SIAM J. SCI. STAT. COMPUT.
Vol. 4, No. 1, March 1983

0196-5204/83/0401-0010 $01.25/0

AUTOMATIC SELECTION OF METHODS FOR SOLVING STIFF AND NONSTIFF SYSTEMS OF ORDINARY DIFFERENTIAL EQUATIONS*

LINDA PETZOLD†

Abstract. This paper describes a scheme for automatically determining whether a problem can be solved more efficiently using a class of methods suited for nonstiff problems or a class of methods designed for stiff problems. The technique uses information that is available at the end of each step in the integration for making the decision between the two types of methods. If a problem changes character in the interval of integration, the solver automatically switches to the class of methods which is likely to be most efficient for that part of the problem. Test results, using a modified version of the LSODE package, indicate that many problems can be solved more efficiently using this scheme than with a single class of methods, and that the overhead of choosing the most efficient methods is relatively small.

Key words. stiff, nonstiff, ordinary differential equations, initial value problems

1. Introduction. This paper describes a scheme for automatically determining whether an initial value problem $dy/dt = f(y, t)$, $y(t_0) = y_0$, can be solved more efficiently using a class of methods suited for nonstiff problems or a class of methods designed for stiff problems. The decision is based on information which is available at the end of each step of the integration, so that if a problem changes character (i.e., from nonstiff to stiff or vice versa) in the interval of integration, the solver automatically switches to the class of methods which is likely to be most efficient for that part of the problem.

This scheme is useful in several different situations. The user of an ODE solver may not know whether his problem is stiff, or the solver may be called by another code (a package for solving partial differential equations or boundary value problems, for example) where the character of the problem is not known in advance. With the technique described here, the most effective family of methods is chosen automatically. Moreover, many "stiff" problems are often nonstiff in the initial phase, or transient. Integrating through the transient with stiff methods (by a "stiff method," we mean a method designed for stiff problems) is very expensive, whereas nonstiff methods are much better suited for this purpose. As the problem becomes stiff, the code can eventually switch to the stiff methods. In general, problems may be stiff in some intervals and nonstiff in others. This scheme selects the methods that are most efficient for each interval.

Several techniques have been reported in the literature (Shampine [5], [6], [7]) for detecting stiffness. Our objective here is somewhat different, because in addition to detecting stiffness we actually expect the code to shift to the methods which are most appropriate for the problem. Some of the ideas in these papers (especially Shampine [7]) have influenced the approach that was taken here. Shampine [8] outlines a scheme for automatically altering the solution algorithm based on stiffness of the problem for codes based on implicit A-stable formulas. Our scheme is somewhat more general than his in that we automatically select a method from a class of methods where all of the members are not necessarily A-stable.

The basic principles underlying the switching technique are quite simple and are explained in the next section. Some of the difficulties involved in implementing this scheme are described in § 3, along with the approaches we have taken to resolve these problems.

* Received by the editors December 3, 1980, and in revised form September 1, 1981.
† Applied Mathematics Division, Sandia National Laboratories, Livermore, California 94550.

This scheme has been implemented using Adams methods (orders 1–12 with functional iteration to solve the corrector equation) as the family of nonstiff methods, and backward differentiation formulas (BDF) (orders 1–5 with modified Newton iteration) as the family of stiff methods. These seem to be good choices because general purpose codes based on these methods are among the most effective codes available for solving nonstiff and stiff problems, respectively [1], [2], [4]. We have modified the code LSODE, an updated version of the GEAR package written by A. C. Hindmarsh [3], to automatically switch between Adams methods and BDF whenever it is appropriate. In order to illustrate the basic operation of the scheme, some care has been taken to keep the changes to the code to a minimum and to avoid exploiting any properties that are specific to this particular implementation (for example, error estimates based on the Nordsieck vector). It is clear, however, that by exploiting some of the features specific to a particular code, somewhat greater reliability and efficiency of this scheme could be achieved. In addition to a block of code which actually makes the decision and implements the change in method families, several other changes to the code were made so that reliable information about the problem could be obtained at each step. These changes occur in the stepsize and order control logic and in the corrector iteration in the Adams part of the code. In § 4 we describe the results of applying this code to some test problems.

The overhead of making the switching decisions is small. For a truly nonstiff problem, where the method family will never be changed (we always start out using nonstiff methods because they are much cheaper per step and many stiff problems are really nonstiff in the beginning of the interval), this code runs nearly as fast as the unmodified LSODE, using the Adams option with functional iteration. For problems which are stiff in some regions of the interval of integration and nonstiff in others, this code can be much faster than using either Adams methods or BDF over the entire interval. Because the cheaper nonstiff methods are used during the transient of a stiff problem, significantly fewer Jacobian evaluations are required for many stiff problems.

2. Basic strategy. As the integration proceeds our objective is to choose the family of methods which will solve a given problem most efficiently. This decision is made by comparing the method that is currently being used to the method that would be used if the code switched to the other family of methods. To compare the methods, we consider the stepsize that each method could use on the next step, and the cost per step of each method. Since one step of a nonstiff method is typically much cheaper than one step of a stiff method, we should favor using the nonstiff method as long as the stepsizes it uses are not very much smaller than the stepsizes that would be used by the stiff method.

What controls the stepsize for each method? For the nonstiff method several considerations affect the stepsize. First, we must choose a stepsize so that the formula is accurate over the next step (so that a norm of the local truncation error is less than some constant ε). If the code uses functional iteration to solve the corrector equation, the stepsize must be small enough so that the iteration will converge rapidly. Finally, the stepsize must be small enough so that the method will be stable. For the stiff method, the stepsize is chosen so that the formula is accurate over the next step. We will assume that stability and convergence of Newton's method do not restrict the stepsize that could be used by the stiff method. Obviously, this assumption may not always be correct. However, while the stiff method we are currently using may not be stable for the stepsize we would like to use on the next step, another method in

the family of stiff methods probably would be. Hopefully, the order control mechanism would find that method. Skelboe [9] describes an order control strategy to accomplish this; we will not take up this question here. If the stepsize is being controlled by convergence of Newton's method—due, for example, to a very poor approximation to the Jacobian matrix—these assumptions will cause the code to stay with the stiff methods even if they are doing very poorly. We see no good way to avoid this problem.

The first task is to estimate the stepsize that each method could use to achieve the requested accuracy. Let N be the nonstiff method and S the stiff method. If N and S are both linear multistep methods of order q, for example, then we can require the norm of the principal part of the local truncation error to be less than ε. Suppose we are currently using method N with step size h_{CURRENT}, that N is stable for this problem with this stepsize, and that $\|LTE_N\|$ is our estimate for the local truncation error. Then, it is well known that h_N and h_S (the stepsizes that the nonstiff and stiff methods could use on the next step) should satisfy

$$h_N \leqq \left(\frac{\varepsilon}{\|LTE_N\|}\right)^{1/(q+1)} h_{\text{CURRENT}} \tag{1}$$

and

$$h_S \leqq \left(\frac{\varepsilon (C_N/C_S)}{\|LTE_N\|}\right)^{1/(q+1)} h_{\text{CURRENT}}. \tag{2}$$

C_N and C_S are constants depending on the methods N and S.

The stepsize of N may also be affected by considerations such as stability and convergence of functional iteration, so we must find out what effects, if any, these conditions will have. To accomplish this, we need an estimate for $\|\delta f/\delta y\|$ or an estimate of the spectral radius $\rho(\delta f/\delta y)$.

When the stiff method is used, a Jacobian matrix is available and $\|\delta f/\delta y\|$ can be computed directly. The norm used is the matrix norm which is consistent with the vector norm that is used in the code—a weighted norm where the weights depend upon error tolerances. The norm can be computed cheaply (relative to the cost of a matrix factorization) whenever a new Jacobian matrix is formed. Thus, the norm which is available at any given time may correspond to a time several steps back, but it is not likely to be too severely in error because the stiff method reevaluates the Jacobian whenever it changes significantly.

If we are using the nonstiff method, a lower bound for $\|\delta f/\delta y\|$ can be obtained very cheaply during the corrector iteration, using the ideas of Shampine [7]. The bound is formed concurrently with our estimate of the rate of convergence of the iteration. The basic idea is that if the iteration is written as

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

then

$$\frac{\|y^{(k+1)} - y^{(k)}\|}{\|y^{(k)} - y^{(k-1)}\|} \leqq h\gamma \left\|\frac{\delta f}{\delta y}\right\|.$$

The maximum of these ratios,

$$\frac{1}{h\gamma} \frac{\|y^{(k+1)} - y^{(k)}\|}{\|y^{(k)} - y^{(k-1)}\|}$$

obtained over the current step, is a lower bound for $\|\delta f/\delta y\|$. These bounds tend to be quite good (near the spectral radius of $\delta f/\delta y$), although they fluctuate when the dominant eigenvalues of $\delta f/\delta y$ are complex. Unless the difference between two iterates is so small that our estimate would be polluted by roundoff error, we force the nonstiff part of the code to take at least two corrector iterations, in order to generate a lower bound for $\|\delta f/\delta y\|$ on each step.

In any case, suppose a lower bound K for $\|\delta f/\delta y\|$ has been generated or $\|\delta f/\delta y\|$ has been computed directly. Then h_N must be small enough so that functional iteration will converge at a sufficiently rapid rate r, for example $r \leqq \frac{1}{2}$. Thus h_N must satisfy $h_N \gamma \|\delta f/\delta y\| \leqq \frac{1}{2}$, so we will require

$$h_N \leqq \frac{1}{2\gamma K}. \tag{3}$$

Stability also constrains the stepsize for the nonstiff method. If r_q is the radius of the largest half disc contained in the stability region of method N, we must have

$$h_N \rho\left(\frac{\delta f}{\delta y}\right) \leqq r_q,$$

or the computation can become unstable. Thus, we require h_N to satisfy

$$h_N \leqq \frac{r_1}{2K} \tag{4}$$

where K is our lower bound for $\|\delta f/\delta y\|$ and the factor $\frac{1}{2}$ is included so that we can be reasonably sure that h_N would lead to a stable computation (since K is only a lower bound). We actually require h_N to satisfy (4) when computing with the nonstiff method, not just for deciding whether to use the stiff or nonstiff methods. The reasons for this will be discussed in the next section.

Once these estimates have been generated, it is a simple matter to decide whether to use method N or method S. The stepsize that N could use on the next step is the largest h_N that satisfies conditions (1), (3) and (4). The stepsize that S could use is the largest h_S that satisfies (2). Supposing that N is cheaper per step than S, so that we would be willing to take as many as M_+ steps with N for each step that would have to be taken with S, then we will shift to method S (if we are currently using N) if

$$h_S \geqq M_+ h_N. \tag{5}$$

It is important to guard against changing families of methods too frequently, for it might happen that the computation is no longer stable. To avoid this, we stay with method S possibly a little bit longer than what is really optimal; that is, shift from S to N if

$$h_N \geqq M_- h_S. \tag{6}$$

In our code, we have taken $M_- = 1$.

Another factor that is working to prevent the code from shifting back and forth very often is that the constant K which is computed in N is a lower bound for $\|\delta f/\delta y\|$, while in S, $\|\delta f/\delta y\|$ is computed directly. This has a conservative effect in switching from S to N. As a final precaution, we force the code to wait 20 steps after a change in method families before considering a change again. This provides time for the error estimates to settle down after the switch before trying to use them to make another

such decision. The constants M_+ and M_- can be chosen to be functions of the dimension of the system of equations.

3. Implementation considerations. In any practical implementation of a scheme such as the one described above, it is important that the information upon which the code is basing its decisions be reliable, and not misleading. The code should recognize situations where it is not possible to obtain reliable information, and take some appropriate action. In this section we will describe some of the problems involved in ensuring that the information which our scheme requires, namely the local truncation error and a reasonable lower bound for $\|\delta f/\delta y\|$, are reliable, and in recognizing those situations where the information obtained may be misleading.

Several problems with estimating the local truncation error must be dealt with. The first is a problem involving instability. Normally, when the stepsize for the nonstiff method is limited by stability, it tends to oscillate about the largest stable value. (This is explained in detail in Shampine and Gordon [5].) When the stepsize is inside the region of absolute stability of the method, errors are damped. When it is outside the stability region, there is an error growth which causes the error estimates to increase. This eventually brings the stepsize back into the stability region (because the code adjusts the stepsize to keep the error estimate less than ε). In this way, most codes do in some sense detect instability and handle it automatically. When this happens, however, the code has no way of distinguishing whether the error estimates actually reflect the smoothness of the solution or are polluted by terms arising from instability. Thus, instability can make a problem appear to the code to have a solution that is much less smooth than it really is. This is obviously a very undesirable situation for our scheme, which is asking the question after every step, "How smooth is the solution relative to the size of the largest eigenvalues of the problem?"

There would seem to be several ways around this apparent dilemma. The simplest solution is to switch to the stiff methods at the first sign that the stepsize is being limited by stability. Unfortunately, this causes the code to use the stiff methods for many problems that are only marginally stiff and could be solved much more efficiently using the nonstiff methods.

Assuming then that our objective is to use the nonstiff methods as long as the stepsize is not being limited *very* severely by stability, what can we do to ensure that the estimates are not misleading? Because we intend to use the nonstiff method for some time while the problem is stiff, some rather subtle difficulties can occur. For example, even if we are using an A-stable corrector with functional iteration for solving a problem which is becoming stiff, the error estimates may become polluted by terms arising from instability, unless we are very careful about deciding when the corrector iteration has converged. This is because the stability that is of interest here is not the stability of the implicit method by itself (with the corrector solved exactly), but the stability of the method with only a finite, but not necessarily constant, number of corrector iterations.

For example, to illustrate the problems with stability, suppose the trapezoidal method, with functional iteration and automatic stepsize control, is used for solving a stiff problem, and that the corrector iteration is terminated when the norm of the difference between two iterates is less than δ (δ is some constant related to the error tolerance ε, like $\delta = \varepsilon/10$). As the problem becomes more and more stiff, the corrector iteration eventually fails to converge, and the stepsize is reduced. This may happen several times, until finally the stepsize is small enough that the iteration may converge, *not* because the iteration is contracting, but because the difference between the prediction and the first correction is less than δ. There are two ways to interpret this

behavior. If the stepsize is limited by convergence of the iteration, then the error estimate must be less than ε—possibly much less—if it is to be an accurate indication of the smoothness of the solution. However, since the error estimate is based on the difference between the predictor and the corrector, and this difference may be accurate only up to the error δ incurred from terminating the corrector iteration early, there is a limitation on how small an error estimate the code can resolve. Another way of seeing this is the following: As long as the difference between the prediction and the first correction is less than δ, the algorithm we are using is in some sense not the trapezoidal method. It is, instead, a prediction followed by one corrector iteration based on the trapezoidal method. This method is not A-stable. Thus, errors are amplified and the error estimate is misleading. These problems can be avoided if 1) the code is forced to take two corrector iterations per step to estimate a rate of convergence, and 2) the step is rejected if the rate of convergence is not rapid enough, *even though the difference between two successive iterates may be quite small.* This is an implicit limitation on the stepsize of the form $h\|\delta f/\delta y\| \leqq C$, for C some constant, because steps are rejected during the corrector iteration that do not satisfy this criterion. A problem with this strategy is that when convergence of the iteration is limiting the stepsize, there are likely to be many rejections of this type, and this is costly.

The solution that we have adopted for these problems of polluted error estimates is to explicitly limit the step size so as to try to ensure stability. Zlatev and Thomsen [11] employ a strategy of this type to avoid repeated step failures, although their code uses a user-supplied estimate of the magnitude of the largest eigenvalue of the Jacobian. The form that this takes in our modification to LSODE is that when a new stepsize and/or order is selected in the nonstiff part of the code, the stepsize that could be used in the next step for each order is computed as the minimum of the stepsize required for accuracy and the stepsize needed to ensure stability. (We require $hK \leqq r_q/2$, where K is our lower bound for $\|\delta f/\delta y\|$, and r_q is the radius of the largest disc contained in the stability region of the Adams PECE method of order q. There is no good reason for using the PECE stability region here—in fact, it is probably more reasonable to use the intersection of Adams–Moulton stability regions with regions where a rapid rate of convergence of the iteration could be obtained. The code is not very sensitive to these numbers, so long as they are the correct order of magnitude, and they decrease monotonically with the order from order 2 or 3 upwards.) Since this code chooses the order which can use the largest stepsize, an effect of this explicit limitation on the stepsize is that when stability is limiting the stepsize, the order is automatically lowered (unless it is already at second order) because the restrictions are less severe for lower order methods.

Sometimes the error estimate cannot be trusted because it may be polluted by roundoff error. This occurs frequently in three different situations: 1) At the very beginning of the computation, the stepsize is often much smaller than what is needed for accuracy, and it takes some time for the code to increase it. During this time, error estimates are quite small, and may be indistinguishable from zero by roundoff. 2) After passing over a discontinuity, the code may be taking very small steps, while the problem after the discontinuity is very smooth. The same problem as in 1) then occurs. 3) With the limitation on the stepsize to ensure stability, as the problem becomes more and more stiff the error estimates become smaller and smaller. If the tolerance is tight, and/or if the constant M_+ in (5) is relatively large, then the error estimates can be driven down to a level which is sensitive to roundoff. The code should switch to the stiff methods in situation (3), but not in (1) or (2).

The problem of detecting when the error estimate may be polluted by roundoff

does not appear to be trivial. We have taken a simple approach to this problem and some further work is necessary on this topic. We say that the estimate is indistinguishable from zero if the norm of the difference between the predictor and the corrector is less than one hundred times the norm of the predictor times the unit roundoff error of the machine. When this condition is detected, then if the last stepsize chosen had to be reduced to ensure stability, we switch from the nonstiff to the stiff methods.

In order to guard against switching back from the stiff methods immediately after passing this test, a similar test in the reverse direction is necessary. We switch from the stiff to the nonstiff methods if $H_N/H_S \geqq M_-$, and the estimated predictor-corrector difference, for the stepsize that the nonstiff method would use, is not so small as to be indistinguishable from zero as measured by the test described above.

The lower bound for $\|\delta f/\delta y\|$ can also be misleading because of roundoff. This has been noted by Shampine [7], and we use a test similar to his to describe whether to form the bound. If $\|y(k+1)-y(k)\| \leqq 100 \cdot u \cdot \|y(0)\|$, the bound is not formed and the corrector is considered to have converged (the iteration is terminated). Since a recent lower bound is important for our scheme, then if the bound has not been generated recently and the last stepsize chosen had to be reduced to ensure stability, we switch from the nonstiff to the stiff methods.

If the error tolerance ε is so small that the norm of the difference between the predictor and the corrector is forced to be smaller than one hundred times the norm of the predictor times the unit roundoff error of the machine, then the error estimates are indistinguishable from zero, and no lower bounds for $\|\delta f/\delta y\|$ are formed. In this situation it is impossible to tell whether the problem is stiff. To avoid these problems, we double ε if $\varepsilon \leqq 100 \cdot u \cdot \|y\|$ at the start of any step.

Another difficulty with lower bounds for $\|\delta f/\delta y\|$ occurs in problems whose dominant eigenvalues have large imaginary parts. For these problems, the lower bounds can fluctuate between values much smaller than the spectral radius of the Jacobian, and much larger. In response to this problem, and in the interests of being conservative, we use the maximum of all lower bounds generated since the last time a change in stepsize or order was considered (at most, $q+2$ steps). If a lower bound has not been generated during that time, the code may decide to switch to the stiff methods. If the switch is not made, the most recent nonzero maximum is used. These fluctuating estimates can cause problems for the stepsize and order selection mechanisms in the code. For instance, the stepsize may be restricted and the order lowered based on an unusually large estimate, when possibly the next time such a change is considered the estimate is much smaller. Since the earlier computation was probably very stable (because of the large estimate restricting the stepsize), the response of the code to the smaller estimate is likely to be to increase the stepsize and raise the order (this is because, without the effects of instability on the error estimates, high order differences of the solution tend to be smaller than low order differences). K. Stewart [10] has suggested using averages of the lower bounds; this looks like a very good idea, but in the case of wildly fluctuating estimates there still seem to be problems. Either an average that heavily weights large values, or the maximum over a large number of past steps would also help to minimize this difficulty. Order selection algorithms now used in nonstiff codes do not appear to be adequate when the stepsize is restricted to ensure stability and/or convergence of functional iteration. Different order selection algorithms should probably be used in this situation.

4. Practical experience. In this section we will first complete the description of the modifications that were made to LSODE, and then describe the results of using the modified code to solve some problems.

A change from one family of methods to the other is considered after every successful step, unless it has been less than twenty steps since the last switch. The test is skipped if the current method is an Adams method of order greater than five. The reasoning behind this is that the code is not likely to be using such a high order method for solving a stiff problem. If the problem is stiff, then stability will be restricting the stepsize, and the order should be lowered rapidly so that soon the test will be made (there is some danger in this logic, but there have been no problems with this in practice).

The method in the other family that we consider switching to is the one which is of the same order as the method that we are currently using. This is in some ways an arbitrary decision, because information is always available to consider any method with order less than or equal to the current order. Most of the time when the switch is made from Adams to BDF, the order is already quite low, so there is not much choice about which BDF to use. When switching from BDF to Adams there should not be any stability problems with the new method, because the stepsize is restricted to ensure stability. There is a potential source of problems when changing from Adams to BDF, however. The stepsize will generally be increased substantially in this direction, and it is possible that for the stepsize chosen, the BDF may be unstable (especially if the problem has eigenvalues near the imaginary axis). If this happens, it is possible that the code could be led into diagnosing that the problem is less smooth than it really is, and would then switch back to the nonstiff methods. This problem has not been encountered in practice.

The actual switch in method families has been implemented in the most obvious way using the Nordsieck data representation. New method coefficients are calculated, and the old Nordsieck vector is used as if that family of methods had been used all along. Thus, in the first step of the "BDF" after using Adams methods, a true BDF is not really being used, because that would require the polynomial represented by the Nordsieck vector to interpolate past values of the solution, whereas the Adams methods use a polynomial whose derivatives agree with the derivatives of the solution at past times. If the switch is done often, this could cause stability problems, but there is no problem if it is done only a few times. There are several devices in the code which have been described earlier designed to prevent it from thrashing between families of methods. Thrashing has not presented a serious problem in our experience.

The norms used in the code were all changed to weighted l_1 norms, and $\|\delta f/\delta y\|$ is computed in the stiff part of the code with the norm which is consistent with the weighted vector l_1 norm. (The weights are the same as the ones used in the unmodified LSODE, and depend on the error tolerances.) The constant M_+ was taken to be five in the tests described below, and $M_- = 5/M_+ = 1$. The test problems are relatively small (all have dimension less than or equal to 51 and most are much smaller than that) so for these problems our algorithm is conservative about diagnosing some problems as stiff. With these parameters and the range of tolerances used in the tests, the code occasionally runs into the roundoff limitations described earlier.

The modified code was tested on the nonstiff DETEST problems [4] and on the stiff DETEST problems [1], [2]. In addition, we solved van der Pol's equation,

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 2.0, \\ y_2' &= \eta(1-y_1^2)y_2 - y_1, & y_2(0) &= 0.0, \end{aligned} \tag{7}$$

with $\eta = 100.0$ on the interval $[0, 1000]$. This problem was chosen because it alternates between being stiff and nonstiff several times during the interval of integration, so that it is a good illustration of the code's ability to switch back and forth between the

two families of methods. All computations were done in single precision on a CDC 6600 with pure absolute error tolerances. The initial stepsize was determined automatically by the code (using the algorithm in LSODE) in all cases. An initial stepsize of 1.0E-12 was used for all problems. Detailed results for the DETEST problems along with results for unmodified LSODE for the same problems are available from the author. Both codes achieved comparable accuracies for the test problems. Conclusions based on these tests are summarized below.

Very few of the problems of nonstiff DETEST were diagnosed as stiff. The times that this happened the problems were solved in a comparable amount of time or faster (in terms of function evaluations, steps, and execution time) than unmodified LSODE using functional iteration.

On most of the nonstiff problems, the modified code used fewer steps, but more function evaluations, than LSODE with $MF = 10$ (Adams methods with functional iteration). This is mainly a consequence of forcing two corrector iterations per step. The limitations on the stepsize to ensure stability do not appear to seriously limit the efficiency of the code on these problems. The modified code was slightly less efficient than LSODE on the sum total over all of the nonstiff problems. This is to be expected, as there is some price to be paid for making the tests to diagnose stiffness for problems that are not stiff. A summary of results for the nonstiff DETEST problems is shown in Table 1. We have also included in this table the results of using LSODE with $MF = 22$ (BDF with Newton's method using finite-difference Jacobian), as an example of how a code which might be used if the problems were suspected to be stiff would perform on the test problems.

TABLE 1
Nonstiff test problems, summary.

Code	EPS	Exec. time	FCN calls	No. of steps
Switching*	10^{-3}	5.265	7,891	3,234
	10^{-6}	12.041	17,189	7,681
	10^{-9}	24.589	30,987	14,819
	Overall	41.894	56,067	25,734
LSODE ($MF = 10$)†	10^{-3}	4.334	5,412	3,557
	10^{-6}	10.417	11,081	8,948
	10^{-9}	24.243	22,581	19,595
	Overall	38.994	39,074	32,100
LSODE ($MF = 22$)‡	10^{-3}	10.263	8,503	3,909
	10^{-6}	25.488	19,237	10,454
	10^{-9}	61.403	43,926	28,595
	Overall	97.154	71,666	42,958

* The stiff methods in the code use modified Newton iteration with finite-difference generated Jacobian matrices.

† The option $MF = 10$ uses Adams methods with functional iteration.

‡ The option $MF = 22$ uses BDF and modified Newton iteration with finite-difference generated Jacobian matrices.

On the stiff problems, our experience indicates that the tests work very well at loose and moderate tolerances. With EPS $= 10^{-3}$, the code switched to the stiff methods at a reasonable time for every problem, and with EPS $= 10^{-6}$ the results were very

good except for problem E4. At tighter tolerances, there were minor difficulties mainly with B5 and E4. Both of these problems have eigenvalues with relatively large imaginary parts, especially B5 which has eigenvalues $-10 \pm 100i$. So far as we have been able to tell, the difficulties with these problems appear to be related to fluctuating lower bounds for $\|\delta f/\delta y\|$. The code switched back and forth between the two families of methods once on problems E2, EPS $= 10^{-3}$, and F4, EPS $= 10^{-3}$. This is the correct action for problem E2, which is van der Pol's equation (7) with $\eta = 5$. Problem F4 is the Field–Noyes chemical oscillator [2], and the response of the code to this problem appears to be incorrect, although this does not seriously degrade the efficiency of the code (over using LSODE with $MF = 22$).

A better test for deciding when the error estimate is polluted by roundoff would probably increase the reliability of this scheme at tight tolerances, although it performs well already for most problems, and we do not expect many users to ask for stringent tolerances when solving stiff problems. For practically all stiff problems, this new switching technique uses many fewer Jacobian evaluations, and it is definitely more efficient than LSODE ($MF = 22$) at loose tolerances. Since most stiff problems we would expect to see in a practical situation would be somewhat larger than the test problems (all have dimension less than or equal to ten) and would be solved with loose to moderate tolerances, this technique is useful. At stringent error tolerances, our scheme uses more function evaluations than LSODE ($MF = 22$), probably due again to forcing two corrector iterations per step in the nonstiff part of the code (during the transient). A summary of results for the stiff DETEST problems is given in Table 2, and detailed results are available from the author.

TABLE 2
Stiff test problems, summary.

Code	EPS	Exec. time	FCN calls	JAC calls	No. of steps
Switching*	10^{-3}	8.244	8,331	488	3,821
	10^{-6}	21.606	20,676	707	9,984
	10^{-9}	54.213	50,763	1,894	22,751
	Overall	84.063	79,770	3,089	36,556
LSODE ($MF = 22$)†	10^{-3}	14.488	10,143	753	5,368
	10^{-6}	27.414	19,227	1,285	11,136
	10^{-9}	61.564	43,129	2,700	27,103
	Overall	103.466	72,499	4,738	43,607

* The stiff methods in the code use modified Newton iteration with finite-difference generated Jacobian matrices.

† The option $MF = 22$ uses BDF and modified Newton iteration with finite-difference generated Jacobian matrices.

To illustrate how the code performs on a problem which repeatedly changes character during the interval of integration, we have included the results of applying the code to van der Pol's equation (7) with $\eta = 100.0$ on the interval [0, 1000]. A plot of the first component of the solution to this problem is shown in Fig. 1. During the times when the solution is changing rapidly, the problem is nonstiff, and when it is changing more slowly, it is stiff. Statistics on the performance of the modified code for this problem are shown in Tables 3a, b.

TABLE 3
van der Pol's equation test results

	Switched from BDF to Adams		Switched from Adams to BDF	
	Time	Step	Time	Step
EPS = 10^{-6}			.04095	34
	80.79	172	81.25	417
	162.21	555	162.59	680
	162.62	813	162.67	859
	243.64	981	244.09	1190
	325.05	1312	325.42	1423
	325.44	1519	325.50	1583
	406.46	1721	406.84	1846
	406.87	1979	406.92	2025
	487.89	2147	488.34	2356
	569.30	2478	569.75	2711
	650.72	2833	651.17	3042
	732.14	3164	732.59	3381
	813.54	3519	813.92	3644
	813.95	3777	814.00	3823
	894.97	3945	895.42	4154
	976.38	4276	976.76	4387
	976.78	4497	976.84	4546
EPS = 10^{-9}			.06328	73
	80.40	341	81.18	618
	81.19	840	81.28	953
	161.89	1214	162.70	1639
	243.32	1912	244.12	2332
	324.68	2584	325.53	3018
	406.14	3292	406.95	3719
	485.38	3917	485.47	3938
	487.51	4016	488.37	4433
	566.98	4655	569.79	5186
	650.38	5476	651.21	5913
	731.77	6171	732.63	6659
	813.23	6898	814.05	7369
	894.66	7641	895.46	8079
	976.07	7335	976.88	8766

Code	Error tolerance	No. of steps	FCN calls	JAC calls
Switching*	10^{-6}	4565	9311	372
	10^{-9}	8802	17465	456
LSODE† ($MF = 22$)	10^{-6}	5810	9124	707
	10^{-9}	15851	21617	1330

* The stiff methods in the code use modified Newton iteration with finite-difference generated Jacobian matrices.

† The option $MF = 22$ uses BDF and modified Newton iteration with finite-difference generated Jacobian matrices.

We are still making improvements to the switching code. Anyone desiring a copy of the code described in this paper (or possibly an updated version of this code) is encouraged to write to the author.

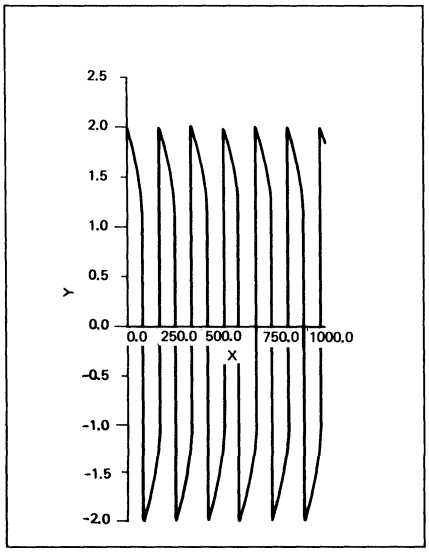

FIG. 1. *Van der Pol equation—first component.*

5. Summary. A scheme has been described for automatically determining whether a system of ordinary differential equations can be solved more efficiently using a class of methods suited for nonstiff problems or a class of methods designed for stiff problems. A code using this new technique is nearly as efficient for solving problems which are known in advance to be nonstiff (stiff) as codes designed for nonstiff (stiff) problems. Switching between families of methods is often more efficient than using a stiff method alone for problems which are nonstiff in some regions of the interval of integration and stiff in other regions. This scheme is useful for solving problems where the character of the problem is not known in advance, because the methods which are likely to be most efficient are selected automatically.

Acknowledgments. The author would like to thank L. F. Shampine for several stimulating discussions on this topic. Discussions with T. H. Jefferson, R. J. Kee, C. W. Gear, K. Stewart, and K. Jackson have also been very helpful in straightening out some of the ideas presented here.

REFERENCES

[1] W. H. ENRIGHT T. E. HULL AND B. LINDBERG, *Comparing numerical methods for stiff systems of ODE's*, BIT, 15 (1975), pp. 10–48.

[2] W. H. ENRIGHT AND T. E. HULL, *Comparing numerical methods for the solution of stiff systems of ODE's arising in chemistry*, in Numerical Methods of Differential Systems, Academic Press, New York, 1976, pp. 45–63.

[3] A. C. HINDMARSH, LSODE *and* LSODI, *two new initial value ordinary differential equation solvers*, ACM SIGNUM Newsletter, 15, 4, Dec. 1980.

[4] T. E. HULL, W. H. ENRIGHT, B. M. FELLEN AND A. E. SEDGWICK, *Comparing numerical methods for ordinary differential equations*, SIAM J. Numer. Anal., 9 (1972), pp. 603–637.

[5] L. F. SHAMPINE AND M. K. GORDON, *Computer Solution of Ordinary Differential Equations*, W. H. Freeman, San Francisco, 1975.

[6] L. F. SHAMPINE, *Stiffness and nonstiff differential equations solvers*, II: *Detecting stiffness with Runge–Kutta methods*, ACM Trans. Math. Software, 3 (1977), pp. 44–53.

[7] ———, *Lipschitz constants and robust ODE codes*, in Computational Methods in Nonlinear Mechanics, North-Holland, Amsterdam, 1980, pp. 427–449.
[8] ———, *Type-insensitive ODE codes based on implicit A-stable formulas*, SAND79-244, Sandia National Laboratories, Livermore, CA, 1979.
[9] S. SKELBOE, *The control of order and steplength for backward differentiation methods*, BIT 17 (1977), pp. 91–107.
[10] K. STEWART, Personal communication, Jet Propulsion Laboratory.
[11] Z. ZLATEV AND P. G. THOMSEN, *Automatic solution of differential equations based on the use of linear multistep methods*, ACM Trans. Math. Software, 5 (1979), pp. 401–414.