SAND80-2367J
Unlimited Release

UC-32

IMPLEMENTATION OF ROSENBROCK METHODS

Lawrence F. Shampine

Prepared by Sandia Laboratories, Albuquerque, New Mexico 87185
and Livermore, California 94550 for the United States Department of
Energy under Contract DE-AC04-76DP00789.

Printed November 1980

Sandia National Laboratories

2900-Q(3-80)

DISCLAIMER

Issued by Sandia Laboratories, operated for the United States Department of Energy by Sandia Laboratories.

PAGES 1 to 2

WERE INTENTIONALLY

LEFT BLANK

UC-32
SAND80-2367J

Implementation of Rosenbrock Methods*

L. F. Shampine
Applied Mathematics Research Department
Sandia National Laboratories**
Albuquerque, NM 87185

Abstract

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 such a 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.

Key Words and Phrases: Rosenbrock method, ordinary differential equations (ODEs), stiffness, software for ODEs.

CR Categories: 5.17

---

* This article sponsored by the U. S. Department of Energy under Contract DE-AC04-76DP00789.

** A. U. S. Department of Energy Facility.

Implementation of Rosenbrock Methods

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 implementation. 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

$$y' = f(x,y),$$

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

$$(I - \gamma h f_y(y_o))\, k_i = h f(y_o + \sum_{j=1}^{i-1} \alpha_{ij} k_j) + h f_y(y_o) \sum_{j=1}^{i-1} \gamma_{ij} k_j \qquad i = 1,\ldots,s$$

(1)

$$y_1(x_o + h) = y_o + \sum_{i=1}^{s} C_i k_i \,.$$

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_o at x_o to y_1 at $x_o + h = x_1$.

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 [14] 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. In [23] some progress is reported and a fuller discussion of the issues is given. 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 non-stiff, the integration by DEGRK is nearly as efficient as that by RKF45 [24, 25], an effective code for non-stiff 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 production-quality 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. This appears to be a matter deserving more attention than we give it here. We shall present a practical and cheap approximation of the condition which may be of value for other methods as well.

With the additional information available to Rosenbrock codes, it is possible to devise an exceptionally robust procedure for the selection of the initial step size, at least in the context of a type-insensitive code.

It is extremely difficult to compare codes for the solution of stiff ODEs and this is especially true when comparing codes based on quite different presumptions about the problem class. Some numerical results for DEGRK will be presented and in a few cases corresponding results for a BDF code are given. Some research directions are indicated by the results of this investigation.

2. Getting Partial Derivatives

In the solution of

$$y' = f(x,y),$$

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. In section 7 we describe a software device which may make this more true.

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 [16, 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. Such problems are sufficiently important that there are a number of packages written for this specific class. One such package is that of Edsberg [30, pp. 81-94]. It writes the problems as

$$y' = Ap, \quad y(o) \text{ given} \tag{1a}$$

where A is an M x N matrix with integer entries and p is an N vector with

$$p_j = k_j \prod_{i=1}^{M} y_i^{r_{ji}} \quad . \tag{1b}$$

Here the $r_{ji} \geq 0$ are integers describing the reactions and the $k_j > 0$ are rate constants. This autonomous system has $f_x = o$ and f_y is readily computed from the observation that

$$\frac{\partial p_j}{\partial y_i} = r_{ji} \frac{p_j}{y_i} \quad .$$

This class was a major reason that we began a study of Rosenbrock methods.

Another class of problems which may well be suitable are the linear problems

$$y' = J(x)y + g(x).$$

The Jacobian J(x) 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 [20] which we use as a numerical example in section 13 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.

We are supposing that partial derivatives are not a lot more expensive to evaluate than the function. This is because they must be evaluated at every step with a Rosenbrock method and only infrequently with the BDF. Of course if a Rosenbrock method took sufficiently fewer steps than a BDF method, it could compensate for a more expensive step. Still, it seems that Rosenbrock methods are not likely to be very competitive except in the circumstances we postulate. Approximating partial

derivatives by numerical differentiation generally results in a rather expensive evaluation. Typical schemes for dense Jacobians use N extra function evaluations to form a Jacobian for a system of N equations. Except for N small, this makes a step with a Rosenbrock method much more expensive than a typical step with a BDF. If the Jacobian has a useful structure, such as banded or sparse, it may be much cheaper to form the Jacobian than in the dense case. Even so, it is comparatively expensive except for very narrow bands or very special sparse structure.

We note the successful computations of Kaps and Rentrop [14] using differencing and remark only that all the problems in their test set are small. In DEGRK we chose not to provide an option for numerical differentiation for the reasons just given.

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. It is usually suggested that if the problem arises as

$$\frac{dy}{dx} = f(x,y),\ y(a) \text{ given} \tag{1}$$

one convert this to

$$\begin{aligned} \frac{dy}{dt} &= f(x,y),\ y(a) \text{ given} \\ \frac{dx}{dt} &= 1 \quad ,\ x(a) = a\ . \end{aligned} \tag{2}$$

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 Jacobian of (2) is, in partitioned form,

$$J = \begin{pmatrix} f_y, & f_x \\ o, & o \end{pmatrix} \tag{3}$$

Clearly the eigenvalues 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 set carefully collected groups of linear and non-linear problems. D1 is in the non-linear group only because of the conversion from its original form. 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 non-linear equation.

It is about as easy to implement the form (1) in a Rosenbrock code as the autonomous form when done in the manner of the next section. In many papers it has been considered obvious that one use 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. Efficient Representation

The usual form of the Rosenbrock formulas (1.1) 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 which has been attributed to Wolfbrandt. The resulting general form which we write for non-autonomous equations is

$$E = I - \gamma h\, f_y(x_o, y_o) \tag{1a}$$

$$Ek_i = f(x_o + A_i h,\ y_o + h \sum_{j=1}^{i-1} a_{ij} k_j) + B_i h f_x(x_o, y_o) + \sum_{j=1}^{i-1} C_{ij}\, k_j \qquad i = 1, \ldots, s \tag{1b}$$

(1c) $$y_1 = y_o + h \sum_{i=1}^{s} m_i k_i$$

There is another way to save a significant amount of arithmetic in the formation of E. To actually solve the linear systems (1a, 1b) we scale so that we work with

$$f_y(x_o, y_o) - \frac{1}{\gamma h} I$$

instead of E. 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 [30, 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 (1) 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. (There is another reason we take up in the next section.) Because of the expense of a failed step, the step size selection algorithm is 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.

5. Conditioning

The Rosenbrock methods require 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 [22, 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. The matter merits more attention than we give it here.

By definition $\text{cond}(E) = \|E\| \, \|E^{-1}\|$. In general

$$\rho(M) \leq \|M\|$$

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:

I $\quad |h\lambda| \ll 1,$

II $\quad \mathrm{Re}(\lambda) \leq 0.$

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

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

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

$$\rho(E) \doteq \max_k |h\lambda_k| \geq |h\lambda_j| \gg 1.$$

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

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

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 non-stiff 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 $\|E\|_1$. In general if one solves $Ev = w$ for v, he gets a lower bound for $\|E^{-1}\|$ from

$$\|v\| = \|E^{-1}w\| \le \|E^{-1}\| \, \|w\|$$

on dividing by $\|w\|$. 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 as in section 4 so that

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

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

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

and as above

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

We shall approximate this lower bound by $h\gamma$. For stiff problems, $|h\lambda_N| \ll 1$ so this is a good approximation. Indeed for the chemistry problems of (2.1), the Jacobian is always singular because of conservation laws, so that $\lambda_N = 0$ and this is not an approximation at all.

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

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

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

Finally then

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

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. When solving the problems of the test set [8] it is mostly the case that the matrix does not become extremely ill-conditioned. The most ill-conditioned problem we have noticed was the integration of the Rosenbrock problem in its original variables to approximate steady-state on the interval $[0, 4\text{x}10^8]$, c.f. Hindmarsh and Byrne [16, pp. 147 - 166]. We found lower bounds as large as $6.2\text{x}10^{11}$. Another fairly ill-conditioned problem was that of Bui [2] on the interval $[0, 5]$ for which a bound of $4.1\text{x}10^3$ was observed. To provide some quantitative comparison, we solved both of these problems at the two pure absolute error tolerances 10^{-2} and 10^{-4}. Whenever the LINPACK condition estimate COND $\geq 10^3$, we computed the ratio $h\gamma\|f_y\|/\text{COND}$. The Rosenbrock problem has a singular Jacobian so we expect our assumptions to be well satisfied. The agreement with the lower bound of LINPACK is remarkable. The lower bounds always agreed to at least 3 digits. The Jacobian of Bui's problem is not singular and the estimated lower bounds differed more. At the

tolerance of 10^{-2} the ratio ranged from 0.2 to 1.4. At the tolerance of 10^{-4} the ratio ranged from 1.1 to 1.2. 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.

Now that we have a cheap, useful condition indicator, what do we do with it? The trouble is that a large condition number alerts us to possible difficulties, but it does not provide very precise information. This matter is discussed by the LINPACK project in [7, p. I.9]. A rule of thumb is suggested there that if the computer word has about t decimal digits and if the condition is 10^k, then the answers are accurate to no more than t-k digits. Even if this were so, what would be the appropriate action? We could reduce the step size to reduce the conditioning, but this makes the integration correspondingly more expensive. We could resort to residual correction. This requires the storage of the Jacobian, which we do not do in DEGRK, and significantly increases the number of linear systems to be solved. We could turn to another version of the code in a higher precision. This is considerably more expensive than residual correction because then all computations are done in multiple precision rather than the relatively small proportion needed to compute the residual. Unfortunately, on many machines ODEs are normally solved in the highest precision provided by the hardware, and the residual computation has to be done by software multiple precision. At the very least this causes portability problems, and it may be comparatively expensive.

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.

6. 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 [24, 25], we have written for non-stiff 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 [24, 25] 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 [14] 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 5 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 5 of the 9 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 [14] and a related pair in the text [27].

We have not used the pairs selected by Kaps and Rentrop for two main reasons which are amplified in other sections. In the section on stability we go into the matter more fully, but here we just observe that the fourth order formulas they selected are just barely stable at infinity. In the GRK4T pair and the pair in [27], 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 12, 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. In section 10 we take up the reason why it is important that a method for stiff problems evaluate the differential equation throughout the step. The Kaps-Rentrop choices do not satisfy the criterion so we considered other choices. Within the family of Proposition (3.20) there is just one possibility which satisfies the design criterion of section 10. 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 in the efficient form (4.1) is

$$y' = f(x,y)$$

$$E = I - \tfrac{1}{2} h\, f_y(x_o,y_o)$$

$$Ek_1 = f(x_o,y_o) + \tfrac{1}{2} h\, f_x(x_o,y_o)$$

$$Ek_2 = f(x_o + h,\ y_o + hk_1) - \tfrac{3}{2} h\, f_x(x_o,y_o) - 4k_1$$

$$Ek_3 = f(x_o + \tfrac{3}{5} h,\ y_o + \tfrac{24}{25} hk_1 + \tfrac{3}{25} hk_2) + \tfrac{121}{50} h\, f_x(x_o,y_o) + \tfrac{186}{25} k_1 + \tfrac{6}{5} k_2$$

$$Ek_4 = f(x_o + \tfrac{3}{5} h,\ y_o + \tfrac{24}{25} hk_1 + \tfrac{3}{25} hk_2) + \tfrac{29}{250} h\, f_x(x_o,y_o) - \tfrac{56}{125} k_1 - \tfrac{27}{125} k_2 - \tfrac{1}{5} k_3$$

$$y_3(x_o + h) = y_o + h(\tfrac{97}{108} k_1 + \tfrac{11}{72} k_2 + \tfrac{25}{216} k_3)$$

$$y_4(x_o + h) = y_o + h(\tfrac{19}{18} k_1 + \tfrac{1}{4} k_2 + \tfrac{25}{216} k_3 + \tfrac{125}{216} k_4)$$

$$y_4(x_o + h) - y_3(x_o + h) = h(\tfrac{17}{108} k_1 + \tfrac{7}{72} k_2 + \tfrac{125}{216} k_4)$$

7. Software Interface

Recently the author and H. A. Watts [26] presented a design for a software interface to a package of ODE solvers called DEPAC. At this time the package contains 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 this package. In this way it 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 shall mention other minor matters elsewhere, but there is one important difference we take up here.

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. Consider the examples of section 2 where one sees that he 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 Non-Stiff?

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 non-stiff 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 non-stiff or stiff. This 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 [21]. Briefly, if $h_{acc} \gg h_{stable}$, 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) disc of radius ρ. (Van der Houven calls ρ the generalized stability boundary [12, p. 83].) If λ is any eigenvalue of the Jacobian f_y with $Re(\lambda) \leq 0$ 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 for a matrix $M=(M_{ij})$ is

$$\|M\|_1 = \max_j \sum_i |M_{ij}| .$$

This is a simple, cheap computation. Both the Fehlberg (4,5) formulas are stable if we require

$$h\|f_y\| \leq 2.4 . \tag{1}$$

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 (see section 9) 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 [24,25]. Unlike RKF45, DEGRK does not use the "stretching" device, but it does use a "look-ahead." As described in section 5, the step size might be reduced to improve the conditioning of the matrix E in (4.1). 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 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 (1), 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 (1) holds. If the step size had to be more than halved for this reason, we switch to the Rosenbrock method. Our hypothesis when switching to the Rosenbrock method is that the step size is being held back pretty significantly because of stability and it, rather than accuracy, is probably the dominant consideration. We expect, then, that the Rosenbrock method will succeed at this step size which is half (or less) of what will work for the Fehlberg method.

It is not very likely that a problem would call for a step size h such $h\|f_y\| \doteq \rho$ 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 non-stiff 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

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

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 5 steps. With the Rosenbrock scheme, this saves a number of matrix norm computations. With the Fehlberg scheme, this saves a good many unnecessary Jacobian evaluations. If the problem is

unequivocally non-stiff, 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 non-stiff problem by only 5%. 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 non-stiff 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. When solved with DEGRK at a pure absolute error tolerance of 10^{-2} the problem is marginal. Four of the 6 (!) steps needed to solve the problem were taken with the F(4,5) pair. The maximum value of $h\|f_y\|$ encountered was about 4. The Jacobian was evaluated at every step because this is a borderline problem. At the tolerance 10^{-4} all 10 steps were taken with the F(4,5). The maximum value of $h\|f_y\|$ was about 2 and the Jacobian had to be evaluated at 7 of the 10 steps. At the tolerance

10^{-6} all 19 steps were taken with the F(4,5) pair. The maximum value of $h\|f_y\|$ was again about 2 and the Jacobian had to be evaluated at 7 of the 19 steps. 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% we expect in a clear-cut case.

For the sake of variety we shall report some results with the Kaps-Rentrop pair GRK4A advanced with the third order formula. The code is DEGRK with the pair given in section 6 replaced by GRK4A. The B family of problems in the test set [8] are linear with non-real 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 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. We solved B4 and B5 at the pure absolute tolerances 10^{-2}, 10^{-4}, 10^{-6} and measured the central processor time for the solution of each problem. When we forced the code to solve B4 without using the Fehlberg formulas, it cost 1.489 time units to do the integration at all three tolerances. With the Fehlberg formulas, this fell to 0.527. The corresponding figures for B5 are 4.366 and 0.950 respectively. When the Fehlberg formulas were used, more function evaluations were made, e.g., 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 [17] given analytical Jacobian and the same tolerances. All the numerical results obtained were of accuracy comparable to DEGRK. The cost of solving B4 was 1.301 time units and of solving B5, 18.689 units.

Lest the reader think that the results reported for B4 and B5 be somehow due solely to the oscillatory nature of the solutions and the non-real eigenvalues, we mention similar results for the A family of linear problems with real eigenvalues. When solving A3 with the pure absolute error tolerances 10^{-2}, 10^{-4}, 10^{-6} the total cost was 1.141 time units if the F(4,5) formulas were not used in the Rosenbrock code and 0.831 if they were. The BDF code required 1.025 units. When solving A4 the cost was 1.509 if the F(4,5) formulas were not used in the Rosenbrock code and 1.104 if they were. The BDF code required 2.791 units.

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 non-stiff problems will be analyzed in section 13.

9. Initial Step Size

The initial step size is a critical one because it determines whether whether the code "sees" the scale of the problem. The algorithms for estimating local error and adjustment of step size do well providing that only small adjustments are needed at each step.

We have long felt it important that the code select the initial step size automatically. This is obviously a convenience for the user. A suitable initial step size depends on the formula and the problem so that it is not easy for a user to obtain the information needed to make a good selection, even if he knew how. It is common that users solve a family of problems. Experience with an initial step size applied to one member of a family may provide valuable information about the integration of another. For this reason and because even the most careful automatic procedure can break down, we did provide the user a way to supply a guess in DEPAC. This is done by limiting the first step so that it does not go past the first output point.

In DEGRK we insist that the first step be taken with the explicit Runge-Kutta method F(4,5). We reduce the step size as needed so that both of the formulas of this pair are stable. This is an effective device for assuring ourselves that we shall "see" how fast the solution can change at the initial point. It is accomplished by evaluating the Jacobian at the initial point and insisting that the step size h satisfy

$$\|hf_y\|_1 \leq 2.4 \tag{1}$$

As explained in section 8, this implies the step size h is stable for both F(4,5) formulas. We note that (1) may be much more stringent than necessary. This if fine because we are mainly interested in a step size which is small enough that we can trust the start of the integration.

Having evaluated f, f_y, f_x, at the initial point, we are in a position to take a "virtual" step with a Taylor series T(1,2) pair:

$$y_1 = y_o + hy_o' = y_o + hf(a,y_o)$$

$$y_1 = y_o + hy_o' + \frac{h^2}{2} y_o'' = y_o + hf(a,y_o) + \frac{h^2}{2}[f_x(a,y_o) + f_y(a,y_o)\, f(a,y_o)]$$

As usual, the error control makes this an annoyingly complicated matter. The error estimated in solution component i is

$$|est_i| = \left|\frac{h^2}{2}[f_x(a,y_o) + f_y(a,y_o)\, f(a,y_o)]_i\right| \tag{2}$$

DEGRK allows two error control parameters $rtol_i$ and $atol_i$ to be specified for a mixed relative-absolute test on each solution component. The matter is handled a little differently in the start than at a general step. For the first step we try to take the error relative to the solution at the beginning of the step:

$$|est_i| \leq atol_i + rtol_i |y_{o,i}| = wt_i \tag{3}$$

In the normal case $wt_i > 0$. The weight wt_i is fixed, as is all of est_i except for the factor h^2. Thus we can immediately deduce the largest h such that (3) holds. In the usual error control of the code, the average magnitude of the solution at the two ends of the step is used. This protects against a solution component vanishing "accidentally," but it is inconvenient for the first step. In particular, the weight then depends on the step size h and selection of h is no longer so simple.

It is all too common that a user ask for pure relative error, $atol_i = 0$, even though the solution component $y_{o,i} = 0$ at the initial point. Of course then $wt_i = 0$ in (3). In such a case we take the error relative to the solution at the end of the step:

$$wt_i = rtol_i \, |hf(a,y_o)|$$

Again we can see the largest h such that $|est_i| \leq wt_i$, but notice that the order is reduced in this situation.

It can happen that the solution component has a double zero at the initial point, $f(a,y_o)_i = 0$, in which case both choices of weight vanish. The first order Taylor series method produces a numerical solution which is identically zero for such a component so pure relative error control is not possible. We simply say such a component provides no scale information.

Except for the extremely rare case that the user specifies pure relative error for every solution component and every component has (at least) a double zero at the initial point, we find a step size suitable for the T(1,2) pair. We do not give up in the extreme case because it is quite possible we shall be able to integrate it. It is just that this part of the step selection procedure provides no useful step size information. The local error of the first order method behaves like $\sim h^2\phi$ in general and that of the fourth order method like $\sim h^5\psi$. As a heuristic to go from a scale suitable for T(1,2) to a scale suitable to F(4,5) we assume that the error of the fourth order method is equal to that of the first order method raised to the 5/2 power. From this we deduce the largest step size which would apparently succeed with the fourth order method. If it is smaller than the bounds previously stated, we use it.

Finally we increase the step size as necessary so that it be meaningful in the precision being used. Specifically in DEGRK, we insist it be at least as large as 26 units of roundoff in the initial point a.

10. Design Criteria for One-Step Methods

Runge-Kutta and Rosenbrock methods evaluate the differential equation 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. Some computational results brought to the author's attention the fact that the Kaps-Rentrop Rosenbrock formulas do not span the interval. On subsequent investigation it was found that this is common for formulas aimed at stiff problems. Unfortunately it is with stiff problems that trouble is most likely.

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 mockups of photocatalyzed atmospheric reactions (see, e.g. [5]) which are illustrative. The simpler has the form

$$y'(t) = d - by + aE(t). \tag{1}$$

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.

If the method does not evaluate the differential equation at t_{n+1}, it can do an exceedingly poor job of locating a quasi-discontinuity. To expose the trouble let us consider a simple example, the implicit midpoint rule:

$$\text{solve} \qquad y_{n+\frac{1}{2}} = y_n + \frac{h}{2} f(t_n + \frac{h}{2}, y_{n+\frac{1}{2}}),$$

$$\text{advance} \qquad y_{n+1} = y_n + h\, f(t_n + \frac{h}{2}, y_{n+\frac{1}{2}}).$$

Suppose the local error is estimated by doubling. In order to describe simply what is going on, let us consider the problem (1) and speak loosely as though the functions $E(t)$, $y(t)$ were actually discontinuous. We take the time origin at sunset and suppose $y(t)$ has attained its steady state value d/b. Let us try a step size of just less than 8 hours. In the first step we evaluate the differential equation after 4 hours and we find the numerical solution to be d/b. In the second step we evaluate just short of 12 hours and the formula again says that the solution is d/b. In the double step of 16 hours the evaluation is done at 8 hours where the intermediate solution is d/b and so apparently confirms the "more accurate" solution to be d/b. Of course the formula does not "see" the discontinuous change at sunrise. The result is that the location of sunrise has been missed by 4 hours!

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 differential equation is evaluated.

The example of the midpoint rule is not at all contrived. 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 [13] 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 [30, pp. 201-215] has based an extrapolation code on a modified midpoint rule

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

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

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

One 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. It is interesting that Alexander increased the number of stages to get better stability properties. As a consequence of the desired stability properties, he had to evaluate at t_{n+1} and so avoided the defect.

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 [29], and of Kaps and Rentrop [14].

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 differential equation past some point, or its definition changes there. The DEPAC [26] 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 is 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 prolog 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.

11. Adjustment of Step Size

The principles of the adjustment of step size for explicit Runge-Kutta methods are discussed at length in [24,25]. 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 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 non-stiff. It would then move back to the smooth solution at which time it would begin to increase the step size rapidly. We believe this is partly due to the stability properties which we take up in section 12. It would not be particularly inefficient except for another phenomenon. 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 [19] have taken up this matter. Ueberhuber [28] 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

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

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

$$\exp(h\lambda) = R(h\lambda) + \mathcal{O}(|h\lambda|^{p+1}) ,$$

so

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

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

$$\frac{le}{y_n} \doteq - R(h\lambda) .$$

The behavior of R in the neighborhood of infinity must be investigated anyway because of the stability implications as in section 12, but here we are interested in the implications for accuracy. For methods stable at ∞, $|R(\infty)| \leq 1$. Writing

$$R(h\lambda) = c_o + \frac{c_1}{h\lambda} + \frac{c_2}{(h\lambda)^2} + \ldots$$

we see that if the local error is not acceptable, it may require large changes of step size to reduce it significantly. The Rosenbrock methods we implemented all have $c_o \doteq 0.3$. 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. Many common formulas have $c_o = 0$, but none of the popular ones have $c_1 = 0$. Thus this difficulty with the asymptotic behavior is of some generality. It is surprising to many that the local error may well be a <u>decreasing</u> function of h for $|h\lambda| \gg 1$. This illustrates that our understanding of the control of error by adjustment of step size is not complete.

We have responded to the situation in two ways. On a failed step we are pessismistic about the assumed asymptotic behavior. Because of the work involved it is better to attempt a stepsize 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 that 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 non-stiff 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

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

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.

12. 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 proceeding with both formulas. Kaps has told us that in the tests of [14] 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 experience, 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. With the pure absolute error tolerances 10^{-2}, 10^{-4} there was no striking difference. The number of steps gives a fair impression of the relative work. The numbers of steps at the two tolerances were 24, 90 with the third order formula and 28, 114 with the fourth order formula. Although not negligible, the difference does not compare to that observed when pure relative error tolerances of 10^{-2}, 10^{-4} were used. Then the numbers of steps were 158, 769 and 253, 922 respectively. 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 different selection of formula pair in section 6 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).

13. 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 12. 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; ξ = 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 [17]. At tolerance 10^{-2} the respective central processor times were 0.205 and 0.497 units. At tolerance 10^{-4} they were 0.754 and 1.02. At tolerance 10^{-6} they were 4.34 and 1.67. 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 [14].

The parameter choice K = 5, ξ = 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 in this measure increases rapidly as the tolerance becomes more stringent for a non-stiff problem.

The parameter choice K = 5, ξ = 500, N_f = 50 results in the stiffest problem. The maximum value of $h\gamma\|f_y\|$ encountered by DEGRK in the integrations at tolerances 10^{-2}, 10^{-4}, 10^{-6} are respectively, 7353, 4478, 1734. According to the results of section 5 this implies 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. At tolerance 10^{-2}, 2 of the 21 steps were taken with the F(4,5) pair; at tolerance 10^{-4}, 12 of 77; and at tolerance 10^{-6}, 36 of 636. Notice the rapid increase in the number of steps as the tolerance is made more stringent. This is characteristic of a fixed order method. The results suggest the code quite inappropriate at the tolerance 10^{-6}. This is also suggested by the number of rejected steps which were respectively 0, 8, and 111. At the crudest tolerance DEGRK is somewhat competitive even in terms of function and Jacobian evaluations. Then it needed 90 function and 21 Jacobian evaluations whereas the BDF code needed 78 function and 15 Jacobian evaluations. The difference of performance in this measure increases rapidly as the tolerance becomes more stringent for a stiff problem.

It is especially hard to compare codes on difficult problems, but we shall present one example which has its interesting points. Scott and Watts [16, 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 5 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 $h\gamma\|f_y\|$ as an indication of the stiffness. For the initial value 0.99000 this maximum was about 4000 at tolerance 10^{-2} and 7000 at tolerance 10^{-4}. For the initial value 0.99026 this maximum was about 50 at tolerance 10^{-2} and 20 at tolerance 10^{-4}.

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 cost in central processor time at the tolerances 10^{-2}, 10^{-4} were 0.054, 0.221 with the BDF code and 0.148, 0.223 with DEGRK. The result at tolerance 10^{-2} was not so bad

with the BDF code as with the other problem, but one component was off by a factor of more than 3 so the solution was not very helpful. At tolerance 10^{-4} the maximum error with the BDF code was 4.0×10^{-1}. The error at tolerance 10^{-2} was not very good with DEGRK either, 8.6×10^{-1}, although closer in performance to the tolerance 10^{-4} than 10^{-2} with the BDF code. The error at tolerance 10^{-4} with DEGRK was 7.1×10^{-3}.

The kind of results seen on this problem did not surprise 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.

14. A Personal Assessment

In the course of this investigation the author has formed some opinions about production codes based on Rosenbrock methods. A few will be mentioned because they suggest certain lines of development.

The most straightforward improvement to DEGRK would be the development of a Rosenbrock formula with better properties. Specifically, it seems that maximal damping at infinity ($R(\infty) = 0$) and a higher order are needed. Higher order, maximally damped formulas have already been given [15]. Unfortunately they are not accompanied by an error estimator. The general principle of doubling is applicable, but does not strike the author as promising. It involves two matrix factorizations and a considerable number of stages per step. Very recent work [18] suggests that perhaps the approach is practical. The history of explicit Runge-Kutta methods suggests that too much emphasis is being placed on a minimal number of stages. In view of the significant linear algebra costs and the partial derivatives evaluation at every step, it appears better to aim for fewer steps with more stages.

In some circumstances the Rosenbrock methods appear to enjoy an advantage with respect to the BDF in terms of overhead. The author suspects that this is partly due to the different assumptions made about the problem class addressed rather than being intrinsic. The Rosenbrock methods are being implemented as fixed order codes while the BDF are usually implemented as variable order codes. The distinction has important implications independent of the underlying methods. The situation is analogous to the relative merits of explicit Runge-Kutta and variable order Adams methods.

We have seen that a crude, but useful, way to recognize and respond to stiffness automatically is possible. The author believes that other techniques he is currently developing will prove at least as effective for the BDF.

The Rosenbrock methods handle gracefully stiff problems with Jacobians that change pretty often. It is not clear at this time the practical significance of this difference. Part of the difficulty is that there is not enough information available about "typical" problems. Just how constant are the Jacobians? Do we focus our attention on problems with nearly constant Jacobians because our theoretical understanding of them is better, or are they truly representative? Another difference difficult to evaluate is the different role of ill-conditioning in the linear systems to be solved. This appears a worrisome matter for accurate integrations with high order Rosenbrock methods, but it is possible that severe ill-conditioning is not common or, for some reason not taken up in this paper, does not greatly affect the results.

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(1979), 483-493.

3. Bui, T. D. and Bui, T. R. Numerical methods for extremely stiff systems of ordinary differential equations. Appl. Math. Modelling 3 (1979), 355-358.

4. Butcher, J. C. A transformed implicit Runge-Kutta method. J. ACM 26 (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, NJ, 1971.

10. Hall, G. and Watt, J. M. Modern Numerical Methods for Ordinary Differential Equations. Clarendon Press, Oxford, 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. Inst. for Numer. Anal., Tech. Univ. of Denmark, Lyngby, Denmark, 1979.

12. van der Houven, P. J. Construction of Integration Formulas for Initial Value Problems. North-Holland, Amsterdam, 1977.

13. Hulme, B. L. and Daniel, S. L. COLODE: a colocation subroutine for ordinary differential equations. SAND74-0380. Sandia National Laboratories, Albuquerque, NM, 1974.

14. 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.

15. Kaps, P. and Wanner, G. A study of Rosenbrock-type methods of high order. Inst. für Math. and Geometrie, Universität Innsbruck, 1979.

16. Lapidus, L. and Schiesser, W.E. Numerical Methods for Differential Systems. Academic, New York, 1976.

17. NAG Manual, Mark 7. NAG Central Office, 7 Banbury Road, Oxford, England, 1979.

18. Poon, S., Kaps, P., and Bui, T. D. A Comparison of ROW methods for stiff O.D.E.'s using Richardson extrapolation and embedding techniques for stepsize control. Dept. Comp. Sci., Concordia Univ., Montreal, 1980.

19. Prothero, A. and Robinson, A. On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations. Math. Comp. 28 (1974), 145-162.

20. Rodrigues, A. and Beira, E. C. Staged approach of percolation processes. AIChE J. 25 (1979), 416-423.

21. Shampine, L. F. Stiffness and nonstiff differential equation solvers, II: detecting stiffness with Runge-Kutta methods. ACM Trans. Math. Software 3 (1977), 44-53.

22. Shampine, L. F. Implementation of implicit formulas for the solution of O.D.E.'s. SIAM J. Sci. Stat. Comp. 1 (1980), 103-118.

23. Shampine, L. F. Type-insensitive ODE codes based on implicit A-stable methods. Math. Comp., to appear.

24. Shampine, L. F. and Watts, H. A. The art of writing a Runge-Kutta code, Part I, in J. R. Rice, ed. Mathematical Software III. Academic, New York, 1977.

25. Shampine, L. F. and Watts, H. A. The art of writing a Runge-Kutta code, II. Appl. Math. Comp. 5 (1979), 93-121.

26. Shampine, L. F. and Watts, H. A. DEPAC - design of a user oriented package of ODE solvers. SAND79-2374. Sandia National Laboratories, Albuquerque, NM, 1980.

27. Stoer, J. and Bulirsch, R. Introduction to Numerical Analysis. Springer, New York, 1980.

28. Ueberhuber, C. W. Implementation of defect correction methods for stiff differential equations. Computing 23 (1979), 205-232.

29. Villadsen, J. and Michelsen, M. L. Solution of Differential Equation Models by Polynomial Approximation. Prentice-Hall, Englewood Cliffs, NJ, 1978.

30. Willoughby, R. A. Stiff Differential Systems. Plenum Press, New York, 1974.

Distribution:

Prof. R. Alexander
Rensselaer Polytechnic Institute
Mathematical Sciences Dept.
Troy, NY 12181

Prof. D. G. Bettis
TICOM
University of Texas at Austin
Austin, TX 78712

Prof. T. A. Bickart
Elec. and Comp. Engr. Dept.
Link Hall
Syracuse University
Syracuse, NY 13210

Dr. T. D. Bui
Dept. of Computer Science
1455 de Maisonneuve Blvd. West
Montreal, Quebec H3G 1M8
CANADA

Prof. Dr. R. Bulirsch
Mathematisches Institut der
Technischen Universität
8 München 2
Arcisstrasse 21
Postfach 20 2420
West Germany (BRD)

Prof. J. C. Butcher
Dept. of Computer Science
University of Auckland
Private Bag
Auckland
NEW ZEALAND

Dr. G. D. Byrne
Computer Technology and Services Div.
Exxon Research and Engineering Co.
P.O. Box 45
Linden, NJ 07036

Prof. F. Chipman
Department of Mathematics
Acadia University
Wolfville, Nova Scotia
CANADA B0P 1X0

Dr. Alan Curtiss
AERE Harwell
Computer Science and Systems Div.
Didcot, Berkshire OX 11 ORA
ENGLAND

Prof. G. Dahlquist
Department of Numerical Analysis
Royal Institute of Technology
S-10044 Stockholm
SWEDEN

Prof. Dr. P. Deuflhard
Universität Heidelberg
Institut für Angewandte Mathematik
Im Neuenheimer Feld 294
6900 Heidelberg 1
West Germany

J. R. Dormand
Dept. of Mathematics and Statistics
Teesside Polytechnic
Middlesbrough Cleveland TS1 3BA
England

Dr. R. England
I.I.M.A.S.
Universidad Nacional
Autónoma de México
Apartado Postal 20-726
Mexico 20, D. F.
MEXICO

Dr. W. H. Enright
Department of Computer Science
University of Toronto
Toronto, Ontario
CANADA M5S 1A7

Prof. C. W. Gear
Department of Computer Science
University of Illinois at U-C
Urbana, IL 61801

Dr. I. Gladwell
Department of Mathematics
University of Manchester
Manchester M13 9PL
ENGLAND

Prof. G. K. Gupta
Division of Computer Applications
Asian Institute of Technology
P.O. Box 2754
BANGKOK, THAILAND

Dr. George Hall
Department of Mathematics
University of Manchester
Manchester M13 9PL
ENGLAND

Dr. J. P. Hennart
I.I.M.A.S.
Universidad Nacional
Autónoma de México
Apartado Postal 20-726
Mexico 20, D.F.
MEXICO

Dr. A. C. Hindmarsh
L-310
Lawrence Livermore Laboratory
Livermore, CA 94550

Prof. P. J. Van Der Houwen
Stichting Mathematisch Centrum
Ze Boerhaavestraat 49
Amsterdam 1005
The Netherlands

Prof. T. E. Hull
Department of Computer Science
University of Toronto
Toronto, Ontario
CANADA M5S 1A7

Dr. K. R. Jackson
Dept. of Computer Science
Yale University
New Haven, CT 06520

Dr. R. Jeltsch
Mathematics Department
Ruhr-Universität Bochum
4630 Bochum 1
Postfach 102148
West Germany (BRD)

Dr. P. Kaps
Institut für Mathematik I
Universität Innsbruck
Technikerstr. 13
A-6020 Innsbruck
AUSTRIA

Dr. R. W. Klopfenstein
RCA Laboratories
Princeton, NJ 08540

Dr. F. T. Krogh
125/128
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91103

Prof. J. D. Lambert
Department of Mathematics
University of Dundee
Dundee DD1 4HN
SCOTLAND

Dr. B. Lindberg
Dept. of Computer Science
Royal Institute of Technology
Stockholm 70
SWEDEN

Dr. W. Liniger
IBM T. J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Dr. sc. R. März
Bereich Num. Mathematik
Sektion Mathematik
Humboldt-Universität zu Berlin
DDR 1086 Berlin, PSF 1297
East Germany (DDR)

Dr. O. Nevanlinna
Department of Mathematics
Faculty of Technology
University of Oulu
SF-90570 OULU 57
FINLAND

Dr. S. P. Nørsett
Universitet I Trondheim
Norges Tekniske Høgskole
Institutt for Numerisk Matematikk
N 7034 Trondheim
NORWAY

Dr. A. Prothero
Thornton Research Centre
P.O. Box No. 1
Chester CHI 3 SH
ENGLAND

Dr. P. Rentrop
Institut für Mathematik
der Technischen Universität
D-8 München 2
Arcisstrasse 21
Postfach 20 24 20
West Germany (BRD)

Prof. J. Rosenbaum
Virginia Commonwealth University
Mathematical Sciences Dept.
Richmond, VA 23284

Dr. R. Sacks-Davis
Dept. of Computer Science
Monash University
Clayton, Victoria 3168
AUSTRALIA

Dr. A. E. Sedgwick
Dept. of Mathematics
Dalhousie University
Halifax, NS B3H 4H8
CANADA

Dr. P. Seifert
Technischen Universität Dresden
Sektion Mathematik
Wissenschaftsbereich
Numerische Mathematik
DDR-8027 Dresden, Mommsenstr. 13
East Germany (DDR)

Prof. Dr. H. J. Stetter
Institut fur Numerische Mathematik
Technische Hochschule Wien
A-1040 Wien, Gusshausstr. 27-29
AUSTRIA

Dr. P. G. Thomsen
Inst. for Numerical Analysis
Technical University of Denmark
DK 2800 Lyngby
DENMARK

C. W. Ueberhuber
Institut für Numerische Mathematik
der Technischen Universität
Gusshausstr. 27
A-1040 Wien
AUSTRIA

Dr. M. van Veldhuizen
Wiskundig Semarium
Vrije Universitett
de Boelelaan 1011
Amsterdam
the Netherlands

Dr. J. H. Verner
Dept. of Mathematics
Queen's University
Kingston K7L 3N6
CANADA

J. G. Verwer
Stichting Mathematisch Centrum
P.O. Box 4079 1009AB
Amsterdam
the Netherlands

Dr. G. Wanner
Dept. de Mathematique
Universite de Geneve
2-4 Rue du Lievre
CH-1211 Geneve 24
SWITZERLAND

Dept. of Computer Sciences
Report Section
Royal Institute of Technology
S-100 44
Stockholm 70, Sweden

2646 M. R. Scott
2646 H. A. Watts
5600 D. B. Shuster
Attn: 5610 A. A. Lieber
5620 M. M. Newsom
5630 R. C. Maydew
5640 G. J. Simmons
5641 R. J. Thompson
5642 L. F. Shampine (50)
8266 E. A. Aas
8332 R. E. Huddleston
8332 L. Petzold
3141 T. L. Werner (5)
3151 W. L. Garner (3)
For DOE/TIC (Unlim. Release)
3154-3 R. P. Campbell (25)
For DOE/TIC

Org.	Bldg.	Name	Rec'd by *	Org.	Bldg.	Name	Rec'd by

* Recipient must initial on classified documents.