JOURNAL OF COMPUTATIONAL PHYSICS 54, 74–86 (1984)

# Stiffness and the Automatic Selection of ODE Codes*

L. F. SHAMPINE

*Applied Mathematics Research Department, Sandia National Laboratories,*[†] *Albuquerque, New Mexico 87185*

Received June 15, 1981

The author describes the basic ideas behind the most popular methods for the numerical solution of ordinary differential equations (ODEs). He takes up the qualitative behavior of solutions of ODEs and its relation to the propagation of numerical error. Codes for ODEs are intended either for stiff problems or for non-stiff problems. The difference is explained. Users of codes do not have the information needed to recognize stiffness. A code, DEASY, which automatically recognizes stiffness and selects a suitable method is described.

## 1. INTRODUCTION

In this paper we shall first describe the basic ideas of the most popular methods for the numerical solution of the initial value problem for a system of ordinary differential equations (ODEs). Then we shall take up the qualitative behavior of the solutions of ODEs and its relation to the propagation of numerical error. At the present time codes for ODEs are divided into those intended for non-stiff problems and those intended for stiff problems. They are very inefficient when applied to the wrong type of problem. We shall try to give the reader some feeling as to what stiffness is. It is a complex matter, and unfortunately the user of a code does not have available all the information necessary to take the best possible action. Because of the situation, the automatic recognition and response to stiffness is an important and active area of mathematical software research. A crude, but useful, way to select automatically a suitable ODE code during the course of an integration will be presented.

* This article sponsored by the U. S. Department of Energy under Contract DE-AC04-76DP00789.
[†] A U. S. Department of Energy Facility.

0021-9991/84 $3.00

## 2. Typical Numerical Methods

We shall be interested in solving the system of ordinary differential equations

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

Many quantities here and later are vectors, but there will be no need for special notation. The popular numerical methods produce approximate solution values $y_n \doteq y(x_n)$ on a mesh $\{x_n\}$ in the interval $[a, b]$. They step through the interval by proceeding from an approximation at $x_n$ to one at $x_{n+1} = x_n + h$, a step of length $h$. (We suppose the direction is chosen so that $h > 0$ for notational simplicity.)

The simplest procedure is the forward Euler method which can be derived as follows. Suppose we have $y_n \doteq y(x_n)$. Define the local solution $u(x)$ to be the solution of

$$u' = f(x, u), \qquad u(x_n) = y_n.$$

If we expand $u$ in a Taylor series about $x_n$, we have

$$u(x_{n+1}) = y_n + hf(x_n, y_n) + \frac{h^2}{2} u''(x_n) + \dots,$$

where we use $u'(x_n) = f(x_n, y_n)$. The forward Euler method is

$$y_{n+1} = y_n + hf(x_n, y_n). \tag{2}$$

The local error is defined in general to be

$$\text{le} = u(x_{n+1}) - y_{n+1}$$

and is here

$$\text{le} = \frac{h^2}{2} u''(x_n) + \dots \,. \tag{3}$$

The forward Euler method is representative of the popular explicit Runge–Kutta methods. A method like that of Fehlberg which is implemented in the code RKF45 [1] of Watts and Shampine has the same form, but evaluates the equation several times in the course of a step.

If one did the Taylor expansion about the end of the step, he would arrive at the backward Euler formula

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}). \tag{4}$$

Asymptotically, as $h \to 0$, this formula has the same local error as the forward Euler formula. There is, however, an important qualitative difference. The new solution vector $y_{n+1}$ appears on both sides of Eq. (4) so that the formula is implicit. How one solves these algebraic equations at each step is crucial to the use of the formula.

One way to solve (4) is to predict a solution using the explicit formula (2),

$$y_{n+1}^{(1)} = y_n + hf(x_n, y_n)$$

and then use simple, or functional, iteration

$$y_{n+1}^{(m+1)} = y_n + hf(x_{n+1}, y_{n+1}^{(m)}), \qquad m = 1, 2, \ldots . \tag{5}$$

Here, and later, we need the important concept of a Lipschitz constant. We say the function $f(x, y)$ satisfies a Lipschitz condition with constant $L$ for $a \leqslant x \leqslant b$, all $y$ if

$$\|f(x, u) - f(x, v)\| \leqslant L \|u - v\| \tag{6}$$

for all $u, v$. If one does a Taylor expansion of $f$,

$$f(x, u) - f(x, v) = J(u - v),$$

where $J$ is the Jacobian matrix

$$J = \left(\frac{\partial f_i}{\partial y_j}\right),$$

here evaluated at $x$ and at unknown points between $u$ and $v$, he sees that

$$L = \sup \|J\|$$

is a suitable Lipschitz constant. In this form it is perhaps more easily recognized that $L$ measures how fast $f$ can change with respect to the variables $y$. A Lipschitz condition is normally assumed to hold when solving (1) because it guarantees a unique solution exists, and it is fundamental to the analysis of numerical methods. The assumption can be weakened, but there is no need to go into the matter here.

On subtracting (5) from (4) and using the Lipschitz condition (6), we arrive at

$$\|y_{n+1} - y_{n+1}^{(m+1)}\| \leqslant hL \|y_{n+1} - y_{n+1}^{(m)}\|.$$

This says that if $hL < 1$, simple iteration converges. When iterated to convergence, we have here an example of an Adams–Moulton method as implemented, e.g., in Gear's code DIFSUB [2]. The more accurate Adams formulas involve $y_{n-1}, y_{n-2}, \ldots,$ but this is here irrelevant.

The iterations of (5) do not actually lead to a more accurate approximation of the solution of the differential equation. Another possibility is to stop at a fixed number of iterations. This is what is known as a predictor–corrector method. If we quit at $m = 2$, we have here an example of an Adams–Bashforth–Moulton pair as implemented, e.g., in the code ODE [3] of Shampine and Gordon.

For the solution of stiff problems we are interested in step sizes $h$ with $hL \gg 1$. It is obviously necessary to use a different iteration. The standard one is the simplified Newton iteration which uses an approximate Jacobian matrix $J$,

$$J \doteq \frac{\partial f}{\partial y}(x_{n+1}, y_{n+1})$$

in a linearization of (4):

$$y_{n+1}^{(m+1)} = y_n + h[f(x_{n+1}, y_{n+1}^{(m)}) + J(y_{n+1}^{(m+1)} - y_{n+1}^{(m)})].$$

In comparison with a simple iteration, this is very expensive. The matrix $J$ must be formed, the iteration matrix $I - hJ$ must be decomposed into triangular factors, and a linear system must be solved for each iteration. On the other hand, very large step sizes $h$ may be possible. We have here an example of the backward differentiation formulas (BDF). The more accurate formulas involve previous values $y_{n-1}, \ldots$, but this is here irrelevant. The example is representative of the implementation of Gear in DIFSUB and its modernization in the GEAR package and later LSODE, both by Hindmarsh [4, 5].

The methods of Runge–Kutta and Adams are classical, but are still among the most effective for non-stiff problems. The backward differentiation formulas were exploited later for the solution of stiff problems when these problems were recognized as a special class for which the classical methods were impractical. The BDF are certainly the most popular today for the solution of stiff problems. For later reference we mention here the software package DEPAC [6] which includes a Runge–Kutta code DERKF built upon RKF45, an Adams–Bashforth–Moulton code DEABM built upon ODE, and a backward differentiation formula code DEBDF built upon LSODE.

## 3. Error Propagation

The stability of the differential equation problem itself is crucial to its numerical solution. A classical result from the theory of ODEs states that if $u(x)$, $v(x)$ are solutions of (1), then

$$\|u(x) - v(x)\| \leqslant \exp(L(x-a)) \, \|u(a) - v(a)\| \tag{7}$$

for $x \geqslant a$. This bound tells us how fast solutions can spread apart in terms of the Lipschitz constant and the distance involved. It poses a fundamental limit on the accuracy possible for numerical solution. We start integrating (1) with the exact initial value $y_0 = y(a)$. On stepping to $a + h$ we approximate $y(a+h)$ by $y_1$. If no further error is made in the remainder of the integration, the result at $b$ could be in error by as much as $\exp(L(b-a-h)) \, \|y_1 - y(a+h)\|$. If $L(b-a)$ is large, this is a very serious limitation.

The bound (7) is sharp but is almost always completely unrealistic. This is because it must account for problems with solutions diverging as fast as possible. Normally we do not solve unstable problems. The typical behavior is that solution curves spread and come together so that the net result is that the problem is moderately stable.

There is a corresponding concept of the stability of a numerical formula. When applied to a problem (1) satisfying a Lipschitz condition, do the numerical solutions diverge? One establishes results like

$$\|u_n - v_n\| \leqslant K \|u_0 - v_0\|, \tag{8}$$

where $\{u_n\}$ is the result of the formula applied to Eq. (1) with initial value $u_0$ and $\{v_n\}$ is similarly defined. More is needed. We must be able to assert the existence of a constant $K$ in (8) which holds uniformly for all sufficiently small step sizes in order to establish convergence. There is a body of analytical techniques for proving such asymptotic stability results.

For simplicity we have described stability in (7) and (8) with respect to perturbation of initial conditions only. We must also establish it with respect to perturbations of the equation. In addition the formula must in a reasonable way approximate the differential equation; the local error defined in Section 2 must tend to zero as the step size does. Putting all these aspects together one can establish a convergence result as follows for Euler's method (forward or backward) with constant step size $h$:

$$\|y(x_n) - y_n\| \leqslant \frac{M_2 h}{2} \left( \frac{\exp(L(x_n - a)) - 1}{L} \right),$$

where

$$M_2 = \max_{[a,\, b]} |y''(x)|.$$

The exponential term arises from stability considerations. The first term arises from the local error (3) divided by $h$. Basically it comes from the facts that an error like (3) is made at each step and there are $(b-a)/h$ steps.

In practice the step size is adjusted at every step so that the local error is no more than a given tolerance $\varepsilon$. This is to be achieved with the largest step size $h$ possible. Then, e.g.,

$$\left\| y(x_n) - y_n \right\| \leqslant \varepsilon \left( \frac{\exp(L(x_n - a)) - 1}{L} \right).$$

In later sections these expressions will provide useful insight.

## 4. What Is Stiffness?

We feel that it is important to distinguish between the theoretical and practical definitions of stiffness. The *practical* definition of stiffness is the historical one: If codes based on the classical methods of Runge–Kutta and Adams are much more expensive than those based on the BDF, the problem is stiff. In view of the fact that the classical methods are very much cheaper per step than the BDF, the question boils down to asking, when might the BDF use a much larger step size? In this section we shall consider what determines the step size that a modern code can use.

The example methods outlined in Section 2 all have the same local error asymptotically. If the accuracy request determines the step size, the problem cannot be stiff. Generally speaking, the more accuracy one asks for, the less stiff a problem will appear.

A simple example will prove illuminating. Consider the solution of the single equation $y' = \lambda y$ with $\lambda \ll -1$. The solution for $x \geqslant 0$ is $y(x) = \exp(\lambda x) y(0)$. For the example formulas,

$$|\mathrm{le}| \doteq \left| \frac{h^2}{2} y''(x) \right| = \left| \frac{h^2}{2} \lambda^2 \exp(\lambda x) y(0) \right| .$$

If one tries to control the error in an absolute sense, $|\mathrm{le}| \leqslant \varepsilon$, we see that for small $x$, $y''(x)$ is large and a small step size is needed to resolve the boundary layer. However, as the integration proceeds, $y''(x)$ tends to 0 exponentially fast so that the accuracy requirement falls away and extremely large $h$ are feasible. If one imposes a relative error control,

$$\left| \frac{\mathrm{le}}{y(x)} \right| \doteq \left| \frac{h^2 \lambda}{2} \right| \leqslant \varepsilon,$$

the situation is completely different—the step size is restricted on grounds of accuracy for all $x$. Evidently then, stiffness depends on the nature of the error control.

We have seen that it is possible to encounter problems for which accuracy can be achieved with very large $h$. Is this at all likely? Consider problems of the form

$$y' = Jy + g(x), \tag{9}$$

where $J$ is a constant matrix. Any two solutions $u$, $v$ of (9) satisfy $(u - v)' = J(u - v)$. Suppose $J$ is similar to a diagonal matrix of its eigenvalues $\lambda_i$ (in general, complex numbers):

$$TJT^{-1} = D = \operatorname{diag}\{\lambda_i\}.$$

The change of variables $z = T(u - v)$ leads to

$$z' = Dz \qquad \text{or} \qquad \{z_i' = \lambda_i z_i\}.$$

Uncoupled like this, we see that the solution curves $u$ and $v$ approach one another if $\mathrm{Re}(\lambda_i) < 0$ for all the eigenvalues $\lambda_i$.

This familiar stability analysis will be applied later to the corresponding numerical procedure. What it says now is that all solutions $y(x)$ converge to a particular solution $u(x)$, and their derivatives $y^{(P)}(x)$ converge to the $u^{(P)}(x)$ too. If $u(x)$ is smooth, then accuracy may not dominate in the choice of the step size for integrating it. If it does not, this will be true of *all* solutions $y(x)$. When some $\lambda_j$ is "large," there will be a period of rapid change— an initial transient or boundary layer—as $y(x)$ approaches $u(x)$. There the problem is not stiff, but eventually the integration does become stiff. Notice that what "smooth" means depends very much on the particular numerical method employed.

This analysis is applied to the general problem $y' = f(x, y)$ locally by considering the approximating problem at $(x_n, y_n)$:

$$y' = \frac{\partial f}{\partial y}(x_n, y_n)(y - y_n) + f(x_n, y_n) + \frac{\partial f}{\partial x}(x_n, y_n)(x - x_n). \tag{10}$$

The eigenvalues of the local Jacobian $\partial f/\partial y$ tell us something about the stability of the original problem. This heuristic analysis is useful, but it should not be taken too seriously. As usual with a local linearization, certain phenomena are not revealed in the model equation.

Experience has shown that it is very common that the local linearization be stable, indeed super stable. It is by no means unusual to encounter eigenvalues $\lambda_i$ with $\mathrm{Re}(\lambda_i) \leqslant -10^{10}$. This is not some artifact of the mathematics, it simply reflects the stability of the physical processes being modeled, and the $\lambda_i$ show the time scales on which the various processes can evolve.

Now that we have seen how it can happen that the restriction on the step size due to accuracy can be weak, we ask, what else can restrict $h$? One possibility is output. When we described solution methods in Section 2, we implicitly assumed that answers at mesh points selected by the code would suffice. This often not the case. People want answers at specific $x$ and may want a lot of answers, e.g., in plotting. One possibility is to adjust the step so as to place a mesh point at every desired output point. This can severely affect the efficiency of those methods for solving differential equations which do a lot of work for each step. Other methods are able to interpolate on the mesh and so produce output values cheaply with little impact on the cost of integration. The typical Adams and BDF codes are examples. One must be alert to the fact that where and how frequently he wants answers may affect the stiffness.

A related issue is the length of the interval of integration. If a few steps with a Runge–Kutta code suffice for the whole interval, it matters not at all that a BDF code could use an enormously larger step size.

The iteration method implemented in a code plays an important role. When the backward Euler formula is evaluated by simple iteration, the step size must satisfy $hL < 1$. When $L$ is large, this can be a very severe restriction. Recalling the relationship between $L$ and the Jacobian matrix and using the general result that $|\lambda_i| \leqslant \|J\|$ for any eigenvalue $\lambda_i$ of $J$, we see that $|h\lambda_i| < 1$ is necessary. This restriction holds quite independently of the smoothness of the solution. As we saw with the model problems (9), this consideration might well be an extremely severe restriction. The predictor–corrector form has a similar restriction which is manifested in another way. Of course, with the simplified Newton scheme this kind of restriction does not arise. The point here is that how a formula is implemented in a code may profoundly affect the step size possible.

Let us now take up the stability of the numerical method when applied to those problems of the form (9) which are stable. The same method of analysis applies to all the standard numerical methods. Changing variables shows that the forward Euler method applied to $z' = Dz$ is uncoupled just as the differential equations are, and for equation $i$, the formula is

$$z_{n+1,i} = z_{n,i} + h\lambda_i z_{n,i}.$$

Here we find that the difference vector $z$ grows unless $|1 + h\lambda_i| \leqslant 1$. The stability region for this method is the disc $S = \{h\lambda \mid |1 + h\lambda| \leqslant 1\}$. The integration is stable only if $h\lambda_i \in S$ for all eigenvalues $\lambda_i$ of the Jacobian $J$. A similar analysis for the backward Euler method leads to

$$z_{n+1,i} = z_{n,i} + h\lambda_i z_{n+1,i}.$$

One finds that

$$\left| \frac{1}{1 - h\lambda} \right| \leqslant 1$$

for all $h\lambda$ with $\mathrm{Re}(\lambda) < 0$. Thus whenever the differential equation (9) is stable, this numerical method is too.

In this way all the standard numerical methods have associated with them stability regions $S$ such that $h\lambda_i \in S$ is necessary for stability of the method, where $\lambda_i$ is any eigenvalue of the local Jacobian with $\mathrm{Re}(\lambda_i) < 0$. The classical methods of Runge–Kutta and Adams all have finite stability regions. Ideally we would have the whole left half complex plane as a stability region, like the backward Euler formula has. Unfortunately it is hard to achieve this with efficient formulas. In particular, although the more accurate BDF have infinite stability regions, they do have "holes" where they are not stable.

Whether or not a stability restriction is present depends on the formula used and details of the problem, specifically the behavior of the eigenvalues of the Jacobian. A problem can suffer a severe stability restriction with a good formula, for example, the fifth order BDF, and none at all with another, e.g., the fourth order BDF.

In conclusion we have seen that stiffness in a *practical* sense depends on the details of the mathematical problem, the computational problem, the numerical method, and the implementation of the numerical method. Furthermore, the type can change as the integration proceeds. The usual description of a problem as "stiff" means only that if one chooses to solve the whole problem with a single code, he is better off to use one based on BDF than on Runge–Kutta or Adams methods. Almost always there are boundary layers present which are better handled by the classical methods.

## 5. Automatic Selection of Code

In this section we shall describe a crude, but useful, technique for the automatic selection of code. With the assistance of Baca, the author has written a code DEASY for this purpose. It fits into the DEPAC software package referenced in Section 2. Its basic task is to monitor the integration and to select the Runge–Kutta code DERKF or the BDF code DEBDF, whichever is the more appropriate. It is presumed that any user of DEASY does not want to concern himself with the technical details of the numerical solution. For this reason DEASY uses DEBDF in a mode generating a dense Jacobian by numerical differencing internally.

Any automatic scheme must predict the behavior of the integration in the future based on what has been observed up to the current point. The problem can, and often does, change character. Thus we must reconsider the decision as to the best method as often as practical, depending on the cost of the decision and the cost of changing methods.

We propose selecting the code according to the Lipschitz constant $L$. The integration is always started with the Runge–Kutta code DERKF. If the integration is at $x$ and is to continue to $b$, a switch from DERKF to DEBDF is made if $L(b-x) \geqslant 300$. A switch from DEBDF to DERKF is made if $L(b-x) \leqslant 100$. No switch is permitted if it is estimated that no more than 15 steps remain in the integration.

Supposing that this program is feasible, are we doing the "right" thing? We believe it is important that the first few steps be small enough to resolve any solution curve, so as to recognize the scale of the problem. When possible, the step size will be increased rapidly thereafter. This done most efficiently with a one-step method like that of DERKF. Also, if the problem is not stiff, DEBDF will never be called and the expensive approximation of a Jacobian will be completely avoided. If $L$ is "small" compared to the interval of integration, it is clear that the Runge–Kutta code is the more efficient. If $L$ is "large," the situation is not so clear. It might be large because the differential equation is stable or unstable. If the differential equation is stable, it is best to use DEBDF, although we have noted a good many other reasons why the step size might also be restricted. If the problem is unstable, it is not clear which code is best. This is a matter which has seen little attention. High order would be useful, but this does not separate these particular codes. It is at least plausible that the additional information furnished by the Jacobian might allow DEBDF to do better. Experience does not separate the codes either. Our choice here seems adequate. Fortunately one is not asked to solve problems which are unstable for long intervals.

Our proviso that we not switch unless more than 15 steps remain is a natural one in any case, but it leads into another issue. How do we adjust the step size when switching methods? It is possible to do something sophisticated relating the methods, but we did not consider it worth the trouble. All the codes in DEPAC automatically select an initial step size. The user can limit it if he wishes, and we did limit it by the step size being used currently by the other method. Otherwise, we simply relied on the automatic start. Naturally we need to give the codes a reasonable number of steps to get going. Our decision to proceed in this simple way is in part justified by our belief that the type changes infrequently.

In DERKF we compute a large *lower* bound for $L$. To do this we evaluate $f$ at most five times. By checking every 15 attempted steps, the cost is held to no more than a 5 % increase in function evaluations. The bound is obtained by the use of a non-linear power method. When the code has reached $x_n$, it has $y_n = y^{(0)}$ and $f(x_n, y_n) = f^{(0)}$. We define $y^{(1)}$ by

$$y^{(1)} - y^{(0)} = \frac{1}{\rho_0} f^{(0)},$$

where the scalar $\rho_0$ is chosen to make the difference $y^{(1)} - y^{(0)}$ small. For $m = 1, 2, \ldots$ define

$$\begin{gathered} f^{(m)} = f(x_n, y^{(m)}), \qquad \rho_m = \frac{\|f^{(m)} - f^{(0)}\|}{\|y^{(m)} - y^{(0)}\|}, \\ y^{(m+1)} - y^{(0)} = \frac{1}{\rho_m}(f^{(m)} - f^{(0)}). \end{gathered} \tag{11}$$

Notice that the definition of $\rho_m$ makes $\|y^{(m+1)} - y^{(0)}\| = \|y^{(m)} - y^{(0)}\| = \cdots = \|y^{(1)} - y^{(0)}\|$. By definition of a Lipschitz constant, $\rho_m \leqslant L$ for each $m$. If we do a Taylor expansion of $f$ in (11), we find

$$y^{(m+1)} - y^{(0)} = \frac{1}{\rho_m} J_m (y^{(m)} - y^{(0)}),$$

where $J_m$ is the Jacobian evaluated at $x_n$ and at points along the line between $y^{(m)}$ and $y^{(0)}$. By virtue of keeping $y^{(m)}$ close to $y^{(0)}$, we can approximate $J_m \doteq J$ to see what is happening. Then

$$y^{(m+1)} - y^{(m)} \doteq \frac{1}{\rho_m \rho_{m-1} \cdots \rho_0} J^m (y^{(1)} - y^{(0)}).$$

This is the power method applied to computing the largest eigenvalue of $J$. It is known that $\rho_m \to \rho(J)$, the magnitude of the largest eigenvalue of $J$, if the eigenvalue is not complex.

A different view is obtained when we choose a norm. In the Euclidean vector norm, a simple computation shows that $\rho_m^2$ is a Rayleigh quotient for the largest

TABLE I

Comparison of Automatic Selection of Code on a Set of Non-stiff Problems to a Code Intended for Such Problems

| Absolute Tolerance | DERKF FCN | DERKF CP | DEASY FCN | DEASY CP |
|---|---|---|---|---|
| $10^{-3}$ | 4010 | 1.40 | 4150 | 1.44 |
| $10^{-4}$ | 5589 | 1.85 | 5804 | 1.90 |
| $10^{-5}$ | 7912 | 2.42 | 8247 | 2.59 |
| $10^{-6}$ | 11324 | 3.30 | 11859 | 3.58 |

eigenvalue of $J_m^T J_m$ which is just $\|J_m\|_2^2$. For this reason we can expect $\max \rho_m$ to be a large lower bound.

For more details about this process the reader can consult [7]. At most five iterations are allowed in DEASY. If the problem has a Lipschitz constant large enough to cause a switch, in our experience it is revealed in one or two iterations. Thus we can often quit early to hold down the cost.

To go in the reverse direction is easier. In DEBDF an approximate Jacobian $J$ is actually formed from time to time. The only difficulty is that we want an *upper* bound and $\|J\|_2$ is not practical to compute. We actually use

$$\|J\|_2 \leqslant \|J\|_E = \left(\sum_{i,j} J_{ij}^2\right)^{1/2}$$

which is readily available. Again it is possible to interrupt the computation of the bound when we see that a switch cannot occur and so reduce the overhead.

Some numerical examples will illustrate that the convenient code DEASY is quite useful. Hull *et al.* [8] assembled a set of 25 non-stiff test problems. In Table I we report the cost of solution of all the problems, except D5, in terms of the number of

TABLE II

Comparison of Automatic Selection of Code on a Problem with Unstable Regions to a Runge–Kutta Code

| Absolute tolerance | DERKF FCN | DERKF CP | DEASY FCN | DEASY CP |
|---|---|---|---|---|
| $10^{-3}$ | 564 | 0.146 | 682 | 0.327 |
| $10^{-4}$ | 804 | 0.214 | 1015 | 0.563 |
| $10^{-5}$ | 1156 | 0.298 | 1389 | 0.745 |
| $10^{-6}$ | 1711 | 0.431 | 1883 | 1.066 |

TABLE III

Comparison of Automatic Selection of Code on a Set of Stiff Problems to a Code Intended for Such Problems

| | DEBDF | | DEASY | |
|---|---|---|---|---|
| Absolute tolerance | FCN | CP | FCN | CP |
| $10^{-3}$ | 7234 | 9.904 | 7662 | 9.707 |
| $10^{-4}$ | 8628 | 11.875 | 9602 | 12.551 |
| $10^{-5}$ | 10094 | 13.710 | 11185 | 14.573 |
| $10^{-6}$ | 12304 | 17.109 | 12655 | 16.913 |

function evaluations FCN and central processor time CP on a CDC6600. The absolute error tolerance is the same for all components of the systems.

DEASY found that none of these problems had a large Lipschitz constant and correctly chose to integrate with DERKF internally. It did say that D5 has a large Lipschitz constant, which is also correct. This is a problem of two bodies in elliptic motion. At the initial data the eigenvalues of the Jacobian are $\pm\sqrt{2000}$, $\pm i\sqrt{1000}$, and $\|J\|_2 = 2000$. The interval is [0,20]. The problem is unstable at the initial point $x = 0$ and also near 6.5 and 12.7. DEASY uses the BDF code near these points. The results are displayed in Table II. As we have noted, which code is best in a region of instability is not clear. In this instance using the BDF code was acceptable but not optimal.

Enright *et al.* [9] have collected a set of 25 stiff problems. We integrated them except for B2, B3, E2 and E4. As explained in detail in [10], the first three listed are *not* stiff and E4 is poorly posed. The test set [8] accounts for non-stiff problems, so B2, B3 and E2 were not included here. The code DEBDF (not, however, DEASY) gives anomalous results with E4, so it was not included. The results are displayed in Table III.

These numerical results show that even a crude automatic selection of a code can enable a user to ignore the technical details of solution, yet still solve problems effectively. The additional cost of this convenience is modest.

## REFERENCES

1. L. F. SHAMPINE AND H. A. WATTS, Sandia National Laboratories Report SAND76-0585, 1976.
2. C. W. GEAR, "Numerical Initial Value Problems in Ordinary Differential Equations," Prentice-Hall, Englewood Cliffs, N.J., 1971.
3. L. F. SHAMPINE AND M. K. GORDON, "Computer Solution of Ordinary Differential Equations: the Initial Value Problem," Freeman and Co., San Francisco, 1975.
4. A. C. HINDMARSH, Lawrence Livermore Laboratory Report UCID-30001, rev. 3, 1974.

5. A. C. Hindmarsh, *SIGNUM Newsletter* **15** (1980), 10–11.
6. L. F. Shampine and H. A. Watts, Sandia National Laboratories Report SAND79-2374, 1980.
7. L. F. Shampine, *in* "Computational Methods in Nonlinear Mechanics" (J. T. Oden, Ed.), Chap. 19, North–Holland, Amsterdam, 1980.
8. T. E. Hull *et al.*, *SIAM J. Numer. Anal.* **9** (1972), 603–637.
9. W. H. Enright *et al.*, *BIT* 15 (1975), 10–48.
10. L. F. Shampine, Sandia National Laboratories Report SAND80-2772, 1980.