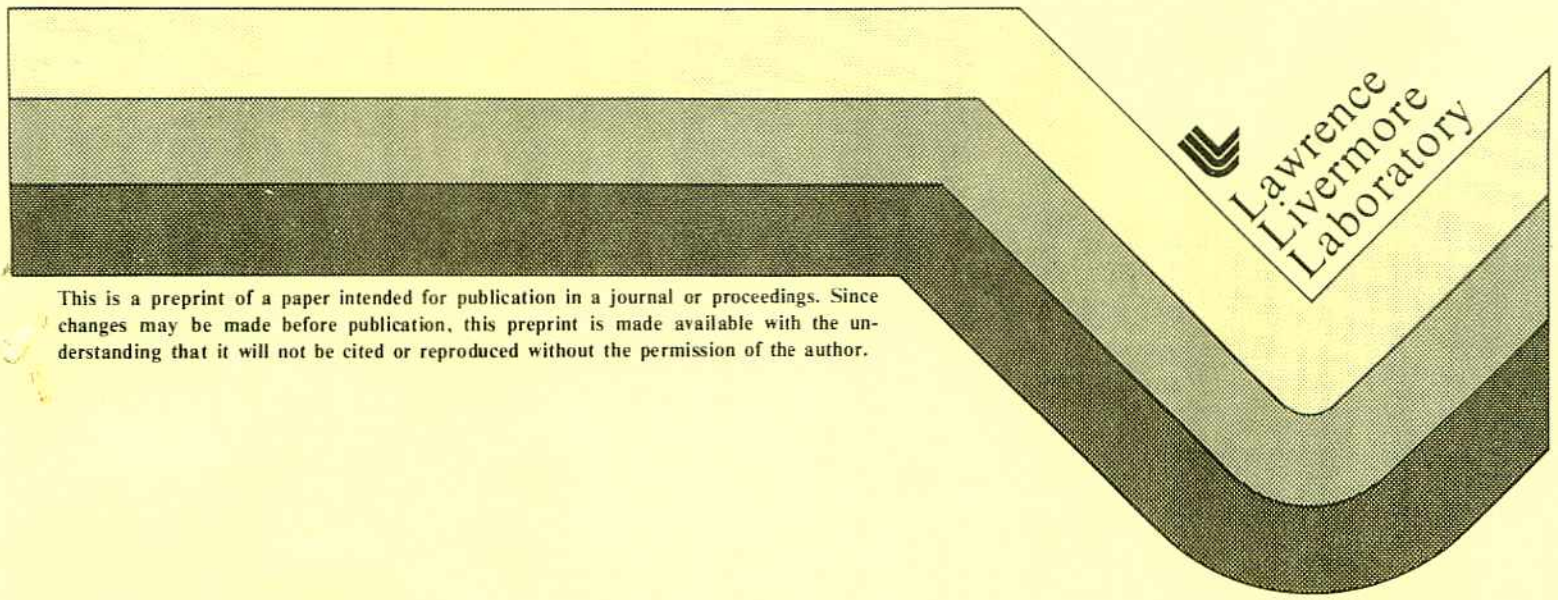
UCRL-83259
PREPRINT

# ON NUMERICAL METHODS FOR STIFF DIFFERENTIAL EQUATIONS--GETTING THE POWER TO THE PEOPLE

A. C. Hindmarsh

This paper was prepared for presentation at
COMPCON 1980, February 25-28, 1980, San Francisco

December 1979



# ON NUMERICAL METHODS FOR STIFF DIFFERENTIAL EQUATIONS -- GETTING THE POWER TO THE PEOPLE*

A. C. Hindmarsh

Lawrence Livermore Laboratory
Livermore, CA 94550

## Introduction And Example

As recently as 1960, the commonly held perception of ordinary differential equations (ODE's) in practical applications was that almost all of them could be solved with simple numerical methods widely available in textbooks. Many still hold that perception, but it has become more and more widely realized, by people in a variety of disciplines, that this is far from true. The biggest single reason for this is the recognition of stiff ODE systems as a frequent and widespread occurrence, and of methods that are effective for stiff problems, but are not so simple. Stiff systems are now known to arise commonly in chemical kinetics, structural and mechanical systems, electronic circuits, and many other areas, and these systems are being solved numerically as a fairly routine matter with appropriate choices of techniques and software, in a wide spectrum of computing environments. As this recognition was beginning to take place, however, two closely related challenges presented themselves. One was to investigate and extend the known classes of numerical ODE methods, and in the process discover and identify, by way of mathematical analysis, those that seem appropriate for stiff problems. The other was to implement the appropriate methods, using the growing technology in computing machinery and associated software, so as to make the power of the methods available to those with the problems to be solved. Although efforts in both categories are continuing at a strong pace, one class of methods, that of the so-called BDF methods, was found quite early to be effective on stiff problems, and its various implementations have since become far more successful than any others at bringing sophisticated ideas to bear on the stiff system problems faced by users at large. Following a description of the stiffness problem and a brief glimpse of BDF methods, what follows is a discussion of the ways in which these methods have achieved this success, and ways in which they and related ideas might do a better job in the future of meeting this challenge--of getting the power to the people where stiff ODE's are concerned.

First it is essential to understand stiffness itself. The essence of the phenomenon can be illustrated with a simple example problem. Consider the familiar LRC circuit, in which the current I satisfies the ODE

$$\ddot{I} + R\dot{I} + I/C = 0 \qquad (1)$$

*Work performed under the auspices of the U.S. Dept. of Energy by the Lawrence Livermore Laboratory under contract W-7405-ENG-48.

where the dot denotes differentiation with respect to time t. By defining constants $\alpha = \sqrt{LC}$ and $\beta = 2\ L/R$ (both having the dimensions of time), we can rewrite the equation as

$$\ddot{I} + 2\dot{I}/\beta + I/\alpha^2 = 0. \qquad (2)$$

The general solution is

$$I(t) = A_1 \exp(-t/\tau_1) + A_2 \exp(-t/\tau_2),$$

where $A_1$ and $A_2$ are constants and the $\tau_i$ are the roots of the quadratic equation

$$\tau^2 - 2\tau\alpha^2/\beta + \alpha^2 = 0.$$

Thus

$$\tau_1/\alpha = \alpha/\beta - \sqrt{(\alpha/\beta)^2 - 1} = (\tau_2/\alpha)^{-1}$$

$$\tau_2/\alpha = \alpha/\beta + \sqrt{(\alpha/\beta)^2 - 1}. \qquad (3)$$

Assume now that $\alpha >> \beta$. Then

$$\tau_2/\alpha \approx 2\ \alpha/\beta >> 1, \quad \tau_1/\alpha \approx \beta/2\ \alpha << 1.$$

The $\tau_i$ are the time constants of the system, in that they indicate the time scales of the two exponential decay terms, independent of the initial conditions. The long time constant, $\tau_2$, dictates the time span over which any particular solution will have to be given in order to be complete, while the short one, $\tau_1$, dictates the time scale of the initial rapid response, or fast transient, in the solution. To take a specific case, Fig. 1 shows the solution for the case

$$\alpha/\beta = 100,\ I(0) = 0,\ \dot{I}(0) > 0. \qquad (4)$$

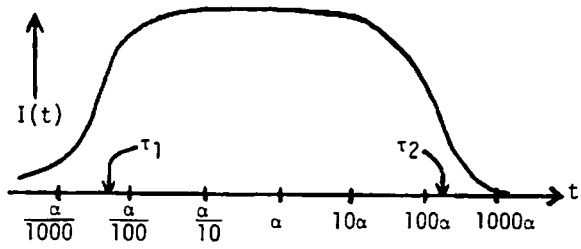



Figure 1.

The fast and slow responses are clearly shown in this semi-log plot.

The presence of a decay time constant that is very short, relative to the total time span of interest, is the defining feature of stiff problems. Note that beyond the initial fast tran-

sient, this short time constant is not at all evident in any particular solution curve, starting at realistic values of $I(0)$ and $\dot{I}(0)$. Yet it is inherent in the problem at all times, in the following sense: Given a particular solution $I(t)$ and any point $t_0 > 0$, the solution of the ODE starting at values perturbed slightly from $I(t_0)$ and $\dot{I}(t_0)$ will in general have a fast transient at $t_0$, on a time scale of the short time constant $\tau_1$, and then be smooth (on a scale of $\tau_2$) thereafter.

To obtain a more general definition of stiffness, first return to the ODE (2) and rewrite it as a first order system in the vector $y = (I,\dot{I})^T$, namely

$$\dot{y} = \begin{pmatrix} 0 & 1 \\ -1/\alpha^2 & -2/\beta \end{pmatrix} y = Ay. \qquad (5)$$

For linear systems of this form, with <u>any</u> matrix A, one can (almost always) write the solution as a linear combination of exponentials $\exp(\lambda t)$, where the values of $\lambda$ are the eigenvalues of A. If the equation is a stable one, these $\lambda$ will all have negative real parts, and the positive quantities $\tau = -1/Re(\lambda)$ are the associated time constants. (For the specific case in (5), with $\alpha > \beta$, these values of $\tau$ are exactly those obtained before, in (3).) Again, the occurrence of widely separated values of the $\tau$ is characteristic of stiffness.

Now consider the initial value problem for a general first-order ODE system,

$$\dot{y} = f(y,t). \qquad (6)$$

Here y is a vector of length N, and the independent variable t will be thought of as time, but need not actually be. The Jacobian matrix of the system is the N x N matrix of partial derivatives of f,

$$J(y,t) = \frac{\partial f(y,t)}{\partial y} . \qquad (7)$$

If $x(t)$ and $y(t)$ are two neighboring solutions of (6), their difference $z(t)$ is given roughly by the ODE

$$\dot{z}(t) = J(y(t),t)\, z(t). \qquad (8)$$

Thus an approximate local analysis leads to systems of the form $\dot{z} = Az$. We therefore say that the problem (6) is stiff if at least one of the time constants $\tau$, given by $\tau = -1/Re(\lambda)$ with $\lambda$ an eigenvalue of $J(y,t)$, is very small compared to the t range of interest. Since J and thus the $\tau$ can vary with t, and also vary as one changes the particular solution $y(t)$, one must qualify this definition in various ways.

Complete formality here is of little value to the scientist or engineer with a real problem to solve. The fundamental question to ask is: On the basis of at least a qualitative knowledge of the physical processes being modelled by the ODE system, is there a relatively fast decay process included in the system? If so, the system is most likely stiff. If intuition does not give a determination, a pragmatic approach will. If any of the classical textbook ODE methods, or software packages based on one of the methods, is attempted, and if the time step size $\Delta t$ that must be used in order to get decent answers seems inordinately small when measured on a plot of the solution, then the problem is most likely stiff. Thus, for example, such a method applied to the example problem (2)-(4) will require values of $\Delta t$ on the order of $\tau_1$ all the way through the problem, while the plot of the solution would suggest using values on the order of $\tau_2 \approx 4 \cdot 10^4\, \tau_1$ after the intial transient. Indeed the difference between using an appropriate method and an inappropriate one on a stiff problem can easily be a factor of $10^4$ or more in the number of time steps needed to finish the problem.

## BDF Methods - A Glimpse

While I will not attempt to present any stiff methods in any detail, it is useful to get at least a glimpse of what they entail. Two simple examples of methods for the general ODE problem (6) (initial value problem) will illustrate the central issues.

Consider first the simple linear problem (5). If a linear change of variables is made in y, the problem can be rewritten as two uncoupled scalar equations, each of the form

$$\dot{y} = \lambda y. \qquad (9)$$

The values of $\lambda$ are as before--the eigenvalues of A. The offending $\lambda$ is the one for which $\tau = -1/\lambda = \tau_1$ is small, or $\lambda$ is negative and large in magnitude, so consider only that equation.

The simplest of all ODE methods is Euler's, given by

$$y_n = y_{n-1} + h\dot{y}_{n-1}, \qquad (10)$$

where the subscripts are time step indices, $y_n$ approximates $y(t_n)$, $h = t_n - t_{n-1}$ is a constant step size in t, and $\dot{y}_{n-1}$ denotes $f(y_{n-1}, t_{n-1})$. For (9), the result is

$$y_n = (1 + h\lambda)y_{n-1}, \quad y_n = (1 + h\lambda)^n y_0.$$

Since the true solution is a decaying exponential, this Euler solution will be clearly useful only if $|1 + h\lambda| < 1$, or

$$h/\tau_1 < 2. \qquad (11)$$

For the case at hand, where $\tau_1$ is small, this is a prohibitive restriction.

The Backward Euler method, given by

$$y_n = y_{n-1} + h\dot{y}_n \qquad (12)$$

is an <u>implicit</u> one in the sense that the formula involves $y_n$ (in $\dot{y}_n = f(y_n,t)$) on the right as well as on the left side. For (9), the Backward Euler solution is

$$y_n = (1 - h\lambda)^{-1} y_{n-1}, \quad y_n = (1-h\lambda)^{-n}\, y_0.$$

Since $1 - h\lambda = 1 + h/\tau > 1$ for <u>any</u> choice of $h > 0$, this numerical solution is qualitatively correct for any h. For the general problem (6), however, this benefit comes only with the cost of solving an algebraic system for $y_n$ at each step. As real problems tend to be complex, nonlinear, and/or large in size, this cost can be quite significant, and this aspect of the method receives considerable attention in the various implementations. In contrast to what is commonly done in

the nonstiff case, it is not sufficient to use simple functional iteration,

$$y_{n(m+1)} = y_{n-1} + hf(y_{n(m)}, t_n), \qquad (13)$$

starting from some guess $y_{n(0)}$ (such as from (10)). For the test equation (9) this becomes

$$y_{n(m+1)} = y_{n-1} + h\lambda y_{n(m)}.$$

By setting $y_n = (1-h\lambda)^{-1} y_{n-1}$ and rewriting the above as

$$y_{n(m+1)} - y_n = h\lambda(y_{n(m)} - y_n),$$

it is clear that this iteration converges only if $|h\lambda| < 1$, again a prohibitive restriction on h, comparable to that in (11) for the ordinary Euler method. Instead, an iteration related to Newton's method, and therefore involving the Jacobian matrix (7) is generally used instead. For general f, such an iteration involves a sequence of linear algebraic systems of the form

$$(I-hJ)(y_{n(m+1)} - y_{n(m)}) =$$

$$y_{n-1} + hf(y_{n(m)}, t_n) - y_{n(m)}, \qquad (14)$$

where I denotes the identity matrix, and J is some approximation to $J(y_n,t_n)$, held fixed through the iteration sequence. This iteration has the desirable property that if f is linear in y, and $J = \partial f/\partial y$, then one gets the correct value for $y_n$ in one iteration. The two Euler methods have only first order accuracy. That is, in the limit of small h, the error at the end of a fixed t interval is O(h). Methods of higher order of accuracy are common, but relatively few of them have the property (shared by the Backward Euler Method) that is essential for stiff problems, namely that where the solution is smooth the step sizes can be correspondingly large, rather than constrained by the smallest time constant in the system. One group of methods that have this property is given by the Backward Differentiation Formulas, or BDF's. The BDF of order q is given by

$$y_n = \sum_{i=1}^{q} \alpha_i y_{n-i} + h\beta_0 \dot{y}_n , \qquad (15)$$

where the $\alpha_i$ and $\beta_0$ are constants (depending on q). (The name BDF comes from looking at (15) as a formula for $\dot{y}_n$ given $y_n$ and its past values $y_{n-i}$.) The case q = 1 is identical to (12), so these give generalizations of the Backward Euler method of orders q = 1, 2, ... . At high orders the desirable properties of the BDF's degenerate, and the implementations discussed below all limit q to 5.

The formula (15), as written, assumes h to be fixed. However, for most problems it is highly advantageous to vary h during the problem. This can be easily accomplished by approximating the needed past values (the $y_{n-i}$ in (15)) spaced at intervals of the new step size h' by way of the interpolating polynomial associated with the existing data spaced at the step size h. An alternative approach[1] is to use a variable-step analog of (15) to start with,

$$y_n = \sum_{i=1}^{q} \alpha_{ni} y_{n-i} + h_n \beta_{no} \dot{y}_n, \qquad (16)$$

in which the $\alpha$ and $\beta$ coefficients depend on the spacing of the $t_{n-i}$ (i = 0, ..., q), and this spacing is arbitrary. In both cases, a feature of the implementations that is nearly as valuable as the varying of h is the varying of the order q. Both are chosen dynamically in an attempt to maximize the efficiency of the numerical solution.

## Software Packages

The Backward Differentiation Formulas, together with known techniques for solving the associated implicit system of algebraic equations, for selecting the step size h, and for selecting the order q, have been made available to the computer user community in the form of software packages since about 1968. Such general-purpose ODE solvers have proved to provide a highly effective means of making the power of these methods available simultaneously to users in all disciplines in which stiff problems arise. Moreover, they greatly simplify the process of replacing old solution techniques with new and better ones as they come along, because the use of this software imposes a separation of the problem being solved from the method used to solve it.

Two basic packages using BDF methods are available for general use on stiff systems. These are the GEAR package, which is based on an earlier code called DIFSUB written by C. W. Gear, and EPISODE, which is newer. Although these two solvers look nearly identical to the user, they differ radically internally, in that GEAR uses (15) with step changing by interpolation while EPISODE uses the variable-step generalization of the BDF's in (16). In solving stiff systems, both use a modified Newton iteration which generalizes (14) and involves a matrix

$$P = I - h\beta_0 J \quad (\text{or } I - h_n\beta_{no} J). \qquad (17)$$

In carrying out the iteration, both GEAR and EPISODE assume that J is a full (dense) N x N matrix. (More will be said shortly on alternative assumptions.)

GEAR and EPISODE are written in reasonably portable Fortran, as demonstrated by the fact that they are currently in use at hundreds of installations on a wide variety of machines. In each case, the use of the solver requires the user to write a subroutine defining the right-hand side function f(y,t), and a calling program which communicates with a driver subroutine in the solver package. The latter call provides the value of N, the initial conditions, an error tolerance parameter (two in the case of EPISODE), a method option flag, and the value of t at which answers are desired. The solver returns answers as desired (or a flag indicating why it could not), and repeated calls are to be made to continue the solution. As an option, the user may also provide the Jacobian values J(y,t) needed in (17) by supplying another subroutine, or the solver may be made to generate these internally by means of difference quotients.

Another feature of these solvers that users have found extremely useful is an option for nonstiff systems. This option selects variable-order implicit Adams methods instead of BDF's, and functional iteration ((13) or its generalization to higher order formulas) instead of modified Newton iteration. Thus a user who has both stiff and nonstiff problems can select a suitably efficient method for either, simply by his choice of the method flag parameter.

For problems that demand frequent and considerable changes in step size, e.g., because of waves or spikes in the solution, the EPISODE package is a more efficient and more reliable choice than GEAR. For problems with fairly smooth solutions, both will do, but GEAR tends to have the greater efficiency.[2]

As mentioned at the beginning, stiff systems arise in a variety of applications. One type of application that is responsible for a large fraction (perhaps half) of the stiff systems being solved at present has received much special attention. This is the area of time-dependent partial differential equations (PDE's), or coupled systems of such equations, in which the spatial derivatives have been discretized in some way, leaving a large system of ODE's in time. This procedure, coupled with the use of an appropriate ODE solver, is referred to as the numerical method of lines, and has been developed for a variety of spatial geometries and discretization techniques. It poses a challenge to ODE solvers because the resulting systems tend to be quite large, as well as (usually) being stiff and nonlinear. Meeting this challenge has led to a sequence of variants of the GEAR and EPISODE solvers, which are also widely used.

The easiest category of PDE's to which this approach applies is that of systems of diffusion-like equations in one dimension. Here, any of several simple finite difference treatments of the spatial derivatives (and of the boundary conditions) leads to an ODE system $\dot{y} = f$ in which the Jacobian J is tightly banded about the main diagonal. Two variant solvers, called GEARB and EPISODEB, were written to take advantage of this, with great savings in storage and run time. These two packages, especially the older one, GEARB, have been used extensively, both directly by users and indirectly as a part of automatic general-purpose PDE packages. If instead one uses collocation, finite elements, or the Galerkin method for the space variable, the result is an implicit ODE system $A(y,t)\dot{y} = g(y,t)$, where A is an N x N matrix. Here A and $\partial g/\partial y$ are again banded, and two other variants, GEARIB and EPISODEIB, solve such systems. They treat the problem in the form given, which is much more economical than writing $\dot{y} = f = A^{-1}g$ and using one of the other solvers. GEARIB (modified slightly) is used quite successfully in a general purpose 1-D PDE solver called PDECOL[3], which uses collocation and polynomial splines to treat the spatial variable.

Two other variants of GEAR have been written with sparse Jacobian structures in mind. One, called GEARS, allows for a completely general sparse structure, and uses direct sparse linear system solvers. Another, called GEARBI, is for the case of a blocked Jacobian structure, and uses a block-iterative method (block-SOR) for the linear systems. The block structure is one that occurs frequently in 2-D PDE systems treated by finite differences on a rectangular mesh.

Further details of these solvers and their uses can be found in Ref. 4 and literature cited there.

## The Impact of the Hardware Environment

The computing environment in which stiff ODE systems are solved has a significant impact on the optimal choice of the means by which they are solved. This is sometimes due directly to the finite wordlength of the computer, sometimes to the nature of the memory components, and sometimes to the nature of the languages and supporting library software available.

To start with, the computer wordlength and associated roundoff error properties play a direct role in limiting the accuracy and degree of stiffness for which a numerical solution can be obtained. Specifically, suppose BDF methods are used on a machine with unit roundoff u. Beyond the rapid transient, the step size h is large compared to the smallest time constant in the system, $\tau_1$, and the degree of stiffness can be quantized by the stiffness ratio $S = h/\tau_1$. (Some definitions use the largest time constant in place of h, but the two are comparable when h is of reasonable size for the long-term solution.) Now the algebraic system that must be solved to get $y_n$ from past data can be shown to be ill-conditioned when S is large, in the sense that small perturbations in the data correspond (in the worst case) to large perturbations in the solution $y_n$--larger by a factor roughly equal to S. Thus the computed value of $y_n$ must be considered to have errors on the order of Su (measured in whatever norm is appropriate). On the other hand, the implementations of these methods control the error committed on taking one step (or an estimate of that error) to be less than some user-supplied tolerance parameter $\epsilon$ (again measured in an appropriate norm). Thus the problem is unlikely to be numerically solvable (with reasonable efficiency) unless Su is well below $\epsilon$. Given a problem which exceeds this limit, the observed behavior of any of the solvers described above is for the step size h to fail to reach the values expected of it, in terms of the smoothness of the long-term solution. A solution will usually still be obtained, but at the cost of restricted step sizes. Thus, for example, a problem with $S = 10^{10}$ (very stiff) is unlikely to be solvable efficiently on a machine having 8 significant decimal digits in single precision, and on a 14-digit machine it can probably be solved (efficiently) only with a few digits of accuracy.

On the basis of these considerations, most of the various ODE solvers listed have been prepared in both single and double precision versions in order to meet the variety of environments. (In the EPISODE family, the two versions of each solver are combined into one source, using comment cards, special flags, and a con-

verter routine, also in Fortran, which shifts the flags so as to change the source from single precision to double or vice versa.) As a general rule, the double precision version is recommended when solving stiff problems on machines having 9 or fewer significant decimal digits in the single precision floating point arithmetic.

At LLL and elsewhere, the memory configuration on the CDC 7600 machines has impacted the stiff system solvers in at least one instance. In order to handle very large systems (N > 14000) arising from coupled kinetics-transport PDE systems in two dimensions, the GEARBI package was rewritten so as to use the Large Core Memory (LCM) for some of the large work arrays. The resulting solver, called GEARBIL, copies data between LCM and the more rapid-access Small Core Memory in large blocks for maximum efficiency.

The move toward pipeline machine architectures has affected the ODE solvers considerably. On the 7600, a library of fast vector operation modules called STACKLIB (using stack-contained loops wherever possible), together with vector extensions to the Fortran language, were developed in order to simulate vector hardware instructions on the CDC STAR-100 computers. Serendipitously, they also improved the efficiency of codes on the 7600 considerably. The GEAR package was therefore rewritten to take advantage of these features. The result, GEARV, does in fact run faster for large problems, but not dramatically for stiff problems because the solution of the linear algebraic systems (done by elimination in GEAR) does not lend itself well to pipeline organization. A separate version of GEAR, called GEARST, was written for the STAR-100, but for stiff problems it is slower than GEAR on the 7600, for the reason given above. The GEAR package has also been successfully run on the CRAY-1 computer with virtually no alterations.

## Future Developments

In the coming years, the computational power available to users with difficult stiff ODE problems is likely to change in several respects. New discoveries in mathematical techniques, more powerful and more versatile hardware, and better ways of bringing these two areas together will all play valuable roles.

In the area of mathematical research on numerical methods, many opportunities are evident, of which I will mention only one. The nonlinear algebraic system problem resulting from each step with a BDF method is in principle amenable to a variety of techniques for solving such systems, one of which is the modified Newton scheme now used. More exotic choices, especially ones that take advantage of certain structural features in the system, are being studied, and others will no doubt be discovered. Even with the modified Newton choice, greater economy (in both storage and time) in the solution of the linear system at each iteration will be possible as new methods for structured linear systems are developed.

The effect of the hardware environment on the choice of numerical method has been and will be very important. This is exemplified by the case of tridiagonal matrix solvers on pipeline and parallel machines, where timing comparisons on the 7600 and STAR-100 caused renewed interest in methods previously discarded.[5]

The current advances in the power and complexity of integrated circuits is bound to impact the ODE area. In the forseeable future, however, the role of integrated circuit modules is likely to be limited to subtasks within ODE algorithms, rather than entire algorithms, because the state of these algorithms is still very fluid at present. Thus hardware modules for tasks *such as* solving linear systems could be very beneficial to the performance of stiff system solvers, as is currently demonstrated by the substitution of hand-coded assembly language modules for Fortran routines. Such benefits would of course require that the integrated circuit modules be accessible from the high-level language (e.g., Fortran) in which the ODE solver itself is written.

Finally, there have been and will be efforts made to formulate standards for ODE solvers. A tentative standard has been proposed for the user interface (the communication between user and solver) of Fortran ODE solvers.[6] If and when achieved and adopted widely by the authors of these solvers, this will greatly ease the burden users now bear in facing a variety of solvers with widely differing call sequences, etc. Eventually, standards for the internal structure of ODE solvers are also likely. This will have the effect of identifying standard subtasks common to many solvers, for which fast modules (hardware or software) would be well worth building.

## References


[1] G. D. Byrne and A. C. Hindmarsh, "A Polyalgorithm for the Numerical Solution of Ordinary Differential Equations," ACM-Trans. Math. Software, 1 (1975), 71-96.

[2] G. D. Byrne, A. C. Hindmarsh, K. R. Jackson, and H. G. Brown, "A Comparison of Two ODE Codes: GEAR and EPISODE," Computers & Chem. Eng., 1 (1977), 133-147.

[3] N. K. Madsen and R. F. Sincovec, "Algorithm 540. PDECOL, General Collocation Software for Partial Differential Equations," ACM-Trans. Math. Software, 5 (1979), 326-351.

[4] A. C. Hindmarsh, "A Collection of Software for Ordinary Differential Equations," in the Proceedings of the ANS Topical Meeting on Computational Methods in Nuclear Engineering, Williamsburg, VA, April 23-25, 1979; also available as LLL Report UCRL-82091.

[5] G. H. Rodrigue, N. K. Madsen, and J. I. Karush, "Odd-Even Reduction for Banded Linear Equations," J. Assoc. Comp. Mach. 26 (1979) 72-81.

[6] A. C. Hindmarsh, "A Tentative User Interface Standard for ODEPACK," LLL Report UCID-17954, October 1978.

*Technical Information Department* · Lawrence Livermore Laboratory
University of California · Livermore, California 94550