ALGORITHMS AND SOFTWARE FOR ORDINARY DIFFERENTIAL EQUATIONS AND DIFFERENTIAL-ALGEBRAIC EQUATIONS, PART II: HIGHER-ORDER METHODS AND SOFTWARE PACKAGES

Alan C. Hindmarsh and Linda R. Petzold

Department Editor: William J. Thompson

This two-part article describes methods for solving systems of differential equations. Part I, which appeared in the previous issue, reviews the explicit Euler method, discusses "stiffness," and describes how and why the implicit Euler method can provide useful solutions of stiff systems. Part I concludes with a consideration of errors and error estimates. Part II extends the discussion to higher-order methods of both the multistep and one-step varieties. Part II gives special attention to large stiff systems and differential-algebraic systems. The article concludes with a description of relevant software packages that are freely available from Netlib on the Internet.

Because of their simplicity, we have been using the explicit and implicit Euler methods to illustrate some basic concepts. Both have first-order accuracy: the global errors are $O(h)$ for a maximum step size of h. In most problems, however, computational efficiency can be considerably increased by using higher-order methods that are generalizations of these simple methods. The importance of higher-order methods is that they are often able to achieve the same level of accuracy as lower-order methods but with many fewer steps and, hence, with much more efficiency. The higher-order methods fall primarily into two classes, multistep methods and one-step methods.

Alan C. Hindmarsh is a mathematician in the Center for Computational Sciences and Engineering at Lawrence Livermore National Laboratory, Livermore, CA 94550. Linda R. Petzold is a professor in the Department of Computer Science, University of Minnesota, Minneapolis, MN 55455. Hindmarsh and Petzold have been responsible for developing numerous software packages for the solution of ordinary differential equations and differential-algebraic equations.

Multistep methods

Multistep methods make use of several past values of $\mathbf{y}$ and/or $\mathbf{f}$ to achieve a higher order of accuracy for the ODE of Eq. (1) in Part I. The general form of a k-step multistep method is

$$\sum_{j=0}^{k} \alpha_j \mathbf{y}_{n-j} = h \sum_{j=0}^{k} \beta_j \mathbf{f}_{n-j}. \tag{1}$$

where α_j and β_j are constants that depend on the order, and possibly on previous step sizes, and $\alpha_0 \neq 0$. The quantities $\mathbf{y}_{n-j}$ and $\mathbf{f}_{n-j}$ represent values of $\mathbf{y}$ and $\mathbf{y}'$, respectively, at the points t_{n-j}. The method is explicit if $\beta_0 = 0$ and implicit otherwise. Here, $h = h_n = t_n - t_{n-1}$.

Several important classes of multistep methods have proven very efficient and robust for solving various types of ODE systems. Adams methods make use of past values of $\mathbf{f}$, and are written

$$\mathbf{y}_n = \mathbf{y}_{n-1} + h \sum_{j=0}^{k} \beta_j \mathbf{f}_{n-j}. \tag{2}$$

Equation (2) gives a method of order $k+1$; i.e., global errors are $O(h^{k+1})$.

The Adams methods are the best known multistep methods for solving general *nonstiff* systems. Several popular codes are based on these methods, which are stable up to order 12 for nonstiff problems. Each step requires the solution of a nonlinear system,

$$\mathbf{y}_n = \mathbf{a}_n + h\beta_0 \mathbf{f}(t_n, \mathbf{y}_n),$$

($\mathbf{a}_n$ = past history terms). But rather than use the modified Newton procedure described earlier, this system is nearly always solved by simple *functional iteration*. Here, from a

predicted value $\mathbf{y}_{n(0)}$, one simply iterates on the function in Eq. (2) whose fixed point is sought:

$$\mathbf{y}_{n(m+1)}=\mathbf{a}_n+h\beta_0\mathbf{f}(t_n\mathbf{y}_{n(m+1)}).$$

This converges reasonably well for nonstiff systems and has the advantage that no linear system has to be solved. For this reason, most people refer to the Adams/functional-iteration combination as an explicit method, even though the underlying formula is implicit.

The most effective multistep methods for solving *stiff* systems are the *backward differentiation formulas* (BDFs). BDF methods make use of past values of y to advance the solution, according to the formula

$$\mathbf{y}_n=\sum_{i=1}^{k}\alpha_i\mathbf{y}_{n-i}+h\beta_0\mathbf{f}_n. \tag{3}$$

The reason for the name is that, on identifying $\mathbf{f}_n$ with $\mathbf{y}'_n$, Eq. (3) is a formula for approximating $\mathbf{y}'(t_n)$ (differentiation) in terms of current and past (backward) values of $\mathbf{y}_j$.

The BDF method based on Eq. (3) has order k; i.e., global errors are $O(h^k)$, and it is stable up to order six. The nonlinear system at each time step is almost always solved by some form of Newton iteration, which usually accounts for much, if not most of the total cost of obtaining the solution. Each Newton iteration involves the solving of an $N\times N$ linear system

$$A\Delta\mathbf{y}=\text{residual vector}$$

for a correction to $\mathbf{y}_n$, in which the coefficient matrix is

$$A=I-h\beta_0J,\quad J=\partial\mathbf{f}/\partial\mathbf{y}, \tag{4}$$

and J is evaluated at some nearby value of t and $\mathbf{y}$. (I denotes the $N\times N$ identity matrix.) Functional iteration is ruled out in the stiff case, because stiffness produces a large value for the Lipschitz constant L of $\mathbf{f}$ with respect to $\mathbf{y}$ (the maximum of the norm of $\partial\mathbf{f}/\partial\mathbf{y}$), and convergence of functional iteration requires $hL<1$; thus step sizes h are severely restricted, just as they are for a nonstiff method such as explicit Euler.

Many widely used codes for solving ODE systems are based on this class of methods. A representative code is the solver LSODE.[1] In addition, BDF methods are very well suited for solving DAE systems. For the general form $\mathbf{F}(t,\mathbf{y},\mathbf{y}')=0$, this means requiring $\mathbf{y}_n$ to satisfy

$$\mathbf{F}\left(t_n,\mathbf{y}_n,\frac{\mathbf{y}_n-\sum_{i=1}^{k}\alpha_i\mathbf{y}_{n-i}}{h\beta_0}\right)=0. \tag{5}$$

Several popular DAE codes are based on these methods, the most well-known being the solver DASSL.[2] (More will be said about software in a later section.)

The BDF methods were originally proposed and used in *fixed-step* form, where the coefficients α_i and β_0 in Eq. (3) depend only on the order k. In the implementations (e.g., LSODE), the step sizes h are actually allowed to change periodically in accordance with a test on the estimated local error, and the steps following a step size change use interpolated values for the $\mathbf{y}_{n-i}$ at the new step size. However, many problems demand frequent changes of step size, and for them the fixed-step BDF methods can lose efficiency or even reliability. In fact, diurnal chemical kinetics problems, such as in the ozone model given in Part I, first demonstrated the need for variable-step forms for BDF methods. Two different variable-coefficient forms of BDF methods have been developed. In both, the method coefficients are recomputed at every step as a function of the actual step sizes $h_n, h_{n-1},\ldots$ used over the last k steps. But in one version, the so-called *fixed-leading-coefficient* version of BDFs, the value of β_0 does not vary; it depends only on k. This has important consequences for the Newton iteration used, which will be discussed later. The ODE solver VODE[3] and the DAE solver DASSL are representative of codes that use this form of the BDF methods.

Multistep methods are more complex than one-step methods, both to analyze and to implement. Their stability depends on the behavior of the solutions to the difference equation (1). This equation has several fundamental solutions. Coefficients in this method must be chosen so that the extraneous solutions to the difference equation (that is, solutions that do not approximate the solution of the ODE or DAE) do not grow. A robust and efficient implementation of a code based on multistep methods is far from straightforward. Issues that must be dealt with include deriving stable variants of the formulas that are applicable for variable step sizes, estimating errors and changing the step size and order of the method as the problem changes, obtaining suitable starting values, deciding when to terminate the nonlinear iteration, and determining appropriate starting step sizes. These issues are even more complicated for DAE systems, for which much of the ODE methodology is inapplicable.

One-step methods

The second class of higher-order methods is that of one-step methods. Unlike the multistep methods, these methods do not make use of past values of $\mathbf{y}$ or $\mathbf{f}$ [for the ODE system of Eq. (1) in Part I] to achieve a higher order. Instead, they depend on evaluations of the differential equation at judiciously chosen locations within the current time step. Such methods are known as Runge–Kutta methods, or extrapolation methods, which are actually a special case of general Runge–Kutta methods. Runge–Kutta methods were discussed in considerable detail by John Butcher in a previous PNA column,[4] but we will give a short description here for the sake of completeness. A single step with a Runge–Kutta method for the ODE $\mathbf{y}'=\mathbf{f}$ is defined by a set of equations of the form

$$\mathbf{y}_n=\mathbf{y}_{n-1}+h\sum_{i=1}^{s}b_i\mathbf{k}_i,$$
$$\mathbf{k}_i=\mathbf{f}\left(t_{n-1}+c_ih,\mathbf{y}_{n-1}+h\sum_{j=1}^{s}a_{ij}\mathbf{k}_j\right),\quad i=1,2,\ldots,s. \tag{6}$$

This defines an s-stage Runge–Kutta method. Such a method can be either explicit ($a_{ij}=0$ for $j\geq i$) or implicit,

and some implicit choices are useful for stiff problems. One-step methods offer advantages over multistep methods for some problems. For problems with frequent discontinuities, one-step methods are easier to restart at a high order. For stiff systems with highly oscillatory modes, one-step methods are stable with a higher order of accuracy than multistep methods.

A difficulty in implementing one-step methods is finding an efficient solution of the nonlinear system, which is in general larger than for multistep methods. Another is obtaining the solution at points between time steps. This latter task is easily accomplished with multistep methods via a polynomial that passes through past values of $\mathbf{y}$ or $\mathbf{f}$. For most problems, it is quite difficult to write a one-step code that is competitive with the best multistep codes. Implicit Runge–Kutta methods are potentially useful for some DAE systems also, but there is in general an additional set of order conditions which the method coefficients must satisfy to achieve a given order.[5]

Large stiff systems

ODE systems that are both stiff and large (in number of ODEs) are especially challenging, even if given in the explicit form of Eq. (1) in Part I. As indicated above, an implicit method then leads to a nonlinear algebraic system that must be solved at every time step. The size and complexity of such systems may make conventional treatments prohibitive in computational cost or memory storage, or both. Considerable research is currently devoted to this class of problems.

For a given time step, we can write the nonlinear system as $\mathbf{F}(\mathbf{y})=0$, where $\mathbf{F}$ is related to the function $\mathbf{f}$ in $d\mathbf{y}/dt=\mathbf{f}(t,\mathbf{y})$ by the equation $\mathbf{F}(\mathbf{y})=\mathbf{y}-\mathbf{a}_n-h\beta_0\mathbf{f}(t_n,\mathbf{y})$. By the well-known process of Newton's method, we generate successive approximations to the desired solution vector $\mathbf{y}$ by adding corrections that are defined by an approximate linear system. This reduces the problem to a sequence of large linear systems, which we write simply as

$$A\mathbf{x}=\mathbf{b}. \tag{7}$$

Here $\mathbf{b}$ is a vector of residuals [the negative of $\mathbf{F}(\mathbf{y})$ for the current approximation to $\mathbf{y}$], A is a matrix related to the Jacobian J of $\mathbf{f}$, namely $A=I-h\beta_0 J$, and $\mathbf{x}$ is the unknown vector of corrections to $\mathbf{y}$.

Instead of relegating this problem to a standard linear-system solution algorithm, an approach that can be much more effective is the use of *iterative methods*. One starts with a guess $\mathbf{x}_0$ (we use $\mathbf{x}_0=0$), and corrects it successively to get iterates $\mathbf{x}_1$, $\mathbf{x}_2$,... . Many iterative methods for linear systems are known, but some are much more appealing than others in the setting of large stiff systems. Such methods are known as Krylov subspace iteration methods. Their crucial property is that at each iteration they require only the value of the matrix–vector product $A\mathbf{v}$ for a given vector $\mathbf{v}$. That is, if m iterations have been done, so that one has $\mathbf{x}_0,\mathbf{x}_1,\ldots,\mathbf{x}_m$ (or some equivalent set of vectors), a vector $\mathbf{v}$ is generated as a linear combination of these vectors, and the next iterate $\mathbf{x}_{m+1}$ is a linear combination of $A\mathbf{v}$ and the older vectors. Many methods of this type (such as conjugate gradient iteration, for example) are known to work well when A has certain special properties (such as symmetry), but only a few are good candidates when no such assumptions about A are made. These are the most useful choices, because no special properties can be assumed about the function $\mathbf{f}$ from which A is obtained.

Given a suitable Krylov method, it can be exploited to best advantage by finding an efficient way to calculate products $A\mathbf{v}$ that does not entail calculating the matrix A itself. To do this, we note that A is just the matrix of partial derivatives of $\mathbf{F}(\mathbf{y})$, just as J is that of $\mathbf{f}$. This implies that for a suitably small constant ϵ, $[\mathbf{F}(\mathbf{y}+\epsilon\mathbf{v})-\mathbf{F}(\mathbf{y})]/\epsilon$ is a good approximation to $A\mathbf{v}$. The value of $\mathbf{F}(\mathbf{y})$ is already available, and the value of $\mathbf{F}(\mathbf{y}+\epsilon\mathbf{v})$ is easily expressed in terms of $\mathbf{f}(t,\mathbf{y}+\epsilon\mathbf{v})$. Thus the Krylov iteration proceeds by making one evaluation of $\mathbf{f}$ and some simple vector operations at each iteration until convergence of the iterates is achieved to within a suitable tolerance. When Newton's method and Krylov iteration are combined with, for example, a BDF method for the ODE system, the result is a *matrix-free* method for stiff systems. In contrast to traditional stiff-system methods, such a method involves no explicit construction or storage of the matrices J or A.

Working from the solver LSODE, which uses BDF methods for stiff ODE systems, we wrote another solver that combines the Krylov methods described above with BDF integration.[6,7] When tested, it worked well on many of the test problems but failed badly on many others. The reason is that Krylov methods are just not powerful enough, by themselves, to handle with acceptable efficiency the wide variety of matrices A that can occur. However, they can be assisted greatly by a technique known as *preconditioning*.

Suppose we can find a matrix P (the preconditioner matrix) that resembles A to some extent but is much easier to construct and operate with. In particular, suppose that we can solve linear systems $P\mathbf{x}=\mathbf{b}$ reasonably efficiently. To solve $A\mathbf{x}=\mathbf{b}$, we write an equivalent system, say $(AP^{-1})(P\mathbf{x})=\mathbf{b}$, with a different matrix $A'=AP^{-1}$ and a different solution vector $\mathbf{x}'=P\mathbf{x}$, and apply the Krylov method to the problem $A'\mathbf{x}'=\mathbf{b}$. Each iteration requires the evaluation of a product $A'\mathbf{v}=AP^{-1}\mathbf{v}$, but that is achieved by solving $P\mathbf{w}=\mathbf{v}$ for $\mathbf{w}$ and then approximating $A\mathbf{w}$ as before. If the iteration converges to a vector $\mathbf{x}'$, then the vector we want is $\mathbf{x}=P^{-1}\mathbf{x}'$, or the solution of $P\mathbf{x}=\mathbf{x}'$. Convergence is more likely to occur now, because A' is closer to the identity matrix, depending on how close P is to A. This arrangement is called preconditioning on the right (since P^{-1} multiplies A on the right), but one can just as easily precondition on the left, by writing $(P^{-1}A)\mathbf{x}=P^{-1}\mathbf{b}$. In fact, one can precondition on both sides, with two preconditioners P_1 and P_2 whose product approximates A.

To incorporate the idea of preconditioning, we wrote another LSODE variant, called LSODPK, containing a selection of preconditioned Krylov methods to solve the linear-system problem.[8] The Krylov methods available are

preconditioned-conjugate-gradient (PCG), the Arnoldi method, and the generalized-minimal-residual method (GMRES).[9] LSODPK works well on many test problems that could not be handled without preconditioning. Because the choice of preconditioner can best be made by exploiting the structure of the problem, the user of LSODPK must supply the preconditioner. That is, in terms of the ODE system itself, the user must identify the most important contributions to the Jacobian matrix J (that is, to the stiffness of the ODE system), find a way to represent and operate with these contributions in an economical manner, and then use them to build one or two preconditioner matrices P_1 and P_2. For a complicated problem, the user's job may seem to require as much effort as constructing a complete solution method for the problem from scratch. But it does not, because it focuses on the linear system aspect of the solution only, while the solver takes care of accounting for the errors associated with the choice of preconditioners, for the nonlinear iteration surrounding the linear system, and for the accuracy of the time-stepping procedure.

Although the construction of good preconditioners depends heavily on the nature of the problem, considerable experience has been built up with respect to certain classes of problems. For ODE systems that arise from the spatial discretization of time-dependent systems of PDEs, two natural choices are typically available. First, the terms in the PDEs that reflect how the different PDE components are coupled to each other at each spatial point give rise to one type of preconditioner, which we call the interaction preconditioner. Second, the terms that reflect how each PDE component is transported in space can be used to construct another type of preconditioner, which we call the transport preconditioner. For example, in the ozone model given at the beginning, the chemical kinetics terms R_i lead to an interaction preconditioner and the diffusion terms lead to a transport preconditioner. If both contributions are important, then either they can be regarded as the two preconditioners P_1 and P_2 needed by LSODPK or their product can be used as a single preconditioner on either side.

One particular problem solved by this approach is a system of PDEs on a two-dimensional spatial grid with a discretized frequency variable that represents a laser-oscillator model. We developed a pair of preconditioners, first by considering the interaction and transport contributions separately, but later with a modification motivated by the Jacobian structure whereby some interaction coefficients were moved to the transport preconditioner. The size of the ODE system varied up to 38 745, and LSODPK generated solutions with complete success.

After seeing how successful the combination of Krylov and BDF methods was with the LSODPK solver, we generated a similar combination with the VODE solver, called VODPK .[10] In this case, the Krylov method chosen is the GMRES method. In using VODPK on a large stiff system, the power and generality of variable-coefficient BDF methods, Newton iteration, and GMRES iteration is combined with a user-supplied preconditioner (or preconditioner pair) that incorporates problem-specific information where it is most needed.

Differential–algebraic systems

Many physical phenomena are most naturally described by a system of differential-algebraic equations of the form

$$\mathbf{F}(t,\mathbf{y},\mathbf{y}')=0. \tag{8}$$

This type of system occurs frequently as an initial-value problem in modeling electrical networks, the flow of incompressible fluids, mechanical systems subject to constraints, robotics, distillation processes, power systems, trajectories, control systems, and in many other applications. Differential-algebraic systems are different from ODE systems in that, while they include ODE systems as a special case, they also include problems that are quite different from ODEs. Some of these systems can cause severe difficulties for numerical methods. Consequently, the numerical solution of these systems is a very active area of research. We outline some of the key ideas here; they are described in greater detail in Ref. 11.

In a sense, the more singular a DAE system is, the more difficult it is to solve numerically. The *index* of a DAE system is a measure of its degree of singularity. Roughly speaking, ODE systems $\mathbf{y}'=\mathbf{f}(t,\mathbf{y})$ have *index zero.* Differential equations coupled with algebraic constraints [that is, $\mathbf{y}'=\mathbf{f}(\mathbf{y},\mathbf{z}),0=\mathbf{g}(\mathbf{y},\mathbf{z})$] have *index one* if $\mathbf{g}=0$ can be solved for $\mathbf{z}$ given $\mathbf{y}$ (that is, if $\partial\mathbf{g}/\partial\mathbf{z}$ is nonsingular) and otherwise have an index higher than one. The index can also be defined for systems that are not expressed in the semiexplicit form of differential equations coupled with algebraic constraints. Additional difficulties can arise for these systems because the singularity may be moving from one part of the system to another.

A simple example of a higher-index system is given by the equations describing the motion of a pendulum in Cartesian coordinates. Let L denote the length of the bar, λ the force on the bar (suitably normalized), and x and y the coordinates of the infinitesimal ball of mass one located at the free end of the bar. Then x, y, and λ solve the DAE system

$$\begin{aligned} x''&=\lambda x,\\ y''&=\lambda y-g,\\ 0&=x^2+y^2-L^2, \end{aligned} \tag{9}$$

where g is the gravitational constant.

The index of this system is three. While this simple system can be easily rewritten as a standard ODE system by converting to radial coordinates, this is often not practical for the much larger systems that are automatically generated by simulation packages designed to model complicated physical networks.

An even simpler example of a higher-index system, which illustrates some of the ways in which these singular systems are quite different from ODEs, is given by

$$\begin{aligned} y&=g(t),\\ x&=y'. \end{aligned} \tag{10}$$

The index of this system is two. While it looks superficially similar to an ODE system, there are important differences. The solution is less continuous than the input function $g(t)$. There is no family of solutions corresponding to an arbitrary choice of initial values. Rather, the initial values (in fact, all values) are completely determined in terms of the function g and its derivative. Finally, it is clear that there is an implied differentiation to obtain x. Since numerical differentiation is notoriously ill-conditioned (sensitive to small errors), difficulties for numerical ODE methods can be expected when there is a higher-index subsystem present in the system.

Over the past decade, a theoretical framework has been developed for understanding the order, stability and convergence of linear multistep and Runge–Kutta methods applied to general index-one and to index-two and index-three systems that can be written in a semiexplicit triangular form that commonly occurs in applications. Not all ODE methods are appropriate for DAEs; the theory shows which methods are stable and accurate. Often for DAEs there is also a choice of formulations of the equations. Different formulations may have the same exact solution but differ considerably in their properties for numerical solution. Recent work has focused on finding appropriate formulations for classes of problems in applications that are advantageous for stability and accuracy of the numerical solution.[12]

The development of codes for DAEs is not a straightforward task because of difficulties in the computation arising from the singular part of the system and the coupling to the differential part, which do not occur for ODE systems. In particular, starting, error estimation, and solving the nonlinear system all present potential difficulties even for index-one systems, and especially for higher-index systems. We have developed a Fortran package called DASSL,[2] which uses fixed-leading-coefficient BDF methods for index-one DAEs. Complete details of the algorithm are available in the book by Brenan *et al.*[11] DASSL has been used successfully for solving a wide range of problems at various universities, laboratories, and in industry, both in the U.S. and in several foreign countries. With some modification as described in Ref. 11, DASSL can also be used to solve index-two systems. Codes for DAEs based on Runge–Kutta methods have also been developed; see for example Ref. 5. These methods are particularly effective for problems with frequent discontinuities.

In contrast to the situation for ODEs, initial conditions for DAEs must be *consistent*, in the sense that they must satisfy the constraints of the system and possibly also some of the derivatives of the constraints. For example, for the pendulum problem (9), the constraint and its first and second time derivatives must be satisfied at the initial time, leading to

$$\begin{aligned} 0 &= x^2 + y^2 - L^2, \\ 0 &= xx' + yy', \\ 0 &= \lambda L^2 - gy + (x')^2 + (y')^2. \end{aligned} \tag{11}$$

Currently, the user computes these consistent initial conditions, using his or her knowledge of the problem and a nonlinear system solver. We are working on a software package to be used in combination with DASSL or its extensions, which would make this task more routine for many index-one systems. Methods for finding consistent initial conditions for higher-index systems are described for example in Ref. 13.

The success of Krylov iteration methods combined with the ODE solvers LSODE and VODE has inspired the same approach for DAE systems. Accordingly, we developed a variant of DASSL, called DASPK, that combines the preconditioned GMRES Krylov iterative method with the BDF methods of DASSL, as applied to DAE systems.[14]

Software packages

Even the best numerical method is unlikely to find wide acceptance until it is embodied in a computer code that is made available for general use. In that spirit, much of our work on methods for ODE and DAE systems has been accompanied by the development of software packages. It is important to understand that this process is not simply a direct translation of a set of formulas into a suitable programming language. Initially, it entails a multitude of decisions on representing and manipulating the relevant data most efficiently and on carrying out all of the numerical processes that together constitute a complete algorithm. The resulting computer code is tested on a wide variety of problems to see that it performs as expected. Then, at some point, it is given to users, along with suitable documentation, so that it can be tried out on realistic problems. All of these phases generate feedback that may result in revision or rewriting of parts of the code. A code often goes through several such feedback-revision cycles during its lifetime.

Various general-purpose packages have been written by the authors of this article to solve systems of ODEs and/or DAEs. These packages are listed in Table I. Details of the algorithms are available in the various references. Nearly all of the packages listed are available from the Energy Science and Technology Software Center in Oak Ridge. A survey paper on stiff ODE solvers discusses various software, applications, examples, and related issues.[15] Reference 11 discusses DAE issues, applications, and software.

A great deal of useful software for solving ODEs and a wide variety of other numerical and non-numerical problems are available freely on the Internet via Netlib.[16] This includes most of the codes listed here. One can obtain an index of Netlib ODE software by

```
mail netlib@ornl.gov
Subject: send index from ode
```

The netlib system will then mail back an index of ODE solvers and descriptions. To obtain one of these solvers (for example to obtain DDASSL—double-precision DASSL), send the following message

```
mail netlib@ornl.gov
send ddassl from ode
```

On many X-window systems, an interactive version of

Table I. General-purpose multistep packages available from the authors for solving systems of ODEs and/or DAEs.

Solver	Problem	Comments
LSODE	$\mathbf{y}'=\mathbf{f}(t,\mathbf{y})$	User specifies stiff or nonstiff method; allows dense or banded Jacobian matrix in stiff case
VODE	$\mathbf{y}'=\mathbf{f}(t,\mathbf{y})$	Like LSODE, but with variable-coefficient methods internally
LSODES	$\mathbf{y}'=\mathbf{f}(t,\mathbf{y})$	LSODE variant for general sparse Jacobian
LSODA	$\mathbf{y}'=\mathbf{f}(t,\mathbf{y})$	Automatically, dynamically determines where problem is stiff, and chooses appropriate method; allows dense or banded Jacobian matrix
LSODAR	$\mathbf{y}'=\mathbf{f}(t,\mathbf{y})$	Same as LSODA but includes additional root-finding stopping criteria
LSODI	$M(t,\mathbf{y})\mathbf{y}'=\mathbf{g}(t,\mathbf{y})$	Solves linearly implicit ODE or DAE system; allows dense or banded coupling
LSOIBT	$M(t,\mathbf{y})\mathbf{y}'=\mathbf{g}(t,\mathbf{y})$	Same as LSODI but allows block-tridiagonal coupling
DASSL	$\mathbf{F}(t,\mathbf{y},\mathbf{y}')=0$	Solves index-one DAE systems; allows dense or banded coupling
DASRT	$\mathbf{F}(t,\mathbf{y},\mathbf{y}')=0$	Same as DASSL but with additional root-finding stopping criteria
LSODPK	$\mathbf{y}'=\mathbf{f}(t,\mathbf{y})$	LSODE variant; has preconditioned Krylov iterative methods for linear systems
LSODKR	$\mathbf{y}'=\mathbf{f}(t,\mathbf{y})$	Like LSODPK, but with root-finding and automatic Newton/functional iteration switching
VODPK	$\mathbf{y}'=\mathbf{f}(t,\mathbf{y})$	VODE variant; has preconditioned Krylov iterative methods for linear systems
DASPK	$\mathbf{F}(t,\mathbf{y},\mathbf{y}')=0$	DASSL variant; allows selection of direct methods or preconditioned Krylov iterative methods for linear systems
CVODE	$\mathbf{y}'=\mathbf{f}(t,\mathbf{y})$	Rewrite of VODE and VODPK in C

netlib called Xnetlib is available.

The Fortran solver called LSODE is the outcome of a lengthy evolutionary process.[1,17] (LSODE was written in 1979, but the comprehensive documentation[1] was only recently completed.) LSODE solves ODE initial value problems that are given in the explicit form of Eq. (1) in Part I. It allows a user to select between an Adams method (for nonstiff systems) and a BDF method (for stiff systems), using the fixed-step-interpolatory form for both of these methods. When solving a stiff system, and therefore when dealing with the Jacobian matrix J in Eq. (4), LSODE assumes that the matrix is either full (dense) or banded (has nonzero elements located near its main diagonal). Users can either supply J with coding of their own or let LSODE generate an approximation to J internally. Jacobians generated internally are computed as finite-difference quotient approximations. In the dense case, this uses N extra **f** evaluations, and in the banded case with bandwidth M it uses M extra **f** evaluations.

The variable-coefficient solver called VODE was written more recently.[3] VODE looks nearly identical to LSODE as far as its usage is concerned, but the internal algorithm is considerably different. VODE uses the fixed-leading-coefficient form of variable-step BDF methods and the fully variable-coefficient form of the Adams methods. In addition, it includes a feature not in LSODE that can drastically decrease the number of evaluations of the Jacobian J. VODE normally saves a separate copy of J, and when the modified Newton iteration fails to converge, and the apparent reason is the change in the coefficient $h\beta_0$ in the Newton matrix of Eq. (4), the matrix is updated without a re-evaluation of J.

VODE and LSODE are "standard choices" for ODE initial-value problems. Some applications, however, give rise to other problem forms that VODE and LSODE cannot handle. For example, a large stiff system may have a Jacobian that is sparse (most elements are nonzero) but not tightly banded. For that case, there is a sparse variant of LSODE called LSODES. It uses parts of the Yale Sparse Matrix Package to solve the linear systems, and it includes an algorithm to generate difference-quotient Jacobian approximations with a reduced number of **f** evaluations.

Another common situation is one in which the problem changes with time from stiff to nonstiff and back again. For that case, there is another variant, called LSODA; this code switches automatically between stiff and nonstiff methods in a dynamic manner. Yet another variant, LSODAR, addresses the case where the ODE solution is to be stopped at a root of some other function (or set of functions) of **y**, as when a particle trajectory is stopped at the boundary of a geometrical region. Another way of dealing with the change between stiff and nonstiff is to switch dynamically between Newton iteration and functional iteration while using the BDF integration method. This kind of switch has been used in another LSODE variant, LSODKR.

Two other variants of LSODE, called LSODI and LSOIBT, are tailored for the case in which the ODE system is not given in the explicit form of Eq. (1) in part I, but in an implicit form with a matrix M multiplying the time derivative. This system is written

$$M(t,\mathbf{y})\,\frac{d\mathbf{y}}{dt}=\mathbf{g}(t,\mathbf{y}). \tag{12}$$

For example, if a PDE problem is treated by the finite-element method for the spatial discretization, then M is the mass matrix. Even if M is invertible, so that one could write an equivalent system $d\mathbf{y}/dt=M^{-1}\mathbf{g}(t,\mathbf{y})$, this is usually not an efficient way to solve the problem. Instead, one can efficiently treat Eq. (12) directly by the same methods used in LSODE, slightly reformulated. LSODI does this under the assumption that the matrices involved (M and the various Jacobian matrices) are either full or banded. LSOIBT treats the same problem form, but assumes that the matrices involved are "block-tridiagonal," meaning that the nonzero elements occur in blocks lying on and beside the main diagonal, a common occurrence in semidiscrete forms of PDE problems.

The LSODE solver, together with the variants of it just described, form a "systematized collection" of solvers called ODEPACK.[17] Their outward appearance (the user interface) is standardized by the use of identical names and meanings for features that are common to two or more of the codes. They are also standardized internally by, among other things, the use of shared Fortran subroutines for various subordinate tasks. See ODEPAK amd SPODEPACK in Netlib.

Large stiff ODE systems are often beyond the reach of the solvers in ODEPACK and require iterative methods for the linear systems involved. For this case, there are two variants of LSODE, called LSODPK[8] and LSODKR, and a variant of VODE, called VODPK.[10] All three use Krylov subspace methods with user-supplied preconditioning. In addition, LSODKR includes root-finding (as in LSODAR).

Several solvers have been written for DAE problems. In the linearly implicit case Eq. (12), with M singular, LSODI and LSOIBT have been used with some success. But they were not designed for DAE systems and are less reliable for them than the DASSL package.[11] DASSL, which also uses a BDF method, treats the linear systems as full or banded, but in various details it addresses the issues of DAE problems directly. A variant of DASSL with a root-finding ability added, called DASRT, is also available.

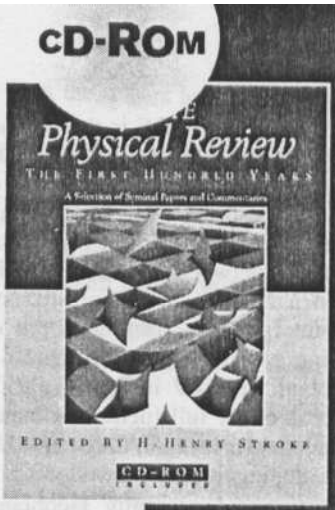
For large DAE systems, where iterative methods are more suitable than direct methods for the linear systems, we have written a variant of DASSL called DASPK,[14] which includes the GMRES Krylov method with user-supplied preconditioning as an option. DASPK actually includes the direct methods of DASSL as well. For use on massively parallel machines, two modified versions of DASPK have been written—one using Fortran 90 (with data parallelism), and one using message-passing.[18] Codes for computing consistent initial conditions for index-one DAEs and for computing the sensitivity of solutions to DAEs and large-scale DAEs with respect to given parameters, are currently in progress.

In recent years, there has been a trend to away from

writing software in Fortran and toward writing in the C language. In response, we have been working on a rewrite in C of the VODE and VODPK solvers (combined), called CVODE.[19] CVODE is composed of a central integrator module that has no knowledge of the nature of the linear system solver (direct or iterative, full or banded, etc.) and a set of linear solver modules from which the user selects prior to starting the integration. An additional motivation for this C rewrite of VODE/VODPK is our plan to extend this package to a parallel version of the solver for distribution-memory MIMD machines.

References

1. K. Radhakrishnan and A. C. Hindmarsh, NASA Reference Publication No. 1327, and LLNL Report No. UCRL-ID-113855, March 1994 (unpublished) .
2. L. R. Petzold, in *Scientific Computing,* edited by R. S. Stepleman *et al.* (North-Holland, Amsterdam, 1983), pp. 65–68.
3. P. N. Brown, G. D. Byrne, and A. C. Hindmarsh, SIAM J. Sci. Stat. Comput. **10**, 1038 (1989).
4. J. C. Butcher, Comput. Phys. **8**, 411 (1994).
5. E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems* (Springer, New York, 1991).
6. P. N. Brown and A. C. Hindmarsh, SIAM J. Num. Anal. **23**, 610 (1986).
7. P. N. Brown, SIAM J. Num. Anal. **24**, 407 (1987).
8. P. N. Brown and A. C. Hindmarsh, J. Appl. Math & Comp. **31**, 40 (1989).
9. Y. Saad and M. H. Schultz, SIAM J. Sci. Stat. Comp. **7**, 856 (1986).
10. George D. Byrne, in *Computational Ordinary Differential Equations,* edited by J. R. Cash and I. Galdwell (Oxford University Press, Oxford, 1992), pp. 323–356.
11. K. Brenan, S. Campbell, and L. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations* (Elsevier, New York, 1989).
12. U. Ascher and L. R. Petzold, SIAM J. Sci. Comput. **14**, 95 (1993).
13. B. J. Leimkuhler, L. R. Petzold, and C. W. Gear, SIAM J. Num. Anal. **28**, 205 (1991).
14. P. N. Brown, A. C. Hindmarsh, and L. R. Petzold, SIAM J. Sci. Comp. **15**, 1467 (1994).
15. G. D. Byrne and A. C. Hindmarsh, J. Comput. Phys. **70**, 1 (1987).
16. J. Dongarra and E. Grosse, Comm. ACM **30**, 403 (1987).
17. A. C. Hindmarsh, in Ref. 2, pp. 55–64.
18. R. S. Maier, L. R. Petzold, and W. Rath, Concurrency: Pract. Exp. (to be published).
19. S. C. Cohen and A. C. Hindmarsh, CVODE User Guide, LLNL Report No. UCRL-MA-118618, 1994 (unpublished).



The Physical Review
The First Hundred Years
A Selection of Seminal Papers and Commentaries

Edited by H. Henry Stroke, New York University

Revisit vital seminal research from the first hundred years of *The Physical Review* in this landmark book and CD-ROM package.

Each of the twelve chapters, introduced by an eminent physicist, represents a major subfield of physics. From elementary particles to condensed matter physics, this book includes 200 reprinted papers with over 1,000 papers available on the accompanying CD-ROM. Readers can search the CD-ROM and book for the historic work in their own fields, or find a well-organized tour of the key thinking in other disciplines.

Designed for researchers, professionals, teachers, students, historians, and interested general readers, *The First Hundred Years* is a vital reference for any scientific collection and an enduring tribute to the authors and editors of *The Physical Review* and *Physical Review Letters.*

CD-ROM System Requirements: 386 or 486 IBM-compatible PC, VGA or SVGA monitor, hard disk drive, CD-ROM drive supported by Windows, 5MB of free space, 4MB RAM, MS-DOS 3.3 or later, Windows 3.1 or later, Microsoft DOS Extensions (MSCDEX) 2.1 or later.

April 1995, 1-56396-188-1, cloth, 1,200 pages, illustrated bound in CD-ROM (not sold separately) **$75.00**

Available from your library wholesaler or call 800-809-2247

Fax: 802-864-7626
Or mail order to:
American Institute of Physics
c/o AIDC, P.O. Box 20, Williston, VT 05495

American Institute of Physics
500 Sunnyside Boulevard
Woodbury, NY 11797

AIP PRESS

Write in number 9 on the Reader Service Card