five of the nine equations of condition for a fifth-order formula. The parameter γ essentially determines the stability properties of the pairs constructed from either proposition. The authors intended that the integration be advanced with the fourth-order formula. They give two formula pairs in [13] and a related pair in the text [24].

We have not used the pairs selected by Kaps and Rentrop for two main reasons. One is that the fourth-order formulas they selected are just barely stable at infinity. In the GRK4T pair and the pair in [24], the companion third-order formula is not stable at infinity. The GRK4A pair does have a third-order formula with reasonable damping at infinity. For this reason we chose first to implement the GRK4A pair, but advancing with the third-order formula. As we report in Section 11, this is a better way to proceed for difficult problems.

We would have been happier with GRK4A if the fourth-order formula were also strongly damped at infinity, but we were prepared to accept this until we ran into what we consider a serious defect: The Kaps–Rentrop formulas do not evaluate f at points spanning all of $[x_n, x_{n+1}]$. This is a general difficulty which we shall discuss in Section 9. Within the family of Proposition (3.20) there is just one possibility without this defect. As it turns out, both formulas have the same damping at infinity which is very nearly as good as the third-order formula in GRK4A. Furthermore both formulas are A-stable. Kaps and Rentrop gave their formulas in decimal form. We went through the tedious computations to obtain this other pair in rational form. It is pleasing that they turned out to be so simple. This increases portability. The formula pair is

$$\begin{aligned}
y' &= f(x, y)\\
E &= I - \tfrac{1}{2}hf_y(x_0, y_0)\\
Ek_1 &= f(x_0, y_0) + \tfrac{1}{2}hf_x(x_0, y_0)\\
Ek_2 &= f(x_0 + h, y_0 + hk_1) - \tfrac{3}{2}hf_x(x_0, y_0) - 4k_1\\
Ek_3 &= f(x_0 + \tfrac{3}{5}h, y_0 + \tfrac{24}{25}hk_1 + \tfrac{3}{25}hk_2) + \tfrac{121}{50}hf_x(x_0, y_0)\\
&\quad + \tfrac{186}{25}k_1 + \tfrac{6}{5}k_2\\
Ek_4 &= f(x_0 + \tfrac{3}{5}h, y_0 + \tfrac{24}{25}hk_1 + \tfrac{3}{25}hk_2) + \tfrac{29}{250}hf_x(x_0, y_0)\\
&\quad - \tfrac{56}{125}k_1 - \tfrac{27}{125}k_2 - \tfrac{1}{5}k_3\\
y_3(x_0 + h) &= y_0 + h\left(\tfrac{98}{108}k_1 + \tfrac{11}{72}k_2 + \tfrac{25}{216}k_3\right)\\
y_4(x_0 + h) &= y_0 + h\left(\tfrac{19}{18}k_1 + \tfrac{1}{4}k_2 + \tfrac{25}{216}k_3 + \tfrac{125}{216}k_4\right)\\
y_4(x_0 + h) - y_3(x_0 + h) &= h\left(\tfrac{17}{108}k_1 + \tfrac{7}{72}k_2 + \tfrac{125}{216}k_4\right).
\end{aligned} \tag{3}$$

6. EFFICIENT REPRESENTATION

The usual form of the Rosenbrock formulas (2) apparently requires the storage of the Jacobian matrix and a matrix-vector multiplication at each stage. These costs can be avoided by a simple manipulation of the formula as suggested by Wolfbrandt [29, p. 90] and others [14]. The resultant form is exemplified by the pair (3).

A significant amount of arithmetic can be saved in the formation of E by scaling. In the case of (3) we simply work with $f_y - (2/h)I$ instead of E whenever a linear system is to be solved. For the solution of stiff ODEs we think this is a more natural scaling anyway. Scaling in this way is advocated by Gourlay and Watson [28, pp. 123-133] for a BDF code and is used in a sparse, semi-implicit Runge-Kutta code [11], but it does not seem to be well known yet.

Solution of (3) involves the formation and factorization of E and then the s solutions for the k_i. The question that interests us right now is whether to keep a copy of the Jacobian f_y or to write over it in forming and factoring E. Because a Rosenbrock method presumes that f_y changes at every step, it is recomputed after every successful step. So the only obvious reason for saving f_y is to reuse it when repeating a rejected step. Because of the expense of a failed step, step size selection algorithms are usually rather conservative so as to make failed steps uncommon. We expect Rosenbrock methods to be applied to problems for which computation of f_y is not much more expensive than computation of f. Thus recomputation of f_y at failed steps should not be a very big waste for the kind of code and problem we have in mind. In compensation we roughly halve the storage required by the code. We deemed this to be a bargain in DEGRK.

7. SOFTWARE INTERFACE

Recently the author and H. A. Watts [23] presented a design for a software interface to a package of ODE solvers called DEPAC. At the time the package contained three solvers: DERKF, a Runge-Kutta Fehlberg code; DEABM, an Adams-Bashforth-Moulton variable order code; and DEBDF, a BDF variable order code. The generalized Runge-Kutta Fehlberg and Rosenbrock code DEGRK was written to fit into DEPAC and is now being distributed with it. Accordingly, DEGRK was provided with all the user convenience and protection specified in the package. For the most part, the interface is an obvious mixture of the interfaces for the Runge-Kutta and BDF codes along with appropriate descriptive comments. Some matters are pertinent only to DEGRK. One is to discover and report that ill-conditioning is causing the step size to be restricted. The package design was intended to incorporate such additional interrupts.

We have chosen a different form for the partial derivative routine than is customary. In part this is necessary. A BDF routine needs only the Jacobian f_y; a Rosenbrock routine needs f_x too. The difference could have been concealed by using the autonomous form, but we think it better to emphasize the difference. Thus the partial derivative routine returns with the matrix f_y and the vector f_x. We require f to be evaluated in this subroutine at the same time. This is *in addition* to providing a separate subroutine for the evaluation of f. The device is intended to increase the efficiency of the code and to make it more likely that partial derivatives are not a lot more expensive than a function evaluation. It depends on the fact that the code never requires evaluation of the partial derivatives without also requiring evaluation of the function at the same argument. The gain to be made is that often the function evaluation is cheap if combined with the evaluation of the partial derivatives. For the examples of Section 2 one almost has to evaluate f in the course of evaluating f_y and f_x. If the user chooses to program the partial derivative subroutine to take advantage of this fact, and if the call list is as we take it, a function evaluation is obtained at a considerably reduced cost. If the user does not want to be bothered, or if it is not cheaper to combine the f and the partial derivative evaluations, he can simply insert a call to the f subroutine in his subroutine for the partial derivatives. This costs the user some linkage and a little complication in writing the partial derivative subroutine, but the cost is not large. When applicable, the device could be quite helpful.

8. STIFF OR NONSTIFF?

Within the class of problems postulated, it is relatively easy to decide at any step whether to use an explicit or Rosenbrock one-step method. We shall describe what we did in DEGRK and the reader will see that the ideas are broadly applicable. Although crude, the decision procedure is remarkably useful.

We have found that an effective code for nonstiff problems can be based on a pair of formulas of orders 4 and 5 involving 6 stages which were devised by Fehlberg. We would like to be able to switch from such a code to a procedure suitable for stiff problems when it would be more efficient and back when it would not. Naturally we expect to pay something for the convenience of such a type-insensitive code, but we hope that the cost will be almost negligible if the problem is unequivocally nonstiff or stiff. This switching turns out to be feasible.

The first question we answer is when to switch to a method suitable for stiff problems—in our case a Rosenbrock formula pair. The explicit Runge-Kutta formula is inefficient only when a step size h_{acc} suitable for achieving the requested accuracy must be reduced to h_{stable} to keep the computation stable. We can decide when to switch if we can estimate h_{acc} and h_{stable}. One's immediate reaction is likely to be that all general-purpose codes estimate h_{acc}, and we need only consider h_{stable}. Unfortunately this is not so. We have discussed the behavior of Runge-Kutta codes in the presence of stability restrictions elsewhere [19]. Briefly, if $h_{\text{acc}} \gg h_{\text{stable}}$, the code will increase the step size until the computation becomes unstable. The growing error is seen by the local error estimator and the step size reduced until the computation is again stable. For such a step size propagated error is actually damped out and eventually the smooth behavior of the true solution appears in the numerical solution. As this behavior is manifested, the code realizes its step size is smaller than h_{acc} and increases the step size. The cycle repeats itself. It is gratifying that the error never gets out of hand, but the difficulty we must face here is that the step size which the code estimates as appropriate for the accuracy is ordinarily far smaller than h_{acc}. To obtain a reasonable estimate of h_{acc}, we must force the code to work within its region of absolute stability. Thus a critical issue is to obtain a good estimate or a reliable bound for h_{stable}.

Most explicit Runge-Kutta methods have stability regions which contain a (half) disk of radius ρ. (Van der Houven calls ρ the generalized stability boundary [26, p. 83]. If λ is any eigenvalue of the Jacobian f_y with $\text{Re}(\lambda) \leq 0$ and $|h\lambda| \leq \rho$, the method is absolutely stable with step size h. We obtain a computable relation from the bound $|\lambda| \leq \|f_y\|$. In DEGRK we use the L_1 norm which is a simple, cheap computation. *Both* the Fehlberg (4, 5) formulas are stable if we require

$$h\|f_y\| \leq 2.4. \tag{4}$$

This condition is forced on the step size when the explicit Runge-Kutta method is used so as to guarantee the computation is stable. Then we can be sure that the step size estimated by the formula pair as appropriate for the requested accuracy actually approximates h_{acc} and can be used to decide when to switch.

DEGRK is organized as follows: There is a step size to be attempted which was estimated in a special module for the first step or in the module used to attempt the previous step. This step size may be reduced so as to produce output at desired points. This matter is described in [21, 22]. Unlike RKF45, DEGRK does not use the "stretching" device, but it does use a "look-ahead." As described in Section 4, the step size might be reduced to improve the conditioning of the matrix E in (2). These adjustments to the step size are done before the method is selected because the choice is critically dependent on the step size. In a module it is decided which method to use and the step size is possibly reduced further. Next control goes to one of the two modules for attempting a step by the two methods. If the step is a success, the module used estimates what step size is appropriate for the next step. If the step is a failure, a step size for another try is selected. After a failure control is returned to the point where this description began. This is the reason we said "to attempt" the previous step.

There are three cases. The first step is always taken with the explicit Runge-Kutta method so as to get on scale. Also it may be necessary to try several times if the estimated step size is badly off, and this is much cheaper to do with the explicit formula. The other two cases depend on the method used for the preceding step.

Suppose the preceding step was taken with the Rosenbrock method. If the step size satisfies (4), we switch to the Fehlberg scheme and otherwise continue with the Rosenbrock method. This implies that the explicit Runge-Kutta formula will be used for all sufficiently small step sizes. There is a question as to how to adjust the step size on the change of formula. Here we do not adjust it at all. The Fehlberg pair is of higher order and is an accurate pair of more stages. We postulate that if it is stable, it is more accurate than the Rosenbrock pair. Indeed, because we might be well within the stability region of the method, the F(4, 5) pair might be a lot more accurate than necessary with this step size. Because we adjust step size at every step it is not necessary that we have a good scheme for altering the step size when we change formula. On the other hand, we do need to prevent frequent changes so as to allow the code time to match the step size to the accuracy required.

If the preceding step was taken with the Fehlberg formula, we reduce the step size as necessary so that (4) holds and stability is guaranteed. If the step size had to be reduced pretty significantly, we can expect that stability, rather than accuracy, is determining the step size. Then it is reasonable to believe that the Rosenbrock method will be successful at this step size even though it is a lower order method than the Fehlberg method. In DEGRK we switch to the Rosenbrock method if the step size satisfying (4) is less than half that deemed appropriate for the Fehlberg method.

It is not very likely that a problem would call for a step size h such that $h \| f_y \| \doteq \rho$ for many steps, but to make frequent switches less likely, we have made it easier to switch to the Fehlberg formula than vice versa. In point of fact, frequent switches would not be important at all except for the crudity of the "adjustment" of step size on a switch.

To hold down the overhead, especially for nonstiff problems, we do not evaluate the Jacobian nor its norm at every step. We keep track of whether the Jacobian has been evaluated at the current step and whether its norm has been evaluated. In the module for selecting the method, we check if the step size is close to the critical point, specifically if

$$\tfrac{1}{2}\rho \le h \| f_y \|_{\text{old}} \le 4\rho.$$

If it is, we form, if necessary, a current f_y and we form, if necessary, a current $\| f_y \|$ for our decision. In any event we form a current value of $\| f_y \|$ every five steps. With the Rosenbrock scheme, this saves a number of matrix norm computations. With the Fehlberg scheme, this saves a good many unnecessary Jacobian evalu-

Table I. Solution by DEGRK of the Problem E2, Which Is Nonstiff but Borders Being Stiff at Crude Tolerance

Tolerance	Number of steps	max $h \| f_y \|$	Number of f_y evaluations
10^{-2}	6	4	6
10^{-4}	10	2	7
10^{-6}	19	2	7

ations. If the problem is unequivocally nonstiff, we shall evaluate the Jacobian every five steps. For the six stage F(4, 5) methods this represents 30 function evaluations. We are presuming of the class of problems that evaluation of the function and the Jacobian together is not a lot more expensive than evaluating the function alone. To get some idea of the costs, suppose that the evaluation of both function and Jacobian is $2\frac{1}{2}$ times the cost of evaluating a function alone. In such a case, evaluating the Jacobian to test for stiffness increases the cost in function evaluations of solving an unequivocally nonstiff problem by only 5 percent. We consider this to be a negligible cost for the convenience of a type-insensitive code. We remark that, roughly speaking, DEGRK behaves like the efficient code RKF45 when it is confronted with an unequivocally nonstiff problem.

Clearly the cost of testing goes up when the code is working close to the switching point. One might evaluate the Jacobian at every step, even though the integration is carried out with the explicit formula pair. On the other hand, the conservative nature of the algorithm means that a problem may be treated as stiff when it would actually be more efficient to use the explicit Runge-Kutta scheme. This strikes us as an unavoidable price which should not be a large one.

We shall consider a few examples to illustrate the usefulness of switching. First let us consider the problem E2 of the test set [8]. This is van der Pol's equation, but it is *not* undergoing relaxation oscillations and we consider it not to be stiff. According to the authors of the test set, the maximum magnitude of an eigenvalue is at most 15 and the length of the interval is only 1. Results are displayed in Table I. When solved with DEGRK at a pure absolute error tolerance of 10^{-2}, the problem is marginal. Four of the six (!) steps needed to solve the problem were taken with the F(4, 5) pair. The Jacobian was evaluated at every step because this is a borderline problem. At the tolerances 10^{-4} and 10^{-6} all steps were taken with the F(4, 5) pair. At the crudest tolerance when the problem was most ambiguous, the code made 36 function evaluations so that the 6 evaluations of the partial derivatives (the associated f evaluation is included in the 36 reported) was a significant but acceptable cost. At the most stringent tolerance there were 121 f evaluations and the number of partial derivative evaluations approaches the 5 percent we expect in a clear-cut case.

For the sake of variety we shall report some results in Table II that were computed with the Kaps-Rentrop pair GRK4A advanced with the third-order formula. The code is DEGRK with the pair given in Section 5 replaced by GRK4A. The B family of problems in the test set [8] are linear with nonreal eigenvalues. B2-B5 is a family of one parameter with the eigenvalues getting larger and moving closer to the imaginary axis as one goes from B2 to B5. B5 is

Table II. CPU Times for Solution at All Three Tolerances 10^{-2}, 10^{-4}, 10^{-6} for Several Problems: The Effect of Recognizing Nonstiff Regions is Shown

	B4	B5	A3	A4
GRK4A and F(4, 5)	0.527	0.950	0.831	1 104
GRK4A alone	1.489	4.366	1.141	1.509
BDF	1.301	18.689	1.025	2.791

a trap for high-order BDF formulas which suffer a stability restriction with this problem. The Rosenbrock formulas we have implemented are all A-stable. When the Fehlberg formulas were used, more function evaluations were made; for example, at 10^{-4} on B5 the function evaluations increased from 652 to 768, but the number of partial derivative evaluations dropped as did the LU decompositions and solutions of linear systems. The real-time considerations make it impossible to define an optimal switching point between formulas, but our results suggest that we have made an adequate choice. By way of indicating the possibilities of the kind of code we investigate, we made the same computations with the BDF code of the NAG library [16] given analytical Jacobian and the same tolerances. All the numerical results obtained were of accuracy comparable to DEGRK. Lest the reader think that the results reported for B4 and B5 are somehow due solely to the oscillatory nature of the solutions and the nonreal eigenvalues, we also display results for some of the A family of linear problems with real eigenvalues.

Evidently switching formulas to account for a lack of stiffness is of significant value for these example problems, even though they are considered to be "stiff" test problems. A further family of stiff and nonstiff problems will be analyzed in Section 12.

9. DESIGN CRITERIA FOR ONE-STEP METHODS

Runge-Kutta and Rosenbrock methods evaluate the function several times in the course of a step of length h from x_n to $x_n + h$, say at $x_n + A_i h$, $i = 1, 2, \ldots$. The author and his colleague H. A. Watts have pointed out in connection with explicit Runge-Kutta methods that it is desirable that the evaluations span the interval $[x_n, x_{n+1}]$. This is so that discontinuities can be "seen" by the formula.

It is typical of stiff problems that they exhibit small regions in which the solution changes so fast that it is almost discontinuous on a time scale suitable for the rest of the problem. We shall describe these boundary layers or transition regions here as quasi-discontinuities. Relaxation oscillations are a familiar example. Another kind of example comes from a forcing function. Hindmarsh and Byrne have considered a couple of mock-ups of photocatalyzed atmospheric reactions (see, e.g., [5]) which are illustrative. The simpler has the form

$$y'(t) = d - by + aE(t).$$

The forcing function $E(t)$ is zero during the 12-hour night. At sunrise it increases in seconds to a value almost constant during the day and reverts to 0 at sunset. The problem is so stiff that the solution is nearly always in steady state, in particular it has the constant value d/b at night. Thus the forcing function $E(t)$ and the solution $y(t)$ are nearly square waves.

With the more familiar methods we expect a code to locate a quasi-discontinuity very sharply. During a period of slow variation a code for stiff problems will take very large time steps. On such a time scale a boundary layer "looks" like a discontinuity. We expect, and find in the widely used codes, that codes will have repeated step failures at such a quasi-discontinuity until the step size is reduced to the point that the solution is not changing rapidly on the new time scale. Of course this means that the boundary layer is located accurately and resolved to the degree necessary.

Quasi-discontinuities cannot be regarded as pathological for stiff problems and it is clear that serious errors in their solution are possible with any formula which does not evaluate at t_{n+1}. Specifically, if during a smooth portion of an integration a method might use a step size of h, a quasi-discontinuity could be located improperly by as much as $(1 - A_j)h$, where $t_n + A_j h$ is the point closest to t_{n+1} at which the evaluation takes place.

Among the fully implicit Runge-Kutta methods, considerable attention has been directed at those based on the Gaussian points because they achieve maximal order and A-stability. They are all defective in the way we have pointed out, with the midpoint rule being the worst case. Hulme and Daniel [12] have a code implementing both Gaussian and Radau formulas with doubling as an error estimator. Our observation applies directly. It is interesting to note that in the recent derivation of some formulas by Butcher [4] (and implemented by Burrage, Butcher, and Chipman) the defect is not considered and it is quite possible. However, the additional constraints applied to achieve better stability properties had the side effect of avoiding the defect.

Lindberg [28, pp. 201-215] has based an extrapolation code on a modified midpoint rule

$$y_{n+1} = y_n + hf\left(t_n + \frac{h}{2},\ \frac{y_{n+1} + y_n}{2}\right).$$

It is interesting that there has been some discussion [10, p. 165] as to whether the basic formula ought to be this rule or the trapezoidal rule

$$y_{n+1} = y_n + \frac{h}{2}\,[f(t_n, y_n) + f(t_{n+1}, y_{n+1})].$$

The argument advanced in favor of the midpoint rule is that it is unnecessary to evaluate $f(t_{n+1}, y_{n+1})$. In the present context we see that this is an argument *against* the midpoint rule.

The midpoint rule is an example of a semi-implicit formula. Some computationally interesting examples of such formulas considered by Crouzeix, Alexander [1], and Norsett exhibit one or more defects arising from an attempt to achieve various other computationally desirable properties. Crouzeix's (2, 3) A-stable DIRK formula does not evaluate at t_{n+1}. The (3, 4) formula evaluates in the *future*, as does Norsett's formula.

The defect we have noted is practically standard with Rosenbrock formulas; see, for example, the formulas used in the codes of Bui [3], of Villadsen and Michelsen [27], and of Kaps and Rentrop [13].

In the course of these studies we noted that a number of codes are based on one-step methods which evaluate outside the step, either in the past, some $A_i <$

0, or in the future, some $A_i > 1$. This has traditionally been avoided without any special comment, but in view of the recent use of such formulas, a few remarks seem to be in order. If a problem arises in autonomous form, there is no obstacle to evaluating outside the step. As we have commented earlier, most theoretical work is done with the autonomous form and it is easy to understand how a researcher might overlook an evaluation outside the interval. Several of Bui's Rosenbrock formulas evaluate in the past. This is not greatly different from a method with memory. There is an obvious difficulty with starting and after (effectively) restarting due to discontinuities. Bui's code apparently assumes that evaluation in the past will cause no problem, but this is not always true. Alexander [1] notes that a semi-implicit formula of Crouzeix evaluates in the future. Norsett's pair as implemented by Houbak and Thomsen [11] does this too. There is not then a starting problem, but there is a termination problem. It is not uncommon that it is not possible to evaluate the function past some point, or its definition changes there. The DEPAC [23] software design specifically provides users a way to warn the code that this is the case. Any formula which evaluates in the future needs to take special action in such a case.

We have not thought of any easy and reliable remedy for the defect of not evaluating at the end of the step when solving stiff problems. Perhaps we should remark that it is the combination of formula and error estimator that counts. If the formula did not evaluate at the end, but the estimator did, there would be no difficulty. We take a serious view of this defect. Evaluating outside the step is not so serious. For many problems no special action is needed. Easy remedies seem feasible because the difficulty is similar to a familiar one, but this does get away from the (relative) simplicity of one-step methods. We feel that as an absolute minimum of protection to the user, the prologue of any code based on such a formula should warn the user of the situation so that he can recognize when the code is not applicable.

10. ADJUSTMENT OF STEP SIZE

The principles of the adjustment of step size for explicit Runge-Kutta methods are discussed at length in [21, 22]. We have followed them in the portion of DEGRK concerned with the F(4, 5) formulas. However, if a step size should fail more than once, we reduce the step size by the fixed factor of 0.2. This is because the asymptotic behavior expected is not evident, else we would not have multiple failures. With no other information we resort to the fastest reduction ordinarily allowed.

There are some new issues when solving stiff problems that we have not seen discussed. One is losing the scale of the problem. For some particularly difficult problems, the Rosenbrock formulas we have implemented have needed to restart repeatedly. The code would be integrating a smooth solution with a very large step size and suddenly find it necessary to reduce the step size to the point that the problem is nonstiff. It would then move back to the smooth solution, at which time it would begin to increase the step size rapidly. We observed several cases when the algorithm for step size adjustment appropriate to the F(4, 5) formulas required more than 25 reductions of step size to finally obtain a successful step.

The problem with the results mentioned is a general one. When solving stiff problems the observed order may not be that of the formula applied to nonstiff problems. Prothero and Robinson [17] have taken up this matter. Ueberhuber [25] has tried to cope with it in another context. It is easy to see that there is a difficulty by considering a one-step method applied to the specific scalar equation

$$y' = \lambda y.$$

If at x_n we have a computed solution y_n, the typical one-step method leads to

$$y_{n+1} = R(h\lambda)\, y_n,$$

where R is a rational function. The local error

$$\mathrm{le} = y(x_n + h) - y_{n+1} = (\exp(h\lambda) - R(h\lambda))\, y_n.$$

When $|h\lambda| \ll 1$, we have

$$\left| \frac{\mathrm{le}}{y_n} \right| = \mathcal{O}(|h\lambda|^{p+1})$$

for a method of order p. However, when solving stiff systems we are interested in this differential equation for $\mathrm{Re}(\lambda) < 0$, $|h\lambda| \gg 1$ and the situation is radically different. First we note that

$$-\frac{\mathrm{le}}{y_n} \doteq R(h\lambda) = c_0 + \frac{c_1}{h\lambda} + \frac{c_2}{(h\lambda)^2} + \cdots .$$

The Rosenbrock methods we implemented all have $c_0 \doteq 0.3$. One of the problems we integrated had $|h\lambda| \sim 10^{10}$, so it is not surprising that the local error did not behave like a third-order formula. It is surprising to many that the local error may well be a *decreasing* function of h for $|h\lambda| \gg 1$. To get the kind of behavior we expect, it may be necessary to reduce $h\lambda$ enormously.

We have responded to the situation in two ways. On a failed step we are pessimistic about the assumed asymptotic behavior. Because of the work involved, it is better to attempt a step size too small and succeed than one too large and fail. On a first failure, we simply halve the step size. Should this fail, we reduce the step size attempted by a factor of 0.2. Should this step size fail, we, in effect, restart by reducing the step size so that $\| hf_y \| = \rho$, thus forcing the code to change to the explicit Runge-Kutta formula. This drastic action is because we have accumulated evidence that the scale of the problem has been lost. For reliability we reduce the step size to the point where any integral curve can be resolved.

On a successful step we estimate an appropriate step size for continuing, but limit it depending on how stiff the problem is. The explicit formula for nonstiff regions permits a step size increase as large as a factor of 5. The larger $\| hf_y \|$ is, the more conservative we choose to be because we are working in a region where our theoretical underpinnings are shaky. Specifically in DEGRK, we limited the increase of step size to

$$1.2 + \frac{3.8}{1.0 + \dfrac{\| hf_y \|}{50}}.$$

Thus if the problem is barely stiff, the increase is limited to a factor of 5, and if it is extremely stiff, to a factor of 1.2.

11. STABILITY PROPERTIES

The stability of methods for the solution of stiff problems has been the subject of intensive research. Nevertheless, our understanding of the matter is far from answering the needs of practice. Early work rigorously applies only to problems of the form $y' = Jy$ with a constant J which can be diagonalized by a similarity transformation. The common numerical methods can be analyzed by the same transformation so that one can test stability by considering the method as applied to $y' = \lambda y$ for λ a (complex) eigenvalue of J. Rosenbrock methods applied to this test equation lead to a rational function $R(h\lambda)$ of the step size h and λ. If $|R(h\lambda)| \leq 1$, the computation is stable and otherwise, unstable. The application of this analysis to more complicated problems is heuristic. Although experience shows it to be useful, one should not put too much faith in it.

The reason we give this background is that the Kaps–Rentrop formula pairs have $|R(\infty)| \doteq 1$ for the formula they intended for advancing the solution. When solving stiff problems we are very interested in step sizes h such that for some eigenvalue λ of the Jacobian, $|h\lambda| \gg 1$. The author much prefers to use formulas for which the stability is not so marginal, so as to be a little more confident that they will be applicable to problems less artificial than the test equation.

Besides the matter of stability, there is the related matter of how accurate formulas are for $|h\lambda| \gg 1$. At least for the test equation, this can be studied in detail in terms of how well $R(h\lambda)$ approximates $\exp(h\lambda)$. If $|R(\infty)| \doteq 1$, there is no qualitative agreement for $|h\lambda| \gg 1$. If $|R(\infty)|$ is significantly less than 1, the numerical solution is at least damped.

We preferred to advance the solution with the third-order formula of the GRK4A pair because it has $|R(\infty)| \doteq 0.31$. We actually tried advancing with each formula of the pair. Kaps has told us that in the tests of [13] it was more efficient to use the fourth-order formula. This is easy to understand because the test set [8] is not particularly demanding and rewards high order. Our experience was somewhat different because our code used the Fehlberg scheme part of the time. Whenever the Fehlberg scheme could be used, one would expect that the higher order formula of the Rosenbrock pair would be advantageous. In our computations with the test set [8] there was no important distinction due to which formula of the Rosenbrock pair was used. The matter was different when harder problems were tried.

A good example of our experiences, though not the most dramatic, is the problem of Bui [2] integrated to $x = 5$. We made runs in which the solution was advanced with the third-order formula and corresponding runs with the fourth-order formula of the GRK4A pair. The results are displayed in Table III. With pure absolute error tolerances there was no striking difference. The number of steps gives a fair impression of the relative work. Although not negligible, the difference does not compare to that observed when pure relative error tolerances were used. Considering the cost of a step, this represents an important difference in the performance of the formulas and caused us to prefer the more damped formula.

We would prefer that both formulas of the (3, 4) pair be strongly damped at infinity. Also, we would prefer to advance the solution with the fourth-order formula to take advantage of the higher order. This is partly why we made a

Table III. The Number of Steps Required to Solve Bui's Problem with the GRK4A Pair When the Integration Is Advanced with the Formula of Order 3 and of Order 4

		Order 3	Order 4
Absolute error	10^{-2}	24	28
	10^{-4}	90	114
Relative error	10^{-2}	158	253
	10^{-4}	769	922

different selection of formula pair in Section 5 than did Kaps and Rentrop. With our choice both formulas are A-stable and both have $|R(\infty)| \doteq 0.33$. This is very nearly the same damping at infinity as that of the third-order formula of GRK4A, but now we can advance the solution with the higher order formula (which by construction is a relatively accurate formula of order 4).

12. MORE NUMERICAL RESULTS

As we said in the Introduction, it is not our object to compare the performance of the code DEGRK to popular BDF codes. Some results were reported in Sections 8 and 11. We shall present here a few additional results intended to say something about the algorithms used in DEGRK and to suggest that Rosenbrock methods might be competitive in suitable circumstances.

In Section 2 we stated a problem from the chemical engineering literature which depends on three parameters K, ξ, N_f. In the article referenced a set of computations is reported for the nine problems resulting from the choices $K = 5$; $\xi = 0.1, 5, 500$; $N_f = 0.1, 5, 50$. The solutions are well scaled so an absolute error test is reasonable. We solved all nine problems at a given tolerance with DEGRK and then with the BDF code of the NAG library [16]. The results are displayed in Table IV. Spot checking of the apparent accuracies suggests that DEGRK is producing a somewhat more accurate result, but that the accuracies are roughly comparable. These results and others of the kind show that DEGRK may be more efficient in a real time sense for suitable problems provided one does not ask for a great deal of accuracy. Kaps and Rentrop came to a similar conclusion in [13].

The parameter choice $K = 5$, $\xi = 0.1$, $N_f = 0.1$ results in the least stiff problem. At all three tolerances the F(4, 5) formulas are used at every step. At tolerance 10^{-2} there are only 3 steps, and 3 Jacobian evaluations were made. At tolerance 10^{-4} the decision is less ambiguous because of the smaller step size needed to get the accuracy. There were then only 6 steps and 2 Jacobian evaluations. At tolerance 10^{-6} there were 12 steps and 3 Jacobian evaluations. Because so few steps are made in solving this problem, the number of Jacobian evaluations is relatively large. As we would expect, the more stringent the tolerance, the less stiff the problem looks and the fewer Jacobians are needed in our test. It is no surprise that DEGRK is more efficient than the BDF code in terms of function and Jacobian evaluations. At tolerance 10^{-2} DEGRK required 21 function evaluations along with the 3 Jacobian evaluations, whereas the BDF code needed 35 function evaluations and 8 Jacobian evaluations. The difference of performance

Table IV CPU Times for Solution of a Chemical Engineering Problem for 9 Sets of Parameters: Some Sets Result in a Nonstiff Problem; Others in a Stiff Problem

Tolerance	DEGRK	BDF
10^{-2}	0.205	0.497
10^{-4}	0.754	1.02
10^{-6}	4 34	1.67

Table V. Solution by DEGRK of the Problem of Table IV with the Parameter Set Resulting in the Stiffest Problem

Tolerance	max $h\gamma \lVert f_y \rVert$	Steps with F(4, 5)	Total steps
10^{-2}	7353	2	21
10^{-4}	4478	12	77
10^{-6}	1734	36	636

in this measure increases rapidly as the tolerance becomes more stringent for a nonstiff problem.

The parameter choice $K = 5$, $\xi = 500$, $N_f = 50$ results in the stiffest problem. Results are displayed in Table V. According to the results of Section 4, the values of $h\gamma \lVert f_y \rVert$ imply some fairly ill-conditioned systems in the evaluation of the Rosenbrock formula. As is typical, more stringent accuracy requests lead to smaller step sizes and better conditioned systems. Thus, in a way, we can expect more accurate solutions when we really need them. A significant number of steps were taken with the explicit method at each tolerance. Notice the rapid increase in the number of steps as the tolerance is made more stringent. This is characteristic of a fixed-order method.

It is especially hard to compare codes on difficult problems, but we shall present one example which has its interesting points. Scott and Watts [15, pp. 197–227] report a difficult initial value problem arising from the solution by shooting methods of a boundary value problem describing a kidney function. The system of five equations shows a dramatic difference in cost when using the Adams suite ODE/STEP, INTRP on variation of one initial value from 0.99026 to 0.99000. In large measure the difference in behavior is due to stiffness, although in another study we found that both problems are stiff. The integrations are very sensitive so high accuracy was necessary in the application. Such high accuracy makes DEGRK inappropriate, but we thought it interesting to explore the problem at relatively crude tolerances because of the differing stiffness.

For each of the two different initial values cited, we solved the problem at the two pure relative error tolerances 10^{-2}, 10^{-4}. DEGRK must take the first step of an integration with the explicit Runge-Kutta pair, but for these integrations the problems were so stiff that it took no other steps with the explicit formula. The problem with initial value 0.99000 is significantly stiffer. We computed in every case the maximum value of $h\gamma \lVert f_y \rVert$ as an indication of the stiffness. For the initial value 0.99000 this maximum ranged from 4000 to 7000. For the initial value 0.99026, it ranged from 20 to 50.

We also solved the problems with the BDF code from the NAG library. A difficulty is that the computed results are of differing accuracies. We computed solutions at the pure relative error tolerance of 10^{-6} with the BDF code and regarded them as the "true" solutions in what follows. In the application it is the value of the solution at the end of the integration which is critical, so we concentrated on it.

For the problem with initial value 0.99000, the BDF code computed a solution cheaply at tolerance 10^{-2}, 0.049 units of central processor time, but it was worthless. For example it reported the first two solution components to be about 1.89×10^{0}, 5.81×10^{-1}, when they, in fact, are about 1.38×10^{2} and 7.21×10^{-3}. At the tolerance 10^{-4} the cost was 0.295 units and the maximum relative error was about 1.3×10^{-1}. When DEGRK was given the tolerance 10^{-2} it took more time, 0.180 units, but it produced a result almost as good as that with tolerance 10^{-4} in the BDF code, namely, a maximum error of 1.7×10^{-1}. When DEGRK was given the tolerance 10^{-4} it took less time, 0.248 units, than the BDF code and got a lot more accuracy, namely, a maximum error of 2.0×10^{-3}. The situation was similar, though rather less dramatic, for the initial condition 0.99026.

The kind of results seen on this problem did not surpise the author because he adopted rather conservative tactics in DEGRK and furthermore some of the algorithms have a tendency to result in more accuracy than required. The line of BDF codes starting with DIFSUB [9] are not so conservative. The situation makes it hard to compare DEGRK directly to BDF codes, but this is not the object of the present paper. We do think the results presented show that Rosenbrock codes are competitive with BDF codes in appropriate circumstances and that DEGRK, in particular, is in some respects successful.

REFERENCES

1. ALEXANDER, R. Diagonally implicit Runge-Kutta methods for stiff O.D.E.'s *Siam J. Numer. Anal 6* (1977), 1006-1021.
2. BUI, T.D. Some *A*-stable and *L*-stable methods for the numerical integration of stiff ordinary differential equations. *J. ACM 26,* 3 (July 1979), 483-493.
3. BUI, T.D., AND BUI, T.R. Numerical methods for extremely stiff systems of ordinary differential equations. *Appl. Math Model. 3* (1979), 355-358.
4. BUTCHER, J.C. A transformed implicit Runge-Kutta method. *J. ACM 26,* 4 (Oct. 1979), 731-738.
5 BYRNE, G.D., HINDMARSH, A.C., JACKSON, K.R., AND BROWN, H G. A comparison of two ODE codes: GEAR and EPISODE. *Comp. and Chem. Eng. 1* (1977), 133-147.
6. CLINE, A.K., MOLER, C.B., STEWART, G.W., AND WILKINSON, J H. An estimate for the condition of a matrix. *SIAM J. Numer. Anal. 16* (1979), 368-375.
7. DONGARRA, J.J., BUNCH, J.R., MOLER, C.B., AND STEWART, G.W. *LINPACK User's Guide.* Society for Industrial and Applied Mathematics, Philadelphia, 1979.
8. ENRIGHT, W.H., HULL, T.E., AND LINDBERG, B. Comparing numerical methods for stiff systems of O.D.E.'s. *BIT 15* (1975), 10-48.
9. GEAR, C.W. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall, Englewood Cliffs, N.J., 1971.
10. HALL, G , AND WATT, J.M. *Modern Numerical Methods for Ordinary Differential Equations* Clarendon Press, Oxford, England, 1976.
11. HOUBAK, N., AND THOMSEN, P G SPARKS a FORTRAN subroutine for the solution of large systems of stiff ODE's with sparse jacobians. NI-79-02, Institute for Numerical Analysis, Tech. Univ. of Denmark, Lyngby, 1979.
12. HULME, B L., AND DANIEL, S.L. COLODE a colocation subroutine for ordinary differential equations. SAND74-0380, Sandia National Laboratories, Albuquerque, N. Mex., 1974.

13 KAPS, P., AND RENTROP, P Generalized Runge-Kutta methods of order four with stepsize control for stiff ordinary differential equations. *Numer. Math* *33* (1979), 55–68.
14 KAPS, P , AND WANNER, G. A study of Rosenbrock-type methods of high order. Institut fur Mathematık and Geometrie, Universitat Innsbruck, Innsbruck, Austria, 1979.
15. LAPIDUS, L., AND SCHIESSER, W.E. *Numerıcal Methods for Dıfferential Systems.* Academic, New York, 1976
16. NAG CENTRAL OFFICE 7 Banbury Road, Oxford, England, *NAG Manual, Mark 7,* 1979.
17. PROTHERO, A., AND ROBINSON, A. On the stabılıty and accuracy of one-step methods for solving stiff systems of ordinary differential equations. *Math Comp. 28* (1974), 145–162.
18. RODRIGUES, A., AND BEIRA, E.C Staged approach of percolation processes. *AIChE J. 25* (1979), 416–423
19. SHAMPINE, L.F. Stıffness and nonstiff differential equation solvers, II. Detecting stiffness wıth Runge-Kutta methods. *ACM Trans. Math. Softw. 3,* 1 (March 1977), 44–53.
20. SHAMPINE, L.F. Implementation of implıcıt formulas for the solution of O.D.E.'s. *SIAM J. Scı. Stat. Comp. 1* (1980), 103–118.
21. SHAMPINE, L.F., AND WATTS, H.A. The art of wrıtıng a Runge-Kutta code, Part I. In *Mathematıcal Software III,* J. R. Rıce (Ed.). Academic, New York, 1977.
22. SHAMPINE, L.F., AND WATTS, H.A. The art of wrıtıng a Runge-Kutta code, II. *Appl. Math. Comp 5* (1979), 93–121.
23. SHAMPINE, L.F., AND WATTS, H.A. DEPAC—Design of a user oriented package of ODE solvers. SAND79-2374, Sandia National Laboratories, Albuquerque, N. Mex., 1980.
24. STOER, J., AND BULIRSCH, R. *Introductıon to Numerıcal Analysıs.* Springer-Verlag, New York, 1980.
25. UEBERHUBER, C.W Implementation of defect correction methods for stiff differential equations. *Comput. 23* (1979), 205–232.
26. VAN DER HOUVEN, P.J. *Construction of Integratıon Formulas for Initial Value Problems* North-Holland, Amsterdam, The Netherlands, 1977.
27. VILLADSEN, J., AND MICHELSEN, M.L. *Solutıon of Dıfferentıal Equation Models by Polynomial Approxımatıon.* Prentıce-Hall, Englewood Cliffs, N.J., 1978.
28. WILLOUGHBY, R.A. *Stıff Dıfferentıal Systems.* Plenum Press, New York, 1974.
29. WOLFBRANDT, A. A study of Rosenbrock processes with respect to order conditions and stiff stabılıty. 77.01R, Dep. Computer Scıence, Univ. of Goteborg, Goteborg, Sweden, 1977.

Receıved November 1980; revised November 1981; accepted February 1982