MATHEMATICS OF COMPUTATION
VOLUME 39, NUMBER 159
JULY 1982, PAGES 109–123

Type-Insensitive ODE Codes Based on Implicit $A(\alpha)$-Stable Formulas*

By L. F. Shampine

Abstract. Previous work on A-stable formulas is extended to $A(\alpha)$-stable formulas, which are far more important in practice. Some important improvements in technique based on another interation method and an idea of Enright for the efficient handling of Jacobians are proposed. Implementation details and numerical examples are provided for a research-grade code.

1. Introduction. This paper is a sequel to the paper [10] which treats A-stable formulas. Although largely independent, it cannot be properly understood without the companion paper. For this reason and to avoid repetition of some lengthy arguments, it is presumed that the reader has studied [10].

Many of the most popular formulas for the solution of stiff ODE problems are not A-stable so that extending the analysis of [10] is of great practical value. The backward differentiation formulas (BDF) of orders 3–6 are examples which are $A(\alpha)$-stable, and may be thought of as prototypes for the formulas treated here.

The task at hand is qualitatively different from that of [10]. There stiffness amounted to selecting an efficient iteration method for the evaluation of implicit formulas. Stability played no obvious role there, but here the behavior of the integration can be affected by stability.

Part of the results reported here are equally relevant to A-stable formulas. We describe improvements of technique based upon an idea of Enright and still another iteration method. Some of the practical details not fully specified in [10] will be developed. Some numerical results with A-stable formulas will be presented.

The changes to [10] to accomodate $A(\alpha)$-stable formulas are more conceptual than practical. We shall first discuss the practical significance of formulas which are not A-stable. Then we shall investigate those factors affecting step size and iteration method when a stability restriction is possible. It develops that whatever practical difficulty which might arise is due to the formula and is not aggravated by our switching iteration methods. The inherent trouble with $A(\alpha)$-stable formulas must be taken up in our investigation, and as a consequence we propose ways to ameliorate it. Some numerical results will be given.

2. How Troublesome Are $A(\alpha)$-Stable Formulas?. Our aim is to select an efficient scheme for the evaluation of implicit formulas. It seems unlikely that one would base

---

Received April 2, 1981; revised November 20, 1981.
1980 *Mathematics Subject Classification.* Primary 65L05.
Key words and phrases. ODE codes, stiffness, $A(\alpha)$-stable.
*This work performed at Sandia National Laboratories supported by the U. S. Department of Energy under contract number DE-AC04-76DP00789.

0025-5718/82/0000-0077/$04.50

a general purpose code intended to handle stiff problems on formulas which were not $A(\alpha)$-stable. From here on we suppose the formula $A(\alpha)$-stable. When one chooses to work with an $A(\alpha)$-stable formula, one has, in effect, chosen to restrict the domain of applicability of the code to those problems such that the Jacobian, evaluated anywhere along the solution, has eigenvalues only in the appropriate sector of the complex plane. Should some Jacobian have eigenvalues too close to the imaginary axis, a poor performance is expected (and often observed).

The behavior of an $A(\alpha)$-stable formula on a problem for which it is intended is the same as that of an A-stable formula applied to the same problem. This obvious remark shows us that the considerations of [10] are unaltered for such problems. In this sense there is no real difficulty with the application of our ideas to $A(\alpha)$-stable formulas.

As mentioned, it is expected that the typical code based on an $A(\alpha)$-stable formula will behave badly when it actually suffers a restriction on the step size due to stability. The comparative testing of [2] shows what can happen with the BDF. The problem B5 leads to a stability restriction for the BDF of orders greater than 4. (It is hoped in the popular variable order BDF codes that the restriction will be avoided by resorting to the more stable lower order formulas. Unfortunately this does not always happen [13].) In the present context we are only responsible for seeing that our schemes for selecting the iteration method do not make the trouble worse. The situation has seen very little attention. Because of the close connection with the present study, we shall point out improvements of the technique employed in current codes in this situation.

3. Efficient Use of Jacobians. In [8] we promoted saving Jacobians for reuse in the simplified Newton iterations. W. H. Enright [1] has proposed a scheme which offers important advantages in the present context. We shall point out here some arguments in its favor not stated by Enright and also a convenient software device for its efficient use.

We are solving the system

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

The implicit formula has the generic form

$$y = h\gamma f(y) + \psi, \tag{1}$$

where y is the solution value for the step to x_{n+1}, γ is a constant characteristic of the formula, ψ lumps together previously computed quantities, and the independent variable x is suppressed. The direction of integration is chosen so that $h > 0$. The iteration schemes considered in [10] are

$$(I - h\gamma J)(y^{m+1} - y^m) = \psi + h\gamma f(y^m) - y^m. \tag{2}$$

Here $J = 0$ is simple iteration, and J, a nontrivial approximation to the Jacobian matrix f_y, is a simplified Newton iteration.

First we describe Enright's basic idea. The simplified Newton iteration must repeatedly solve linear systems with the iteration matrix $I - h\gamma J$. What is going on is seen more clearly if the linear system is scaled to result in the matrix $J - (1/h\gamma)I$. (We recommend this as a cheaper and more natural scaling for stiff problems

anyway.) One does a similarity reduction to the Hessenberg form of the iteration matrix,

$$J - \frac{1}{h\gamma} I = LHL^{-1}.$$

Here L is unit lower triangular and H is upper Hessenberg. (For notational simplicity, here and elsewhere we leave out permutation matrices resulting from pivoting.) One can solve the necessary linear systems with this factored matrix instead of the usual LU factorization of lower and upper triangular matrices. The advantage comes when one wants to change step size or method so that $h\gamma \to h'\gamma'$. Then

$$J - \frac{1}{h'\gamma'} I = L\left(H + \left(\frac{1}{h\gamma} - \frac{1}{h'\gamma'} \right) I \right) L^{-1}.$$

Thus the factorization can be changed trivially to correspond to the new $h'\gamma'$.

The Hessenberg factorization requires an unimportant amount of additional storage. It does require some extra work to solve the linear systems which Enright proposes to reduce by further factoring $H = LU$ into bidiagonal L and upper triangular U. The additional factorization facilitates solving the linear systems repetitively, but, on a change of $h\gamma$, one must reconstruct H by multiplying out LU, alter its diagonal, and refactor it.

Our colleague H. A. Watts raised the question to us as to whether it was better to pay the storage cost of keeping a copy of H and so avoid the expense and inaccuracy of frequently reconstructing H. This depends on the user's requirements, but we noted a very convenient way to handle the matter within the DEPAC software interface design [12]. In this design the user provides to the code a working array and the length of the array. He is instructed as to the minimum length of the array which will suffice for the solution of the problem. Here this length is that corresponding to reconstruction of H. The instructions could then add that if the user can provide a specific larger amount of storage, the code will run somewhat faster. This will not surprise him. The code itself simply checks the amount of storage provided and uses the appropriate technique internally. The user is never bothered about what is being done inside the code. Storing the Hessenberg matrix should often lead to a perfectly acceptable amount of total storage and lead to a noticeable improvement in speed.

The advantage of Enright's idea is that the frequent change of step size is not accompanied by a corresponding expensive factorization of the iteration matrix. He did not point it out, but the device alone goes a long way towards making a code type-insensitive. If the code is applied to a nonstiff problem, the Jacobian is unimportant in the iteration matrix—$J = 0$ (simple iteration) would suffice. Any reasonable algorithm for deciding when to form a new J should lead to relatively few Jacobian evaluations when the problem is not stiff and of course, correspondingly few matrix factorizations. Another important situation in which Enright's scheme is very useful is when the step size is being restrained for reasons of stability. We shall amplify this later after describing what is happening to the step size then.

In what follows we shall assume Enright's idea is used. Unfortunately, the Hessenberg reduction does not preserve structure in the Jacobian. When the Jacobian

is highly structured, a copy should be kept and an LU factorization used. The cheaper linear algebra and the greatly reduced storage then compensate for the extra matrix factorizations.

4. Effects of a Stability Restriction. What happens when a method with a finite stability region is presented with a stiff problem? Some arguments can be found in [5], [11] but the basic idea is simple and plausible: If the step size corresponds to being outside the stability region, propagated error is amplified until the local error estimator detects it and reduces the step size. If the step size corresponds to being inside the stability region, propagated error is damped until the error estimator realizes that the step size can be increased. The consequence is that the average step size corresponds to being on the stability boundary.

As in [10] let h_{acc} be the largest step size which would permit the local accuracy requirement to be satisfied. The difficulty for us is that most of the time the numerical solution contains a substantial component of propagated error. It looks "rough", and the estimate h_{est} of h_{acc} may be quite a lot smaller than h_{acc}, which corresponds to the smooth solution being tracked. If the code were to hold the step size constant long enough while in the stability region, a good estimate of h_{acc} could be obtained. The situation is exaggerated in our research code based on the θ-family of one-step formulas. Past solution values *are* used in forming h_{est}, but the memory is very short and erratic values of h_{est} are observed.

The idea of Enright sketched in Section 3 appears to be very helpful when the step size is being restrained by stability. The step size is continually changing, but there is no need for a new Jacobian. Unfortunately, the matter is a little tricky. Going outside the stability region may be accompanied by a convergence failure rather than a rejected step, particularly when using the stringent acceptance criterion of [6], [15]. The typical algorithm in present codes regards this as a signal to form a new Jacobian. In the algorithm described in Section 7 we have attempted to avoid these unnecessary Jacobian evaluations. It is to be appreciated that this difficulty does not arise from our ideas about switching iteration methods.

5. A More Efficient Iteration Method. When solving (1) by simple iteration, the rate of convergence is determined by $\| h\gamma f_y \|$. In [10] we proposed using the weighted maximum norm of the approximate Jacobian J to estimate this. To avoid notational complication, we shall write down only the unweighted norm:

$$\| h\gamma J \| = \max_i | h\gamma | \sum_j | J_{ij} | .$$

Simple iteration was used whenever

$$\| h\gamma J \| \leq r < 1 \tag{3}$$

for some r deemed an acceptable rate of convergence. As we remarked at the time, it seemed a shame to switch from Newton to simple iteration and make no further use of J.

The Jacobi iteration for solving (1) is

$$(I - h\gamma D)(y^{m+1} - y^m) = \psi + h\gamma f(y^m) - y^m,$$

where $D = \operatorname{diag}\{J_{ii}\}$. The equivalent of (3) is now

$$\|(I - h\gamma D)^{-1} h\gamma (J - D)\| = \max_i \frac{|h\gamma| \sum_{j \neq i} |J_{ij}|}{|1 - h\gamma J_{ii}|} \leqslant r < 1. \tag{4}$$

It is just as easy to use the Jacobi iteration as simple iteration. According to the criterion used, it is at least as effective. To see this, suppose (3) holds. Then the Jacobi iteration is well defined because

$$1 > r \geqslant \|h\gamma J\| \geqslant |h\gamma J_{ii}| \quad \text{each } i$$

hence

$$1 - h\gamma J_{ii} > 0 \quad \text{each } i.$$

A simple computation shows that if (3) holds, then (4) must also hold.

The Jacobi iteration can be extremely advantageous. If a diagonal element $J_{ii} \leqslant 0$ and it dominates the off-diagonal elements sufficiently,

$$\sum_{j \neq i} |J_{ij}| \leqslant r |J_{ii}|,$$

the row puts no restriction on the step size to achieve the rate of convergence r in (4). Clearly, arbitrarily stiff problems can be solved with the Jacobi iteration provided that they have the right Jacobian structure. We do not know how important this is in practice. A very popular method of analysis in chemical kinetics makes use of a quasi steady state hypothesis. A solution component $y_i(x)$ satisfies an equation of the form

$$y_i' = P_i(y_1, \ldots, y_n) - Q_i(y_1, \ldots, y_n) y_i.$$

It is assumed that both P_i and Q_i are roughly constant and that $Q_i \gg 1$. Considering the corresponding row of the Jacobian of the system, we see that such an equation places no restriction on the step size which might be used with the Jacobi iteration.

We propose using simple iteration until the first Jacobian is formed and thereafter using the Jacobi iteration. Every time a Jacobian J is formed we must save the diagonal in a separate array because we plan to overwrite J if we ever do a matrix factorization involving it. In our research code we aim for a rate $r = 0.5$. When the Jacobian is formed, we determine the largest step size, h_{itmax}, which according to (4) would yield this rate of convergence. At the same time we compute $\|J\|$ which, through monitoring of $h\|J\|$, gives us a measure of the stiffness of the problem.

After forming a Jacobian J, we may switch back and forth between a Jacobi iteration and a simplified Newton iteration using the same J. Our intention is basically to use the Jacobi iteration whenever $h \leqslant h_{\text{itmax}}$. As we discussed at length in [10], the information used to determine h_{itmax} in (4) may become out of date as the Jacobian changes along the solution curve. A useful refinement is to improve the value for h_{itmax} based on whatever observations are available. Whenever we actually use the Jacobi iteration and should need at least two iterations, we obtain an estimate of the rate of convergence,

$$\max_{m \geqslant 0} \frac{\|y^{m+2} - y^{m+1}\|}{\|y^{m+1} - y^m\|}.$$

The rate of convergence is essentially linear in the step size, so we can estimate the largest step size which would yield convergence at a rate $r = 0.5$. If this estimate is smaller than the current h_{itmax}, we reduce h_{itmax} to this value. If it is larger, we take no action. This is because we are certain the rate of contraction is no faster than the observed rate, but we cannot exclude the possibility that the true rate is slower.

6. Interaction of Iteration Method and Stability. In this section we identify the tasks our algorithm for selecting the iteration method should address. The step size h_{acc} is the largest step size which would yield the desired local accuracy. There are two potential restrictions on this step size: The step size h_{iter} is the largest for which the iteration contracts in the norm of the code. The step size h_{stable} is the smallest step size h for which $h\lambda_i$ is on the boundary of the stability region for some eigenvalue λ_i of f_y with $\text{Re}(\lambda_i) \leq 0$.

When we considered A-stable methods in [10], the restriction due to the iteration method was the only one present because h_{stable} is infinite by definition. The restriction can be quite severe. When simple iteration is used, h_{iter} is defined by $h_{\text{iter}}\gamma\| f_y\| = 1$. Because $\| f_y\| \geq |\lambda_i|$ for all eigenvalues λ_i of f_y, the step size must satisfy $|h_{\text{iter}}\lambda_i| \leq 1/\gamma$. This is exactly the restriction imposed by stability for a method with an absolute stability region consisting of the half-disc of radius $1/\gamma$. Stating this necessary condition on h_{iter} in this way gives some insight as to the role of the iteration method. The restriction can be still more severe because the norm of the Jacobian can be much larger than the spectral radius of the Jacobian. For some standard test problems, the difference amounts to several orders of magnitude.

Problems for which $A(\alpha)$-stable methods are appropriate have h_{stable} infinite and the situation is the same as with an A-stable method. The basic task is to recognize when we can use a cheap iteration—simple or Jacobi—and when we must use a simplified Newton iteration. This must be done inexpensively, at virtually every step, and switching iteration methods must be efficient. In [10] we described how to do this and here we refine the procedure with Enright's idea and the use of the Jacobi iteration.

Because we want robust codes, we must consider what happens when some eigenvalue of f_y is close enough to the imaginary axis that h_{stable} is finite. One possibility is that $h_{\text{acc}} \ll h_{\text{stable}}$. Such a case is no different from one with h_{stable} infinite and needs no more comment. If, on the other hand, h_{acc} is comparable to, or greater than, h_{stable}, the formula actually exhibits a stability restriction.

The essential difficulty we face when the formula exhibits a stability restriction is that the estimate h_{est} of h_{acc} varies above and below h_{stable} as described in Section 4. We shall try to devise an algorithm which will avoid unnecessary evaluation of Jacobians in this situation. This is a general difficulty unrelated to our switching of iteration methods. Indeed, if $h_{\text{iter}} \ll h_{\text{stable}}$, there is no interaction at all. Typical numerical methods have a region of absolute stability containing a half-disc of radius β (van der Houven [4, p. 83] calls β the general stability boundary). If one compares β to $1/\gamma$ for typical methods, it is seen that often there will be no interaction of the two restrictions on step size if simple iteration is used. On the other hand, if the structure of f_y is favorable, h_{iter} can be much larger than h_{stable} when the Jacobi iteration is used. Thus interaction of a cheap iteration method and the stability restriction is probably not common, but is certainly possible. For this

reason, we must be careful that our scheme for switching iterations does not lead to a lot of unnecessary work should the step size used vary about h_{iter}.

7. An Algorithm. In [10] we did not fully specify when the change of iteration method should be made, nor did we go into details of implementation. In our numerical experiments we have found that the decisions are not critical, except in the presence of a stability restriction. Here we shall describe one approach which seems to work satisfactorily. The algorithm might appear to be rather different from that outlined in [10], but on close examination is seen to be essentially the same when an A-stable formula is used. Besides the fact that $A(\alpha)$-stable formulas are now treated, an additional iteration method and a different way of handling Jacobians contribute to the different description.

The various decisions interact with one another so that some reflection is necessary to fully appreciate the algorithm. To help the reader follow it, we split this section into subsections which correspond roughly to modules in a computer implementation. Translation into code is still not completely straightforward because we detail only those matters relevant to this paper. In each subsection we provide arguments for proceeding as we do. The approach we take is reasonable and works adequately, but there is room for improvement. It is hoped that presenting the algorithm in this way will facilitate future development of it.

7.1. *Initial Iteration Method.* The first step must be taken with simple iteration. No Jacobian is formed initially, and if simple iteration will suffice for the entire integration, the algorithm described will never form a Jacobian, one of our goals. However, if a Jacobian ever is formed, we shall not use simple iteration again. For the reasons given in Section 4, we prefer to use the Jacobi iteration method instead.

Just how the initial step size is selected is extraneous to this paper. On the other hand, our algorithm applies a sensitive test that this step size reflects the initial scale of the problem by insisting that simple iteration converge. This cheap iteration allows the code to recover inexpensively from an initial step size which is far too large.

7.2. *Evaluate Implicit Formula.* Here we have accepted values (x_n, y_n) and wish to compute an approximate solution y_{n+1} at $x_{n+1} = x_n + h$. An iteration has already been selected for the attempt to evaluate the implicit formula for y_{n+1}.

First a prediction y_{n+1}^0 is made, and then we begin iterating. How one decides when the implicit formula has been solved adequately is extraneous to this paper, but we do need to point out that some procedures allow the possibilities that the predicted value or the first iterate be accepted; see, for example, [6], [8], [15].

Whenever two or more iterations are made, a rate of contraction is estimated. The possibility of failure is not even considered until two iterations are made so that a rate estimate is available. It is very helpful that the rate for both the simple and the Jacobi iteration can be regarded as proportional to the step size. In [8] we argued that it is reasonable to proceed as if this were also the case with the rate for the simplified Newton interation. In our algorithm the step size in the factored iteration matrix is always that being attempted. This is the case of the analysis of [8] which is best justified. From proportionality and an observed rate, h_{iter}, the largest step size which would yield convergence with the current iteration method, is estimated.

If the observed rate is not less than 1, the iteration is certainly not contracting, so we go to subsection 7.2.1 dealing with a failure to converge. If the prediction is poor, or if the rate is slow, many iterations might be necessary. It is best to specify a maximum allowed number of iterations and to proceed to 7.2.1 if it is attained. In our research code this maximum was 5. We actually used the observed rate to predict if convergence would not be attained within the allowed number of iterations. This was done as described in [8, p. 109]. It proved quite helpful to terminate the iteration early in this way.

If convergence is secured, we proceed to 7.3 where the accuracy of y_{n+1} will be checked. If two or more iterations were made, an estimate of h_{iter} is available. If fewer than two iterations were made, no estimate of h_{iter} is ordinarily available. However, the rest of the algorithm may cause subsection 7.2 to be repeated: when convergence is not obtained, and when it is obtained, but y_{n+1} is not accurate enough, the step from x_n is attempted again with a smaller step size. An h_{iter} estimated in one try is at our disposal for subsequent tries provided the iteration method was not changed, as will be seen to be typical. Thus it could happen that convergence is obtained in fewer than two iterations, but an estimate of h_{iter} is available.

7.2.1. *Convergence Failure.* In traditional algorithms a new Jacobian must be formed at this time because, to save storage, it is overwritten by the iteration matrix and destroyed in the subsequent factorization. Saving a copy of the Jacobian and Enright's idea both allow us to reuse Jacobians so that we may hope to avoid the considerable expense of forming unnecessary Jacobians. Enright's idea avoids the storage penalty associated with saving a copy of the Jacobian and also avoids the cost of a factorization.

There are two basic tools for securing convergence. One is to reduce the step size. This has two good effects. The predicted value y_{n+1}^0 is improved and the rate of contraction is improved. The other tool is to form a new Jacobian. Here the simplified Newton iteration with a Jacobian evaluated at some previously computed point must be regarded as a different iteration from that using a Jacobian evaluated at a current point. In particular, we cannot predict the effect on the rate of contraction due to forming and using a current Jacobian; it is hoped that the rate will be better.

We argue that in conjunction with other aspects of our algorithm, reduction of the step size is always the appropriate response to failure of convergence. We insist that the very first step be carried out with simple iteration for reasons amplified in another subsection, so do not consider forming a Jacobian then. After the first step one has experience in a preceding step from which to predict what should happen in the current step. Of course, the step size selected for the current step is predicted to yield the desired accuracy. In our algorithm it is also predicted from solid evidence to yield convergence of the iteration method. The only time we cannot make such a prediction is when a Jacobian is evaluated at a current point and a simplified Newton method is tried in the current step. Later we shall amplify "current point" and note here only that, in this last circumstance, we certainly do not want to form a new Jacobian. Thus we reduce the possibilities to considering an appropriate response when we fail to get convergence, even though we have predicted on the basis of experience that the step size should yield accuracy and convergence.

It may happen that we just missed getting convergence because the rate is a little too slow or the predicted value a little too far away. There is no justification then for going to the expense of forming a new Jacobian when a relatively small change of step size will suffice.

If the rate of contraction is very slow, perhaps the process is not even contracting, or y^0_{n+1} is very far away, some of our basic assumptions justifying our predictions must be invalid. One possibility is that $y(x)$ or $f(x, y)$ does not change smoothly on $[x_n, x_{n+1}]$; for example, some term in f has a discontinuity. If the numerical solution can be advanced at all, the appropriate response seems to be to reduce the step size. Evaluating a Jacobian as traditionally done might well be a complete waste of effort; see subsection 7.4. Another possibility is that the problem has not changed character, the formula has. If the step size corresponds to being outside the stability region of the method, y_{n+1} may be a very poor approximation to $y(x_{n+1})$. This can lead to convergence failure in two ways: The predicted y^0_{n+1} might reflect the smooth behavior of $y(x)$ and so be far away from y_{n+1}. Even if the rate of contraction is adequate, a convergence failure might then occur because too many iterations are needed. Alternatively, the Jacobian might vary rapidly as a function of the dependent variables so that an approximate Jacobian which is a good approximation to $f_y(x_{n+1}, y(x_{n+1}))$ might be far away from $f_y(x_{n+1}, y_{n+1})$. This could mean that the prediction of a good rate of contraction is incorrect and there is a resulting failure of convergence. It is crucial that we not form a Jacobian when the stability region is the source of the trouble. It might help us to compute y_{n+1}, but the effort is misdirected. Should we get convergence, we expect then to reject y_{n+1} on grounds of accuracy. The appropriate response is to reduce the step size which will restore stability and the accuracy of y_{n+1}. Thus, in every case, an appropriate response is to reduce the step size without forming a new Jacobian.

From proportionality we estimate how much to reduce the step size to secure an adequate rate of convergence. In our research code we aim at a rate of 0.5. A significant reduction should be made to account for a possible lack of proportionality of rate and for a predicted solution which is not quite good enough. In our research code we required a reduction of at least a factor of 0.5.

In our experience a signal that a very rapid change of step size is needed often arises from a failure of some basic assumption. For this reason we limit the step size reduction. In our research code we did not allow the step size to be reduced more than by a factor of 0.1. Of course, very slow observed contraction, or divergence, result in this maximal reduction.

In only one situation do we change iteration method in this subsection. The step size is always halved or more. If the step was attempted with Newton's method, we have at our disposal an estimate of the largest step size for which the Jacobi iteration with this Jacobian matrix would converge at an acceptable rate. Should the step size be reduced enough that we can switch, we do so. This is one way the algorithm can respond to a change in character of the problem. There is no cost associated with the switch, and we have excellent reason for believing the Jacobi iteration will succeed and yield rapid convergence. The benefit is an important reduction in the overhead involved in solving linear systems of equations and in changing step size.

7.3. *Test Local Error.* Having formed y_{n+1} successfully, its accuracy is now estimated and h_{est}, an estimate of the largest step size which would yield the desired

accuracy is formed. If y_{n+1} is not accurate enough, we go to 7.3.1 and otherwise, to 7.3.2.

7.3.1. *Reject the Step.* If y_{n+1} is not accurate enough, h_{est} is smaller than the step size h actually tried. The step to x_{n+1} is rejected and another try made with a smaller step size. There are a variety of practical issues involved in selecting the new step size with the consequence that it is not simply h_{est}. However, the new step size h will be smaller than one which gave acceptable convergence in the computation of y_{n+1}. The iteration method used and the prediction scheme should therefore be more effective with this new step size. The only reason for considering a change of iteration method is that it might be possible to switch from a simplified Newton to a Jacobi iteration. If this is feasible, we do so as described at the end of subsection 7.2.1.

Having selected a new step size and switched to a cheaper iteration when feasible, we go to the beginning of 7.2 and try again.

7.3.2. *Accept the Step.* When we decided to accept y_{n+1}, we formed h_{est}. There are a variety of practical issues involved in selecting a step size for the next step with the consequence that it is not simply h_{est}. Several are significant to our algorithm. It is not prudent to let the step size increase greatly in a single step. When a simplified Newton iteration is being used and only a small increase in the step size appears possible, the increase in efficiency due to a slightly bigger step size is not worth the cost of changing the iteration method. There is a common tactic which is of fundamental importance in our special situation, so we shall expand upon it.

It is quite useful to note whether there was an unsuccessful attempt to step from x_n before the present one succeeded. A failure to get convergence, or a rejected step, tell us that the attempted step size was too optimistic. It can happen that h_{est} estimated from a shorter, successful step size is also too optimistic: A not uncommon situation is that the problem, i.e., $y(x)$ or $f(x, y)$, changes dramatically in the course of the first step size tried, e.g., there is a discontinuity in f at some x_d. The smooth behavior up to x_d suggests a large step size is possible, but this results in step failures until one tries a step landing short of x_d. The smooth behavior up to this point causes the cycle to be repeated. A simple but helpful tactic is not to allow a step size increase after any step involving an unsuccessful try. The device is of general value, but notice how it fits in here. It is a response to a dramatic change of problem. It is an equally valid response to a dramatic change of the behavior of the formula, namely, stepping outside the stability region. The tactic smooths out the behavior of the step sizes in a most desirable way.

Having chosen a step size we would like to try, we now ask if we need to improve the iteration method in order to evaluate the implicit formula with such a step. As we noted in 7.2, it may happen that convergence was so good in the step just taken that we were unable to estimate h_{iter}. In view of the fact that the new step size cannot be greatly larger than this one for which convergence was so good, there is no point to changing iteration methods. In such a case we proceed to the next step via 7.5.

We anticipate that the current iteration method will converge at rate, say, 0.5 if $h \leq 0.5\ h_{\text{iter}}$ (or h_{itmax} if the Jacobi iteration is used—see Section 4). If we predict adequate convergence in this way, we retain the iteration method and go to 7.5. Otherwise we go to subsection 7.4 to improve the iteration method.

7.4. *Improve Iteration Method.* Here we turn to a more powerful iteration method so that we can proceed with a step size appropriate to the smoothness of the solution. There are three cases corresponding to the three methods that might have been used to take the successful step. In each we choose to form an approximate Jacobian at a current point, but the reasons are rather different.

If the step was taken with simple iteration, we have no choice because both the alternatives require the Jacobian. If the step was taken with a simplified Newton iteration, the only improvement we consider is to form a current Jacobian.

If the step was taken with the Jacobi iteration, the diagonal of an existing Jacobian was used. It is tempting to switch to the simplified Newton iteration with the old Jacobian matrix. The trouble is that there is no information available to predict whether this will suffice. It is true that the Jacobi iteration allows us to solve some stiff problems for which it is reasonable to presume that the Jacobian is roughly constant for many steps. If the Jacobian has not changed much, using the whole Jacobian would be adequate. We consider the Jacobi iteration to be ordinarily only a minor improvement to simple iteration and expect it to be used basically on nonstiff portions of the integration. In such portions we expect the solution and the Jacobian to change significantly. For this reason, on grounds of simplicity, and to support decisions made in subsection 7.2.1, we choose to form a new Jacobian in this case, too.

When it appears desirable to form a new approximate Jacobian to aid in stepping from x_n to x_{n+1}, traditional algorithms approximate $f_y(x_{n+1}, y^0_{n+1})$. If Jacobians are saved, there is a good reason to approximate $f_y(x_n, y_n)$ instead. If all is going well, the two possibilities behave the same. It is a fundamental hypothesis that the Jacobian is changing slowly, so there should be little difference when computing y_{n+1}. There is even less reason to distinguish the procedures when the Jacobian is used for the computation of later steps, $y_{n+2}, y_{n+3}, \ldots$. A real difference arises when the attempt to compute y_{n+1} is unsuccessful. Because the character of the problem might have changed on $[x_n, x_{n+1}]$, it is necessary to reevaluate $f_y(x_{n+1}, y^0_{n+1})$ with the new step size. Proceeding as we suggest, there is no reason to form a new Jacobian. The traditional algorithms do not save the Jacobian so one must reevaluate anyway, in which case one might as well evaluate at the current predicted solution.

On forming the Jacobian, h_{itmax} is estimated for the Jacobi iteration as described in Section 4. If $h \leq h_{\text{itmax}}$, we predict the Jacobi iteration will contract at a rate of at least 0.5. In such a case we choose the Jacobi iteration for the next step. Otherwise we choose the simplified Newton method with this new Jacobian.

Having selected a step size and an iteration method we proceed to 7.5.

7.5. *Process Output.* After successfully computing (x_{n+1}, y_{n+1}), the result is returned to the user if it was requested. The code would be reentered at 7.2 if the user wishes to continue. The user might shorten the step size proposed for the next step for a variety of reasons. The code might also shorten it in this section to deal with future output whether or not control is returned to the user. If control is not returned to the main program, it now passes directly to subsection 7.2.

The point we make here is that the step size proposed may be reduced for some reason before attempting the next step. This can only improve the performance of the prediction scheme and of the iteration method. It also implies that factorization

of iteration matrices for the Newton method should be done in subsection 7.2, when the step size to be attempted is definitive.

8. Some Numerical Results. We have written a research code based on the θ-family of formulas

$$y_{n+1} = y_n + h\big[\theta f(x_n, y_n) + (1-\theta) f(x_{n+1}, y_{n+1})\big].$$

When applied to $y' = Ay$ with constant step size h, the stability function $R(\lambda)$ is given in

$$y_{n+1} = R(\lambda) y_n = \frac{1+\theta\lambda}{1-(1-\theta)\lambda} y_n,$$

where $\lambda = Ah$. Our experiments were performed with $\theta = 0$, the backward Euler method, which is A-stable and has $R(\infty) = 0$; $\theta = 0.1$ which is A-stable, but less damped since $R(\infty) = -1/9$; and $\theta = 0.9$ which has a (small) finite stability region. The local truncation error is

$$\text{lte} = \big[\tfrac{1}{2} - (1-\theta)\big] h^2 y''(x) + O(h^3).$$

The leading constant is then 0.5, -0.4, $+0.4$ in the three cases, respectively.

In general the code is written in accordance with the proposals of [6], [8], [15]. The term y'' in the local truncation error is estimated by a second divided difference of solution values. Enright's idea is used for the simplified Newton scheme. Analytical Jacobians were used in all our experiments.

The code was written for research purposes so that the results presented here are intended to illustrate switching iteration methods rather than the behavior of a piece of software. Some computations were made with the production BDF code LSODE [3]. To make it more comparable we restricted the order to 1, so refer to the results as computed with LSODE1. Unfortunately, the codes are tuned very differently and at order 1 the cost is a sensitive function of the accuracy achieved. A direct comparison is also difficult because LSODE does a lot of unnecessary Jacobian evaluations which result in very fast convergence and fewer function evaluations. Here we shall use the LSODE1 results merely to indicate the general range of Jacobian evaluations as made by a traditional algorithm.

Our numerical results were obtained from selected members of the test set [2]. The observations of [9] are very pertinent. For example, it is pointed out in [9] that the problem E2 is not really stiff, so it provides an interesting test of our algorithm. The problem was solved with the three scalar, pure absolute error tolerances 10^{-2}, 10^{-3}, 10^{-4}. The formula with finite stability region, $\theta = 0.9$, did not form a Jacobian in any case so that the integrations were all carried out entirely with simple iteration. The A-stable methods, $\theta = 0$ and 0.1, formed exactly one Jacobian for each integration. However, all steps were carried out with either the simple or Jacobi iteration for all the integrations. This performance is about all we could hope for from our switching algorithms. LSODE1 used 7–11 Jacobians when doing these integrations.

For E2 the solutions with $\theta = 0.1$ and 0.9 were of comparable efficiency, and both were notably more efficient than the solution with $\theta = 0$. This was expected from the local truncation errors. With the truly stiff problems it is too expensive to carry out

the entire integration with a formula with finite stability region. For this reason the runs with $\theta = 0.9$ were limited to 100 successful steps for all our examples. Computations with this formula simulate the use of an $A(\alpha)$-stable formula on a problem which exhibits a stability restriction. The experiments we report give a good idea of the performance of our algorithm in a situation we regard as very unusual in practical computation.

The problem A2 has a constant Jacobian. At the tolerances 10^{-2}, 10^{-3}, the formula with $\theta = 0.9$ formed one Jacobian, but all 100 steps were taken with a cheap iteration method. At the tolerance 10^{-4}, no Jacobian was formed at all. One reason for using A2 as a test problem was to point out that, with Enright's device and our algorithm for deciding when to form a Jacobian, we cannot form more than two Jacobians when the differential equation has a constant Jacobian, as it does here. This gratifying observation is easily seen to follow from the description of Section 7 and is indicative of the soundness of the approach. In all the runs with $\theta = 0$ and 0.1, exactly two Jacobians were formed; LSODE1 forms 20–38.

This problem also illustrates the fact that switching methods can be advantageous even for a stiff problem. For example, with $\theta = 0.1$ at tolerance 10^{-2} there were 45 successful steps. Of these 12 were taken with simple iteration, 18 with Jacobi, and 15 with Newton. This is quite a lot of steps with cheap iterations for such a crude tolerance. At the tolerance 10^{-4}, the 333 successful steps were split among the iterations as 116, 196, and 21. The results with $\theta = 0$ are quite similar but correspondingly more expensive as expected from the local truncation error.

The problem A2 has an obvious limit solution. The numerical results at the end of the interval agreed with the limit solution to about one more digit than the local error tolerance in every case. With the A-stable formulas there were at most two convergence failures and two rejected steps in a run. With $\theta = 0.9$ there were no convergence failures, but there were a significant number of rejected steps. The most was 7 at the tolerance 10^{-2}. This is to be expected as a consequence of the interaction with the stability region as discussed in Section 6.

We believe the Liniger-Willoughby problem D1 should be solved in the original variables for reasons presented in [9]. When this is done, the problem is linear with a nonconstant Jacobian. The problem was solved with scalar, pure absolute error tolerances 10^{-2} and 10^{-3}.

In every case the code took exactly two steps with simple iteration. In the case of the formula with a finite stability region, $\theta = 0.9$, one Jacobian was formed for each integration, but the Newton scheme was never used. The stability region had no serious effects: there were no convergence failures; one integration had 5 rejected steps, the other 4.

The solutions of the Liniger-Willoughby problem with the two A-stable formulas were quite similar, with $\theta = 0.1$ somewhat more efficient. There were 1–2 convergence failures in each integration. There were a significant number of rejected steps, the worse case being 14 rejections associated with 96 successful steps at tolerance 10^{-2} with $\theta = 0.1$. We have also found a lot of steps rejected with other kinds of methods applied to this problem. In [7] we found that how cautious the step size selection algorithm is plays an important role in this matter. The algorithm implemented in the θ-family code is bold, using a procedure suitable for nonstiff

problems. Our algorithm for selecting the iteration method does not cause a new Jacobian to be formed on a step rejection so that the number of Jacobians formed ranged from 4 to 5, despite a relatively large number of step rejections. By way of comparison, LSODE1 used 24–31.

The Robertson problem D2 is a nonlinear problem of chemical kinetics, which we solved for the scalar, pure absolute error tolerances 10^{-2}, 10^{-3}, 10^{-4}. The formula with a finite stability region, $\theta = 0.9$, did not form a Jacobian for the tolerance 10^{-2} but formed just one for the other two tolerances. All steps were taken with a cheap iteration method. There were no convergence failures and at most 4 rejected steps in an integration. The A-stable formulas had no convergence failures and at most one rejected step in an integration. The number of Jacobians formed ranged from 2 to 3. LSODE1 formed 15–45. Prothero [14, p. 135] presents some similar computations with a more traditional code based on a member of the θ-family. The results are not directly comparable, but it is worth remarking that his code uses on the order of 20 Jacobians.

9. Conclusion and Acknowledgements. We believe that the analysis and numerical results presented here demonstrate the feasibility of making some kinds of ODE codes type-insensitive by the automatic selection of iteration method. Extension of the results of [10] to $A(\alpha)$-stable formulas is more of conceptual than practical importance. However, we have in addition fully specified the algorithms of [10] and introduced several important technical improvements. Although stability restrictions are of little practical importance with the popular $A(\alpha)$-stable formulas, we have shown a way to enhance the performance of codes then, quite aside from the issue of selection of iteration method.

The author and his associates L. R. Petzold and H. A. Watts are engaged in a project to develop a new piece of mathematical software for the solution of ODEs. Towards this end a number of investigations are being made independently and in various combinations of team members. The work reported here is an independent investigation done by the author, with computing assistance provided by L. S. Baca. The author has benefited from frequent discussions of the investigation with the other members of the team. Integration of the results with the work of the others has led to notable improvements.

Numerical Mathematics Division
Sandia National Laboratories
Albuquerque, New Mexico 87185

1. W. H. ENRIGHT, "Improving the efficiency of matrix operations in the numerical solution of stiff ordinary differential equations," *ACM Trans. Math. Software*, v. 4, 1978, pp. 127–136.

2. W. H. ENRIGHT, T. E. HULL & B. LINDBERG, "Comparing numerical methods for stiff systems of O.D.E.'s," *BIT*, v. 15, 1975, pp. 10–48.

3. A. C. HINDMARSH, "LSODE and LSODI, two new initial value ordinary differential equation solvers," *SIGNUM Newsletter*, v. 15, 1980, pp. 10–11.

4. P. J. VAN DER HOUVEN, *Construction of Integration Formulas for Initial Value Problems*, North-Holland, Amsterdam, 1977.

5. L. F. SHAMPINE, "Stiffness and non-stiff differential equation solvers," in *Numerische Behandlung von Differentialgleichungen* (R. Ansorge et al., eds.), Birkhauser, Basel, 1975.

6. L. F. SHAMPINE, "Evaluation of implicit formulas for the solution of ODEs," *BIT*, v. 19, 1979, pp. 495–502.

7. L. F. SHAMPINE, "Implementation of Rosenbrock methods," *ACM Trans. Math. Software*. (To appear.)

8. L. F. SHAMPINE, "Implementation of implicit formulas for the solution of ODEs," *SIAM J. Sci. Statist. Comput.*, v. 1, 1980, pp. 103–118.

9. L. F. SHAMPINE, "Evaluation of a test set for stiff ODE solvers," *ACM Trans. Math. Software.*, v. 7, 1981, pp. 409–420.

10. L. F. SHAMPINE, "Type-insensitive ODE codes based on implicit A-stable formulas," *Math. Comp.*, v. 36, 1981, pp. 499–510.

11. L. F. SHAMPINE & M. K. GORDON, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, Freeman, San Francisco, Calif., 1975.

12. L. F. SHAMPINE & H. A. WATTS, *DEPAC—Design of a User Oriented Package of ODE Solvers*, Report SAND79-2374, Sandia National Laboratories, Albuquerque, N. M., 1980.

13. S. SKELBOE, "The control of order and steplength for backward differentiation formulas," *BIT*, v. 17, 1977, pp. 91–107.

14. A. PROTHERO, "Introduction to stiff problems," Chap. 9 in *Modern Numerical Methods for Ordinary Differential Equations* (G. Hall and J. M. Watt, eds.), Clarendon Press, Oxford, 1976.

15. J. WILLIAMS, *The Problem of Implicit Formulas in Numerical Methods for Stiff Differential Equations*, Report 40, Dept. of Math., Univ. of Manchester, Manchester, England, 1979.