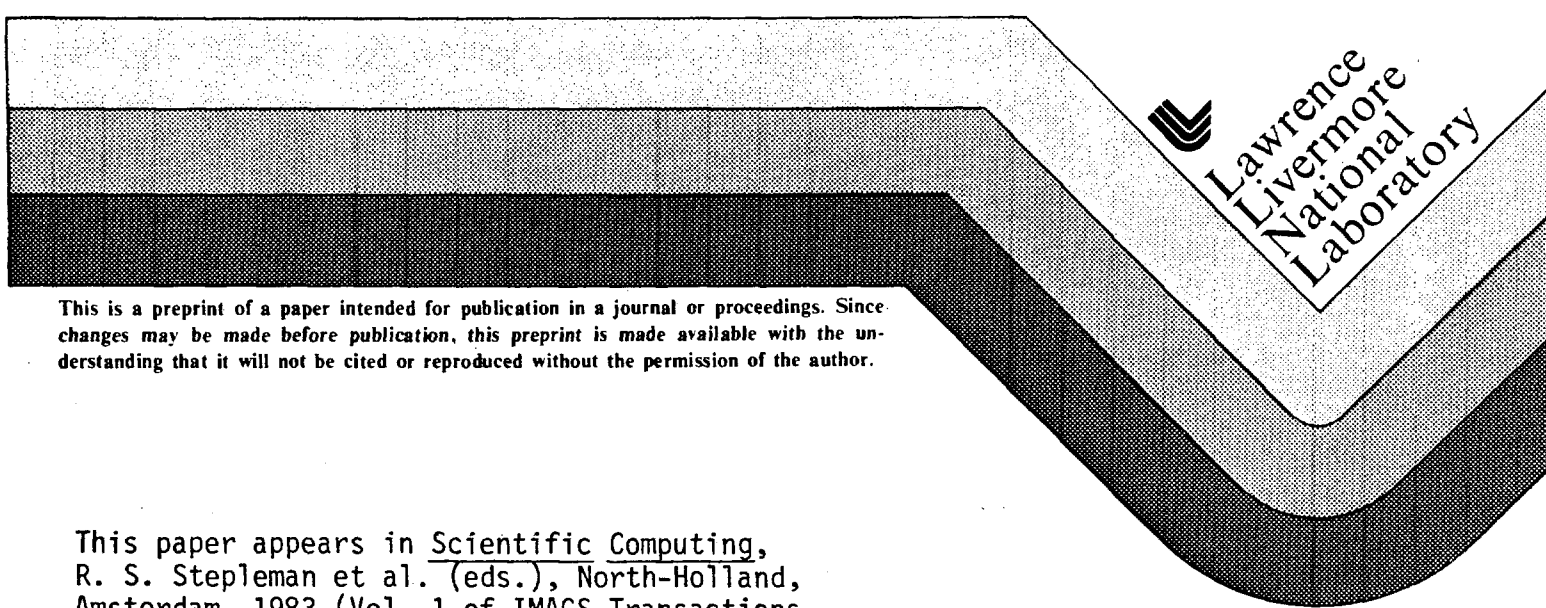
UCRL-88007
PREPRINT

ODEPACK, A SYSTEMATIZED COLLECTION OF ODE SOLVERS

Alan C. Hindmarsh

August 1982



This paper appears in Scientific Computing, R. S. Stepleman et al. (eds.), North-Holland, Amsterdam, 1983 (Vol. 1 of IMACS Transactions on Scientific Computation), pp. 55-64.

ODEPACK, A SYSTEMATIZED COLLECTION OF ODE SOLVERS*

Alan C. Hindmarsh

Lawrence Livermore National Laboratory
Livermore, CA 94550, USA

The growing number of good general purpose solvers for initial value problems for ordinary differential equation (ODE) systems has fueled discussions on the idea of a systematized collection of such solvers, ODEPACK. Within recent years, a tentative user interface standard was developed, and an initial collection of five solvers was written. These solvers handle stiff and nonstiff problems in standard (explicit) form, problems in linearly implicit form, full Jacobians, banded Jacobians, general sparse Jacobians, and problems with rootfinding (g-stop) requirements. Two of the solvers have automatic (stiff/nonstiff) method selection. These solvers are described briefly here, and their capabilities are illustrated with an example problem arising from a model of atmospheric kinetics-transport in two dimensions.

1. INTRODUCTION

Initial value problems for ODE systems have prompted a great deal of effort in numerical methods and software development. Stiff ODE systems are now recognized as being particularly common, and are of course much more challenging numerically. Here stiffness can be roughly defined as the presence of one or more fast decay processes in time, with a time constant that is short compared to the time span of interest. Good general purpose solvers have been available for up to 20 years, and the number of such solvers has grown quite sizable. Among the more popular of these are the GEAR and EPISODE packages and their variants [1], developed at LLNL, which use various forms of BDF (backward differentation formula) methods in the stiff case, and implicit Adams methods in the nonstiff case.

Faced with the large number and variety of ODE solvers, both the users and the suppliers of this software have expressed a desire for standardization. In other areas, analogous pressures have resulted in "systematized collections" of software (EISPACK, LINPACK, etc.) which meet high standards of quality and uniformity. Efforts to produce a similar collection of ODE initial value solvers have had some success. A tentative user interface standard was developed, and an initial collection, called ODEPACK, was then generated. The starting point of this collection is a package called LSODE, which is the result of rewriting the GEAR [2] and GEARB [3] solvers in conformity with the standard interface. Several variants of LSODE were then written to solve other problem clases.

*This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48, and supported in large part by the DOE Office of Basic Energy Sciences, Mathematical Sciences Branch.

In the next section, the methods used in the ODEPACK collection are summarized briefly. Following that is a short description of each of the five solvers in the existing collection, as it is now available. Finally, a two-dimensional atmospheric modelling problem is used to illustrate the solvers.

2. SUMMARY OF METHODS

Among the various numerical methods used for solving ODE initial value problems, a few are much more commonly used than others. The Adams multistep methods (explicit and implicit) are suitable for nonstiff systems, especially the implicit Adams methods. Explicit Runge-Kutta methods are also popular, but are also suitable only for nonstiff problems. Implicit Runge-Kutta methods of various types are being widely studied for use on stiff systems. But for large stiff problems, the most popular methods used are based on the so-called backward differentiation formulas (BDF's), which are multistep methods first implemented by C. W. Gear.

In 1968, Gear wrote a subroutine called DIFSUB [4] for initial value ODE problems, that included the BDF methods for stiff systems and implicit Adams methods for nonstiff systems. This program was reorganized, rewritten, and improved upon at LLNL, resulting in the GEAR package [2]. However, when solving a stiff system of size N, of the general form

$$\dot{y} \equiv dy/dt = f(t,y),$$

this package makes use of the Jacobian matrix of partial derivatives,

$$J = \partial f/\partial y,$$

in full NxN form. Thus the GEAR package is useful only for nonstiff and fairly small stiff problems. Because of this, variants of GEAR were developed later to handle large stiff

problems having some sparse structure in the Jacobian. Among these were GEARB [3], for the case of a banded J; GEARS [5], for a general sparse matrix J; and GEARBI [6], for a regularly blocked J, with block-iterative (block-SOR) treatment of the associated linear systems. Another variant, GEARIB [7], was written for linearly implicit problems, i.e. problems of the form $A\dot{y} = g(t,y)$, in which the matrix A and $\partial g/\partial y$ are banded.

As a frame of reference for later descriptions of algorithms and software, we give here a brief summary of the methods used in the GEAR package (and most of its variants and descendants), and also LSODE and its variants. Consider the system $\dot{y} = f(t,y)$, where y is a vector of length N, and consider a discrete time mesh $t_0, t_1, \ldots, t_n, \ldots$. (Of course, the independent variable need not be time, but often it is, and we will think of t as a time or time-like variable here.) For the moment, we consider the step size $h = t_n - t_{n-1}$ to be fixed. Discrete approximations y_n to $y(t_n)$ are to be constructed, with y_0 given, and $\dot{y}_n$ will always denote $f(t_n,y_n)$.

For nonstiff problems, we use the implicit Adams (or Adams-Moulton) formulas

$$y_n = y_{n-1} + h \sum_{i=0}^{q-1} \beta_i \dot{y}_{n-i} .$$

Here q ($1 \le q \le 12$) is the order of accuracy, and the coefficients β_i depend only on q. The formula is implicit in that $\beta_0 > 0$. Solution of this implicit equation is done by functional iteration,

$$y_{n(m+1)} = y_{n-1} + h\, \beta_0\, f(t_n,y_{n(m)}) + h \sum_{i=1}^{q-1} \beta_i \dot{y}_{n-i} ,$$

where an initial guess (or prediction) $y_{n(0)}$ is obtained from an analogous explicit formula. This iteration is terminated by a convergence test. Both the step size h and order q are actually varied during the integration process, by use of estimates of the local errors committed, in relation to a user-supplied tolerance. Changes in h are achieved by interpolation of the multistep data. Note that no NxN matrices are involved in this case.

For stiff problems, we use the BDF

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

$$= a_n + h\, \beta_0\, f(t_n,y_n) ,$$

where again q is the order (here $1 \le q \le 5$), and $\beta_0 > 0$. Stiffness makes functional iteration fail to converge for the step sizes of interest, because of strong dependencies in f upon y. Therefore, we use a modified Newton iteration,

$$-P\,[y_{n(m+1)} - y_{n(m)}] = y_{n(m)} - a_n - h\, \beta_0\, f(t_n,y_{n(m)}) ,$$

where P is an NxN matrix approximating the Jacobian of the algebraic system to be solved:

$$P \approx I - h\, \beta_0\, J , \qquad J = \partial f/\partial y .$$

(Here I denotes the NxN identity matrix.) Again a prediction $y_{n(0)}$ is formed from an analogous explicit formula. This iteration differs from a true Newton method in that J is only evaluated periodically. In fact, J is evaluated only at predicted values $y_{n(0)}$, and only on those steps where a new value appears necessary, on the basis of a convergence failure or other indication. The same value of P (or its LU decomposition, if used) is used over all iterations in any one step, and typically also over several time steps, until a reevaluation of J and P is called for. (In the case of the LSODES solver, P is sometimes updated and LU-decomposed without a reevaluation of J.) Again, h and q are both varied to meet local error tolerance requirements.

In applying the BDF method to large stiff problems, it is important to note that a numerical solution of the linear system

$$Px = r$$

(x = correction vector, r = residual vector)

can very often easily take advantage of a sparse structure in P. This is accomplished either through suitable structured LU decompositions, or through iterative linear system methods that use a given matrix structure. The use of structure is especially important in solving ODE systems that come from time-dependent partial differential equation (PDE) systems by the method of lines, whereby spatial variables are discretized, leaving ODE's in time.

Problems in the linearly implicit form $A(t,y)\,\dot{y} = g(t,y)$ arise frequently. Probably the most common sources are discretizations (by the method of lines procedure) of time-dependent PDE systems in which collocation, Galerkin, finite element, or other weighted residual methods are applied to the spatial variables. In these problems, A is a square matrix, usually nonsingular. We allow $A = A(t,y)$, but often A is constant. When A is nonsingular, this is an ODE system, but otherwise it is a differential-algebraic system. A numerical method for such an implicit system can be gotten from either of the multistep formulas given above, by multiplying both sides by $A(t_n,y_n)$, replacing $A(t_n,y_n)\dot{y}_n$ by $g(t_n,y_n)$, and solving the resulting implicit

relation for y_n. If the original formula has the form

$$y_n = a_n + h \beta_0 \dot{y}_n ,$$

then we obtain an implicit relation of the form

$$S(y) \equiv A(t_n,y) (y - a_n) - h \beta_0 g(t_n,y) = 0,$$

to be solved for $y = y_n$, where a_n is a constant vector. Again, a modified Newton iteration is usually most appropriate for this. However, it helpful first to introduce the residual function

$$r(y) = g(t_n,y) - A(t_n,y) s ,$$

values of which the user is to supply. Here s represents an approximation to $\dot{y}_n$, and we specifically define s to be

$$s = (y_{n(0)} - a_n) / h \beta_0 .$$

That is, s is a predicted value of $\dot{y}_n$ that corresponds to the prediction $y_{n(0)}$ through the original formula: $y_{n(0)} = a_n + h \beta_0 s$. We then find that S(y) and r(y) are related by

$$S(y) = A(t_n,y) (y - y_{n(0)}) - h \beta_0 r(y) .$$

In analogy with the algorithm for stiff explicitly given problems, we evaluate the Newton matrix $S'(y) = \partial S/\partial y$ only at the predicted value $y_{n(0)}$. From the above relation, we find that this matrix is

$$P \equiv S'(y_{n(0)}) = A(t_n,y_{n(0)}) - h \beta_0 r'(y_{n(0)}) ,$$

where r'(y) similarly denotes the Jacobian of r, $\partial r/\partial y$. Note that if A is the identity matrix I, this matrix P reduces to that used in the case $\dot{y} = f$.

The algorithms for solving $A \dot{y} = g$ arrived at in this way are numerically reliable only for certain classes of problems (including in particular those with non-singular A), and not for general differential-algebraic systems of this form [8]. However, for most applications of interest, these methods have been found to perform well, if not for the original system, then for a reorganized form of it.

For all of these methods, the algorithm for selecting the step size h and method order q is basically that used by Gear in [4], based on asymptotic local error analysis, but with some modifications. On each step, an estimate of the local error (at the current order q) is formed from the difference between the predicted and final corrected values of y. This gives a value of h suitable for meeting the given tolerances at order q (and for redoing the step if the tolerances were not met). Periodically, one can also estimate the local error that would be committed at orders q - 1 and q + 1 (dissallowing one of these choices if q is currently 1 or the maximum allowed). These estimates yield values of h suitable for each of the three orders. Then the new h and q are selected on the basis of maximizing h. Except when a step fails the local error tolerance test, changes in h and q are allowed no more frequently than every q + 1 steps, in order to prevent instabilities.

3. THE ODEPACK SOLVERS

3.1 The ODEPACK Concept

The GEAR package and its variants were added to a list of available general purpose initial value solvers that was growing quite sizable by 1975. The length and diversity of this list caused some concern to users and software developers alike. There was much duplication of capabilities offered, but at the same time there was very little in common among the solvers in terms of either their external appearance or their internal structure. This situation was in sharp contrast to that in other areas in which "systematized collections" of Fortran routines were being developed. The earliest examples were EISPACK [9], for computing matrix eigensystems, LINPACK [10], for solving linear systems, and FUNPACK, for certain special functions.

The idea of a systematized collection of initial value ODE solvers, tentatively called ODEPACK, was discussed informally as early as 1974, in workshops attended by people from all over the world [11]. However, it was quickly realized that the task was much larger in the ODE case than in other areas, partly because of the complexity of the subject, and partly because of widely divergent views of what ODEPACK should look like. Starting in 1976, attempts were made to reduce the problem by involving only people at U.S. Department of Energy laboratories, and LLNL received funding to study the feasibility of ODEPACK from the Applied Mathematical Sciences Research Program under the Office of Basic Energy Sciences in DOE.

The natural first step, and a necessary preliminary to any actual development of an ODEPACK, was the setting of standards for the interface between the user and the ODE solvers. The user interface to a solver consists mainly of the call sequence of the routine the user must call, together with definitions of the one or more user-supplied routines called by the solver. To the extent that solvers for various problem types and using various methods must all communicate certain specific things to and from the user, it is possible to formulate a loose set of standards for the user interface. An early proposal is given in [12]. A sequence of workshops and discussions on user interface standards for ODE solvers succeeded in producing a reasonable consensus in 1978 [13,14]. The resulting tentative interface standard was achieved only through considerable compromise by the various participants, which included ODE software authors and users at various DOE laboratories.

At that time, it was agreed that several of the more popular ODE solvers, including GEAR,

GEARB, DE/STEP [15] and RKF45 [16], would be rewritten to conform with the tentative standard interface [13], resulting in a small collection that was at least systematized in its external appearance. The first result of that agreement was a package based on the GEAR and GEARB packages, called LSODE (Livermore Solver for ODE's) [17,18]. Subsequently, four variants of the LSODE solver were written, all in accordance with the tentative standard interface [13], with minor modifications. In the meantime, unfortunately, the other software authors involved withdrew from the agreement, and so this collection does not yet have analogous rewritten versions of their codes.

In what follows, the LSODE package, and the variants of which have been completed to date, are summarized. Other variants planned are also mentioned, and comments on availibility of the solvers are given.

3.2 LSODE: The Basic Solver

LSODE [17,18] combines the capabilities of GEAR and GEARB. Thus it solves explicitly given stiff and nonstiff systems $\dot{y} = f(t,y)$, and in the stiff case it treats the Jacobian matrix $J = \partial f/\partial y$ as either full or banded, and as either user-supplied or internally approximated by difference quotients. By comparison with GEAR and GEARB, LSODE offers a number of new features that make it more convenient, more flexible, more portable, and easier to install in software libraries. Some of these are the following:

(a) Through the redesigned user interface, many new options and capabilities are available, and others are much more convenient than before. Some examples are--more flexible error tolerance parameters, independent flags for starting and stopping options, internally computed initial step size, two work arrays in the call sequence for all internal dynamic work space, user names for f and J in the call sequence, easy changing of input parameters in mid-problem, convenient optional inputs (such as maximum method order), convenient optional outputs (such as step and function evaluation counts), optional provision of derivatives of the solution (of various orders) at any point, and real and integer user data space (of dynamic length) available in the f and J routines (with no extra burden on the casual user).

(b) The user documentation, which is contained in the initial comment cards of the source, is given in a two-level form. A short and simple set of instructions, with a short example program, is given first, for the casual user. Then detailed instructions are given for users with special problem features or a desire for nonstandard options. The latter is also organized so as to allow selective reading by a user who wants only a fraction of the nonstandard capabilities.

(c) When stiff options are selected, linear systems are solved with routines from LINPACK [10], which is becoming a widely accepted standard collection of linear system solvers.

(d) Some retuning of various heuristics was done so that performance should be more reliable than for GEAR/GEARB. For example, LSODE has no minumum step size (unless one is specified as an optional input), but has instead a maximum number of failed attempts at a time step.

(e) The core routine which takes a single step, called STODE, is independent of the way in which the Jacobian matrix (if used) is treated. Thus variant versions of LSODE for other matrix structures (such as LSODES) will share the same subroutine STODE.

(f) The writing of all error messages is done in a small isolated general-purpose message handler called XERRWV. Two other small subroutines are user-callable and allow for optional changing of the output unit number and optional suppression of error messages. This trio of routines is compatible with a much larger error handling package (the SLATEC Error Handling Package) written at Sandia National Laboratories [19].

(g) LSODE easily allows a user to interrupt a problem and restart it later (e.g. in switching between two or more ODE problems). Also, using LSODE in overlay mode is very easy, with no loss of needed local variables.

(h) The various lists of constants needed for the integration, formerly appearing in a subroutine called COSET, are now computed (once per problem). This adds to the portability of LSODE.

3.3 LSODI: Implicit Systems

The LSODI solver [17], written jointly with J. F. Painter (LLNL), treats systems in the linearly implicit form $A(t,y)\dot{y} = g(t,y)$, where A is a square matrix. Many problems, including PDE's treated by finite elements and the like, result in such systems, and it is almost always more economical to treat the system in the given form than to convert it to an explicit form $\dot{y} = f$. LSODI allows A to be singular, but the user must then input consistent initial values of both y and $\dot{y}$. In the singular case, the system is a differential-algebraic system, and then the user must be much more cautious about formulating a well-posed problem, as well as in using LSODI, which was not designed to be robust in this case. LSODI is based on (and supersedes) the GEARIB package, but corrects a number of deficiencies, as follows:

(a) The matrices involved can be treated as either full or banded, by use of the method flag.

(b) The dependence of A on y is automatically and inexpensively accounted for, whether partial derivatives are supplied by the user or computed internally by difference quotients.

(c) When A is singular, the user needs to supply the initial value of dy/dt, but no later values. This array (along with the initial y) is passed through the call sequence. (Admittedly, correct initial data can be difficult to obtain for some types of problems.) When the initial dy/dt is not being supplied, an input flag instructs LSODI to

compute it on the assumption that A is initially nonsingular. Thereafter, no such assumption on A is made, but ill-conditioning in the Newton matrix P can be a problem when A is singular.

(d) The user-supplied residual routine includes a flag which allows the user to signal either an error condition or an interrupt condition. When an error condition is signalled (e.g. when a value of y is illegal and the residual function cannot be evaluated), LSODI attempts to recover, by updating the Newton matrix P and possibly cutting the step size, and attempts to proceed without generating this condition.

(e) To the maximum extent possible, LSODI shares the same user interface as LSODE, and so reflects all the advantages over GEARIB that LSODE has over GEAR and GEARB, in terms of flexibility, convenience, portability, etc.

The differences between the LSODI and LSODE user interfaces occur primarily in the user-supplied subroutines. With LSODI, one must supply a routine to compute the residual function $r(y) = g(t,y) - A(t,y)s$ for a given t, y, and s, and another routine to add the matrix A to a given array. Optionally, the user can supply a routine to compute the Jacobian matrix $\partial r/\partial y$. The use of r(y) as the basic user-supplied quantity, as opposed to constructing r from values of g and A, is designed to allow for both computational and storage economies. Usually, the user can construct r(y) without forming A explicitly, thus saving considerably on storage, and often he can construct it at much lower cost than LSODI would do so from g and A.

Some examples of the use of LSODE and LSODI on systems arising from PDE problems can be found in [18] and [20]. In the latter, experiments by Painter on incompressible Navier-Stokes problems shed some light on the difficulties involved with differential-algebraic systems. A program for general differential-algebraic systems is available from L. Petzold [21].

3.4 LSODES: General Sparse Jacobian

The LSODES package solves explicit systems $\dot{y} = f$, but treats the Jacobian matrix J as a general sparse matrix in the stiff case. LSODES was written jointly with A. H. Sherman (Exxon Production Research Company), and supersedes the sparse variant of GEAR called GEARS. In LSODES, linear systems are solved using parts of the Yale Sparse Matrix Package (YSMP) [22,23]. Recall that the systems to be solved have the form

$$Px = r, \quad P = I - h\beta_0 J ,$$

where x is a correction vector, h is the step size, and β_0 is a scalar depending on the current method order. The solution of these systems involves several phases:

(a) Determination of sparsity structure. This is either inferred from calls to the f routine, inferred from calls to a J routine (if one is supplied), or supplied directly by the user. A user input flag determines which is done.

(b) Determination of pivot order. Diagonal pivot locations are chosen, and the choice is based on maintaining maximum sparsity. This is done by YSMP (ODRV module) [22]. The ordering algorithm (minimum degree algorithm) operates only on a symmetric sparsity structure, and in LSODES the structure used for this is that of $J + J^T$.

(c) Symbolic LU factorization of the matrix P. This is based only on sparsity and the pivot order, and uses the module in YSMP designed for nonsymmetric matrices with compressed pointer storage (CDRV module) [23].

(d) Construction of J. This can be done internally by difference quotients, or with a user-supplied routine. In the difference quotient case, the number of f evaluations needed is kept to a minimum by a column grouping technique due to Curtis, Powell, and Reid [24]. In the other case, the user-supplied routine provides one column of J at a time, in the form of a vector of length N (although only the non-zero elements need be computed and stored), so that users need never deal with the internal data structure for J and P. In any case, J is stored internally in an appropriate packed form. Evaluations of J are done only occasionally, as explained below.

(e) Construction of $P = I - h\beta_0 J$. In contrast to LSODE and GEARS, LSODES does not force a re-evaluation of J whenever the existing P is deemed unsuitable for the corrector iterations. Instead, when the value of J contained in the stored value of P is likely to be usable (and P is not, only because $h\beta_0$ has changed significantly), then a new matrix P is constructed from the old one, with careful attention to roundoff error. This cuts down greatly on the total number of J evaluations.

(f) Numerical LU factorization of P. This is done by YSMP (CDRV module) in sparse form, and the array containing P is saved in the process. Because of the absence of partial pivoting for numerical stability, this operation can conceivably fail. However, this has only rarely been observed in practice, and if it does occur (with a current value of J), the step size h gets reduced and the difficulty disappears.

(g) Solution of $Px = r$. This is done by YSMP (CDRV module) using the existing sparse factorization of P. Because a modified Newton iteration is used, many values of r (i.e., many linear systems) can arise for the same P, and the separation of the various phases takes advantage of that fact.

The first three phases, and part of the fourth (column grouping for difference quotients), are normally done only at the start of the problem. However, the user can specify that the sparsity structure is to be redetermined in the middle of the problem, and then these operations are repeated.

Actually, the matrix operated on by YSMP is not P but $A = P^T$, because P is generated in column order while YSMP requires the matrix to

be described and stored in row order. But this causes no difficulty, because YSMP includes a routine for solving the equivalent transpose problem $x^T A = r^T$ as well as for the direct problem $Ax = b$.

3.5 LSODA: Automatic Method Selection

LSODA is a variant of LSODE of yet another kind. It was written jointly with L. R. Petzold (Sandia-Livermore), and switches automatically between nonstiff (Adams) and stiff (BDF) methods, by an algorithm developed by Petzold [25]. (The suffix A stands for Automatic.) Thus it is more convenient for users who do not want to be bothered with the issue of stiffness. Also, it is potentially more efficient than LSODE (when used with a fixed method option), when the nature of the problem changes between stiff and nonstiff in the course of the solution. In particular, on the initial (nonstiff) transient interval that is almost always present in stiff problems, LSODA uses the more efficient Adams method. In place of the method flag parameter of LSODE, the user of LSODA supplies only a Jacobian type flag (specifying whether J is full or banded, to be user-supplied or internally generated). The work space supplied to the solver can be either static (and thus allow for either method), or dynamic (and altered each time there is a method switch, to an amount specified by the solver).

3.6 LSODAR: Rootfinding

LSODAR combines the capabilities of LSODA with a rootfinder. It allows one to find the roots of a set of functions $g_i(t,y)$ of the independent and dependent variables in the ODE system. (This is sometimes referred to as a "g-stop" feature.) Thus, for example, it could be used in a particle tracking problem to determine when a particle path reaches any of the walls of a container. LSODAR was also written jointly with L. R. Petzold, based on an algorithm [26] developed by K. Hiebert and L. F. Shampine (Sandia-Albuquerque). The user must supply, in addition to the LSODA inputs, a subroutine that computes a vector-valued function $g(t,y) = (g_i,\ i=1,2,\ldots,NG)$ such that a root of any of the NG functions g_i is desired. Of course there may be several such roots in a given output interval, and LSODAR returns them one at a time, in the order in which they occur along the solution. An integer array tells the user which g_i (if any) were found to have a root on any given return. With LSODAR, it is especially important to choose the tolerances conservatively, so that numerical errors in the computed solution y(t) do not deceive the rootfinding algorithm.

3.7 Future Additions

Several other solvers will be added to the ODEPACK collection in the near future, as they are developed in response to the needs of different classes of problems. In particular, the following two solvers are nearly complete and will soon be available:

(a) LSOIBT. This resembles LSODI in that it solves problems of the form $A(t,y)\,\dot{y} = g(t,y)$, but it assumes a block-tridiagonal structure for all the matrices involved. It then uses a linear system solver tailored to block-tridiagonal systems. LSOIBT was developed from LSODI by C. Kenney (China Lake Naval Weapons Center). It was motivated by the method of moving finite elements for parabolic PDE systems, which generates ODE systems $A\,\dot{y} = g$ with block-tridiagonal structure.

(b) LSODIS. This also solves the $A\,\dot{y} = g$ problem, but uses a general purpose sparse matrix treatment of the linear systems, as in LSODES. LSODIS was developed from LSODI and LSODES by S. Balsdon (University of Texas at Austin) [27], and was also motivated by finite element methods.

In addition, plans are under way to rewrite (and algorithmically improve upon) other existing solvers for addition to ODEPACK. Solvers to be so revised include GEARBI [6] and EPISODE [28].

3.8 Availability

The ODEPACK solvers are being made available by way of the National Energy Software Center (Argonne National Laboratory, 9700 South Cass Ave., Argonne, IL 60439, U.S.A.). NESC operates on a subscription basis, and requests should be handled by the NESC installation representative at each site. Separate single and double precision versions are available.

To date, one or more members of the ODEPACK collection have been sent on request to over 200 sites, and the acceptance of the solvers has been extremely positive.

4. AN EXAMPLE PROBLEM

In order to illustrate the various solvers described above, and to demonstrate their relative merits on a realistic problem, we consider here an example problem. The problem is a simple atmospheric model with two chemical species undergoing diurnal kinetics and transport in two space dimensions. The independent variables in the PDE system are horizontal position x, altitute z (both in kilometers), and time t (in sec), with

$$0 \le x \le 20, \quad 30 \le z \le 50,$$

$$0 \le t \le 86400 \text{ (1 day)} .$$

The dependent variables are

$c^1(x,z,t)$ = the concentration of the oxygen singlet [O] , and

$c^2(x,z,t)$ = concentration of ozone [O_3]

(both in moles/cm^3). The concentration of

molecular oxygen [O_2] is assumed constant. The equations of the model are:

$$c^i_t = (K_v(z)c^i_z)_z + K_h c^i_{xx} + R^i(c^1,c^2,t), \qquad (i=1,2),$$

where the subscripts t, z, and x denote partial derivatives. Here R^1 and R^2 represent the chemistry and are given by

$$R^1(c^1,c^2,t) = -(k_1 + k_2c^2)c^1 + k_3(t)c^2 + k_4(t)\cdot 7.4\cdot 10^{16},$$

$$R^2(c^1,c^2,t) = (k_1 - k_2c^2)c^1 - k_3(t)c^2 .$$

The various coefficients are as follows:

$$K_v(z) = 10^{-8}\cdot\exp(z/5), \quad K_h = 4\cdot 10^{-6},$$

$$k_1 = 6.03, \quad k_2 = 4.66\cdot 10^{-16},$$

$$k_3(t) = \begin{cases} \exp[-7.601/\sin(\pi t/43200)], & t < 43200 \\ 0, & t \geq 43200 \end{cases},$$

$$k_4(t) = \begin{cases} \exp[-22.62/\sin(\pi t/43200)], & t < 43200 \\ 0, & t \geq 43200 \end{cases}.$$

Both c^1 and c^2 are required to satisfy homogeneous Neumann boundary conditions (zero normal derivatives) along all the x and z boundaries. The initial conditions (at t = 0) are

$$c^1 = 10^6\,(1 - \bar{x}^2 + \bar{x}^4/2)(1 - \bar{z}^2 + \bar{z}^4/2),$$

$$c^2 = 10^{12}\,(1 - \bar{x}^2 + \bar{x}^4/2)(1 - \bar{z}^2 + \bar{z}^4/2),$$

$$\bar{x} \equiv (x - 10)/10, \quad \bar{z} \equiv (z - 40)/10,$$

which represent mildly peaked distributions satisfying the boundary conditions.

The solution to this problem is a peaked distribution for both variables, changing in time and diffusing somewhat in all directions. With respect to time, the ozone concentration [O_3] varies only a few percent, but [O] has a sharp initial drop, then rises by over three orders of magnitude, and finally drops essentially to zero at sunset.

To solve the above system numerically, we apply the method of lines using a regular rectangular mesh with constant mesh spacings

$$\Delta x = 20/(M_x-1), \quad \Delta z = 20/(M_z-1).$$

Thus the discrete mesh consists of points (x_j,z_k) with

$$x_j = (j-1)\,\Delta x \quad (j = 1,2,\ldots,M_x),$$

$$z_k = 30 + (k-1)\,\Delta z \quad (k = 1,2,\ldots,M_z),$$

and the discrete variable $c^i_{j,k}$ is an approximation to $c^i(x_j,z_k)$. The spatial derivatives are approximated by standard 5-point central differences, and the boundary conditions are similarly replaced by difference relations. To illustrate, consider a nonuniform diffusion term in one dimension, $(K(z)\,c_z(z))_z$. The value of this term at a point $z = z_k$ is given by

$$\frac{K(z_{k+1/2})\,c_z(z_{k+1/2}) - K(z_{k-1/2})\,c_z(z_{k-1/2})}{(z_{k+1/2} - z_{k-1/2})},$$

where

$$c_z(z_{k+1/2}) \equiv (c_{k+1} - c_k)/(z_{k+1} - z_k),$$

$$c_z(z_{k-1/2}) \equiv (c_k - c_{k-1})/(z_k - z_{k-1}),$$

$$z_{k+1/2} \equiv (z_{k+1} + z_k)/2,$$

$$z_{k-1/2} \equiv (z_{k-1} + z_k)/2.$$

(Uniformity of the mesh is not assumed in these difference formulas.) The boundary conditions in the 2-D problem are approximated by setting

$$c^i_{0,k} = c^i_{2,k} \quad \text{(all k)}$$

for the boundary segment $x_1 = 0$, and similarly for the other three boundary segments. These relations allow one to form a well-defined ODE for each of the $c^i_{j,k}$. The resulting ODE system $\dot{y} = f(t,y)$ has size $N = 2M_xM_z$. It is quite stiff because of the presence of a short kinetics time constant (about 1/6 sec). The initial value vector y_0 is taken from the initial condition functions given above. The system Jacobian J is sparse, with roughly $12M_xM_z = 6N$ nonzero elements. As a band matrix, with component ordering first by species, then by x, and lastly by z, it has a half-bandwidth of $2M_x$, and thus a full bandwidth of $4M_x+1$. (It is important to use such an ordering if minimal bandwidth is important; an ordering by grid points and then by species produces a Jacobian that is not banded at all.)

We consider two cases,

$$M_x = M_z = 10, \quad \text{and} \quad M_x = M_z = 20.$$

As to accuracy, a crude model of this type calls for no more than a few significant figures. To be conservative in recognizing that tolerance parameters are applied to local errors, which can accumulate into global error, we might impose a local relative tolerance of 10^{-4}. We must also specify a positive absolute tolerance on the values of c^1 because it decays to negligible values at night. A reasonable absolute tolerance is 10^{-2}.

Three of the ODEPACK solvers are suitable for this particular problem--LSODE, LSODA, and LSODES. In addition, the older package GEARBI is certainly suitable, and in fact was motivated by exactly this type of problem. Recall that LSODES uses a general sparse treatment of the Jacobian matrix, GEARBI uses block-SOR, while LSODE and LSODA will (in this case) treat the

Jacobian as banded. The problem was set up for each of these four solvers and run first on a CDC-7600 computer, then on a Cray-1 computer. On the 7600, only the 10 by 10 grid problem was run, as the larger problem could not be accommodated by all of the solvers within the Small Core Memory (about 57000 words). For all but GEARBI, both the user-supplied Jacobian option and the internal difference quotient Jacobian option were tested. (GEARBI has no difference quotient option.) The results of the runs on the CDC-7600 are given in Table 1. The results of the Cray-1 runs are given in Table 2 for the 10 by 10 grid, and in Table 3 for the 20 by 20 grid. The tabulated quantities are:

R.T.	=	CPU run time in sec
NST	=	number of steps
NFE	=	number of f evaluations
NJE	=	number of J evaluations
NLU	=	number of LU decompositions
W.S.	=	total size of work space arrays

In the tables, USJ denotes the user-supplied Jacobian option, and DQJ denotes the internal difference quotient Jacobian option. An earlier comparison test on this problem is in [5].

Table 1. Results of kinetics-transport test problem (10x10 grid) on the CDC-7600.

Solver	R.T.	NST	NFE	NJE	NLU	W.S.
LSODE (USJ)	23.2	344	519	68	68	14,242
LSODE (DQJ)	28.4	337	3338	69	69	14,242
LSODA (USJ)	21.3	339	584	55	55	14,242
LSODA (DQJ)	24.6	339	2795	55	55	14,242
LSODES (USJ)	13.1	364	529	10	70	12,455
LSODES (DQJ)	13.5	369	602	8	72	12,664
GEARBI	6.3	316	526	50	50	3,004

Table 2. Results of kinetics-transport test problem (10x10 grid) on the Cray-1.

Solver	R.T.	NST	NFE	NJE	NLU	W.S.
LSODE (USJ)	2.52	344	520	68	68	14,242
LSODE (DQJ)	5.16	337	3463	72	72	14,242
LSODA (USJ)	2.89	344	587	54	54	14,242
LSODA (DQJ)	4.78	340	2794	55	55	14,242
LSODES (USJ)	4.86	364	533	14	71	12,455
LSODES (DQJ)	5.34	378	641	11	76	12,664
GEARBI	3.04	316	526	50	50	3,004

Table 3. Results of kinetics-transport test problem (20x20 grid) on the Cray-1.

Solver	R.T.	NST	NFE	NJE	NLU	W.S.
LSODE (USJ)	19.8	401	604	86	86	104,842
LSODE (DQJ)	43.1	402	7647	87	87	104,842
LSODA (USJ)	17.1	312	550	52	52	104,842
LSODA (DQJ)	35.4	344	5486	61	61	104,842
LSODES (USJ)	43.2	385	577	10	90	61,033
LSODES (DQJ)	42.2	390	638	8	77	61,842
GEARBI	16.4	348	544	58	58	12,004

Several points of interest can be noted in these tables.

(a) First, for each of the two problems, the number of steps does not vary greatly from solver to solver, because that is determined almost entirely by the accuracy requirement, and the accuracy achieved is much the same for all these runs. Also, comparison of the 20x20 grid results with the 10x10 grid results shows that the latter have errors (due to the spatial discretization) of up to 2%.

(b) For each problem, the performance characteristics of LSODE and LSODA are similar, as expected, since both use a banded Jacobian here. In most cases, LSODA is faster, primarily because it uses the cheaper nonstiff (Adams) method on the initial transient of the problem, switching to the BDF at about t = 3.6. For the same reason, the number of Jacobian evaluations is significantly lower for LSODA than LSODE. This advantage is offset somewhat by a larger average number of f evaluations per step for LSODA during the integration of the transient (due to the need in LSODA for estimates of the Lipschitz constant).

(c) For LSODE and LSODA, the use of a difference quotient Jacobian incurs some additional expense over the user-supplied Jacobian, owing to its cost of $4M_X+1$ (= 41 or 81) additional evaluations of f for each evaluation of J. On the 7600, this cost penalty is never more than 25%, but on the Cray, it is 65% to 118%. The reason is that the band matrix solvers on the Cray (which are highly optimized versions of the LINPACK routines) are up to 10 times faster than on the 7600, while f evaluations are only about twice as fast on the Cray. (This illustrates the speed gains possible with vector operations on the Cray, in contrast to the evaluation of f here, which was left in a form that does not vectorize at all.) Thus on the Cray, the cost of the f and J evaluations is a much larger fraction of the total. For example, for LSODE (USJ) on the 10x10 grid problem, the cost of the f

evaluations is about 4% of the total on the 7600, and about 19% of the total on the Cray. Clearly, it pays to generate a closed-form Jacobian routine when using LSODE or LSODA to solve a stiff system on the Cray.

(d) The LSODES results show a significant speedup on the 7600 over LSODE and LSODA (by factors of 1.6 to 2.1), but none (or nearly none) on the Cray. The reason is that, in contrast to the band matrix solvers, the sparse matrix solvers do not show more than about a 2-to-1 speedup in moving from the 7600 to the Cray. The speed advantage of LSODES on the 7600 appears to be due entirely to its algorithm of saving old values of P, and thus cutting down greatly on the number of J evaluations, which constitute a sizable fraction of the total cost on the 7600. Note that each computed value of J is used for 26 to 49 steps, as opposed to only 5 to 6 steps for LSODE and LSODA. For a problem that is similar but more costly in function evaluations, this behavior would lead to a significant cost advantage for LSODES on the Cray as well as on the 7600.

(e) On both computers and for both problems, the cost penalty for a difference quotient Jacobian is quite small for LSODES (at most 10%). This is partly because there are so many fewer J evaluations, and partly because each evaluation of J by difference quotients in LSODES costs only 8 evaluations of f here, independent of the grid size.

(f) The storage requirement for LSODES is lower than for LSODE or LSODA, by around 12% on the 10x10 grid, and around 41% on the 20x20 grid. This trend continues for finer meshes. For courser meshes, LSODES would have no storage advantage, reflecting its need for sparsity information arrays and the fact the matrix P is stored separately from its LU decomposition. Thus for the present problem, on the 7600, and on the Cray with difference quotient Jacobian option, LSODES is competitive with LSODE or LSODA in run time and superior in storage.

(g) Overall, the best performance on this problem, however, is that of GEARBI. This should not be a surprise, since the Jacobian has a very regular block structure of which the block-SOR method in GEARBI is taking full advantage, both in storage and computation. The LU decompositions here are only those of the block diagonal part of the Newton matrix (with 2x2 blocks). The total number of block-SOR iterations for the 10x10 grid was 607, or an average of less than 2 per step. For the 20x20 grid this cost rose to 972 iterations, or an average of 2.7 per step. Note that, because there is little opportunity for use of vector operations, the cost for the 10x10 problem dropped by only a factor of 2.1 in going from the 7600 to the Cray, making the GEARBI run times nearly equal to those of LSODE (USJ) and LSODA (USJ). (A more careful organization of the block-SOR algorithm might yield greater speeds on the Cray.) The storage advantage of GEARBI is tremendous, though - a factor of 4.7 for the 10x10 grid and 8.7 for the 20x20 grid. Thus for problems of this general type, which are amenable to block-iterative matrix treatment, solution by GEARBI or a similar algorithm appears strongly competitive with other approaches.

In closing, we mention some truly large problems to which the GEARBI package has been applied. In the early 1970's, a number of atmospheric models were developed at LLNL, involving chemical kinetics and transport in up to 2 space dimensions. Typically, the number of chemical species was 5 to 20, and typical 2-D mesh sizes were about 40 by 40. Thus when finite differenced, these problems generated ODE systems of sizes exceeding 10,000. The smallest kinetics time constants were typically in the range of milliseconds to microseconds, while the largest diffusion time constants were measured in years, making these systems extremely stiff. The GEARBI package, and an extension of it using Large Core Memory on the CDC-7600 (about 400,000 words), were successfully used to solve these problems in a wide variety of applications [29,30,31].

REFERENCES

[1] 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.

[2] A. C. Hindmarsh, GEAR: Ordinary Differential Equation System Solver, LLNL Report UCID-30001, Rev. 3 (December 1974).

[3] A. C. Hindmarsh, GEARB: Solution of Ordinary Differential Equations Having Banded Jacobian, LLNL Report UCID-30059, Rev. 2 (June 1977).

[4] C. W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations (Prentice-Hall, Englewood Cliffs, NJ, 1971), pp. 158-166.

[5] A. H. Sherman and A. C. Hindmarsh, GEARS: A Package for the Solution of Sparse Stiff Ordinary Differential Equations, in A. M. Erisman, K. W. Neves, and M. H. Dwarakanath (eds.), Electrical Power Problems: The Mathematical Challenge (SIAM, Philadelphia, 1980), pp. 190-200.

[6] A. C. Hindmarsh, Preliminary Documentation of GEARBI: Solution of ODE Systems with Block-Iterative Treatment of the Jacobian, LLNL Report UCID-30149 (December 1976).

[7] A. C. Hindmarsh, Preliminary Documentation of GEARIB: Solution of Implicit Systems of Ordinary Differential Equations with Banded Jacobian, LLNL Report UCID-30130 (February 1976).

[8] L. R. Petzold, Differential/Algebraic Equations are not ODEs, SIAM J. on Sci. and Stat. Computing 3 (1982), pp. 367-384.

[9] B. T. Smith, J. M. Boyle, B. S. Garbow, Y. Ikebe, V. C. Klema and C. B. Moler, Matrix Eigensystem Routines--EISPACK Guide, Lecture Notes in Computer Science, Vol. 6, Edition 2 (Springer-Verlag, New York, 1976).

[10] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, LINPACK User's Guide (SIAM, Philadelphia, 1979).

[11] G. D. Byrne, A Report on the ODE Workshop, Held at San Antonio, Texas, January 26-28, 1976, in the ACM-SIGNUM Newsletter, Vol. 11, No. 1 (May 1976), pp. 27-28.

[12] A. C. Hindmarsh and G. D. Byrne, A Proposed ODEPACK Calling Sequence, LLNL Report UCID-30134 (May 1976).

[13] A. C. Hindmarsh, A Tentative User Interface Standard for ODEPACK, LLNL Report UCID-17954 (October 1978).

[14] A. C. Hindmarsh, A User Interface Standard for ODE Solvers, in the Proceedings of the 1979 SIGNUM Meeting on Numerical Ordinary Differential Equations, April 1979, Univ. of Illinois (Dept. of Comp. Sci.) Report 79-1710, R. D. Skeel (ed.), 1979; also in the ACM-SIGNUM Newsletter, Vol. 14, No. 2 (June 1979), p. 11.

[15] L. F. Shampine and M. K. Gordon, Solution of Ordinary Differential Equations - The Initial Value Problem (W. H. Freeman and Co., San Francisco, 1975).

[16] G. E. Forsythe, M. A. Malcolm, and C. B. Moler, Computer Methods for Mathematical Computations (Prentice-Hall, Englewood Cliffs, NJ, 1977), pp. 129-147.

[17] A. C. Hindmarsh, LSODE and LSODI, Two New Initial Value Ordinary Differential Equation Solvers, in the ACM-SIGNUM Newsletter, Vol. 15, No. 4 (December 1980), pp. 10-11.

[18] A. C. Hindmarsh, ODE Solvers for Use with the Method of Lines, in R. Vichnevetsky and R. S. Stepleman (eds.), Advances in Computer Methods for Partial Differential Equations - IV (IMACS, New Brunswick, NJ, 1981), pp. 312-316.

[19] R. E. Jones, SLATEC Common Mathematical Library Error Handling Package, Sandia National Laboratories Report SAND78-1189 (September 1978).

[20] J. F. Painter, Solving the Navier-Stokes Equations with LSODI and the Method of Lines, LLNL Report UCID-19262 (December 1981).

[21] L. R. Petzold, A Description of DASSL: A Differential/Algebraic System Solver, in the Proc. of the IMACS 10th World Congress, Montreal, August 8-13, 1982.

[22] S. C. Eisenstat, M. C. Gursky, M. H.Schultz, and A. H. Sherman, Yale Sparse Matrix Package: I. The Symmetric Codes, Research Report No. 112 (Dept. of Computer Sciences, Yale University, 1977).

[23] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman, Yale Sparse Matrix Package: II. The Nonsymmetric Codes, Research Report No. 114 (Dept. of Computer Sciences, Yale Universtiy, 1977).

[24] A. R. Curtis, M. J. D. Powell, and J. K. Reid, On the Estimation of Sparse Jacobian Matrices, J. Inst. Math. Applic. 13, (1974), pp. 117-119.

[25] L. R. Petzold, Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations, Sandia National Laboratories Report SAND80-8230 (September 1980).

[26] K. L. Hiebert and L. F. Shampine, Implicitly Defined Output Points for Solutions of ODE's, Sandia National Laboratories Report SAND80-0180 (February 1980).

[27] M. K. Seager and S. Balsdon, LSODIS, A Sparse Implicit ODE Solver, in the Proc. of the IMACS 10th World Congress, Montreal, August 8-13, 1982.

[28] A. C. Hindmarsh and G. D.Byrne, Applications of EPISODE: An Effective Package for the Integration of Systems of Ordinary Differential Equations, in L. Lapidus and W. E. Schiesser (eds.), Numerical Methods for Differential Systems (Academic Press, New York, 1976), pp. 147-166.

[29] J. S. Chang, A. C. Hindmarsh, and N. K. Madsen, Simulation of Chemical Kinetics Transport in the Stratosphere, in R. A. Willoughby (ed.), Stiff Differential Systems (Plenum Press, New York, 1974), pp. 51-65.

[30] M. C. MacCracken, DOT-CIAP Final Report, LLNL Report UCRL-51336 (May 1975).

[31] M. C. MacCracken, D. J. Wuebbles, J. J. Walton, W. H. Duewer, and K. E. Grant, Livermore Regional Air Quality Model: I. Concept and Development, J. of Appl. Meterology, 17 (1978), 254-272.