SIAM J. SCI. STAT. COMPUT.
Vol. 6, No. 2, April 1985

004

EXPERIMENTS WITH QUASI-NEWTON METHODS IN SOLVING STIFF ODE SYSTEMS*

PETER N. BROWN†, ALAN C. HINDMARSH‡ AND HOMER F. WALKER†

Abstract. A nonlinear algebraic system must be solved at each step of the integration of a stiff system of ordinary differential equations by methods based on backward differentiation formulas. Quasi-Newton methods are of potential benefit in solving these algebraic problems. Three types of quasi-Newton methods are studied for this purpose—Doolittle LU updates, and Broyden's first and second methods performed implicitly. Detailed algorithms are given. Tests on some large stiff systems show that significant benefits can be obtained for some problems.

Key words. ordinary differential equations, stiff systems, quasi-Newton methods

1. Introduction. The numerical solution of stiff systems of ordinary differential equations (ODE's) relies heavily on methods for solving systems of algebraic equations. If the ODE system is nonlinear, then so are the algebraic systems that one must solve. Quasi-Newton methods, by which we mean primarily those which are in some sense generalizations of the one-dimensional secant method, have been found to be very successful methods for solving nonlinear algebraic systems. Over the last decade, a great deal of progress has been made in determining very effective quasi-Newton methods, especially for classes of problems which have in common some special structure which can be exploited.

Recent developments in quasi-Newton methods have a potential for application in the context of solving stiff ODE's. The most challenging ODE problems, for which the need for efficient algebraic system-solving methods is usually greatest, are generally those for which the algebraic systems to be solved are very large, have a Jacobian matrix which is sparse (e.g., banded), and have significant expense associated with function and Jacobian evaluations. A major source of such ODE problems is the solution of time-dependent partial differential equations by the method of lines (discretizing in space only) [15], [16]. Most quasi-Newton methods require relatively few Jacobian evaluations (or function evaluations if Jacobians are being approximated by difference quotients) and can often be implemented to offer savings on arithmetic as well. Furthermore, algebraic systems with sparse Jacobians have special structure which can be exploited in quasi-Newton methods. Preliminary studies of the use of quasi-Newton methods in a stiff ODE method were done by Hindmarsh and Byrne [17] and Alfeld [1].

In the following, we consider the application of three particular quasi-Newton methods to the solution of stiff ODE's. These methods are intended primarily for use on large algebraic systems with sparse Jacobians. The focus here is on ODE's for which the associated algebraic systems have sparse Jacobians and are so large that not only function and Jacobian evaluations but also storage and the cost of arithmetic are major concerns. The remainder of this introduction provides a very brief background on stiff ODE's, certain procedures for solving them numerically, and quasi-Newton methods.

* Received by the editors December 29, 1982, and in revised form October 12, 1983. This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48, and supported by the DOE Office of Basic Energy Sciences, Mathematical Sciences Branch.

† Department of Mathematics, University of Houston, Houston, Texas 77004.

‡ Mathematics and Statistics Division, L-316, Lawrence Livermore National Laboratory, University of California, Livermore, California 94550.

In the sequel, we describe the specific quasi-Newton methods of interest, their implementation in a particular algorithm for solving stiff ODE's, and the outcome of applying the resulting procedures to several test problems. We conclude with a summary of overall results and an outline of future areas of investigation.

1.1. Stiff ODE's. We consider an initial value problem for an ODE system,

$$\dot{y} = f(t, y), \qquad y(t_0) = y_0 \tag{1.1}$$

where the dot denotes d/dt and y is a vector of size N. The ODE in (1.1) is stiff if, roughly speaking, it contains a rapid decay process—i.e. rapid by comparison to the time scale of interest for the whole problem. Precise definitions of stiffness vary [22], but usually make reference to the Jacobian matrix,

$$J = \partial f/\partial y = J(t, y). \tag{1.2}$$

Stiffness means that at least one eigenvalue of $J(t, y)$ has a very large negative real part, when evaluated on the solution curve.

A large and very popular class of numerical methods for ODE initial value problems is that of linear multistep methods. These have the form

$$y_n = \sum_{j=1}^{K_1} \alpha_j y_{n-j} + h \sum_{j=0}^{K_2} \beta_j \dot{y}_{n-j}, \tag{1.3}$$

where y_k denotes the numerical approximation to $y(t_k)$, $h = t_k - t_{k-1}$ is considered fixed, and $\dot{y}_k$ denotes $f(t_k, y_k)$. (Variable-step analogues of (1.3) also exist but will not be considered here.) The popular BDF (backward differentiation formula) method corresponds to the case $K_2 = 0$, $K_1 = q =$ method order, and this method has been used extensively for stiff systems [11], [13], [16], [24]. We shall restrict our attention here to the context of a general purpose initial value ODE solver called LSODE [14], [16], which uses the BDF method in the stiff case. Thus at each time step, LSODE must solve an algebraic system

$$\begin{aligned} 0 &= F_n(y_n) \equiv y_n - h\beta_0 f(t_n, y_n) - a_n, \\ a_n &\equiv \sum_{j=1}^{q} \alpha_j y_{n-j} \end{aligned} \tag{1.4}$$

in which $\beta_0 > 0$.

In its unmodified form, LSODE solves (1.4) by modified Newton iteration, in which a prediction $y_n(0)$ is formed using an explicit formula of the type (1.3) and corrected by iterations

$$P_n(y_n(m+1) - y_n(m)) = -F_n(y_n(m)). \tag{1.5}$$

Here the iteration matrix P_n is an approximation to $\partial F_n/\partial y_n$, i.e.,

$$P_n \simeq \frac{\partial F_n}{\partial y_n} = I - h\beta_0 J(t_n, y_n), \tag{1.6}$$

that is held fixed for the iterations, and is usually also held fixed over several time steps. The LSODE user has the option of specifying J as either a full or a banded matrix, and as either supplied by a user subroutine or computed internally by difference quotient approximations. In all cases, the linear system (1.5) is solved by doing an LU decomposition of P_n (at the time it is formed) and using that for all iterations (on all steps) until a decision is made to reevaluate P_n. A convergence test is made on the basis of iterate differences. The time step itself must also pass a test on estimated local

truncation error, and the step size h and order q are adjusted dynamically on the basis of such tests. A change of step size from h to h' is accomplished, in effect, by interpolating in the existing solution history (the y_{n-j}, spaced at h) to get history at the new spacing h'.

1.2. Quasi-Newton methods. By a quasi-Newton method for solving an algebraic system $F(y)=0$, $F: R^N \to R^N$, we mean here any method which generates a sequence of approximate solutions $\{y(k)\}_{k=1,2,\ldots}$ from an initial approximation $y(0)$ by means of an iteration of the form

$$y(k+1) = y(k) - B_k^{-1} F(y(k)). \tag{1.7}$$

Such methods are regarded as variants of Newton's method, in which B_k is the Jacobian matrix $\partial F/\partial y(y(k))$, and so B_k is considered to be an approximation to $\partial F/\partial y(y(k))$.

Our particular interest is in quasi-Newton methods which are generalizations of the one-dimensional secant method. There one obtains B_{k+1} from B_k for some k by updating B_k so that for $s(k)=y(k+1)-y(k)$ and $z(k)=F(y(k+1))-F(y(k))$, the secant equation

$$B_{k+1} s(k) = z(k) \tag{1.8}$$

is satisfied as nearly as possible (in some sense) among all matrices satisfying any auxiliary conditions which might be imposed on B_{k+1}. Auxiliary conditions on B_{k+1} usually take the form of a requirement that B_{k+1} reflect some special structure of $\partial F/\partial y$ such as symmetry or a particular pattern of sparsity. The qualifier "as nearly as possible" is necessary because there may not always exist a matrix satisfying both (1.8) and the auxiliary conditions. For a full discussion of these methods, see Dennis and Moré [6] or Dennis and Schnabel [7].

As indicated at the outset, the focus here is on ODE's (1.1) which are stiff and for which N is large and the Jacobian J given by (1.2) is sparse. Our interest is in considering secant-update quasi-Newton methods as alternatives to the modified Newton algorithm (1.5) for solving the algebraic systems (1.4) which arise in applying BDF methods to such ODE's. In the remainder of this introduction, we touch on several quasi-Newton methods which can potentially use the sparsity of J to advantage. More specific descriptions of the methods of particular interest follow in the next section.

The sparse Broyden update given by Schubert [21] and Broyden [4] determines a quasi-Newton method which takes sparsity into account. This update has the property that if B_k in (1.7) has a desired pattern of sparsity, then so does the updated matrix B_{k+1}. A similar update which preserves not only sparsity but also symmetry in determining B_{k+1} from B_k has been given by Marwil [18] and Toint [23]. Methods employing these updates have desirable local convergence properties but, unfortunately, require new Jacobian factorizations after each update if direct linear algebra methods are used to obtain each step $-B_k^{-1}F(y(k))$. Here, we assume that direct methods are used for solving linear systems and, in particular, that matrix factorizations are LU factorizations as in LSODE. It is further understood that the problems of interest here are of such size that LU factorizations involve considerable expense, and so we regard the sparse Broyden and sparse symmetric updates as unsuitable in the present context, although they may well prove useful in situations which warrant the use of iterative methods for solving linear systems.

The sparse Broyden update, however, is fundamental to the first quasi-Newton method considered in the sequel, that which employs the Doolittle LU updating

procedure of Dennis and Marwil [5]. To describe this method briefly, let us suppose that in an iteration (1.7) one has obtained, for some value of k, a Doolittle decomposition

$$B_k = P_k L_k U_k, \tag{1.9}$$

using a partial pivoting strategy. In (1.9), P_k is a permutation matrix, L_k is a unit lower triangular matrix, and U_k is an upper triangular matrix. Then further approximate Jacobians are obtained by taking

$$B_{k+j} = P_k L_k U_{k+j} \tag{1.10}$$

for as many values of j as possible, where the sequence of upper triangular matrices $\{U_{k+j}\}$ is generated through (essentially) sparse Broyden updates. These updates are done for some but not necessarily all values of j; when done, they are determined by the secant equations

$$U_{k+j} s(k+j-1) = L_k^{-1} P_k^{-1} z(k+j-1). \tag{1.11}$$

If the successive matrices B_{k+j} are required to have a particular pattern of sparsity which is associated with certain corresponding patterns of sparsity of L_k and U_{k+j} (as is the case when the matrices B_{k+j} have a particular band structure), then the advantage of the Doolittle updating method in exploiting sparsity is clear: One maintains (sparse) factors of updated matrices having the desired pattern of sparsity without having to pay for additional factorizations or storage.

The other two quasi-Newton methods considered here take another approach to exploiting sparsity. In this approach, one obtains a factorization of B_k for some k and maintains subsequent approximate Jacobians implicitly, i.e., without explicitly updating B_k, its successors, or their factors, by creating (with B_k) and storing certain auxiliary vectors which incorporate update information. (The number of auxiliary vectors needed for each implicit update is equal to twice the rank of the update.)

Since quasi-Newton methods employing implicit updating incur certain storage and arithmetic costs associated with the auxiliary vectors, such methods are most likely to be effective for problems in which the price of some additional storage and arithmetic might be outweighed by the use of low-rank updates which have proved to be highly successful in solving general algebraic systems with full Jacobians. Matthies and Strang [19], Engelman [9], Engelman, Strang, and Bathe [10], and Geradin, Idelsohn, and Hogge [12] report effective implementations of implicit updating methods which employ several generally successful rank one and rank two updates. Here, we consider the implicit implementation of two updates due to Broyden [3]. The first Broyden update is widely regarded as the most successful update for general systems of nonlinear equations. The second Broyden update is considered to be less effective on general systems than Broyden's first update; however, it has been conjectured (see Alfeld [1]) that Broyden's second update performs particularly well in the context of solving stiff ODE's.

2. The quasi-Newton methods. In this section, we describe more specifically the quasi-Newton methods of interest. It is intended here that these methods be applied to a sequence of problems (1.4) for many values of n and that useful information about these problems be carried over from one value of n to the next. For convenience, however, we describe these methods in the context of solving a single system $F(y) = 0$, $F: R^N \to R^N$, with an iteration (1.7), beginning with an initial approximate solution $y(0)$ and an initial approximate Jacobian $B_0 \simeq \partial F / \partial y(y(0))$. The reader is safe in

assuming that $F = F_n$, that $y(0)$ is an initial approximation of y_n, and that all discussion below refers to the same time step, step n.

In describing the quasi-Newton methods below, our principal interest is in the updating algorithms used in them. However, efficient implementations of the updating algorithms must be well coordinated with the algorithms for determining iteration steps, and so the algorithms given here are somewhat broader in scope than updating algorithms per se. All of our quasi-Newton methods assume that B_0 is given in a form convenient for solving linear equations. They also depend on singling out particular values of k in (1.7) at which to update subsequent approximate Jacobians B_k. The rules for determining when to perform updates and when the iterates are sufficiently near the solution are outlined in the next section, in the discussion of our implementation of these methods in LSODE.

The updating algorithms described in the following are based on the well-known first and second updates of Broyden and the sparse Broyden update (see, for example, [6] or [7]). Here, we use variations of these updates which take into account an implicit rescaling of the independent and dependent variables by a nonsingular diagonal scaling matrix. Suppose that D is a given nonsingular diagonal scaling matrix such that $\tilde{y} = Dy$ and the problem $\tilde{F}(\tilde{y}) \equiv DF(D^{-1}\tilde{y}) = 0$ can be considered well-scaled. Such a scaling matrix is determined automatically by LSODE from user-supplied tolerance information. It is fixed throughout each time step, although it may vary from step to step. The manner of incorporating such a rescaling in an update is illustrated in [7, p. 187] for the first Broyden update.

We remark at this point that since our methods all take the full quasi-Newton step $s(k) = -B_k^{-1}F(y(k))$ at each iteration of (1.7), one can save arithmetic by substituting $F(y(k+1))$ for $[z(k) - B_k s(k)]$ wherever the latter expression appears in an update formula. Since it is desirable to keep computational cost of updating as low as possible, we incorporate this labor-saving substitution throughout our descriptions and implementations of the methods of interest here, even though it is not always regarded as advisable in other settings.

2.1. The Doolittle *LU* updating method. Suppose that at the initial iteration of (1.7) one is given a Doolittle decomposition $B_0 = P_0 L_0 U_0$, and suppose that B_0 and its successors are required to have a particular pattern of sparsity which in turn imposes a certain pattern of sparsity on their lower- and upper-triangular Doolittle factors. Denote by $\mathcal{U}$ the subspace of $R^{N\times N}$ consisting of all matrices having the pattern of sparsity required of these upper-triangular factors. For $i = 1, \cdots, N$, let S_i indicate the "sparsity" projection operator on R^N which imposes the sparsity pattern of the ith row of matrices in $\mathcal{U}$ on vectors in R^N, i.e., which for $j = 1, \cdots, N$ replaces the jth component of a vector in R^N by zero if the ijth entry of all matrices in $\mathcal{U}$ must be zero and otherwise leaves it unchanged. Further denote the ith component of $v \in R^N$ by $v^{(i)}$, the ith row of $U \in \mathcal{U}$ by $U^{(i)}$, and the Euclidean norm on R^N by $|\cdot|$.

We are given a nonsingular diagonal scaling matrix D and a parameter ε, $0 < \varepsilon \leqq 1$. (The purpose of ε is explained below.) The following is our algorithm for Doolittle LU updating.

ALGORITHM 2.1.

At step 0 *of* (1.7). Suppose that one has $y(0)$ and $B_0 = P_0 L_0 U_0$. Then compute $F(y(0))$, $s(0) = -B_0^{-1}F(y(0))$, and $y(1) = y(0) + s(0)$, and go on to step 1 if necessary.

At step k *of* (1.7), $k \geqq 1$. Suppose that one has $y(k)$, $s(k-1)$, and $B_{k-1} = P_0 L_0 U_{k-1}$. Then do the following:

(1) Compute $F(y(k))$ and $s(k) = -L_0^{-1}P_0^{-1}F(y(k))$.

(2) If no update is to be made, take $U_k = U_{k-1}$; otherwise do the following for $i = 1, \cdots, N$:

(a) If $\varepsilon |Ds(k-1)| < |DS_i s(k-1)|$, set

$$U_k^{(i)} = U_{k-1}^{(i)} - \frac{s(k)^{(i)}}{[S_i s(k-1)]^T D^2 [S_i s(k-1)]} [S_i s(k-1)]^T D^2;$$

(b) otherwise, take $U_k^{(i)} = U_{k-1}^{(i)}$.

(3) Compute $s(k) \leftarrow U_k^{-1} s(k)$, $y(k+1) = y(k) + s(k)$, and go on to step $k+1$ if necessary.

It is clear that $U_k \in \mathcal{U}$ if $U_{k-1} \in \mathcal{U}$ and that (1.11) is satisfied provided each test $\varepsilon |Ds(k-1)| < |DS_i s(k-1)|$ in (2.a) above is passed for $i = 1, \cdots, N$ (in which case U_k is just the usual scaled sparse Broyden update of U_{k-1} in $\mathcal{U}$). The purpose of these tests on ε is to insure that no correction is made in a row of U_{k-1} when the projected step $S_i s(k-1)$ along that row is too small relative to the full step. These tests play an important role in the convergence analysis given in [5]. It can be argued heuristically that these tests are more than a theoretical convenience as follows: If $B_{k-1} = P_0 L_0 U_{k-1}$ is a good approximation to $\partial F / \partial y(y(k-1))$, then $w = L_0^{-1} P_0^{-1} z(k-1)$ is almost but not quite the result of operating on $s(k-1)$ with an upper-triangular matrix. In light of the qualifier "not quite," one sees that the ratios $w^{(i)} / |S_i s(k-1)|$ can be well-defined but arbitrarily large; thus updating without these tests can do arbitrarily great violence to the approximate Jacobians. Note that large values of ε correspond to updating that is more conservative in that fewer row updates are likely to be done. In the experiments with Doolittle LU updating in LSODE reported in the sequel, we used a value of ε roughly equal to the unit roundoff. Such a small choice of ε implies that very few of the tests on ε will not be passed; a similarly small choice of ε is reported to be effective in the experiments in [5]. We also remark that there is a certain *restart* procedure for periodically obtaining a new Jacobian or approximate Jacobian which is included in the method of Dennis and Marwil and which is necessary for their convergence analysis. Such a restart procedure is not necessary in the updating algorithm above because it is provided for elsewhere in our implementation of the algorithm in LSODE.

2.2. The implicit Broyden updating methods. To describe our implicit implementation of Broyden's first update, we begin by recalling that if D is a nonsingular diagonal scaling matrix and B_{k+1} is obtained by a scaled first Broyden update of B_k, then

$$B_{k+1} = B_k + \frac{[z(k) - B_k s(k)] s(k)^T}{s(k)^T D^2 s(k)} D^2. \tag{2.1}$$

(See Dennis and Schnabel [7, p. 187, formula (8.3.1)].) It follows from the Sherman–Morrison–Woodbury formula (see Ortega and Rheinboldt [20]) that

$$B_{k+1}^{-1} = \left\{ I + \frac{[s(k) - B_k^{-1} z(k)] s(k)^T}{s(k)^T D^2 B_k^{-1} z(k)} D^2 \right\} B_k^{-1}.$$

By extension, if updated matrices $B_1, \cdots, B_l$ are generated from B_0 by (2.1) in conjunction with an iteration (1.7), then one has

$$B_l^{-1} = [I + v(l) w(l)^T] \cdots [I + v(1) w(1)^T] B_0^{-1}, \tag{2.2}$$

where the auxiliary vectors $v(i)$ and $w(i)$ are given by

$$v(i) = \frac{s(i-1) - B_{i-1}^{-1} z(i-1)}{s(i-1)^T D^2 B_{i-1}^{-1} z(i-1)}, \tag{2.3}$$

and

$$w(i) = D^2 s(i-1), \tag{2.4}$$

for $i = 1, \cdots, l$.

Now suppose that at the initial iteration of (1.7) one is given B_0 in factored form or in some other form convenient for the solution of linear equations. In the applications of interest here, for example, B_0 is likely to be specified by its matrix factors together with a set of auxiliary vectors such as those appearing in (2.2). Suppose also that a nonsingular diagonal scaling matrix D is given. The following is our algorithm for implicitly implementing Broyden's first update.

ALGORITHM 2.2.

At step 0 *of* (1.7). Suppose that one has $y(0)$ and B_0 in a form convenient for the solution of linear equations. Then compute $F(y(0))$, $s(0) = -B_0^{-1}F(y(0))$, and $y(1) = y(0) + s(0)$, and go on to step 1 if necessary.

At step k *of* (1.7), $k \geqq 1$. Suppose that one has $y(k)$, $s(k-1)$, B_0, and also $v(1), \cdots, v(l)$ and $w(1), \cdots, w(l)$, if $k > 1$ and l updates have been made for some l, $1 \leqq l \leqq k-1$. Then do the following:

(1) Compute $F(y(k))$, $s(k) = -B_0^{-1}F(y(k))$, and if l updates ($l > 0$) have been made, compute

$$s(k) \leftarrow [I + v(l)w(l)^T] \cdots [I + v(1)w(1)^T]s(k).$$

(2) If no update is to be made, then go to (3); otherwise, compute

$$w(l+1) = D^2 s(k-1),$$

$$v(l+1) = \frac{s(k)}{w(l+1)^T[s(k-1) - s(k)]},$$

$$s(k) \leftarrow [I + v(l+1)w(l+1)^T]s(k).$$

(3) Compute $y(k+1) = y(k) + s(k)$ and go on to step $k+1$ if necessary.

We note that at the end of part (1) of the algorithm at step k, $k \geqq 1$, one has computed $s(k) = -B_{k-1}^{-1}F(y(k))$, where B_{k-1}^{-1} is implicitly taken to be either B_0^{-1} if no updating has been done since the initial step or $[I + v(l)w(l)^T] \cdots [I + v(1)w(1)^T]B_0^{-1}$ if l updates have been made since the initial step. If no update is made at the current step, i.e., if $B_k^{-1} = B_{k-1}^{-1}$ implicitly, then one accepts this $s(k)$ as the iteration step. If an update is made, then one first computes $v(l+1)$ and $w(l+1)$ according to (2.3) and (2.4), respectively, so that $B_k^{-1} = [I + v(l+1)w(l+1)^T] \cdots [I + v(1)w(1)^T]B_0^{-1}$ implicitly, and then updates $s(k)$ to obtain $s(k) = -B_k^{-1}F(y(k))$.

It is evident that each update that is made requires the formation and storage of two vectors. If N is large, then storage and, hence, the number of updates that one can make, may be sharply limited. In any case, there is certainly a maximum number of updates that can be accommodated in practice. When this number is reached, one has a variety of options such as obtaining a new (approximate) Jacobian from scratch, discarding all update vectors and restarting the updating of B_0 from scratch, replacing the early update vectors with current ones, or simply doing no additional updating. We chose the last option in our implementation with a maximum allowable number of updates equal to 5, since our intention was to update only very infrequently and, therefore, we felt it likely that the code would call for a new Jacobian more often than this maximum allowable number of updates would be reached.

It should also be mentioned that some arithmetic is incurred not only in forming the update vectors but also in using them to determine subsequent iteration steps.

However, most of the work of forming the update vectors is also applied to forming iteration steps concurrently. Furthermore, one sees from the algorithm that using the update vectors to form an iteration step or an additional update vector is unlikely to be regarded as costly, especially on a computer which performs vector operations efficiently.

To describe our implicit implementation of Broyden's second update, we first note that this update is most conveniently written in the form of an inverse analogue of (2.1), which is

$$B_{k+1}^{-1} = B_k^{-1} + \frac{[s(k) - B_k^{-1} z(k)] z(k)^T}{z(k)^T D^2 z(k)} D^2$$

$$= B_k^{-1} \left\{ I - \frac{[z(k) - B_k s(k)] z(k)^T}{z(k)^T D^2 z(k)} D^2 \right\}.$$

If updated matrices $B_1, \cdots, B_l$ are generated by this formula in conjunction with an iteration (1.7), then the counterpart of (2.2) is

$$B_l^{-1} = B_0^{-1} [I - t(1) u(1)^T] \cdots [I - t(l) u(l)^T],$$

where

$$t(i) = \frac{[z(i-1) - B_{i-1} s(i-1)]}{z(i-1)^T D^2 z(i-1)}$$

and

$$u(i) = D^2 z(i-1)$$

for $i = 1, \cdots, l$.

Suppose that B_0 is given in factored form or in some other form convenient for the solution of linear equations. Let D be a nonsingular diagonal scaling matrix. Algorithm 2.3 below is our algorithm for implicitly implementing Broyden's second update. We note that remarks similar to those following Algorithm 2.2 above are also appropriate for Algorithm 2.3.

ALGORITHM 2.3.

At step 0 *of* (1.7). Suppose that one has $y(0)$ and B_0 in a form convenient for the solution of linear equations. Then compute $F(y(0))$, $s(0) = -B_0^{-1} F(y(0))$, and $y(1) = y(0) + s(0)$, and go on to step 1 if necessary.

At step k *of* (1.7), $k \geqq 1$. Suppose that one has $y(k)$, $F(y(k-1))$, B_0 and also $t(1), \cdots, t(l)$ and $u(1), \cdots, u(l)$, if $k > 1$ and l updates have been made for some l, $1 \leqq l \leqq k-1$. Then do the following:

(1) Compute $F(y(k))$.

(2) If no update is to be made, compute

$$s(k) = \begin{cases} -B_0^{-1} [I - t(1) u(1)^T] \cdots [I - t(l) u(l)^T] F(y(k)) & \\ & \text{if } l > 0 \text{ updates have been made,} \\ -B_0^{-1} F(y(k)) & \text{otherwise} \end{cases}$$

and go to (3). If an update is to be made, compute

$$z(k-1) = F(y(k)) - F(y(k-1)),$$

$$u(l+1) = D^2 z(k-1), \qquad t(l+1) = F(y(k)) / u(l+1)^T z(k-1),$$

$$s(k) = -B_0^{-1} [I - t(1) u(1)^T] \cdots [I - t(l+1) u(l+1)^T] F(y(k)).$$

(3) Compute $y(k+1) = y(k) + s(k)$ and go on to step $k+1$ if necessary.

3. Algorithmic implementation. In implementing each of the three update algorithms described above, the LSODE package was modified so as to perform occasional quasi-Newton updates. In order to describe precisely the algorithm for this, we must first outline the structure and overall algorithm of LSODE, to the extent that this is relevant here.

3.1. The unmodified algorithm. Aside from several auxiliary routines of secondary importance, the structure of LSODE (unmodified) is shown in Fig. 1, with the dashed line connections ignored. Subroutine LSODE is a driver, and subroutine STODE performs a single step and associated error control. STODE calls PREPJ to evaluate and do an LU factorization of the matrix P_n of (1.6), and subsequently calls SOLSY to solve the linear system (1.5). (Recall that P_n approximates $I - h\beta_0 J(t_n, y_n)$.) Both of these routines call LINPACK routines [8] to do the matrix operations.

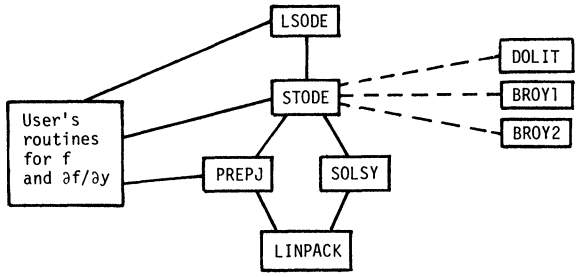

FIG. 1. *Simplified overall structure of* LSODE *package.*

Within STODE, the basic algorithm for step n, in its unmodified form, is as follows:

(1) Set flag showing whether to reevaluate J.
(2) Predict $y_n(0)$.
(3) Compute $f(t_n, y_n(0))$; set $m = 0$.
(4) Call PREPJ if flag is on.
(5) Form $F_n(y_n(m))$.
(6) Call SOLSY and correct to get $y_n(m+1)$.
(7) Update estimate of convergence rate constant C if $m \geqq 1$.
(8) Test for convergence.
(9) If convergence test failed:
  (a) Set $m \leftarrow m+1$.
  (b) If $m < 3$, compute $f(t_n, y_n(m))$ and go to 5.
  (c) If $m = 3$ and J is current, set $h \leftarrow h/4$ and go to 1 (redo step).
  (d) If $m = 3$ and J is not current, set flag to reevaluate J and go to step (3) (redo step).
(10) If the convergence test passed, update history, do error test, etc.

In algorithm step (1) above, the decision is made to reevaluate J (and redo the LU factorization of $P = I - h\beta_0 J$) if either

(a) 20 steps have been taken since the last evaluation of J, or

(b) the value of $h\beta_0$ has changed by more than 30% since J was last evaluated.

In algorithm step (7), the iterate difference $s_n(m) = y_n(m+1) - y_n(m)$ is used, together with $s_n(m-1)$ if $m \geqq 1$, to form the ratio $\text{DELR} = \|s_n(m)\| / \|s_n(m-1)\|$, and

C is updated to be the larger of $.2C$ and DELR. C is reset to .7 whenever J is evaluated. The norm is a weighted root-mean-square norm, with weights determined by user-supplied relative and absolute tolerance parameters RTOL and ATOL. (These weights correspond to the diagonal scaling matrix D referred to in § 2.) The convergence test in step (8) requires the product $\|s_n(m)\| \min(1, 1.5C)$ to be less than a constant which depends only on q. This is based on linear convergence, with the idea that $C\|s_n(m)\|$ is a better estimate of the error in $y_n(m+1)$ than $\|s_n(m)\|$ is. Algorithm step (10) includes step and order selection for the next step (if the error test passed) or for redoing the current step (if it failed), but the details of that are not relevant here.

3.2. The modified algorithm for updating. There are three main additions to this structure, each of which is a call from STODE to one of the updating routines, shown by the dashed lines in Fig. 1. Subroutines DOLIT, BROY1, and BROY2 perform (respectively) Doolittle updates, Broyden's first update, and Broyden's second update.

Any implementation of an updating strategy in LSODE will necessarily have to include rules which decide when to reevaluate J and when to perform an update of P. Within LSODE, at any given step the only feasible measures of the quality of the current P are the following:

(i) The ratio of the current value of $h\beta_0$ to the value as of the last J evaulation.
(ii) The number of steps taken since the last J evaluation.
(iii) RCC $= |(h\beta_0)_n/(h\beta_0)_{n-1} - 1|$, where $(h\beta_0)_k$ denotes the value of $h\beta_0$ at step k.
(iv) DELR $= \|s_n(m)\|/\|s_n(m-1)\|$ = the ratio of iterate differences (when $m \geqq 1$).

The exact rules chosen for reevaluating J and updating P are based on these four quantities, as follows. In the course of the algorithm, a set of six flags is set according to the following rules:

(i) Flag 1 is turned on if either
  (a) 60 steps have been taken since the last evaluation of J,
  (b) the value of $h\beta_0$ has changed by more than 30% since J was last evaluated, or
  (c) the value of $h\beta_0$ has changed by more than 30% from the value on the previous step (i.e., RCC $> .3$).
(ii) Flag 2 is turned on if $.2 <$ RCC $\leqq .3$.
(iii) Flag 3 is turned on if $.1 <$ RCC $\leqq .2$.
(iv) Flag 4 is turned on if $m \geqq 2$ and DELR $\leqq .1$.
(v) Flag 5 is turned on if $m \geqq 2$ and $.1 <$ DELR $\leqq 1.0$.
(vi) Flag 6 is turned on if $m \geqq 2$ and DELR > 1.0.

Then time step n of the integration is given by substituting the following modified steps in the STODE algorithm given in § 3.1 (i.e., replacing step (1) by (1′), etc.):

(1′) (a) Set flag 1 showing whether to reevaluate J.
  (b) Set flag 2 and flag 3 showing whether to update P_n.
  (c) Set IUP $= 1$ if flag 2 or flag 3 is on; otherwise set IUP $= 0$.
(6′) Update the matrix P_n if IUP $= 1$ and $m > 0$; call SOLSY as appropriate; correct to get $y_n(m+1)$.
(8′) (a) If flag 2 is on and $m = 0$, set $m \leftarrow m+1$, compute $f(t_n, y_n(m))$ and go to (5) (forcing at least two corrections).
  (b) Test for convergence.
(9′) If convergence test failed:
  (a) Set $m \leftarrow m+1$.
  (b) If $m < 2$, compute $f(t_n, y_n(m))$ and go to (5).

(c) If $m \geqq 2$, compute DELR $= \|s_n(m)\| / \|s_n(m-1)\|$. Set flag 4, flag 5, and flag 6 according to value of DELR.
(d) If flag 6 is on, go to (h).
(e) If $m < 5$ and flag 4 is on, set IUP $= 0$, compute $f(t_n, y_n(m))$, and go to (5).
(f) If $m < 5$ and flag 5 is on, set IUP $= 1$, compute $f(t_n, y_n(m))$, and go to (5).
(g) If $m = 5$, go to (h).
(h) If J is current, set $h \leftarrow h/4$ and go to (1′) (redo step). Otherwise, set flag to reevaluate J and go to (3) (redo step with the same h).

We note that updating is done when either $.1 <$ RCC $\leqq .3$ or $.1 <$ DELR $\leqq 1.0$ (with $m \geqq 2$), and that at least two corrections are performed when $.2 <$ RCC $\leqq .3$. If DELR $\leqq .1$ and $m \geqq 2$, then no updating is done regardless of what the other strategies imply. Also, the maximum number of iterations allowed per step in the modification is 5 (compared to 3 in unmodified LSODE) to allow for steps in which the current P is initially somewhat out of date but after several updates are performed should be sufficiently good to complete that step and possibly several more. The structure of step (6′) in the above algorithm depends significantly upon the particular updating scheme being employed, following the algorithm given in § 2. Further details are given in [2].

4. Numerical tests. The algorithms described above, and implemented in modified versions of the LSODE package, were tested on various ODE test problems. In this section we give, for each of four problems, a brief description of the problem, numerical results obtained, and some discussion. Three of the four test problems are obtained from time-dependent partial differential equation (PDE) systems solved by the method of lines. Further details on the problem specifications are available in [2]. All of the tests were done on a Cray-1 computer with the CFT compiler.

The algorithms tested included the unaltered LSODE package (as discussed in § 3.1) and versions modified to perform Doolittle and implicit Broyden updates of first and second kinds (as described in § 3.2). In addition, an algorithm was tested which uses the modified Newton strategy from the updating algorithms (of § 3.2) but which never performs matrix updates. This tests the value of the new corrector loop strategy as distinct from the updates themselves.

In what follows, we will use the following abbreviations for the various algorithms:
LSODE: unaltered LSODE package;
DOLIT: LSODE with Doolittle LU updating;
IMPB1: Implicit updating by Broyden's first method;
IMPB2: Implicit updating by Broyden's second method;
LSODE*: LSODE with new strategy but no updates.

Unless otherwise stated, the algorithms are as described in detail in § 3. However, the various heuristic parameters in the updating strategy were varied somewhat in many of the test runs. Where meaningful, results for altered parameter values are also given.

For each problem and each algorithm, a test run was made and yielded various statistics. Those of interest are defined as follows:
R.T. = run time (CPU sec);
NST = number of time steps;
NFE = number of f evaluations;
NJE = number of J evaluations (= number of LU decompositions);
NUP = number of matrix updates.
In all cases, the J evaluations were done by a user-supplied subroutine.

4.1. Test problem 1. This problem is based on a pair of PDE's in two dimensions, representing a simple model of ozone production in the stratosphere with diurnal kinetics. (See also [16] for comparison tests on this problem.) There are two dependent variables c^i representing concentrations of O_1 and O_3 (ozone) in moles/cm^3, which vary with altitude z and horizontal position x, both in km, with $0 \leqq x \leqq 20$, $30 \leqq z \leqq 50$, and with time t in sec, $0 \leqq t \leqq 86{,}400$ (one day). These obey a pair of coupled reaction-diffusion equations:

$$\frac{\partial c^i}{\partial t} = K_h \frac{\partial^2 c^i}{\partial x^2} + \frac{\partial}{\partial z}\left[K_v(z)\frac{\partial c^i}{\partial z}\right] + R^i(c^1, c^2, t) \qquad (i = 1, 2),$$

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

$$R^1(c^1, c^2, t) = -k_1 c^1 - k_2 c^1 c^2 + k_3(t) \cdot 7.4 \cdot 10^{16} + k_4(t) c^2,$$

$$R^2(c^1, c^2, t) = k_1 c^1 - k_2 c^1 c^2 - k_4(t) c^2,$$

$$k_1 = 6.031, \qquad k_2 = 4.66 \cdot 10^{-16},$$

$$k_3(t) = \begin{cases} \exp\left[-22.62/\sin\left(\pi t/43{,}200\right)\right] & \text{for } t < 43{,}200, \\ 0 & \text{otherwise,} \end{cases}$$

$$k_4(t) = \begin{cases} \exp\left[-7.601/\sin\left(\pi t/43{,}200\right)\right] & \text{for } t < 43{,}200, \\ 0 & \text{otherwise.} \end{cases}$$

Homogeneous Neumann boundary conditions are posed. The initial condition functions are polynomials chosen to be slightly peaked in the center and consistent with the boundary conditions:

$$c^1(x, z, 0) = 10^6 \alpha(x)\beta(z), \qquad c^2(x, z, 0) = 10^{12}\alpha(x)\beta(z),$$

$$\alpha(z) \equiv 1 - (.1x - 1)^2 + (.1x - 1)^4/2, \qquad \beta(z) \equiv 1 - (.1z - 4)^2 + (.1z - 4)^4/2.$$

The PDE's are treated by central differencing, on a rectangular grid with uniform spacings, $\Delta x = 20/(J-1)$, $\Delta z = 20/(K-1)$. The differencing for the vertical diffusion term is

$$(1/\Delta z^2)[K_v(z_{k+1/2})(c^i_{j,k+1} - c^i_{jk}) - K_v(z_{k-1/2})(c^i_{jk} - c^i_{j,k-1})].$$

The boundary conditions are simulated by taking $c^i_{0,k} = c^i_{2,k}$ (all k), and similarly on the other boundary segments. The size of the ODE system is $N = 2JK$. The variables are indexed first by species, then by x position, and finally by z position. Thus in $\dot{y} = f(t, y)$, we have $c^i_{jk} = y_m$, $m = i + 2(j-1) + 2J(k-1)$.

For these tests, we chose $J = 20$ and $K = 20$ ($N = 800$). The problem is stiff because of the kinetics, and the Jacobian has half-bandwidths $ML = MU = 2J = 40$. (The diffusion terms are a potential cause of stiffness also, but are not in fact, for these choices of Δx, Δz, K_h, K_v.) A mixed relative/absolute error tolerance was chosen, with RTOL $= 10^{-5}$ and ATOL $= 10^{-3}$.

The results of testing the five algorithms on this problem are shown in Table 1. In this case, all three updating algorithms produced shorter run times than LSODE or LSODE*, with IMPB2 being the fastest. By comparison with LSODE, IMPB2 trades 22 updates for a reduction in Jacobian evaluations by 21—a tradeoff that saves over 6 sec. (22%) of CPU time.

4.2. Test problem 2. This problem is based on a reaction-diffusion system arising from a Lotka–Volterra competition model, with diffusion effects in two space dimensions included. There are two species densities c^i, varying over the spatial habitat

TABLE 1
Test results for problem 1.

Algorithm	R.T.	NST	NFE	NJE	NUP
LSODE	27.76	459	661	85	0
DOLIT	23.20	408	587	67	31
IMPB1	26.73	456	655	80	32
IMPB2	21.64	388	544	64	22
LSODE*	28.06	471	714	83	0

$\Omega = \{(x, z): 0 \leqq x \leqq 1, 0 \leqq z \leqq 1.8\}$, and time t in sec, $0 \leqq t \leqq 10$. These obey

$$\frac{\partial c^i}{\partial t} = d_i\left(\frac{\partial^2 c^i}{\partial x^2} + \frac{\partial^2 c^i}{\partial z^2}\right) + f^i(c^1, c^2) \qquad (i = 1, 2),$$

$$d_1 = .05, \qquad d_2 = 1.0,$$

$$f^1(c^1, c^2) = c^1(b_1 - a_{11}c^1 - a_{12}c^2),$$

$$f^2(c^1, c^2) = c^2(b_2 - a_{21}c^1 - a_{22}c^2),$$

$$a_{11} = 10^6, \quad a_{12} = 1, \quad a_{21} = 10^6 - 1, \quad a_{22} = 10^6, \quad b_1 = b_2 = 10^6 - 1 + 10^{-6}.$$

Homogeneous Neumann boundary conditions are imposed. Initial conditions are chosen consistent with the boundary conditions:

$$c^1(x, z, 0) = 500 + 250 \cos(\pi x) \cos(10\pi z/1.8),$$

$$c^2(x, z, 0) = 200 + 150 \cos(10\pi x) \cos(\pi z/1.8).$$

Given the above parameter values and initial conditions, the solution of this reaction-diffusion system converges as $t \to \infty$ to the equilibrium solution $c^1 = c^1_* \equiv 1 - 10^{-6}$, $c^2 = c^2_* \equiv 10^{-6}$. The two partial differential equations are again treated by central differencing, on a rectangular J by K grid with uniform spacings, with boundary conditions treated as before. The size of the ODE system is $N = 2JK$, and the variables are indexed by species, then by x position, and finally by z position.

For this test, we chose $J = 20$ and $K = 20$ ($N = 800$). The problem is stiff mainly because of the interaction terms, and the Jacobian has half-bandwidths $ML = MU = 2J = 40$. A mixed relative/absolute error tolerance was chosen, with RTOL $= 10^{-6}$ and ATOL $= 10^{-9}$.

The test results on this problem are given in Table 2. Here, in all three updating algorithms, the updates seem to have had no beneficial effect, and simply increased the cost. The reason may be that for this problem the step size grows steadily throughout the problem, at a rate which forces reevaluations of both the Jacobian matrix and P every 8–10 steps, regardless of updating method.

TABLE 2
Test results for problem 2.

Algorithm	R.T.	NST	NFE	NJE	NUP
LSODE	23.92	582	665	66	0
DOLIT	25.20	583	696	69	30
IMPB1	25.20	583	696	69	30
IMPB2	25.33	588	708	70	31
LSODE*	25.63	583	700	72	0

4.3. Test problem 3. This problem is an ODE system in population biology which models the interaction of N species competing for the same limited resource. It is also of Lotka–Volterra type and has the form, with time t in seconds,

$$\frac{du_i}{dt} = u_i\left(b_i - \sum_{j=1}^{N} a_{ij}u_j\right) \qquad (i = 1, \cdots, N).$$

The matrix $A = (a_{ij})$ is taken to be symmetric and banded. For test purposes, we chose $N = 100$, half-bandwidths $ML = MU = 20$, and coefficients

$$a_{11} = \cdots = a_{25,25} = 1, \qquad a_{26,26} = \cdots = a_{50,50} = 10^6,$$
$$a_{51,51} = \cdots = a_{75,75} = 10^{10}, \qquad a_{75,75} = \cdots = a_{100,100} = 10^{11},$$
$$a_{ij} = .0002a_{ii} \quad \text{for } i \neq j, \quad |i-j| \leqq 20.$$

We define $b = (b_i)$ by setting $u^* = (1, \cdots, 1)$ and $b = Au^*$. Then $u(t) \equiv u^*$ is an equilibrium solution of the ODE system, to which the solution of the ODE system converges as $t \to \infty$. We chose initial conditions $u_i(0) = 1.5\ i$, and took $0 \leqq t \leqq 10$. The problem is stiff for the parameters chosen. A mixed relative/absolute error tolerance was chosen with $\text{RTOL} = \text{ATOL} = 10^{-6}$.

The test results on this problem are given in Table 3. Here only IMPB1 and IMPB2 were competitive with LSODE, while DOLIT was not. The explanation may be as offered for Problem 2—steadily and rapidly growing step sizes.

TABLE 3
Test results for problem 3.

Algorithm	R.T.	NST	NFE	NJE	NUP
LSODE	5.32	652	770	80	0
DOLIT	6.22	694	871	100	36
IMPB1	5.39	646	773	83	24
IMPB2	5.31	642	765	81	29
LSODE*	5.66	670	820	85	0

4.4. Test problem 4. Like Problem 2, this problem is based on a reaction-diffusion system arising from a Lotka–Volterra predator-prey model with diffusion effects in two space dimensions. Here the prey and predator species densities vary over $\Omega = \{(x, z): 0 \leqq x \leqq 1, 0 \leqq z \leqq 1\}$ and $0 \leqq t \leqq 3$. The equations are the same as in Problem 2 except with

$$f^1(c^1, c^2) = c^1(b_1 - a_{12}c^2), \qquad f^2(c^1, c^2) = c^2(-b_2 + a_{21}c^1),$$
$$b_1 = 1, \quad a_{12} = .1, \quad a_{21} = 100, \quad b_2 = 1000,$$

and initial conditions

$$c^1(x, z, 0) = 10 - 5\cos(\pi x)\cos(10\pi z),$$
$$c^2(x, z, 0) = 17 + 5\cos(10\pi x)\cos(\pi z).$$

Here the solution becomes spatially homogeneous as $t \to \infty$, and tends to a time-periodic solution of the Lotka–Volterra ODE system modeling the predator-prey interaction without spatial effects, namely $dc^i/dt = f^i$ $(i = 1, 2)$. This last system is alternately stiff and nonstiff depending on the position of the solution in phase space.

The two PDE's are differenced just as in Problem 2, except that the mesh dimensions are $J = K = 10$ ($N = 200$), and hence the Jacobian has half-bandwidths $ML = MU = 2J = 20$. The tolerance parameters used were RTOL $= 10^{-6}$ and ATOL $= 10^{-4}$.

The test results on this problem are given in Table 4. Here IMPB2 gave an 8% reduction in run time, trading 50 updates for 16 fewer Jacobian evaluations. But DOLIT and IMPB1 gave little or no overall cost reduction for this particular algorithm. However, runs were also made with slightly different parameter values in the updating strategy (50% in place of 30% in criterion (b) for flag 1, and .4 in place of .3 in the setting of flag 1 (criterion (c)) and of flag 2; see the detailed algorithm in § 3.2). In these runs, all three updating algorithms ran faster than LSODE, from 15% faster (IMPB2) to 9.5% faster (DOLIT). However, LSODE* ran 12% faster than LSODE also.

TABLE 4
Test results for problem 4.

Algorithm	R.T.	NST	NFE	NJE	NUP
LSODE	8.84	1,248	1,635	129	0
DOLIT	8.80	1,125	1,630	125	53
IMPB1	8.86	1,270	1,677	129	58
IMPB2	8.14	1,180	1,560	113	50
LSODE*	8.79	1,242	1,632	127	0

5. Discussion. We have presented three quasi-Newton methods and discussed their application to solving the nonlinear algebraic equations arising in the solution of stiff ODE systems by BDF methods. Our focus has been on the case in which the ODE system is very large and the Jacobian of the system is sparse, and the quasi-Newton methods considered here were chosen because of their potential for exploiting sparsity. This investigation has not been exhaustive. For one thing, the testing of the methods chosen here has been somewhat limited; for another, there are other quasi-Newton methods, as well as variations of those considered here, which might also be appropriate for this setting.

It must be recognized, however, that the area under investigation is broad and largely unexplored. From the point of view of solving algebraic equations, the ODE setting is markedly different from that of a fixed algebraic problem. In particular, it is clear that for best results a great deal of interaction should take place between the ODE integration algorithm (the step and order selection and its various heuristic decision rules) and the algorithm implementing any given quasi-Newton method (and its heuristics). We do not claim to have achieved an optimal merge of the two, but we believe that we have made the most serious attempt to date at doing so.

Our test results show that, *for some problems*, the combined ODE and quasi-Newton algorithms considered here can offer significant improvements over the unmodified algorithm. We found further that, in our tests, the implicit updates by Broyden's second method came the closest to being consistently beneficial (when updates of any kind were beneficial). The Doolittle method and updates based on Broyden's first method usually (but not always) did a poorer job.

The test results also suggest that, *for some problems*, the quasi-Newton methods studied here may not be capable of reducing the total costs. Specifically, the potential helpfulness of the updates used here seems to be precluded for problems in which the

order and step size values vary rather rapidly during the integration. (However, other quasi-Newton updates, which are applicable only in the small-system case, offer hope of dealing effectively with such rapid variations.) The most favorable results with updates seem to occur when the updated values of $(\partial F_n/\partial y_n)^{-1}$ (actual or virtual) are as accurate (or produce the same speed of convergence) as those that would be gotten by reevaluating $\partial F_n/\partial y_n$ from scratch, but are obtained at much lower cost. With the updates considered here, this fortunate situation seems most likely to occur when sizeable numbers of consecutive integration steps are taken over which the step sizes and Jacobian values change only relatively little. In a general purpose solver like LSODE, a natural approach to the design of the algorithm is to try to detect when such conditions hold and when they do not, and attempt to restrict the use of updates in a dynamic way accordingly. This idea has been a guiding principle in our work, but there is certainly more to be done towards that end.

REFERENCES

[1] P. ALFELD, *Two devices for improving the efficiency of stiff* ODE *solvers*, in Proc. 1979 SIGNUM Meeting on Numerical Ordinary Differential Equations, Univ. Illinois, Dept. Computer Science, Report 79-1710, Urbana, pp. 24-1 to 24-3.

[2] P. N. BROWN, A. C. HINDMARSH AND H. F. WALKER, *Experiments with quasi-Newton methods in solving stiff* ODE *systems*, Lawrence Livermore National Laboratory Report UCRL-88470, December 1982.

[3] C. G. BROYDEN, *A class of methods for solving nonlinear simultaneous equations*, Math. Comp., 19 (1965), pp. 577–593.

[4] ———, *The convergence of an algorithm for solving sparse nonlinear systems*, Math. Comp., 25 (1971), pp. 285–294.

[5] J. E. DENNIS, JR. AND E. S. MARWIL, *Direct secant updates of matrix factorizations*, Math. Comp., 38 (1982), pp. 459–474.

[6] J. E. DENNIS, JR. AND J. J. MORÉ, *Quasi-Newton methods, motivation and theory*, SIAM Rev., 19 (1977), pp. 46–89.

[7] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[8] J. J. DONGARRA, J. R. BUNCH, C. B. MOLER AND G. W. STEWART, LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[9] M. S. ENGELMAN, *Quasi-Newton methods in fluid dynamics*, in Proc. 4th Conference on Mathematics of Finite Elements and Applications, Brunel University, Uxbridge, England, April 27–May 1, 1981.

[10] M. S. ENGELMAN, G. STRANG AND K. J. BATHE, *The application of quasi-Newton methods in fluid dynamics*, Int. J. Numer. Meth. Eng., 17 (1981), pp. 707–718.

[11] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971, pp. 158–166.

[12] M. GERADIN, S. IDELSOHN AND M. HOGGE, *Computational strategies for the solution of large nonlinear problems via quasi-Newton methods*, Computers and Structures, 13 (1981), pp. 73–81.

[13] A. C. HINDMARSH, *Linear multistep methods for ordinary differential equations: method formulations, stability, and the methods of Nordsieck and Gear*, Lawrence Livermore National Laboratory Report UCRL-51186, Rev. 1, March 1972.

[14] ———, LSODE *and* LSODI, *two new initial value ordinary differential equation solvers*, in ACM SIGNUM Newsletter, vol. 15, no. 4 (December 1980), pp. 10–11.

[15] ———, ODE *solvers for use with the method of lines*, in Advances in Computer Methods for Partial Differential Equations—IV, R. Vichnevetsky and R. S. Stepleman eds., IMACS, New Brunswick, NJ, 1981, pp. 312–316.

[16] ———, ODEPACK, *a systematized collection of* ODE *solvers*, in Scientific Computing, R. S. Stepleman et al. eds., North-Holland, Amsterdam, 1983, pp. 53–64.

[17] A. C. HINDMARSH AND G. D. BYRNE, *On the use of rank-one updates in the solution of stiff systems of ordinary differential equations*, ACM SIGNUM Newsletter, vol. 11, no. 3 (October 1976), pp. 23–27.

[18] E. S. MARWIL, *Exploiting sparsity in Newton-like methods*, Ph.D. thesis, Cornell University, Ithaca, NY, 1978.

[19] H. MATTHIES AND G. STRANG, *The solution of nonlinear finite-element equations*, Int. J. Numer. Meth. Eng., 14 (1979), pp. 1613–1626.

[20] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[21] L. K. SCHUBERT, *Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian*, Math. Comp., 24 (1970), pp. 27–30.

[22] L. F. SHAMPINE AND C. W. GEAR, *A user's view of solving stiff ordinary differential equations*, SIAM Rev., 21 (1979), pp. 1–17.

[23] PH. L. TOINT, *On sparse and symmetric matrix updating subject to a linear equation*, Math. Comp., 31 (1977), pp. 954–961.

[24] R. A. WILLOUGHBY ed., *Stiff Differential Systems*, Plenum Press, New York, 1974.