Comp. & Maths. with Appls. Vol. 7, No. 6, pp. 487–495, 1981
Printed in Great Britain.

0097–4943/81/060487–09$02.00/0

A TYPE-INSENSITIVE ODE CODE BASED ON SECOND DERIVATIVE FORMULAS

R. SACKS-DAVIS
Department of Computer Science, Monash University, Clayton, Victoria 3168, Australia

and

L. F. SHAMPINE
Applied Mathematics Research Department, Sandia National Laboratories, Albuquerque, NM 87185, U.S.A.

(*Received February* 1981)

Abstract—Several effective codes for the solution of stiff ordinary differential equations (ODEs) are based on second derivative formulas. They are inefficient for non-stiff problems. It is shown how to modify such codes to make them reasonably efficient. The modifications make them more efficient for stiff problems as well. Users do not have the information to determine stiffness reliably. The modified codes recognize the type automatically at each step and respond appropriately. It is a great convenience for a user to have one code efficient regardless of the type.

1. INTRODUCTION

The present generation of codes for solving the initial value problem

$$Y' = f(Y), \qquad Y(x_0) = y_0$$

for a system of ordinary differential equations (ODEs) is clearly divided into two categories. The codes have been designed to solve either stiff or non-stiff problems, but not both. They are very inefficient when applied to the wrong class of problems. Unfortunately, it is difficult to recognize stiffness, and users do not have at their disposal the necessary information. Having to decide the type of the problem is a serious defect in current ODE codes. It is completely impractical to solve stiff problems with classical methods such as Adams and Runge–Kutta, however such problems almost always have regions of sharp change (boundary layers, transition regions) which are not stiff and so are solved inefficiently by methods intended for stiff problems.

The automatic recognition of the type of a problem was seen long ago to be highly desirable, but only recently has much progress been made. The first successful efforts, see, e.g. [1, 2], allow Adams and Runge–Kutta codes to decide whether stiffness is the reason they are performing inefficiently. Though useful, these results fall far short of what is needed. To proceed further, one must decide what kind of numerical method will be used by the code for each type of problem. In our present state of understanding of the solution of differential equations, the solution method for stiff systems is the more critical choice. In [3] the design of codes based on implicit A-stable formulas which can automatically recognize stiffness and alter their algorithm at every step is discussed. Such codes are insensitive to the type of the problem. Unfortunately, even moderately high order implicit A-stable formulas which can be implemented efficiently are an illusive goal of current research. In this paper we proceed in a different way which demands less of the formulas.

The solution of general stiff ODEs seems to require the use of the Jacobian matrix $\partial f/\partial Y$. Some methods use it indirectly in the evaluation of implicit formulas, and others use it directly in the formulas themselves. For the latter to be practical, it is necessary that the Jacobian be convenient to evaluate analytically. This may or may not be the case, but it is easy for a user to decide. The ability to evaluate the Jacobian analytically offers interesting possibilities which we exploit here. Broadly speaking, there are two approaches to type-insensitive codes. One is actually to change methods. A successful approach of this kind is described in [14] for Rosenbrock formulas. The other is to employ only one kind of method, but to make it relatively efficient for both types of problem. This approach was adopted in [3] and is the one we choose here.

Enright[5] and others have considered the use of second derivative formulas for the solution of stiff ODEs. Stiffly stable formulas of orders up to 9 exist in the class. Furthermore, the formulas are rather accurate, with leading error constants varying from 0.167 (order 2) to 0.424×10^{-3} (order 9)[6]. These properties are far superior to the stability and accuracy properties of popular methods like the backward differentiation formulas. They are even comparable in order and accuracy to the formulas popular for non-stiff problems.

In [6] an efficient variable order implementation, SDSTEP, of Enright's formulas is presented for solving stiff systems of equations. In this paper we show how to recognize stiffness automatically at each step in such codes and how to solve non-stiff problems efficiently. The modified code is intended for problems with Jacobians which are convenient to evaluate analytically. Within this class, the code is efficient whether or not the problem is stiff. In the course of our investigation we also note several minor improvements of technique for the solution of stiff problems.

2. METHODOLOGY

We begin by briefly recalling some aspects of the code, SDSTEP, which is the implementation of Enright's formulas described in[6]. At each step an Adams–Bashforth formula of order k is used to calculate a predicted value, p_{n+1}. Then the corrector equation to be solved at each step is

$$y_{n+1} = \beta_0 h_{n+1} f(y_{n+1}) + \gamma_0 h_{n+1}^2 A f(y_{n+1}) + c_{n+1} \tag{2.1}$$

where

$$A = \partial f / \partial Y|_{p_{n+1}},$$

β_0 and γ_0 are the (constant) leading coefficients of the kth order Enright formula listed in[6], h_{n+1} is the current stepsize, and c_{n+1} denotes those terms in the corrector equation based on previously calculated values. Because the code was designed to solve stiff systems of differential equations, the following Newton iteration is used to solve the corrector equation:

$$W_{n+1}(y_{n+1}^{(m+1)} - y_{n+1}^{(m)}) = \zeta^{(m+1)} \tag{2.2}$$

where

$$\zeta^{(m+1)} = -(y_{n+1}^{(m)} - \beta_0 h_{n+1} f(y_{n+1}^{(m)}) - \gamma_0 h_{n+1}^2 A f(y_{n+1}^{(m)}) - c_{n+1}), \tag{2.3}$$

and

$$W_{n+1} = I - \beta_0 h_{n+1} \tilde{A} - \gamma_0 h_{n+1}^2 \tilde{A}^2.$$

Here $\tilde{A}$ is an approximation to the Jacobian calculated possibly at a previous step.

As we have noted, the basic formulas have good numerical properties. What makes them inefficient for non-stiff problems? The Newton iteration is expensive, but necessary to the evaluation of the implicit formula for stiff problems. However, one ordinarily uses a cheaper iteration for non-stiff problems, namely simple (functional) iteration:

$$y_{n+1}^{(m+1)} = \beta_0 h_{n+1} f(y_{n+1}^{(m)}) + \gamma_0 h_{n+1}^2 A f(y_{n+1}^{(m)}) + c_{n+1}. \tag{2.4}$$

Then y_{n+1} is a point of attraction if

$$\rho(\beta_0 h_{n+1} J + \gamma_0 h_{n+1}^2 A J) < 1$$

where

$$J = \partial f / \partial Y|_{y_{n+1}}$$

and $\rho(M)$ denotes the spectral radius of the matrix M [7]. However, the code uses a specific norm and requires rapid convergence of the iterative scheme, so that the practical requirement for convergence is that

$$\|\beta_0 h_{n+1} \partial f/\partial Y + \gamma_0 h_{n+1}^2 A \partial f/\partial Y\| \leq r < 1$$

hold in a region containing y_{n+1}. Then the iteration will contract in the chosen norm at rate at least r. We shall use this condition and the approximation of $\partial f/\partial Y$ by A to decide when simple iteration is feasible, but we want the test to be as cheap as possible. Even forming A^2 is to be avoided, still more so forming the whole matrix in the condition. A cheaper sufficient condition is to form $\|A\|$ and test

$$|\beta_0| \, \|h_{n+1}A\| + |\gamma_0| \, \|h_{n+1}A\|^2 \leq r \tag{2.5}$$

for a rate r chosen as being sufficiently fast. Suppose $r = 5/8$. All the leading coefficients, β_0 and γ_0, used by the code SDSTEP satisfy $|\beta_0| \leq 1$ and $|\gamma_0| \leq \frac{1}{2}$. Thus the condition (2.5) is certainly satisfied if the simpler condition

$$\|h_{n+1}A\| \leq \tfrac{1}{2}. \tag{2.6}$$

holds. This is exactly the kind of condition on the step size and Jacobian that is usually interpreted as meaning that the problem is non-stiff for this step.

The type of a problem should not change often, and we would like to keep the overhead of testing the type as low as possible. The strategy we have implemented is to use a $\|A\|$ computed at an earlier step in (2.6) if we can. The important thing is that we have an accurate value of $\|A\|$ when $\|h_{n+1}A\|$ is close to the critical value of $\frac{1}{2}$. Thus if $\|A\|$ was computed at an earlier step and if $\|h_{n+1}A\|$ lies in the interval $[\frac{1}{4}, 1]$, we recalculate $\|A\|$ at the current step for a sharp test. Otherwise, there seems to be no real need for a very accurate value of $\|A\|$. In any event a current value of $\|A\|$ is calculated at most 5 steps after the previous calculation. The norm used in the code is the weighted maximum norm.

The amount of work needed for each Newton iteration is substantially greater than that for simple iteration. The calculation of the iteration matrix, W_{n+1}, requires a matrix squaring and subsequent LU decomposition. In the previous version of the code, W_{n+1} is recalculated each time the stepsize or order is changed or when the corrector iterations are converging too slowly. It is to be expected that the use of simple iteration over a number of steps will save a number of these expensive computations. Furthermore, each Newton iteration requires a forward and back substitution to solve the system of linear equations (2.2). Note that simple iteration corresponds to taking $W_{n+1} = I$ in (2.2), so that incorporation of simple iteration into the code is easily achieved by skipping the blocks in the code where W_{n+1} is calculated and where the linear equations are solved.

A matter of practical importance is the convergence criteria used in the corrector iteration. In order to maintain numerical stability when solving stiff systems, it is necessary to iterate until convergence when solving the corrector equation. In the code SDSTEP iteration continues until

$$\|y_{n+1}^{(m+1)} - y_{n+1}^{(m)}\| \leq 4u \|y_{n+1}^{(1)}\|$$

where u is the unit relative round-off error bound, or until

$$\|y_{n+1}^{(m+1)} - y_{n+1}^{(m)}\| \leq C_1 \epsilon \quad \text{and} \quad \|f(y_{n+1}^{(m+1)}) - f(y_{n+1}^{(m)})\| \leq C_2 \epsilon$$

where C_1 and C_2 are small scalars and ϵ is the local error tolerance. On the other hand, the stability difficulties associated with stiff systems do not arise when solving non-stiff equations. Consequently, many codes designed for non-stiff ODE's take a fixed number of iterations when solving the corrector equation. Examples of such explicit methods are the well known Adams PECE codes. Although smaller stepsizes may be required to achieve a given accuracy if a fixed

number of corrector iterations is used, each step of the integration is very cheap. The numerical results we have obtained when solving non-stiff equations have indicated that it is cost-effective to adopt this approach and fix the number of corrector iterations to one.

There is another situation when only one iteration is required. When using Newton iteration to solve the corrector equation, the strategy adopted in SDSTEP is to recalculate the iteration matrix, W_{n+1}, if the stepsize or order has changed since the last time Newton iteration was used. Thus for linear systems of ODEs, W_{n+1} is equal to the exact Newton iteration matrix and convergence is achieved in one iteration.

Simple iteration is a traditional way of evaluating the corrector equation when solving non-stiff problems. A new way is suggested by noting that Newton's iteration has

$$W_{n+1} = I - (\beta_0 h_{n+1} A + \gamma_0 h_{n+1}^2 A^2) = I + 0(\|h_{n+1}A\|).$$

This says that simple iteration can be regarded as an approximation to W_{n+1} valid for small $\|h_{n+1}A\|$. In this point of view it is natural to write

$$W_{n+1} = I - \beta_0 h_{n+1} A + 0(\|h_{n+1}A\|^2)$$

so that

$$W_{n+1}^{-1} = I + \beta_0 h_{n+1} A + 0(\|h_{n+1}A\|^2).$$

This suggests that one use the iteration

$$y_{n+1}^{(m+1)} - y_{n+1}^{(m)} = [I + \beta_0 h_{n+1} A]\zeta^{(m+1)} \tag{2.7}$$

where $\zeta^{(m+1)}$ is defined by (2.3). The same analysis as for simple iteration leads to the condition

$$|\gamma_0 + \beta_0^2| \, \|h_{n+1}A\|^2 + |\beta_0 \gamma_0| \, \|h_{n+1}A\|^3 \leq r.$$

Since $|\gamma_0|$ is rather smaller than $|\beta_0|$ for the formulas considered, the rate of simple iteration is roughly $|\beta_0| \, \|h_{n+1}A\|$ and that of this new iteration, $\beta_0^2 \|h_{n+1}A\|^2$. That is, the new iteration is equivalent to about two iterations of simple iteration. The new iteration has an extra matrix–vector multiplication so one iteration with it requires 1 function evaluation and 2 matrix–vector multiplications. Two iterations with simple iteration require 2 function evaluations and two matrix–vector multiplications.

The choice of iterative scheme for solving the corrector equation also affects the error estimates used in the code. In the previous version of the code, the estimate, E_1, of the local error was given by

$$E_1 = W_{n+1}^{-1}\{h_{n+1} g_{n+1}(y'_{n+1} - p'_{n+1}) - (y_{n+1} - p_{n+1})\}.$$

Here, p_{n+1} and p'_{n+1} are the predicted values of the solution and its derivative at x_{n+1}, g_{n+1} is a scalar and $y'_{n+1} = f(y_{n+1})$. E_1 may be interpreted as the difference between y_{n+1} and a higher order approximation, y^+_{n+1}, to the solution at x_{n+1}. The value y^+_{n+1} is obtained by solving a corrector equation of one higher order than the current order using one step of a modified Newton iteration. When we replace Newton iteration by simple iteration or by the iteration (2.7), we replace E_1 by

$$E_2 = h_{n+1} g_{n+1}(y'_{n+1} - p'_{n+1}) - (y_{n+1} - p_{n+1})$$

and

$$E_3 = (I + h_{n+1}\beta_0 A)\{h_{n+1} g_{n+1}(y'_{n+1} - p'_{n+1}) - (y_{n+1} - p_{n+1})\}$$

respectively. Thus, we solve the corrector equation of higher order using the same iterative

scheme as is used when solving for y_{n+1}. E_2 is the estimate cheapest to evaluate. E_1 requires an extra forward and back substitution and E_3 requires an extra matrix–vector multiplication, making E_1 and E_3 comparable in cost.

It is not clear from the arguments presented which of the two iteration should be preferred for steps treated as non-stiff. If one expected to do several iterations at each step, it would probably be cheaper to use simple iteration in terms of overhead and more expensive in terms of function evaluations. We have already remarked that it appears to be cost-effective to do only one corrector iteration when the problem is non-stiff. This leads one to favor simple iteration. Some experiments we report in the next section support this choice.

The selection of the initial step size is crucial to the reliable solution of an ODE because it specifies the scale of the problem to the code. Modern codes assist the user in this matter. In the code SDSTEP, the first step is taken with a second order formula. For a given initial stepsize, h_0, the code estimates that the local error will be $\|h_0^3 Y'''(x_0)/6\|$. The term $Y'''(x_0)$ is approximated by divided differences at the cost of one extra function evaluation. It is useful to modify this initial stepsize selection so that the first step is treated as non-stiff. This ensures that the code will keep track of how fast the solution can change at the initial point. In view of the previous discussion, all that is required is to add another bound on the stepsize selection so as to ensure that $\|h_0 A\| \leq \frac{1}{2}$. This kind of protection is invaluable when the code might be presented a stiff problem. Such problems ordinarily have a short initial interval of very rapid change (a boundary layer, transition region). An initial stepsize which is out of scale might lead the code to miss this important behaviour entirely and cause the code to track the wrong integral curve after the transient.

3. NUMERICAL RESULTS

The following problem

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

$$\frac{dy}{d\theta} = \frac{(1+\xi)}{\xi} N_f \left\{x-\frac{y}{y+K(1-y)}\right\}, \tag{3.1}$$

arises in the study of percolation processes[8]. The chemical engineers who posed it were interested in a range of values for the parameters ξ, N_f and K. Because the problem is stiff for some parameter values and is non-stiff for others, it illustrates the convenience of a code which does not ask the user to decide the type (see also [4]). We present results for the following three problems on the interval $0 \leq \theta \leq 2$

PROBLEM 1

$$K = 5, \quad \xi = 0.1, \quad N_f = 0.1,$$

PROBLEM 2

$$K = 5, \quad \xi = 5, \quad N_f = 5,$$

PROBLEM 3

$$K = 5, \quad \xi = 500, \quad N_f = 50.$$

Problem 1 is non-stiff whereas Problems 2 and 3 are stiff on portions of the interval. The stiffness ratios (for $y = 0$) for the three problems are approx. 6,200 and 13,000 respectively. The results obtained after solving the problems by the code SDSTEP of[6] are presented in Table 1. The table headings denote the following.

TOL: local error tolerance. An absolute error criterion was used.
STEPS: number of steps.
FN. EVALS: number of function evaluations.
JAC. EVALS: number of Jacobian evaluations.
LU DEC: number of LU decompositions.

MATRIX MULTS: number of matrix multiplications.
MATRIX/VECTOR MULTS: number of "equivalent" matrix–vector multiplications. This is equal to the number of matrix–vector multiplications plus the number of forward/back substitutions plus the number of times a constant multiple of one matrix is added to a second matrix (which is required each time a new iteration matrix is formed).
MATRIX NORMS: number of matrix norms.
ACC. DIGITS: negative logarithm to the base 10 of the absolute global error at the end-point.

Table 1. Performance of code SDSTEP on the chemical engineering problems (3.1)

PROBLEM	TOL	STEPS	FN EVALS	JAC EVALS	LU DEC	MATRIX MULTS	MATRIX/ VECTOR MULTS	MATRIX NORMS	ACC. DIGITS
1	10^{-2}	9	26	10	9	9	48	0	3.7
	10^{-4}	13	48	16	11	11	88	0	5.3
	10^{-6}	22	82	26	14	14	149	0	7.5
	10^{-8}	33	99	35	17	17	191	0	8.9
2	10^{-2}	14	54	16	15	15	103	0	2.8
	10^{-4}	27	104	29	19	19	195	0	4.8
	10^{-6}	47	170	50	26	26	313	0	7.2
	10^{-8}	86	252	89	37	37	485	0	9.1
3	10^{-2}	31	146	38	37	37	282	0	3.0
	10^{-4}	36	153	38	32	32	297	0	5.1
	10^{-6}	64	254	71	53	53	507	0	8.7
	10^{-8}	98	352	103	50	50	708	0	8.6

The code SDSTEP was modified to solve the corrector equation using both simple iteration and Newton iteration as described in Section 2. When using simple iteration, the number of corrector iterations was fixed at one. We shall refer to this version of the code as MODIFIED SDSTEP. The results appear in Table 2.

Observe that as the problems become more stiff, more steps are taken for which Newton iteration is used to solve the corrector equation. This is indicated by the increasing number of LU decompositions and matrix multiplications as one goes from Problem 1 to Problem 3. A comparison with Table 1 shows the large savings made when simple iteration is used on the non-stiff steps. As well as decreasing the number of LU decompositions and matrix multiplications, the use of simple iteration saves a number of forward/back substitutions (as indicated in the column matrix/vector multiplications).

The improvement in performance gained by using simple iteration on the non-stiff steps that was observed in Problems 2 and 3 is fairly representative of the results we obtained after testing the codes on other stiff problems. To gain an idea of the performance of the code on non-stiff problems, we consider the following orbit problem:

$$\begin{aligned} y_1' &= y_3, & y_1(0) &= 1-\epsilon \\ y_2' &= y_4, & y_2(0) &= 0, \\ y_3' &= -y_1/(y_1^2+y_2^2)^{3/2}, & y_3(0) &= 0, \\ y_4' &= -y_2/(y_1^2+y_2^2)^{3/2}, & y_4(0) &= \sqrt{\left(\frac{1+\epsilon}{1-\epsilon}\right)}, \end{aligned}$$

$$\epsilon = 0.3, \qquad 0 \le x \le 20. \tag{3.2}$$

Table 2. Performance of MODIFIED SDSTEP on the chemical engineering problems (3.1)

PROBLEM	TOL	STEPS	FN EVALS	JAC EVALS	LU DEC	MATRIX MULTS	MATRIX/ VECTOR MULTS	MATRIX NORMS	ACC. DIGITS
1	10^{-2}	8	21	9	2	2	22	3	3.1
	10^{-4}	14	34	16	2	2	27	11	4.3
	10^{-6}	24	58	29	0	0	28	7	6.5
	10^{-8}	37	86	43	0	0	42	9	8.2
2	10^{-2}	14	54	16	14	14	114	4	2.7
	10^{-4}	31	101	33	12	12	160	15	5.0
	10^{-6}	59	166	61	14	14	226	28	6.3
	10^{-8}	97	265	102	13	13	331	54	9.1
3	10^{-2}	31	146	38	37	37	319	8	3.0
	10^{-4}	41	174	45	31	31	344	16	4.3
	10^{-6}	67	246	71	38	38	474	28	6.8
	10^{-8}	110	360	116	37	37	591	53	8.5

For a comparison, we also solved the problem with the Adams code STEP[9], a code designed specifically for non-stiff ODEs. The results appear in Tables 3 and 4.

It is observed that MODIFIED SDSTEP solves the non-stiff problem quite efficiently. Its performance is comparable to that of the routine STEP on those problems for which all the steps taken by MODIFIED SDSTEP use simple iteration (rather than Newton iteration) to solve the corrector equation (tolerances 10^{-8} and 10^{-10} in our example). On these problems ithe only extra costs in MODIFIED SDSTEP are those inherent in Enright's formulas (the Jacobian evaluations and matrix/vector multiplications that are required to calculate the second derivative) and the matrix norms required to decide which iteration should be used to solve the corrector equation. As the proportion of steps taken with simple iteration decreases, the performance of MODIFIED SDSTEP relative to STEP deteriorates.

In order to test the codes on a larger set of non-stiff problems we considered the one parameter family of two body problems (3.1) for which $\epsilon = 0.0$, 0.1, 0.3, 0.5, 0.7 and 0.9. For $\epsilon = 0.0$ the orbit defined by (3.1) is circular, whilst for $\epsilon > 0$ the initial conditions cause the orbit to be an ellipse with eccentricity ϵ. Each of these problems was solved at 5 tolerances, giving 30 problems in all.

The performance of MODIFIED SDSTEP can be seen in Table 5. As the local error tolerance becomes more stringent, relatively more steps are taken using simple iteration so that

Table 3. Performance of MODIFIED SDSTEP on the orbital problem (3.2)

TOL	STEPS	FN EVALS	JAC EVALS	LU DEC	MATRIX MULTS	MATRIX/ VECTOR MULTS	MATRIX NORMS	ACC. DIGITS
10^{-2}	61	241	75	21	21	400	35	0.2
10^{-4}	113	358	129	22	22	494	87	2.5
10^{-6}	191	495	208	13	13	485	129	3.9
10^{-8}	284	608	304	0	0	303	131	6.5
10^{-10}	425	890	445	0	0	444	94	7.3

Table 4. Performance of the Adams code STEP on the orbital problem (3.2)

TOL	STEPS	FN EVALS	JAC EVALS	LU DEC	MATRIX MULTS	MATRIX/ VECTOR MULTS	MATRIX NORMS	ACC. DIGITS
10^{-2}	65	134	0	0	0	0	0	---
10^{-4}	119	242	0	0	0	0	0	2.5
10^{-6}	209	423	0	0	0	0	0	4.8
10^{-8}	318	641	0	0	0	0	0	5.6
10^{-10}	433	870	0	0	0	0	0	7.9

Table 5. Percentage of steps taken by MODIFIED SDSTEP using simple iteration as functions of TOL and ϵ

TOL	STEPS	% STEPS SIMPLE ITER	ϵ	MAX OBSVD NORM	STEPS	% STEPS SIMPLE ITER
10^{-2}	443	33	0.0	2.6	549	77
10^{-4}	923	51	0.1	3.8	777	87
10^{-6}	1470	69	0.3	7.3	1074	89
10^{-8}	2187	80	0.5	19.8	1380	87
10^{-10}	3268	87	0.7	90.6	1824	76
			0.9	2428.3	2687	58

MODIFIED SDSTEP is most competitive with STEP at these tolerances. Also, as the parameter ϵ increases, the orbits become more eccentric and large norms are observed in solving the problems. As the values of the maximum observed norm increase, relatively fewer steps are taken with simple iteration and MODIFIED SDSTEP becomes less efficient relative to STEP on these problems. It is interesting to note that the large norms here are due to instability rather than stiffness. When one body approaches closely the other, its location strongly affects the rest of its orbit. It is not clear what kind of method is best for an unstable problem, but high orders seem attractive and this is a strength of SDSTEP. For more details see [2].

An alternative to simple iteration for the non-stiff steps was discussed in Section 2. To study the performance of this approach, the iteration (2.7) was incorporated into the code SDSTEP. The results obtained after testing the resulting code on the orbital problem (3.2) appear in Table 6. The iteration (2.7) has a faster rate of convergence than simple iteration. A comparison with Table 3 indicates that a number of steps (consequently function evaluations and Jacobian

Table 6. Performance of the code which uses the iteration (2.7) on the non-stiff steps in solving (3.2)

TOL	STEPS	FN EVALS	JAC EVALS	LU DEC	MATRIX MULTS	MATRIX/ VECTOR MULTS	MATRIX NORMS	ACC. DIGITS
10^{-2}	56	228	68	21	21	420	35	---
10^{-4}	109	336	122	19	19	647	100	2.2
10^{-6}	168	466	190	18	18	883	144	4.3
10^{-8}	257	568	284	0	0	849	142	6.4
10^{-10}	388	830	415	0	0	1242	95	8.1

evaluations) may be saved with the new method. However, because the iteration (2.7) is more expensive than simple iteration there is a significant increase in matrix/vector multiplications. Similar tradeoffs were observed during more estensive testing, but overall there was no improvement in the results over MODIFIED SDSTEP.

In conclusion, the results obtained with MODIFIED SDSTEP indicate that the new code is more efficient than the previous version when solving stiff ODEs because it takes advantage of portions of the interval that are non-stiff. In addition it solves non-stiff problems rather efficiently, especially at stringent tolerances.

REFERENCES

1. L. F. Shampine, Stiffness and non-stiff differential equation solvers—II: detecting stiffness with Runge–Kutta methods. *ACM Trans. Math. Software* **3**, 44–53 (1977).
2. L. F. Shampine, Lipschitz constants and robust ODE codes, In *Computational Methods in Nonlinear Mechanics* (Edited by J. T. Oden). North-Holland, Amsterdam (1980).
3. L. F. Shampine, Type-Insensitive ODE codes based on implicit *A*-stable methods. *Math. Comp.*, to appear.
4. L. F. Shampine, Implementation of Rosenbrock methods. SAND80-2367J, Sandia National Laboratories, Albuquerque, New Mexico (1980).
5. W. H. Enright, Second derivative multistep methods for stiff ordinary differential equations. *SIAM J. Numer. Anal.* **11**, 321–331 (1974).
6. R. Sacks-Davis, Fixed leading coefficient implementation of SD-formulas for stiff ODEs. *ACM Trans. Math. Software* **6**, 540–562 (1980).
7. J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, New York (1970).
8. A. Rodrigues and E. C. Beira, Staged approach of percolation processes. *AIChE J.* **25**, 416–423 (1979).
9. L. F. Shampine and M. K. Gordon, *Computer Solution of Ordinary Differential Equations.* Freeman, San Francisco, California (1975).