SIAM J. SCI. STAT. COMPUT.
Vol. 3, No. 3, September 1982

0196-5204/82/0303-0007 $01.00/0

DIFFERENTIAL/ALGEBRAIC EQUATIONS ARE NOT ODE'S*

LINDA PETZOLD†

Abstract. This paper outlines a number of difficulties which can arise when numerical methods are used to solve systems of differential/algebraic equations of the form $\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = \mathbf{0}$. Problems which can be written in this general form include standard ODE systems as well as problems which are substantially different from standard ODE's. Some of the differential/algebraic systems can be solved using numerical methods which are commonly used for solving stiff systems of ordinary differential equations. Other problems can be solved using codes based on the stiff methods, but only after extensive modifications to the error estimates and other strategies in the code. A further class of problems cannot be solved at all with such codes, because changing the stepsize causes large errors in the solution. We describe in detail the causes of these difficulties and indicate solutions in some cases.

Key words. ordinary differential equations, singular differential systems, stiff, numerical methods, error estimates, backward differentiation formulas

1. Introduction. A number of difficulties can arise when numerical methods are used to solve systems of differential/algebraic equations (DAE) of the form $\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = \mathbf{0}$. These problems look much like standard ordinary differential equation (ODE) systems of the form $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$ (and of course include these systems as a special case), and many of the DAE systems can be solved using numerical methods which are commonly used for solving stiff systems of ODE's. However, the class of DAE systems also includes problems with properties that are very different from those of standard ODE's. Some of these problems cannot be solved using variable-stepsize stiff methods such as backward differentiation formulas (BDF). Others can be solved using such methods but only after substantial modifications to the strategies usually used in codes implementing those methods. In this paper we explore the causes of the difficulties and describe modifications which enable codes based on BDF to solve a wider class of problems than were previously possible. Additionally, we suggest strategies for detecting the problems which cannot be solved with this technique.

Several authors [1], [2], [3], [4], [5], [6], [7] have written codes designed to deal with either DAE systems of the form

$$\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = \mathbf{0} \tag{1.1}$$

or special cases of this general problem. These codes are based on a technique which was introduced by Gear [1]. The idea of this technique is that the derivative $y'(t)$ can be approximated by a linear combination of the solution $y(t)$ at the current mesh point and at several previous mesh points. For example, $y'(t)$ may be approximated by BDF. The simplest method for solving differential/algebraic systems is the first order BDF, or backward Euler method. In this method the derivative $y'(t_{n+1})$ at time t_{n+1} is approximated by a backward difference of $y(t)$, and the resulting system of nonlinear equations is solved for $\mathbf{y}_{n+1}$,

$$\mathbf{F}(t_{n+1}, \mathbf{y}_{n+1}, (\mathbf{y}_{n+1} - \mathbf{y}_n)/(t_{n+1} - t_n)) = 0. \tag{1.2}$$

In this way, the solution is advanced from time t_n to time t_{n+1}. In this report we will assume that $\mathbf{y}(t_0)$ is known.

We investigate the behavior of the backward Euler method for solving systems of the form (1.1) in detail because it is the simplest member of several classes of methods

* Received by the editors April 6, 1981, and in revised form October 23, 1981.
† Applied Mathematics Division, Sandia National Laboratories, Livermore, California 94550.

which could conceivably be used for solving systems of the form (1.1). Even this simple method exhibits several serious difficulties when used in attempting to solve certain types of differential/algebraic problems.

All of the difficulties that we describe occur in solving simple linear problems. These problems are much more easily understood than nonlinear problems, and we hope that an understanding of the linear models will provide a plan for action in the nonlinear case. It is likely that difficulties in solving nonlinear problems are at least as great as for related linear problems. Thus, in the first few sections of this report we will be concerned only with linear problems. Later sections examine how the difficulties which occur in solving certain linear systems might also affect strategies for solving the general nonlinear problem (1.1).

We summarize in § 2 results of Sincovec et al. [2] on the decomposition of linear differential/algebraic systems of the form

$$E\mathbf{y}' = A\mathbf{y} + \mathbf{g}(t) \tag{1.3}$$

into canonical subsystems and on the properties of solutions of these subsystems. The structure of linear DAE systems can be characterized by a parameter m called the nilpotency of the system. This is important because the type of numerical difficulties which can be expected depends on the nilpotency of the system to be solved. Standard ODE systems have nilpotency $m = 0$.

In § 3 we describe the difficulties which arise in solving some of these canonical subsystems using the backward Euler method with varying stepsizes. We find that problems of nilpotency $m \leqq 2$ can be solved by codes based on variable-stepsize BDF; however, the usual error estimates which are proportional to the difference between the predictor and the corrector are grossly inaccurate. Although the error in the solution tends to zero as the current stepsize is reduced, these error estimates tend to a positive nonzero limit. This causes codes using these estimates to fail unnecessarily on many ($m = 2$) systems. Unfortunately, the situation is much less hopeful for systems of nilpotency $m \geqq 3$, where varying the stepsize can lead to totally incorrect answers. Worse yet, error estimates which have been proposed [2] or used [3] in some codes would allow wrong answers to be computed for these problems, with absolutely no warning. We know of no techniques for handling these ($m \geqq 3$) problems which do not destroy the structure and sparseness of systems written in the form (1.3).

The remainder of the paper is devoted to techniques for solving problems of nilpotency $m \leqq 2$. New error estimates are derived in § 4 which enable codes to solve this extended ($m \leqq 2$) class of problems reliably. In § 5 we take up some practical issues which are of importance in codes for solving nonlinear DAE systems. In particular, strategies for deciding when the Newton iteration has converged and for detecting problems of nilpotency $m \geqq 3$ (i.e., those which cannot be solved by variable-step BDF) are discussed. We make some recommendations in those areas, but there is still much work to be done.

2. Linear differential/algebraic systems. This section reviews the structure of linear differential/algebraic systems and the properties of solutions of these systems. The results discussed in this section are derived and explained in greater detail by Sincovec et al. [2]. We summarize the main points here because they are necessary background for the understanding of the remainder of this report.

The system we consider in this and the next section is

$$E\mathbf{y}' = A\mathbf{y} + \mathbf{g}(t), \qquad \mathbf{y}(t_0) = \mathbf{y}_0. \tag{2.1}$$

Gantmacher [8] has given a complete analysis of the general matrix pencil, $A-\lambda E$, where A and E are $N\times N$ matrices and A or E or both can be singular. The key result in [8] and [2] is that there exist nonsingular matrices P and Q which reduce the matrix pencil $A-\lambda E$ to a canonical form. When P and Q are applied to (2.1), we obtain

$$PEQQ^{-1}\mathbf{y}' = PAQQ^{-1}\mathbf{y} + P\mathbf{g}(t). \tag{2.2}$$

System (2.2) is composed of five types of uncoupled canonical subsystems. Three of the types correspond to cases where no solutions exist or infinitely many solutions exist. It is not even reasonable to try to solve these problems numerically. Fortunately, codes based on BDF reject these problems almost automatically, because the iteration matrix $E-h\beta A$, (where h is the stepsize and β is a scalar which depends on the method and recent stepsize history) is singular for all values of $h\beta$. The remaining two types of canonical subsystems correspond to the case where $A-\lambda E$ is a regular matrix pencil, that is, $\det(A-\lambda E)$ is not identically zero. In [2], these systems are called "solvable" because solutions to the differential/algebraic equation exist and two solutions which share the same initial value must be identical. In what follows, we will deal only with systems where the matrix pencil is regular.

For solvable systems (2.2) is equivalent to

$$\mathbf{y}_1'(t) = E_1\mathbf{y}_1(t) + \mathbf{g}_1(t), \qquad \mathbf{y}_1(t_0) = \mathbf{y}_{1,0}, \tag{2.3a}$$

$$E_2\mathbf{y}_2'(t) = \mathbf{y}_2(t) + \mathbf{g}_2(t), \qquad \mathbf{y}_2(t_0) = \mathbf{y}_{2,0}, \tag{2.3b}$$

where

$$Q^{-1}\mathbf{y}(t) = \begin{bmatrix}\mathbf{y}_1(t)\\ \mathbf{y}_2(t)\end{bmatrix}, \qquad P\mathbf{g}(t) = \begin{bmatrix}\mathbf{g}_1(t)\\ \mathbf{g}_2(t)\end{bmatrix}$$

and E_2 has the property that there exists an integer m such that $E_2^m = 0$, $E_2^{m-1} \neq 0$. The value of m is defined to be the nilpotency of the system. The matrix E_2 is always composed of Jordan blocks of the form

$$\begin{bmatrix} 0 & 1 & & \\ & 0 & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix}, \tag{2.4}$$

and m is the size of the largest of these blocks.

The behavior of numerical methods for solving standard ODE systems of the form (2.3a) is well understood, and will not be elaborated upon here. Since the subsystems are completely uncoupled and the methods we are interested in are linear, it suffices for understanding (2.1) to study the action of numerical methods on subsystems of the form (2.3b), where E_2 is a single block of form (2.4). When E_2 is a matrix of form (2.4) and size n, the system (2.3b) will be referred to as a canonical nonstate ($m = n$) subsystem.

Let us now take a closer look at one of these canonical nonstate subsystems. For example, the simplest ($m = 2$) system is

$$y_2'(t) = y_1(t) + g_1(t), \qquad 0 = y_2(t) + g_2(t). \tag{2.5}$$

This system has the solution

$$y_2(t) = -g_2(t), \qquad y_1(t) = -g_1(t) - g_2'(t). \tag{2.6}$$

We will deal with this example repeatedly in this report. This system differs from conventional ODE systems in several significant ways. First, note that the solution depends on g (and its derivatives) at the current time only. That is, it does not depend on the initial value or on the past history of g. Also, observe that the solution to (2.5) depends on the derivative of $g_2(t)$. Thus, if $g_2(t)$ is differentiable but not continuously differentiable, then $y_1(t)$ is discontinuous. Numerical methods have a great deal of difficulty in dealing with this situation. In view of the obvious differences between nonstate systems and state systems (standard ODE systems), it is not surprising that methods designed to deal with standard ODE systems should experience so many problems with the nonstate systems. In fact, what is surprising is that these methods actually can solve some of the nonstate systems, as we will show later. For the general nonstate canonical subsystem (2.3b) of nilpotency m, the solution is given by

$$\mathbf{y}_2(t) = -\sum_{i=0}^{m-1} E_2^i \mathbf{g}_2^{(i)}(t), \tag{2.7}$$

and it is easy to see that this system shares the properties described above.

One other point to be made before we move on to numerical methods is that differential/algebraic systems are very similar to stiff systems. For example, if $\varepsilon > 0$ small, then

$$(E - \varepsilon A)\mathbf{z}'(t) = A\mathbf{z}(t) + \mathbf{g}(t) \tag{2.8}$$

is a stiff system near to (2.1). Thus, we expect that if the underlying differential/algebraic system ($\varepsilon = 0$) has nilpotency $m \geqq 2$ many of the difficulties which are described in this report should occur in problems like (2.8). Of course, the stiff system can be solved using a small enough stepsize, but this may be very inefficient. We do not know whether stiff problems with this structure occur very often in practice but when they do most codes will have trouble solving them.

3. Discontinuities, errors and error estimates. This section describes several problems which are likely to arise in solving certain linear differential/algebraic systems. First we look at what happens when codes based on BDF with the usual error control strategy are faced with problems of nilpotency $m = 2$. Codes behave rather strangely in these circumstances and may even fail because the error estimates do not reflect the true behavior of the error. Later in this section we examine the difficulties inherent in solving systems of nilpotency $m \geqq 3$ and indicate why these problems are not solvable by codes based on variable-stepsize BDF. For simplicity, all of our examples use the backward Euler method and low order error estimates. However, it is easy to see that the same types of difficulties occur for higher order methods and higher order error estimates.

Our first example is a problem involving a discontinuity in the dependent variable. This problem is not "solvable" since the solution is not defined at the point of the discontinuity (though it exists and is unique everywhere else). However, it is easy to generate such a problem without realizing it even with a differentiable input function. Hence, we feel that a code should at least fail leaving some indication of the cause of the difficulty on this type of problem. Furthermore, we find later that the difficulties in solving this example problem occur in exactly the same way for continuous, "solvable" systems.

Consider solving the system

$$E\mathbf{y}' = A\mathbf{y} + \mathbf{g}(t). \tag{3.1}$$

Several unpleasant problems may occur if some derivative of $\mathbf{g}(t)$ is discontinuous. First of all, recall from (2.7) that a discontinuity in a derivative of $\mathbf{g}(t)$ may lead to a discontinuity in some component of $\mathbf{y}$. If we were solving such a problem numerically, we would hope that the code could find the discontinuity and pass over it or at least would fail at the discontinuity. Instead, a code based on BDF with the usual error control mechanism (error estimates proportional to the difference between the predictor and the corrector) is likely to fail *not* on the step which spans the discontinuity but on the subsequent step. This leaves us with little indication that the discontinuity was the source of the difficulties. These problems occur even when using a one-step method, which normally (for standard form ODE systems) we think of as having no memory.

How can this happen? As an example, consider the problem

$$\begin{gathered} y_2'(t) = y_1(t), \qquad 0 = y_2(t) - g(t), \\ \text{where } g(t) \text{ is given by } g(t) = \begin{cases} 0, & t \leqq 0 \\ ct, & t > 0 \end{cases}. \end{gathered} \tag{3.2}$$

Suppose $y_1 \equiv y_2 \equiv 0$ for $t < 0$ and that the error estimate (which we will use for accepting or rejecting the step and for choosing the next stepsize) is proportional to $\|\mathbf{y}_{n+1} - \mathbf{y}_n\|$. Now, take one step with the backward Euler method, assuming that t_{n-1} is the time corresponding to the last accepted step to the left of zero (that is, assume $t_{n-1} < 0$, $t_n > 0$). Then, at time t_n we obtain

$$y_{2,n} = g(t_n) = ct_n, \qquad y_{1,n} = \frac{g(t_n) - g(t_{n-1})}{h_n} = \frac{ct_n}{h_n},$$

where $h_n = t_n - t_{n-1}$. Now, as t_n approaches zero, h_n is bounded away from zero as long as $t_{n-1} < 0$ is fixed. So the error estimate,

$$\left\| \begin{matrix} y_{1,n} - y_{1,n-1} \\ y_{2,n} - y_{2,n-1} \end{matrix} \right\| = \left\| \begin{matrix} ct_n/h_n - 0 \\ g(t_n) - g(t_{n-1}) \end{matrix} \right\|$$

approaches zero as $t_n \to 0$ and for some $t_n > 0$ the step is accepted.

There is already a problem at this point as $y_{1,n}$ is in error by $y_{1,n} - y_1(t_n) = ct_n/h_n - c = c(1 - t_n/h_n)$ so that unless either t_{n-1} or t_n is exactly zero, we can construct problems (by choosing c large enough) for which the error in $Y_{1,n}$ is arbitrarily large but the step is accepted. However, this is not as bad as it may at first seem because for $t_n \to 0_+$ we get $\mathbf{y}_n \to 0$, which is the correct solution for a nearby time (a time less than zero).

Since the step to t_n is accepted for some $t_n > 0$, $t_{n-1} < 0$, the code will continue, taking another step to t_{n+1},

$$y_{2,n+1} = g(t_{n+1}) = ct_{n+1}, \qquad y_{1,n+1} = \frac{g(t_{n+1}) - g(t_n)}{h_{n+1}}.$$

These solutions are exactly correct (we can expect this to happen only for linear systems with nilpotency m equal to 2), but the error estimate is

$$\left\| \begin{matrix} c(1 - t_n/h_n) \\ ch_{n+1} \end{matrix} \right\|.$$

The first component of this error estimate is independent of h_{n+1} so that we cannot make the estimate as small as we want by reducing h_{n+1}. Thus, for large enough c, the code will fail on this step—even though the solution is exactly correct.

We can get a better understanding of the cause of this problem with the error estimates by observing what happens graphically. Figure 3.1 shows $g(t)$.

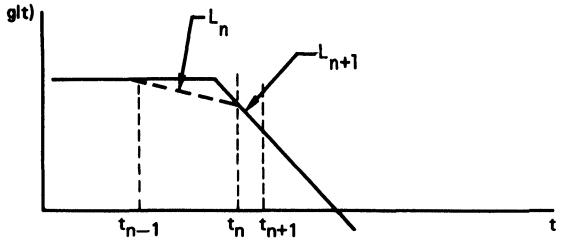

FIG. 3.1

Let L_n be the line joining $g(t_{n-1})$ and $g(t_n)$. The numerical solution $y_{1,n}$ is the slope of the secant line L_n. At the time when we are trying to compute $\mathbf{y}_{n+1}$, this line is fixed because the step from t_{n-1} to t_n has been accepted. The solution for $y_{1,n+1}$ is the slope of the line L_{n+1} between $g(t_n)$ and $g(t_{n+1})$. But as $t_{n+1} \to t_n$ (when the code is reducing the stepsize h_{n+1} to try to obtain a smaller error estimate), this line (L_{n+1}) approaches the tangent of g at t_n. Unless g is linear, the slope of the tangent at t_n is not the same as the slope of the secant from t_{n-1} to t_n. If the difference between these two slopes is bigger than the error estimate, then the code will fail on this step. In any case, error estimates based on the difference between the predictor and the corrector fail to reflect the true magnitude of the error for this problem. Recall that the predictor is just a polynomial extrapolating through past values of the solution, so that the predictor gets arbitrarily close to the most recent solution value as $h_{n+1} \to 0$, and these estimates behave qualitatively in the same way as the simpler estimates we have been considering.

It is easily seen that these difficulties with the error estimate are not limited to problems whose solutions are discontinuous. For example, consider solving the same problem as before except with $g(t)$ given in Fig. 3.2.

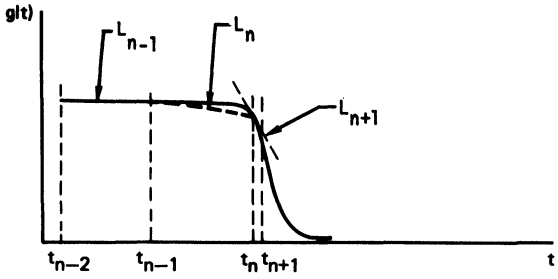

FIG. 3.2

This problem has a steep gradient. Since the slope of L_{n-1} is not much different from the slope of L_n, the step to t_n is accepted. But in the next step, the slope is much different, and we cannot close the gap because the new slope (of L_{n+1}) just gets nearer and nearer to the slope of the tangent at t_n. Since the slope of the tangent is far from the slope of L_n, the code fails on this step.

We conclude that 1) in general, for systems of the form (3.1), the difference between the predictor and the corrector does not approach zero as the current stepsize approaches zero and 2) codes using error estimates based on this difference may fail on problems with steep gradients, and these error estimates seem to bear little resemblance to the errors which are actually incurred for some DAE systems.

Are the problems described above due to poor error estimates or is there some fundamental problem with using backward Euler with varying stepsizes for solving linear DAE systems with nilpotency $m \geqq 2$? (To save writing these will be called ($m \geqq 2$) systems.) To gain a better understanding of the source of this problem, we take a closer look at the errors at each step compared with the error estimates.

Suppose we are starting with initial values at t_n equal to the exact solution. Then what is the error after one step in solving (3.1) with backward Euler? Taking one step, we obtain

$$E\left(\frac{\mathbf{y}_{n+1}-\mathbf{y}_n}{h_{n+1}}\right) = A\mathbf{y}_{n+1} + \mathbf{g}(t_{n+1}). \tag{3.3}$$

If $\mathbf{y}(t_{n+1})$ is the true solution to (3.1) at time t_{n+1}, then we have

$$\mathbf{y}(t_{n+1}) = \mathbf{y}(t_n) + h_{n+1}\mathbf{y}'(t_{n+1}) + \frac{h_{n+1}^2}{2}\mathbf{y}''(\xi), \tag{3.4}$$

where $t_n \leqq \xi \leqq t_{n+1}$. Substituting (3.4) for $\mathbf{y}'(t_{n+1})$ in (3.1), we obtain

$$E\left(\frac{\mathbf{y}(t_{n+1})-\mathbf{y}(t_n)-\frac{h_{n+1}^2}{2}\mathbf{y}''(\xi)}{h_{n+1}}\right) = A\mathbf{y}(t_{n+1}) + \mathbf{g}(t_{n+1}). \tag{3.5}$$

Now, subtracting (3.5) from (3.3), if $\mathbf{e}_{n+1} = \mathbf{y}_{n+1} - \mathbf{y}(t_{n+1})$,

$$E\left(\frac{\mathbf{e}_{n+1}-\mathbf{e}_n+\frac{h_{n+1}^2}{2}\mathbf{y}''(\xi)}{h_{n+1}}\right) = A\mathbf{e}_{n+1}, \tag{3.6}$$

or, rewriting,

$$\mathbf{e}_{n+1} = (E - h_{n+1}A)^{-1}E\mathbf{e}_n - (E - h_{n+1}A)^{-1}E\left(\frac{h_{n-1}^2}{2}\right)\mathbf{y}''(\xi). \tag{3.7}$$

Now, (3.7) says that the error after one step, starting from exact initial values (so that $\mathbf{e}_0 = 0$) is

$$\mathbf{e}_1 = -(E - h_{n+1}A)^{-1}E\left(\frac{h_{n+1}^2}{2}\right)\mathbf{y}''(\xi). \tag{3.8}$$

There are several consequences of this simple expression. For example, for the ($m = 2$) problem (3.2), we have

$$\mathbf{y}''(\xi) = \begin{bmatrix} g'''(\xi) \\ g''(\xi) \end{bmatrix}$$

and

$$(E - h_{n+1}A)^{-1}E = \begin{bmatrix} 0 & -1/h_{n+1} \\ 0 & 0 \end{bmatrix},$$

so that the error after one step is

$$\mathbf{e}_1 = \begin{bmatrix} h_{n+1}g''(\xi)/2 \\ 0 \end{bmatrix}. \tag{3.9}$$

We observe from (3.9) that the error is $O(h_{n+1})$, not $O(h_{n+1}^2)$, and that it depends on $y_2''(t)$ and not $y_1''(t)$. Thus, if we had a good estimate for $\mathbf{y}''$, then the usual error estimate based on $(h_{n+1}^2/2)\mathbf{y}''$ would be asymptotically a gross underestimate. The term $(E - h_{n+1}A)^{-1}E(h_{n+1}^2/2)\mathbf{y}''(\xi)$ is in a sense the local contribution to the global error. Notice that we cannot really define a local error in the usual way as $\mathbf{y}_{n+1} - \mathbf{u}(x_{n+1})$, where $\mathbf{u}(x_n) = \mathbf{y}_n$, because there may not be a solution passing through $\mathbf{y}_n$.

The situation for $(m = 2)$ nonstate subsystems is obviously enough to wreak havoc with any stepsize selection algorithm which assumes that errors are $O(h^{k+1})$, where k is the order of the method. Furthermore, the usual error estimates can cause a code to do very strange things when $g(t)$ has a steep gradient. Despite all of this, the situation is not at all without hope. The error in y_{n+1} at any step can be reduced by reducing h_{n+1}, (recall, however, that the difference between the predictor and corrector is not reduced) so that in principle, if we knew how to adjust the stepsize, the error in y_{n+1} could be controlled by locally adjusting h_{n+1}. In the next section, we will find error estimates which can accomplish this task.

Unfortunately, there are several even more severe problems in solving systems with nilpotency $m \geqq 3$. For the $(m = 3)$ system

$$y_2'(t) = y_1(t), \quad y_3'(t) = y_2(t), \quad 0 = y_3(t) - g(t), \tag{3.10}$$

we have

$$(E - h_{n+1}A)^{-1}E = \begin{bmatrix} 0 & -1/h_{n+1} & -1/h_{n+1}^2 \\ 0 & 0 & -1/h_{n+1} \\ 0 & 0 & 0 \end{bmatrix}$$

and

$$\frac{h_{n+1}^2}{2}\mathbf{y}''(\xi) = \frac{h_{n+1}^2}{2}\begin{bmatrix} g^{iv}(\xi) \\ g'''(\xi) \\ g''(\xi) \end{bmatrix},$$

so that the error in $\mathbf{y}$ after one step starting from the exact solution is

$$\mathbf{e}_1 = \begin{pmatrix} \dfrac{g''(\xi)}{2} + \dfrac{h_{n+1}}{2}g'''(\xi) \\ \dfrac{h_{n+1}}{2}g''(\xi) \\ 0 \end{pmatrix}. \tag{3.11}$$

Restating this, we cannot choose h_{n+1} small enough so that the error in the solution after one step starting from exact initial values is small. This poses a very difficult problem in finding initial values for $m \geqq 3$ systems, if even the exact solution will not do.

This observation seems to conflict with a theorem proved in [2]. Of course, it is not really a conflict but only appears that way because these results are not nearly as strong as what we are accustomed to for standard ODE's. The theorem states that the backward Euler method and k-step BDF converge to the analytic solution of the

system (3.1). Furthermore, it is stated that the error in solving nonstate subsystems with the k-step method is $O(h^k)$, where h is the stepsize. How can the method converge when the error after the first step is independent of h? We must be very careful about interpreting the qualifications on this theorem. The convergence, at least for systems where $m \geqq 3$, starting from the exact solution, only applies to the solution at the endpoint of some fixed interval of integration. This is because the first $m-2$ solution values contain errors which do not become arbitrarily small when h is decreased. The results at later steps depend only on the function $g(t)$ at past steps and not on the initial values so that the solution converges in any interval bounded away from the initial time. The theorem is only true for linear systems, and for these systems, this behavior may be tolerable if we have anticipated it. However, this is a very serious problem if what we are really trying to solve is a nonlinear system. In that case, large errors in the first few steps may persist throughout the entire interval.

The other qualification on the theorem is that it is only true for constant stepsizes. We can see this by applying backward Euler to the $m = 3$ system (3.9),

$$
\begin{aligned}
&y_{3,n+1} = g(t_{n+1}),\\
&y_{2,n+1} = (g(t_{n+1}) - g(t_n))/h_{n+1},\\
&y_{1,n+1} = \left(\left(\frac{g(t_{n+1}) - g(t_n)}{h_{n+1}}\right) - \left(\frac{g(t_n) - g(t_{n-1})}{h_n}\right)\right) \Big/ h_{n+1}.
\end{aligned} \tag{3.12}
$$

Now if $h_{n+1} = h_n = h$, then $y_{1,n+1}$ is given by $(\nabla^2 g/h^2)$ which converges to g'' as $h \to 0$ as the theorem predicts. When $h_{n+1} \neq h_n$ we might hope that $y_{1,n+1}$ would be given by the divided difference $2[g_{n+1}, g_n, g_{n-1}]$, where

$$
2[g_{n+1}, g_n, g_{n-1}] = \frac{\dfrac{g_{n+1} - g_n}{h_{n+1}} - \dfrac{g_n - g_{n-1}}{h_n}}{\dfrac{h_{n+1} + h_n}{2}}, \tag{3.13}
$$

but the method fails to pick up these differences correctly. The error in the approximation to $y_{1,n+1}$ which is caused by changing the stepsize has the form $\frac{1}{2}(1 - h_n/h_{n+1})\mathbf{g}''(\xi) + O(h_{n+1})$. This term becomes arbitrarily large as $h_{n+1} \to O$ (h_n fixed). If we were very careful to select stepsizes so that $h_{n+1} = h_n(1 + O(h_n))$, then this problem could be avoided. However, this seems to be very difficult to accomplish in a practical code and sequences of stepsizes would have to satisfy even more stringent restrictions for systems with $m > 3$.

We have tried to illustrate in this section that while the BDF can, in principle, solve (3.1) (via the theorem [2]), there are a great many qualifications to this statement which can cause serious difficulties in any kind of a practical code. At present, we know of no way to adequately handle (3.1) when $m \geqq 3$ subsystems are present. The most serious problems seem to be in starting the code and in changing stepsize. The remainder of this paper will be concerned with handling ($m \leqq 2$) systems and related nonlinear problems and detecting the other problems that codes based on methods such as BDF cannot solve.

4. Error estimation. In this section we examine several potential candidates for error estimates for differential/algebraic systems. Our aim is to find an estimate which accurately reflects the behavior of the error for linear ($m \leqq 2$) systems. Additionally, we hope to detect ($m \geqq 3$) systems which cannot be solved accurately using BDF with

varying stepsizes and to give an accurate indication of why the code fails to solve these problems.

Several different approaches to error estimation for DAE systems have been reported in the literature. Gear and Brown [3] solve systems of the form

$$\mathbf{f}(t, \mathbf{y}, \mathbf{y}') + P\mathbf{v} = 0, \tag{4.1}$$

where $\mathbf{y}$ and $\mathbf{y}'$ are of length p, $\mathbf{v}$ is of length $q-p$, P is a $qx(q-p)$ matrix and $\mathbf{f}$ is a vector function of length q. A close look at their code [3] reveals that there is no attempt to estimate errors in $\mathbf{v}$. In some respects this makes sense as $\mathbf{v}$ is completely determined by $\mathbf{y}$ on every step so that errors in $\mathbf{v}$ do not cause errors in $\mathbf{y}$. If for some reason we are actually interested in the values of $\mathbf{v}$, we must understand that they may contain very large errors. For example, the ($m = 3$) system (3.9) may be rewritten in the form (4.1), with $v = y_1$, $z_1 = y_2$, $z_2 = y_3$, as

$$z_1' - v = 0, \quad z_2' - z_1 = 0, \quad z_2 - g(t) = 0,$$

and error control is not attempted on v (on y_1). But this is exactly the component that exhibits arbitrarily large errors after a change of stepsize. Also, if we should fail to realize that y_1 occurs linearly, then the behavior of the code is very different.

Recently, a different approach to error control was proposed by Sincovec et al. [2] for linear DAE systems. They observed that errors in nonstate components (subsystem (2.3b)) have a different asymptotic behavior than errors in state components (subsystems (2.3a)) so that stepsize selection schemes which assume a certain asymptotic behavior of the error would have difficulty in controlling the errors in these components. In addition, errors in nonstate components affect the solution only locally, and are not propagated globally to state components. As a consequence, they proposed to "filter out" the part of the error estimate that corresponds to the nonstate solution components. This is accomplished by monitoring a projection of the usual error estimate, where the new estimate is given by

$$\varepsilon_n^* = \|Mc_{n,k}(\mathbf{y}_n^c - \mathbf{y}_n^p)\|. \tag{4.2}$$

Here $\mathbf{y}_n^c$ is the final corrected value of $\mathbf{y}$ at the end of a step, $\mathbf{y}_n^p$ is the predicted value of $\mathbf{y}$ and $c_{n,k}$ is a constant depending on the method and recent stepsize history of the integration. (Note that $\|c_{n,k}(\mathbf{y}_n^c - \mathbf{y}_n^p)\|$ is the error estimate which is generally used in ODE codes.) M is called the canonical state variable projection matrix and is given by

$$M = \lim_{h \to 0} M(h, j), \tag{4.3}$$

where $M(h, j) = ((E - hA)^{-1}E)^j$, where j is greater than or equal to the nilpotency of the system.

It is shown in [2] that this error estimate has the effect of filtering out that part of $(\mathbf{y}^c - \mathbf{y}^p)$ which is associated with nonstate variables (if the system were transformed to canonical form), without the expensive operations of transforming the system into canonical form. It is convenient because LU decompositions of the matrices $(E - hA)$ are always available (for h the current stepsize) because that is the iteration matrix for the Newton iteration. It is somewhat inconvenient in that it is hard to find out what the nilpotency of the system is, and we wonder what to do about systems which are "almost" nilpotent.

Unfortunately, there is one very bad defect in this filtered error estimation scheme and, indeed, in nearly any scheme which fails to control the errors in certain components of the solution. With this particular scheme we can, for example, "solve" any

($m \geqq 3$) system, the stepsize being chosen to control errors in the state components of the system. However, as we have already seen, this can be very misleading because large errors are introduced into some components of the solution whenever the stepsize is changed.

The examples that we have given show that it is a very dangerous practice not to control errors in some components of the solution. The only possible circumstances under which we feel this could be done safely are if 1) we are not interested in the value of that component, *and* we are sure that errors in that component cannot be propagated into any other component later in the integration, or 2) if we are sure that no errors are being made in that component. For example, if in solving (4.1) we are not interested in the value of $\mathbf{v}$, then it is safe to omit those variables from the error control.

We have already noted the similarities between stiff systems and differential/algebraic systems. It is natural to look at error estimation schemes proposed for stiff equations. Curtis [7] noted that for a single linear ODE, $y' = -\lambda(y - f(t)) + f'(t)$ end-step errors are smaller than the usual error estimate by a factor $1/\lambda h$, where $-\lambda$ is the eigenvalue of the Jacobian matrix and h the step size. When λ is large, this problem is very nearly the same as the ($m = 1$) algebraic system $y = f(t)$, which we solve exactly on every step. On the other hand, we saw in § 3 that for some problems, the usual error estimate can severely underestimate the error.

Sacks-Davis [9] noted that for stiff problems, the usual error estimate based on the difference between the predictor and the corrector overestimates the true error. He suggests error estimates for second derivative methods which are asymptotically correct as $h \to 0$, and are reliable and efficient for very stiff problems. The estimates have the form

$$\varepsilon_n^{**} = \| W_n^{-1} c_{n,k} (\mathbf{y}_n^c - \mathbf{y}_n^p) \|, \tag{4.4}$$

where W_n is the iteration matrix for the second derivative method.

If we examine an estimate similar to (4.4) for BDF, where W_n is the iteration matrix for the kth order BDF, then it is easy to see that ε_n^{**} from (4.4) is the same as the estimate (4.2), if $m = 1$ and $M(h_{n+1}, 1)$ is used in place of M and if the matrix E in (3.1) is nonsingular. We also note from (3.7) that the local contribution to the global error for the backward Euler method is

$$s(E - hA)^{-1} E \frac{h^2}{2} \mathbf{y}''(\xi).$$

Because of these observations, we are led to try the estimate

$$\varepsilon_n = \| (E - \beta_{n,k} h_n A)^{-1} E c_{n,k} (\mathbf{y}_n^c - \mathbf{y}_n^p) \|, \tag{4.5}$$

where $\beta_{n,k}$ and $c_{n,k}$ are constants depending on the method used and possibly on the recent stepsize history. For a standard-form ODE, (4.5) is asymptotically equivalent to the usual estimate, $\| c_{n,k} (\mathbf{y}_n^c - \mathbf{y}_n^p) \|$, so that the question we must answer is how well the estimate performs on the nonstate and/or stiff components of a system.

On first glance the estimate (4.5) might seem to contradict statements that were made earlier, as this estimate does not "control the error" in algebraic ($m = 1$) subsystems. These subsystems have the form $y(t) = f(t)$, and they are solved by the method exactly (apart from errors due to terminating the Newton iteration, which are discussed in the next section), so this strategy is not unreasonable. A problem with (4.5) is that if a code interpolates to find the solution at user-specified output points then this estimate does not control the error in the interpolation. These errors seem

to go to zero as the stepsizes get smaller and smaller, but we know of no way to explicitly control them without incurring at the same time all of the problems mentioned in § 3.

How accurately does the estimate (4.5) reflect the error for ($m=2$) nonstate subsystems? From (3.7), we can see that for backward Euler (4.5) accurately estimates the local contribution to the global error as long as $\mathbf{y}''(\xi)$ can be found accurately. Actually, for the canonical ($m=2$) nonstate subsystems (3.2), we have

$$(E-hA)^{-1}E=\begin{bmatrix}0 & -1/h\\ 0 & 0\end{bmatrix},$$

so that only $y_2''(\xi)$ is involved in the estimate. This is the second derivative of the input function $g(t)$, and it can be estimated reasonably well by a divided difference of $g(t)$, so that the estimate (4.5) works. It is fortunate that the estimate does not make use of $y_1''(\xi)$, because difficulties in obtaining this term were in part responsible for the problems discussed in § 3. One other point we make is that for a ($m=2$) subsystem the local contribution to the global error is the global error (neglecting roundoff and errors due to terminating the Newton iteration early). This happens because, from (2.3b), the nonstate subsystems can be written in the form

$$E_2\mathbf{y}'=\mathbf{y}+g(t),$$

where $E_2^2=0$ and then, from (3.7),

$$\mathbf{e}_{n+1}=(E_2-h_{n+1}I)^{-1}E_2\mathbf{e}_n+(E_2-h_{n+1}I)^{-1}E_2\boldsymbol{\tau}_{n+1},$$

where $\boldsymbol{\tau}_{n+1}=(h_{n+1}^2/2)\mathbf{y}''(\xi)$. Rewriting this, we obtain

$$\mathbf{e}_{n+1}=(E_2-h_{n+1}I)^{-1}E_2(E_2-h_nI)^{-1}E_2[\mathbf{e}_{n-1}+\boldsymbol{\tau}_n]+(E_2-h_{n+1}I)^{-1}E_2\boldsymbol{\tau}_{n+1},$$

but the matrix $(E_2-h_{n+1}I)^{-1}E_2(E_2-h_nI)^{-1}E_2$ is identically zero for this case so that

$$\mathbf{e}_{n+1}=(E_2-h_{n+1}I)^{-1}E_2\boldsymbol{\tau}_{n+1}.$$

It is easy to verify that the estimate (4.5) also accurately reflects the behavior of the error for all of the BDF for ($m\leqq 2$) systems. In general, the kth order BDF approximates $h_{n+1}\mathbf{y}'_{n+1}$ by $\sum_0^k \alpha_{i,n+1}\mathbf{y}_{n+1,i}$, where $\alpha_{i,n+1}$ depend on the order and recent stepsize history, so that $\mathbf{y}_{n+1}$ satisfies

$$E\Big(\sum_0^k \alpha_{i,n+1}\mathbf{y}_{n+1-i}\Big)=h_{n+1}A\mathbf{y}_{n+1}+h_{n+1}\mathbf{g}(t_{n+1}), \tag{4.6}$$

or, rewriting,

$$(\alpha_{0,n+1}E-h_{n+1}A)\mathbf{y}_{n+1}=-E\sum_1^k \alpha_{i,n+1}\mathbf{y}_{n+1-i}+h_{n+1}E\mathbf{g}_{n+1}. \tag{4.7}$$

Now, if $\boldsymbol{\tau}_{n+1}$ is defined by

$$\boldsymbol{\tau}_{n+1}=h_{n+1}\mathbf{y}'(t_{n+1})-\sum_0^k \alpha_{i,n+1}\mathbf{y}(t_{n+1-i}) \tag{4.8}$$

($\boldsymbol{\tau}_{n+1}$ is usually called the local truncation error of the method), then

$$(\alpha_{0,n+1}E-h_{n+1}A)\mathbf{y}(t_{n+1})=-E\sum_1^k \alpha_{1,n+1}\mathbf{y}(t_{n+1-i})+h_{n+1}Eg_{n+1}-E\boldsymbol{\tau}_{n+1}. \tag{4.9}$$

Subtracting (4.9) from (4.7), we have, if $\mathbf{e}_n=\mathbf{y}_n-\mathbf{y}(t_n)$,

$$\mathbf{e}_{n+1}=(\alpha_{0,n+1}E-h_{n+1}A)^{-1}E\sum_1^k \alpha_{i,n+1}\mathbf{e}_{n+1-i}+(\alpha_{0,n+1}E-h_{n+1}A)^{-1}E\boldsymbol{\tau}_{n+1} \tag{4.10}$$

so that $(\alpha_{0,n+1}E - h_{n+1}A)^{-1}E\tau_{n+1}$ is the local contribution to the global error for variable-stepsize BDF, and this has the same form as the estimate (4.5).

Since τ_{n+1} is the local truncation error of a variable-stepsize BDF, it depends on the recent stepsize history of the computation. For example, for a variable-step 2nd order BDF,

$$\|\tau_{n+1}\| \leqq h_{n+1}^2(h_{n+1}+h_n)\|\mathbf{y}^{(k+1)}(\xi)\|. \tag{4.11}$$

Thus, the error as $h_{n+1} \to O$ (h_n fixed) has the form $O(h_{n+1}^2)$, not $O(h_{n+1}^3)$. It is important when using this error estimate to get this dependence correctly. If we instead assume (as is done in some methods which change the stepsize via interpolation [3], [5], [6]) that the error depends on the current stepsize as $O(h_{n+1}^3)$, this can cause the code to severely underestimate the error when the solution is just beginning to change rapidly. This is a particular problem for ($m \geqq 3$) nonstate subsystems, where the error does not go to zero as $h_{n+1} \to 0$ (h_n fixed), and the error estimate is given by (for $m = 3$)

$$\varepsilon_{n+1} = \begin{bmatrix} 0 & -1/h_{n+1} & -1/h_{n+1}^2 \\ 0 & 0 & -1/h_{n+1} \\ 0 & 0 & 0 \end{bmatrix} \tau_{n+1}. \tag{4.12}$$

If we had taken $\tau_{n+1} = O(h_{n+1}^3)$, then the error estimate would become smaller as h_{n+1} decreases and we would be led to believe that errors were under control, when in reality there is a large error in the first component. It is especially important for DAE systems and the error estimate (4.5) to use the actual (variable stepsize) principal term of the local truncation error in the error estimate rather than an approximation which assumes that the last k steps were taken with a constant stepsize.

In summary, the error estimate (4.5) seems to be a useful alternative to the usual estimates based on a difference between the predictor and the corrector. It reflects the behavior of the error more accurately than the usual error estimate and overcomes many of the difficulties mentioned in § 3 for systems containing ($m = 2$) nonstate subsystems. Use of this estimate enables codes based on BDF to solve a wider class of problems. The estimate is easily generalized to nonlinear problems. While we have no theory to support this generalization, it seems to have worked well in our experience.

5. Practical issues. It is evident from the problems mentioned in § 3 that DAE systems are in many respects very different from ODE systems. With this in mind, it is not unreasonable to expect that codes for solving the two types of problems must be different in some respects. In this section we question several strategies used in ODE codes which one might be tempted to carry over to DAE codes to discover whether or not they are applicable to these more general problems.

Codes for solving stiff and differential/algebraic systems generally use a modified Newton iteration to solve an implicit equation for $\mathbf{y}_{n+1}$ at each step. For example, in solving

$$E\mathbf{y}' = A\mathbf{y} + \mathbf{g}(t) \tag{5.1}$$

with backward Euler, there is an implicit equation

$$E\left(\frac{\mathbf{y}_{n+1} - \mathbf{y}_n}{h_{n+1}}\right) = A\mathbf{y}_{n+1} + \mathbf{g}(t_{n+1}) \tag{5.2}$$

which is solved for $\mathbf{y}_{n+1}$ at each step.

We will use the linear problem (5.1) as a model, though it is really the more general nonlinear problem (1.1) that we wish to gain some intuition about. Linear

problems are, of course, much easier to solve because Newton's method converges in one iteration if the exact iteration matrix is used. We consider situations where the iteration matrix is not exact to learn more about how to handle nonlinear problems because these problems are much harder to understand.

Now suppose (5.2) is solved by a modified Newton iteration,

$$\mathbf{y}_{n+1}^{(k+1)} = \mathbf{y}_{n+1}^{(k)} - J^{-1}\left(E\frac{(\mathbf{y}_{n+1}^{(k)} - \mathbf{y}_n)}{h_{n+1}} - A\mathbf{y}_{n+1}^{(k)} - \mathbf{g}(t_{n+1})\right), \tag{5.3}$$

where J is an approximation to the matrix $(E/h_{n+1} - A)$ (J may have been computed in some previous step). Several decisions must be made in implementing (5.3). For example, how should we decide when the iteration has converged? If the iteration is not converging, what action should be taken? If it appears that we cannot make the iteration converge, can we discover the problem? We take up these questions here.

In deciding when to terminate the corrector iteration, our main consideration is that the error introduced by terminating this iteration early is small relative to errors due to the difference approximation to $\mathbf{y}'$ and that these errors do not affect the error estimates appreciably. Suppose then that $\boldsymbol{\delta}_{n+1}$ is the error in $\mathbf{y}_{n+1}$ due to terminating the corrector iteration early. That is, if $\tilde{\mathbf{y}}_{n+1}$ is the exact solution to the difference equation and $\mathbf{y}_{n+1} = \mathbf{y}_{n+1}^{(M)}$ is the solution accepted after M iterations, then $\tilde{\mathbf{y}}_{n+1} = \mathbf{y}_{n+1} + \boldsymbol{\delta}_{n+1}$. Then from (5.2),

$$E\left(\frac{\mathbf{y}_{n+1} - \mathbf{y}_n + \boldsymbol{\delta}_{n+1}}{h_{n+1}}\right) = A(\mathbf{y}_{n+1} + \boldsymbol{\delta}_{n+1}) + \mathbf{g}(t_{n+1}). \tag{5.4}$$

The exact solution $\mathbf{y}(t)$ satisfies

$$E\left(\frac{\mathbf{y}(t_{n+1}) - \mathbf{y}(t_n) + \boldsymbol{\tau}_{n+1}}{h_{n+1}}\right) = A\mathbf{y}(t_{n+1}) + \mathbf{g}(t_{n+1}). \tag{5.5}$$

Subtracting (5.5) from (5.4), the global error $\mathbf{e}_n = \mathbf{y}_n - \mathbf{y}(t_n)$ satisfies

$$\mathbf{e}_{n+1} = (E - h_{n+1}A)^{-1}E\mathbf{e}_n + (E - h_{n+1}A)^{-1}E\boldsymbol{\tau}_{n+1} - \boldsymbol{\delta}_{n+1} \tag{5.6}$$

and, rewriting,

$$\begin{aligned}\mathbf{e}_{n+1} = {} & (E - h_{n+1}A)^{-1}E(E - h_nA)^{-1}E(\mathbf{e}_{n-1} + \boldsymbol{\tau}_n) \\ & - (E - h_{n+1}A)^{-1}E\boldsymbol{\delta}_n + (E - h_{n+1}A)^{-1}E\boldsymbol{\tau}_{n+1} - \boldsymbol{\delta}_{n+1}.\end{aligned}$$

We know from ODE theory that for the state variables in the system (2.3a) errors are not amplified greatly from step to step by the BDF and it is sufficient to control $\boldsymbol{\delta}_{n+1}$. Our main concern is what happens to this error for nonstate ($m = 1$ and $m = 2$) subsystems.

For ($m = 1$) subsystems it is logical to control the error in $\mathbf{y}_{n+1}$ (that is, to control $\boldsymbol{\delta}_{n+1}$), as $E = 0$ in this case, so that $\mathbf{e}_{n+1} = \boldsymbol{\delta}_{n+1}$ and $\boldsymbol{\delta}_{n+1}$ is the only error. For a ($m = 2$) nonstate subsystem, the situation is not quite so clear. Let us look at the ($m = 2$) canonical subsystem (2.5). In this case, the matrix $(E - h_{n+1}A)^{-1}E(E - h_nA)^{-1}E$ is identically zero so that the contribution to the global error due to terminating the iteration early is given by

$$-(E - h_{n+1}A)^{-1}E\boldsymbol{\delta}_n - \boldsymbol{\delta}_{n+1}, \quad \text{where } (E - h_{n+1}A)^{-1}E = \begin{bmatrix} 0 & -1/h_{n+1} \\ 0 & 0 \end{bmatrix}.$$

Since $y_1 = y_2'$ is approximated by a backward difference, $y_{1,n+1} \cong (y_{2,n+1} - y_{2,n})/h_{n+1}$, then an error in $y_{2,n}$ is amplified by $1/(h_{n+1})$ in $y_{1,n+1}$.

If we are only interested in iteration error as it relates to the error estimates of § 4, then this does not matter as $y_{1,n+1}$ does not enter into the error estimate so we should control $\boldsymbol{\delta}_{n+1}$. That is, terminate the iteration when

$$\|\boldsymbol{\delta}_{n+1}\| \leqq \varepsilon, \tag{5.9}$$

where ε is a constant depending on the error tolerances requested. On the other hand, for the purposes of controlling the error in y_1, we should control $\boldsymbol{\delta}_{n+1}$ so that

$$\|\boldsymbol{\delta}_{n+1}\| \leqq h_{n+1}\varepsilon \tag{5.10}$$

because the error may be amplified by $1/h_{n+1}$ on the next step. (Though we do not know in advance what the stepsize for the next step will be, we assume it will be the same order of magnitude as the current stepsize.) Starner [6] uses this test in his code though apparently for different reasons. The test (5.10) has the apparent disadvantage that it is not independent of scaling of the independent variable in the system; however, note also that any scaling of this type also scales **y**. For these reasons, it is somewhat unclear whether to use (5.9) or (5.10), but there seem to be several good reasons for choosing (5.10). Shampine [10] discusses ways to bound $\|\boldsymbol{\delta}_{n+1}\|$ based on the differences between iterates, so that finding a bound for this number is not a big problem.

What if the corrector iteration fails to converge (assuming that the iteration matrix is current)? There are several reasons why our answer to this question might be different for a DAE code than for a code aimed at ODE's in standard form. Generally, the strategy taken in ODE codes is to reduce the stepsize when the iteration fails to converge. There are two reasons why this works. First, as we reduce the stepsize, the prediction gets to be a better and better initial guess for the Newton iteration. Second, the iteration matrix for a standard-form ODE $y' = f(t, y)$ is $(I - h\beta\partial F/\partial y)$, and this matrix tends towards the identity as the stepsize h is reduced. This is fortunate, because the identity matrix is very well conditioned so that errors in $\partial F/\partial y$ and in solving the linear systems affect the solution less and less as h is reduced.

The situation with differential/algebraic systems is not quite as desirable. As we have seen, the difference between the predictor and the corrector does not tend to zero as $h_{n+1} \to 0$. Thus, the initial guess may not get better with decreasing h_{n+1}. For these systems the iteration matrix $(\partial F/\partial y' - h\beta\partial F/\partial y)$ looks like $\partial F/\partial y'$ as $h \to 0$. In general, $\partial F/\partial y'$ may be singular, so that as $h_{n+1} \to 0$ the iteration matrix becomes more and more poorly conditioned. This is very troublesome for some systems, where for small enough h roundoff from solving a poorly conditioned linear system causes the corrector iteration to diverge. We are not yet certain about the proper way to handle these problems, but the following alternatives to the usual strategy seem to be worth considering. Instead of decreasing h_{n+1} when the iteration fails to converge, we could instead use a more robust iteration procedure (for example, damped Newton) or, because for at least some component a better initial guess could be obtained by reducing h_{n+1}, we could try reducing h_{n+1}, and if this doesn't help (if after reducing h_{n+1} several times, the initial guess is not much better), then switch to a more robust iteration scheme.

It is possible that the corrector iteration fails to converge despite all of our actions to try to help it. There are a great many more possibilities for the cause of this problem with a DAE system than there are for an ODE system. For an ODE system, we generally assume that this problem is caused by a very poor approximation to the Jacobian matrix. For a DAE system, it could be that we could not get a good initial guess or, maybe, the system contains a ($m \geqq 3$) nonstate subsystem, and the initial guesses are actually diverging from the true solution to the corrector equation. The

iteration may be diverging because the iteration matrix is very poorly conditioned, or the iteration matrix may be a very poor approximation to the Jacobian matrix because of errors in numerical differencing.

Given that all of the above situations are possible, our hope is to be able to diagnose at least some of these possibilities. Let us consider, for example, what happens when a code encounters a system with a ($m \geqq 3$) nonstate subsystem. For a linear or nearly linear problem where corrector convergence presents minimal difficulties, the usual response is for the stepsize to be reduced several times to try to satisfy the error test. Eventually, since the error test will never be satisfied, the stepsize is reduced to the point where the corrector iteration begins to diverge because the iteration matrix is very poorly conditioned. If we see this situation (the error test fails several times—and the error estimate is not reduced very much when h_{n+1} is reduced and then the corrector iteration fails to converge), then it is likely that the cause of the difficulty is this type of problem.

It is possible to use linear algebra routines [11] for solving the linear system which automatically generate an estimate of the condition number of the iteration matrix. This is expensive, especially for banded systems, but it may be worthwhile for differential/algebraic systems because this situation can happen so easily and if it is the cause of the difficulty, then the last thing we would want to do would be to reduce the stepsize.

One problem that we have not discussed very much here is that of finding a set of consistent initial conditions and an initial stepsize which reflects the scaling of the problem. We will not attempt to solve these problems here, but we will indicate why these problems are even more difficult than they may at first seem. From § 3, we know that for ($m \geqq 3$) nonstate subsystems, even the simplest numerical methods with constant stepsizes fail to give correct answers if they are started with the exact solution. Even restricting ourselves to problems which do not contain these subsystems, for nonlinear systems this is still a very difficult problem. From a practical point of view, even with a set of initial conditions and derivatives, we must be very careful about selecting the initial stepsize. If this stepsize is too small, we may fail to solve the problem because the iteration matrix is very poorly conditioned, even though the problem might have been solved successfully given a better (larger) choice of initial stepsize.

A final point which is of practical interest is the cost of computing the error estimate (4.5) and the information that we need to do this. If we restrict ourselves to systems of the form

$$B(t, y)y' - f(t, y) = F(t, y, y') = 0 \tag{5.11}$$

(where y' appears only linearly), we can avoid computing, storing, and multiplying by the matrix $E = \partial F/\partial y'$. (Note that if the iteration matrix $(\partial F/\partial y - h\beta \partial F/\partial y')$ is computed by numerical differencing each column can be computed using only one increment, as we are really finding $(\partial/\partial y_{n+1})F(t_{n+1}, y_{n+1}(y_{n+1} - y_n)/h_{n+1})$ instead of finding two separate matrices $\partial F/\partial y$ and $\partial F/\partial y'$, and adding them together. If this is done, then finding $\partial F/\partial y'$ separately is approximately twice as much work.) This can be done by requiring the user of the code to supply two separate routines. One routine computes $B(t, y)y'$ given (t, y, y')—that is, only those terms in (5.11) that involve y'. The other routine computes the full residual $B(t, y)y' - f(t, y)$, given (t, y, y'). (Alternatively, one routine which computes either of these possibilities, depending on the value of a flag, could be used.) The first routine can be used to compute $E(y_n^c - y_n^p)$ in the error estimate (4.5) given $(y_n^c - y_n^p)$ in place of y'. In many ways this seems to be a simpler

interface and requires less storage than others that have been used for these problems. For example, Hindmarsh and Painter [5] solve problems of the form (5.11), requiring the user to supply routines to compute F, and to add B to a given matrix. With our suggestion, instead of having to find the matrix $B(t, y)$ and add it to a given matrix, the user of the code need only distinguish those terms of (5.11) that involve y'.

Obviously, we do not have answers to all the questions which have been discussed in this section. These seem to be difficulties which have been neglected in the literature. We feel that anyone who is seriously writing a code for solving DAE systems or using such a code should at least be aware of these difficulties.

6. Summary. In this paper we have considered some of the many difficulties which can occur in solving differential/algebraic systems. The behavior of numerical methods applied to these systems differs from what we would expect based on experience from solving ODE's in several important ways. For linear systems of nilpotency $m \leqq 2$, we have noted that the basic algorithms (BDF) which are in use will work, in the sense that as long as stepsizes and orders are chosen so that the method is stable (Gear–Tu [12], Gear–Watanabe [13]) then the computed solution converges to the true solution as the maximum stepsize approaches zero. Error estimates based on the difference between the predictor and the corrector may be grossly inaccurate for these systems and can even cause codes to fail unnecessarily. To overcome these difficulties, we have suggested an error estimate which would more accurately estimate errors for these problems and would eliminate the other difficulties associated with the usual estimates. With the new error estimates, and possibly some changes to other strategies used in ODE codes, we believe that the algorithms (such as variable-stepsize BDF) which have been used in codes for solving differential/algebraic systems in the past could be used to solve DAE systems where solutions behave similarly to solutions of linear problems with nilpotency $m \leqq 2$, and possibly to diagnose other problems which cannot be solved with these algorithms. This is a significant improvement over past codes which could not deal adequately with problems of nilpotency $m = 2$.

On the other hand, we have also found that none of the algorithms such as BDF are adequate for solving (with varying stepsizes chosen automatically by a code) problems with nilpotency $m \geqq 3$. One alarming fact that has come into focus is that some error estimation schemes which have been proposed [2] or used [3] in other codes, would allow wrong answers to be computed for some variables, with absolutely no warning. For these types of systems, the solution does not converge (except under very severe restrictions on how fast the stepsize can change) as the maximum stepsize approaches zero. Instead, large errors may be introduced into the solution whenever the stepsize is changed. Because of this situation, it is wise to use extreme caution in any attempt to avoid controlling error in certain components of the solution of any differential/algebraic system. Finding initial conditions for these problems seems to be extremely difficult because even if we start with initial conditions equal to the exact solution, the solution in the first few steps may be grossly in error. While this may not be fatal for a linear problem, because later the approximation will converge to the true solution, for a nonlinear problem it could be disastrous.

It is well known that some differential/algebriac systems can be thought of as limiting cases of stiff systems (as the stiffness becomes infinite). We have constructed several of these problems and our tests confirm that codes based on BDF exhibit many of the difficulties that we have described here. In particular, for these problems the usual error estimates are very unsatisfactory, especially when the stepsize changes. This tends to cause the stepsize to be reduced until hL (where L is the Lipschitz

constant for the problem) is small. We do not know if this occurs very often in stiff problems which are of interest, but it may be a point worth considering in the design of stiff codes and algorithms.

REFERENCES

[1] C. W. GEAR, *Simultaneous numerical solution of differential-algebraic equations*, IEEE Trans. Circuit Theory, CT-18 (1971), pp. 89–95.

[2] R. F. SINCOVEC, B. DEMBART, M. A. EPTON, A. M. ERISMAN, S. W. MANKE AND E. L. YIP, *Solvability of Large Scale Descriptor Systems*, Boeing Computer Services Company, Seattle, WA, June 1979.

[3] R. L. BROWN AND C. W. GEAR, DOCUMENTATION FOR DFASUB—*A program for the solution of simultaneous implicit differential and nonlinear equations*, UIUCDCS-R-73-575, University of Illinois at Urbana-Champaign, 1973.

[4] T. RÜBNER-PETERSON, *An efficient algorithm using backward time-scaled differences for solving stiff differential algebraic systems*, Institute of Circuit Theory and Telecommunication, Technical University of Denmark, 2800 Lyngby, 1973.

[5] A. C. HINDMARSH AND J. F. PAINTER, Private communication.

[6] J. W. STARNER, *A numerical algorithm for the solution of implicit algebraic-differential systems of equations*, Tech. Rep. 318, Dept. of Mathematics and Statistics, Univ. of New Mexico, May 1976.

[7] A. R. CURTIS, *The* FACSIMILE *numerical integrator for stiff initial value problems*, Harwell Report AERE-R, 9352, 1978.

[8] F. R. GANTMACHER, *The Theory of Matrices*, Vol. 2, Chelsea, New York, 1964.

[9] R. SACKS-DAVIS, *Error estimates for a stiff differential equation procedure*, Math. Comp., 31 (1977), pp. 939–953.

[10] L. F. SHAMPINE, *Implementation of implicit formulas for the solution of ODE's*, this Journal, 1 (1980), pp. 103–118.

[11] A. K. CLINE, C. B. MOLER, G. W. STEWART AND J. H. WILKINSON, *An estimate for the condition number of a matrix*, SIAM J. Numer. Anal., 16 (1979), pp. 368–375.

[12] C. W. GEAR AND K. W. TU, *The effect of variable mesh size on the stability of multistep methods*, SIAM J. Numer. Anal., 11 (1974), pp. 1025–1043.

[13] C. W. GEAR AND D. S. WATANABE, *Stability and convergence of variable order multistep methods*, SIAM J. Numer. Anal., 11 (1974), pp. 1044–1058.