UCID-19937

Matrix-Free Methods in the Solution of Stiff Systems of ODE's

Peter N. Brown
Alan C. Hindmarsh

November 1983

Lawrence Livermore Laboratory

Matrix-Free Methods in the Solution of Stiff Systems of ODE's*

Peter N. Brown[1]

Alan C. Hindmarsh[2]

ABSTRACT

This is an informal preliminary report, on a study of a matrix-free method for solving stiff systems of ordinary differential equations (ODE's). In the numerical time integration of stiff ODE initial value problems by BDF methods, the resulting nonlinear algebraic system is usually solved by a modified Newton method and an appropriate linear system algorithm. In place of that, we substitute Newton's method (unmodified) coupled with an iterative linear system method. The latter is a projection method called the Incomplete Orthogonalization Method (IOM), developed mainly by Y. Saad. A form of IOM, with scaling included to enhance robustness, is studied in the setting of Inexact Newton Methods, and implemented in a way that requires no matrix storage whatever. Tests on several stiff problems, of sizes up to 5000, show the method to be quite effective, and much more economical, in both computational cost and storage, than standard solution methods.

*This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-Eng-48, and supported by the DOE Office of Basic Energy Sciences, Mathematical Sciences Branch.

[1]Mathematics Department, University of Houston, Houston, TX 77004

[2]Mathematics and Statistics Division, LLNL

1. Introduction

The numerical solution of large stiff systems of nonlinear ordinary differential equations (ODE's) requires methods for solving systems of nonlinear algebraic equations. A Newton-like method is often employed to solve the algebraic equations, and this generates a sequence of linear systems of equations to solve at each time step. The solution of these linear systems often makes up most of the work involved in the integration of the ODE's. If direct methods are used to solve the linear systems, then most of the required core memory is for the storage of the Jacobian matrix.

We consider here what we feel is best called a matrix-free method for the solution of these linear systems. In [5], Gear and Saad proposed the use of a Krylov-space projection method known as the Incomplete Orthogonalization Method, or IOM. The IOM algorithm is an iterative method for the solution of a linear system, and as will be seen below, does not require the storage of the coefficient matrix in any form. Its usefulness comes from the fact that the convergence of the sequence of approximations to the solution of the linear system is fastest in the dominant subspace (i.e., in those components corresponding to the eigenvalues in the outermost part of the spectrum of the coefficient matrix). This is a desirable feature in the context of solving stiff systems of ODE's by multistep or multivalue methods, since normally the predicted value of the solution at a time step (which is given by an explicit method) has its largest errors in the stiff components. These errors are then damped out in the first few Newton corrections since the IOM algorithm solves for these components the most accurately. See Saad [6], and Gear and Saad [5] for a more detailed discussion of the convergence properties of the IOM algorithm.

In [5], Gear and Saad discuss the implementation of the IOM algorithm into the Newton iteration and present some preliminary results. There, however, no analysis of the resulting Newton-like iteration scheme is presented. Here we view the incorporation of the IOM algorithm into the Newton iteration as an Inexact Newton Method, a class of methods in which the linear systems in the Newton iteration is solved only approximately. The advantage of this viewpoint is that it lends a theoretical base upon which to build the incorporation, at the same time suggesting how that incorporation might be done. For a discussion of Inexact Newton Methods, see Dembo, Eisenstat and Steihaug [1]. We prove below a generalization of a result in [1], relaxing a condition there on the norm of the residuals in the approximate solutions of the linear systems. This result will provide a partial basis for the modified IOM algorithm presented later.

This is an informal preliminary report. The remainder is organized as follows: In Section 2, we introduce the ODE system and algebraic systems to be solved, give some background on Inexact Newton Methods, and prove the above mentioned result. Then in Section 3, we give some background on and introduce a scaled version of the IOM algorithm, denoted SIOM. The implementation of the SIOM algorithm into the general purpose ODE solver LSODE [3] is discussed in Section 4, and test results on three problems are reported in Section 5. Finally, in Section 6 we conclude with a discussion of our results and some suggestions for future work.

2. Stiff ODE Systems and Inexact Newton Methods

We consider here the numerical solution of the ODE Initial Value Problem

(2.1) $\dot{y} = f(t,y),\ y(t_o) = y_o,$

where the dot denotes d/dt and y is a vector of length N. The ODE in (2.1) is assumed to be stiff, and we will use the popular BDF (Backward Differentiation Formula) methods to solve it. These methods have the general form

(2.2) $y_n = \sum_{j=1}^{q} \alpha_j y_{n-j} + h\beta_o \dot{y}_n, \quad \dot{y}_n \doteq f(t_n,y_n)$,

where q is the method order. The BDF methods are implicit and hence at each time step one must solve an algebraic system

(2.3)
$$0 = F_n(y_n) \equiv y_n - h\beta_o f(t_n,y_n) - a_n ,$$
$$a_n \equiv \sum_{j=1}^{q} \alpha_j y_{n-j}, \quad \beta_o > 0 .$$

In many cases, (2.3) is solved using a modified Newton itertion scheme, where a prediction $y_n(0)$ is formed explicitly, with corrections given by

(2.4)
$$P_n s_n(m) = -F_n(y_n(m)) ,$$
$$y_n(m+1) = y_n(m) + s_n(m) .$$

Here P_n is an approximation to the Newton matrix $\partial F_n/\partial y_n = I - h\beta_o J$, where $J = J(t_n,y_n) = \partial f/\partial y(t_n,y_n)$. P_n is held fixed for the iterations and also held fixed over several time steps. The system (2.4) is typically solved by performing 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 J.

The particular class of problems of interest here is such that most of the work required for the integration is in performing the linear algebra associated with (2.4). Furthermore, much of the core memory needed in the integration is used for the storage of the matrix P_n and its LU factorization. (The latter is usually overwritten on P_n.) For large problems in which the number of unknowns is on the order of several thousand or more, storage considerations may be prohibitive on many computers. Thus, for this class of problems, any method which can approximately solve the system in (2.4) and also reduce the core memory required, merits investigation. The Incomplete Orthogonalization Method (IOM) proposed by Gear and Saad [5] is such a method, which we will view here from a somewhat different perspective than that in [5].

The IOM algorithm itself is a method (described in more detail below) for the approximate solution of a linear system of equations

$$Ax = b ,$$

where A is an N x N matrix, and x and b are N-vectors. The use of the IOM algorithm in the solution of the nonlinear system (2.3) gives rise to a method for solving (2.3) which is more properly viewed as an Inexact Newton Method in which the step of solving the system

$$P_n \, s_n(m) = -F_n(y_n(m))$$

is replaced by one of approximately solving the system

$$(2.5) \qquad F_n'(y_n(m)) \, s_n(m) = -F_n(y_n(m)) \;, \quad F_n' = \partial F_n/\partial y_n \;,$$

which is the linear system one obtains if (unmodified) Newton's method is used to solve (2.3).

Consider a general algebraic system problem, $F(y) = 0$, for which the exact Newton step s is given by

$$F'(y)\, s = -F(y).$$

From Dembo, Eisenstat and Steihaug [1], an Inexact Newton Method for this problem has the following general form:

```
Set $y(0)$ = an initial guess
For $m = 0,1,2,\ldots$ until convergence:
   Find (in some unspecified manner) a vector $s(m)$ satisfying
(2.6)   $F'(y(m))\, s(m) = -F(y(m)) + r(m)$
   Set  $y(m+1) = y(m) + s(m)$ .
```

The <u>residual</u> $r(m)$ represents the amount by which $s(m)$ fails to satisfy the Newton equation (2.5). It is not generally known in advance, but is the result of some inner algorithm which attempts to solve (2.5) exactly but does not. In order to guarantee convergence of the scheme, one must demand some auxiliary conditions on the residual $r(m)$. In [1], it is shown that if

$$\|r(m)\| \leq \eta_m \|F(y(m))\| \;,\; m = 0,1,2,\ldots \;, \tag{2.7}$$

where $0 \leq \eta_m \leq \eta_{max} < 1$, then the sequence of iterates $\{y(m)\}$ converges to a true solution of $F(y) = 0$ linearly, given that the initial guess $y(0)$ is close enough. Here $\|\cdot\|$ stands for any norm on R^N.

For the present context, where actual convergence of the iterates is not necessary, and the cost of obtaining them is high, the condition (2.7) is overly restrictive. Thus it is of interest to find out how much one can relax (2.7) and still obtain enough accuracy in the approximate solution $y(m)$ to y_n. The following is a result along these lines.

Theorem 2.1: Let $F: R^N \to R^N$ be a mapping such that

(2.8) there exists a y^* in R^N with $F(y^*) = 0$;

(2.9) F is continuously differentiable in a neighborhood of y^*; and

(2.10) $F'(y^*)$ is nonsingular.

Let $y(m)$ be a sequence generated by an Inexact Newton Method: $y(0)$ is an initial guess and for $m = 0,1,2,\ldots$, we have

(2.11) $F'(y(m))\, s(m) = F(y(m)) + r(m)$, $y(m+1) = y(m) + s(m)$.

Then there are positive constants ε_0 and K (depending only on F) with the following property: For any positive numbers $\varepsilon \le \varepsilon_0$ and $\delta \le K\varepsilon$, whenever $\|y(0) - y^*\| \le \varepsilon$, and the residuals $r(m)$ in (2.11) satisfy $\|r(m)\| \le \delta$ for all m, then all the iterates $y(m)$ exist and satisfy $\|y(m) - y^*\| \le \varepsilon$ and

$$\limsup_{m \to \infty} \|y(m) - y^*\| \le K^{-1}\delta .$$

Proof: First define $\mu = \|F'(y^*)\|$ and $\lambda = \|F'(y^*)^{-1}\|$, and a new norm $\|y\|_* = \|F'(y^*)y\|$. Then we have

(2.12) $\lambda^{-1}\|y\| \le \|y\|_* \le \mu\|y\|$ for all y in R^N.

Let $\gamma > 0$ be chosen such that

$$b \equiv 2\gamma\lambda(1 + \mu\gamma) < 1.$$

Next observe that by (2.8) - (2.10), there exists $\varepsilon_0 > 0$ so that $\|y - y^*\| \le \varepsilon_0$ implies

(2.13) $\|F'(y) - F'(y^*)\| \le \gamma$,

(2.14) $\|F'(y)^{-1} - F'(y^*)^{-1}\| \le \gamma$, and

(2.15) $\|F(y) - F(y^*) - F'(y^*)(y - y^*)\| \le \gamma \|y - y^*\|$.

Next, for any y with $\|y - y^*\| \le \varepsilon_0$, consider the vector given by

$y^+ = y + s$, with $F'(y)\, s = -F(y) + r$,

where nothing is specified as yet about the size of the residual r. We have the following identity:

$$F'(y^*)(y^+ - y^*) = \{I + F'(y^*)[F'(y)^{-1} - F'(y^*)^{-1}]\}$$
$$\cdot\{r + [F'(y) - F'(y^*)](y - y^*) - [F(y) - F(y^*) - F'(y^*)(y - y^*)]\} .$$

Taking norms and using (2.12) - (2.15), we then obtain

$$\|y^+ - y^*\|_* \leq [1 + \|F'(y^*)\|\cdot\|F'(y)^{-1} - F'(y^*)^{-1}\|]$$
$$\cdot[\|r\| + \|F'(y) - F'(y^*)\|\cdot\|y - y^*\| + \|F(y) - F(y^*) - F'(y^*)(y - y^*)\|]$$
$$\leq (1 + \mu\gamma)(\|r\| + 2\gamma\|y - y^*\|) ,$$

$$\|y^+ - y^*\| \leq \lambda(1 + \mu\gamma)(\|r\| + 2\gamma\|y - y^*\|) ,$$

or

(2.16) $\|y^+ - y^*\| \leq \lambda(1 + \mu\gamma)\|r\| + b\|y - y^*\|$.

We now want to insure that when $\|y - y^*\| \leq \varepsilon_0$ and $\|r\| \leq K\varepsilon$ (for some constant K), then $\|y^+ - y^*\| \leq \varepsilon$ also. From (2.16), it is clear that we will achieve that end if

$$\lambda(1 + \mu\gamma)K\varepsilon + b\varepsilon \leq \varepsilon ,$$

and so we define K by

$$K = (1 - b)/[\lambda(1 + \mu\gamma)] .$$

Now for any positive $\varepsilon \leq \varepsilon_0$, and any y(0) with $\|y(0) - y^*\| \leq \varepsilon$, a sequence of inexact Newton iterates y(m) (m = 1,2,...) satisfying (2.11) with $\|r(m)\| \leq K\varepsilon$ is guaranteed to exist, and for each m, $\|y(m) - y^*\| \leq \varepsilon$.

We can say more about the norms of the errors if we suppose further that $\|r(m)\| \leq \delta$ for some constant δ with $\delta \leq K\varepsilon$. Then, setting

$$a = \lambda(1 + \mu\gamma)\delta ,$$

(2.16) gives

$$\|y(m+1) - y^*\| \leq a + b\|y(m) - y^*\| ,$$

and by induction we obtain

$$\|y(m) - y^*\| \leq a(1 + b + b^2 + \dots b^{m-1}) + b^m\|y(0) - y^*\| .$$

Thus, since $b < 1$,

$$\|y(m) - y^*\| \le a/(1 - b) + b^m \varepsilon ,$$

and finally

$$\limsup_{m \to \infty} \|y(m) - y^*\| \le a/(1 - b) = K^{-1}\delta .$$

QED

Suppose we consider the following Inexact Newton Algorithm:

```
Given $\varepsilon > 0$ and $\delta > 0$, and an initial guess $y(0)$,
    1. Find (in some unspecified manner) $s(m)$ satisfying
           $F'(y(m))s(m) = -F(y(m)) + r(m)$ ,  $\|r(m)\| \le \delta$ .
    2. Set  $y(m+1) = y(m) + s(m)$ .
    3. If $\|F(y(m+1))\| \le \varepsilon$, then stop; otherwise set $m \leftarrow m+1$ and
           go to Step 1.
```

Theorem 2.1 indicates that one should choose $\delta \ll \varepsilon$ in order to guarantee the accuracy of the approximation $y(m)$ for m large. Further, from the proof we have

$$\|y(m+1) - y^*\| \le a + b\|y(m) - y^*\|$$

with $0 < b < 1$ and a proportional to δ. This last inequality implies that when

(2.17) $\qquad a / \|y(m)-y^*\| \ll b$,

then we have

(2.18) $\qquad \|y(m+1) - y^*\| \le \left(\frac{a}{\|y(m)-y^*\|} + b\right) \|y(m)-y^*\| \approx b\|y(m) -y^*\|$.

Thus when (2.17) holds, the sequence is converging almost linearly. This means that the smaller δ is the faster the "convergence" of the sequence $\{y(m)\}$ in the sense that (2.18) holds for more iterates.

3. The Incomplete Orthogonalization Method

The Incomplete Orthogonalization Method (IOM) given in [6] is an algorithm for the approximate solution of the linear system

(3.1) $Ax = b$

where A is an $N \times N$ matrix and x and b are N-vectors. A brief description of the method is given below along with some preliminaries, and the reader is referred to [6] for more details.

By a projection method on the subspace $K_\ell = \text{span}\ \{V_\ell\}$, where $V_\ell = [v_1,\dots,v_\ell]$ is an orthonormal system in R^N, we mean a method which finds an approximate solution x_ℓ of (3.1) by requiring that

(3.2)
$$\begin{aligned} &x_\ell \in K_\ell\ , \\ &(Ax_\ell - b) \perp v_j \quad (j = 1,2,\dots,\ell)\ . \end{aligned}$$

Different choices of the subspace give rise to different projection methods. Let x_0 be an initial guess of the solution x^* of (3.1) and set K_ℓ equal to the Krylov subspace

$$K_\ell = \text{span}\ \{\ r_0,\ Ar_0,\dots,\ A^{\ell-1}r_0\ \}\ ,$$

where $r_0 = b - Ax_0$. Letting $x = x_0 + z$, then z must satisfy

(3.3) $Az = r_0$.

The Krylov subspace projection method then finds an approximate solution z_ℓ of the true solution z^* of (3.3) by requiring that

$$\begin{aligned} &z_\ell \in K_\ell \\ &(Az_\ell - r_0) \perp v_j \quad (j = 1,\dots,\ell)\ , \end{aligned}$$

where $V_\ell = [v_1,\dots,v_\ell]$ is an orthonormal basis of K_ℓ. (V_ℓ also denotes the $N \times \ell$ matrix with columns v_i.) Letting $z_\ell = V_\ell y_\ell$ with $y_\ell \in R^\ell$, we see immediately that y_ℓ must be the solution of the linear system

$$V_\ell^T A V_\ell \cdot y_\ell - V_\ell^T r_0 = 0 \ ,$$

and so

$$x_\ell = x_0 + z_\ell$$

becomes

$$x_\ell = x_0 + V_\ell (V_\ell^T A V_\ell)^{-1} V_\ell^T r_0 \quad . \tag{3.4}$$

It is assumed throughout that the vectors $r_0, Ar_0, \ldots, A^{\ell-1} r_0$ are linearly independent so that the dimension of K_ℓ is ℓ.

We next present an algorithm given by Saad [6], which is an adaptation of an earlier one due to Arnoldi. It constructs an orthonormal basis $V_\ell = [v_1, \ldots, v_\ell]$ of K_ℓ such that $V_\ell^T A V_\ell$ has Hessenberg form:

```
1. Compute $r_0 = b - Ax_0$ and set $v_1 = r_0/\|r_0\|$ .
2. For $j = 1,2,\ldots,\ell$ do:
        $w_{j+1} = Av_j - \sum_{i=1}^{j} h_{ij} v_i \ , \ h_{ij} = (Av_j, v_i)$
        $h_{j+1,j} = \|w_{j+1}\|$
        $v_{j+1} = w_{j+1}/h_{j+1,j}$ .
```

Here $(\cdot,\cdot)$ is the Euclidean inner product and $\|\cdot\|$ the Euclidean norm. Saad (cf. [6]) has shown that $[v_1, \ldots, v_\ell]$ is an orthonormal basis for K_ℓ and that the matrix $V_\ell^T A V_\ell$ is the Hessenberg matrix H_ℓ whose nonzero elements are the h_{ij} defined in the above algorithm. It then follows that the vector $V_\ell^T r_0$ in (3.4) is equal to $\beta V_\ell^T v_1 = \beta e_1$, where $\beta = \|r_0\|$, and $e_1 = (1,0,\ldots,0)^T \in R^\ell$. Therefore, the approximation x_ℓ is given by

$$x_\ell = x_0 + \beta\, V_\ell\, H_\ell^{-1}\, e_1 \ .$$

One practical consideration is the choice of ℓ. A very useful identity in choosing ℓ is the following equation for the residual norm:

(3.5) $\quad \|b - Ax_\ell\| = h_{\ell+1,\ell}\, |e_\ell^T y_\ell|$.

The relation (3.5) follows from the relation

$$AV_\ell = V_\ell H_\ell + h_{\ell+1,\ell} v_{\ell+1} e_\ell^T ,$$

which can be derived from the algorithm. An interesting feature of the relation (3.5) is that one does not have to form x_ℓ or y_ℓ in order to compute $\|b - Ax_\ell\|$. If we perform an LU factorization of H_ℓ, writing $H_\ell = LU$, and assume that no pivoting was necessary, then it can be shown that

$$h_{\ell+1,\ell}\, |e_\ell^T y_\ell| = h_{\ell+1,\ell}\, \beta \left| \left(\prod_{i=1}^{\ell-1} \ell_i\right)/u_{\ell\ell} \right| ,$$

where the ℓ_i $(i=1,\dots,\ell-1)$ are the successive pivots (the subdiagonal elements of L). In general, one can show that when no pivoting has been necessary for $i \in I$, where $I \subset \{1,\dots,\ell-1\}$, then

(3.6) $$h_{\ell+1,\ell}\, |e_\ell^T y_\ell| = h_{\ell+1,\ell}\, \beta \left| \left(\prod_{i\in I} \ell_i\right)/u_{\ell\ell} \right| .$$

See [6] for more details.

The use of (3.5) to estimate the error $\|b-Ax_\ell\|$ then leads to the following algorithm:

Algorithm 3.1 (Arnoldi's Algorithm):

```
1. Compute $r_0 = b - Ax_0$ and set $v_1 = r_0/\|r_0\|$ .
2. For $\ell = 1,2,\dots,\ell_{max}$ do:
```

```
(a)  $w_{\ell+1} = Av_\ell - \sum_{i=1}^{\ell} h_{i\ell}v_i \ , \ h_{i\ell} = (Av_\ell, v_i)$
     $h_{\ell+1,\ell} = \|w_{\ell+1}\|$
     $v_{\ell+1} = w_{\ell+1}/h_{\ell+1,\ell}$
(b)  Update the LU factorization of $H_\ell$ .
(c)  Use (3.6) to compute $\rho_\ell = h_{\ell+1,\ell}|e_\ell^T y_\ell| = \|b - Ax_\ell\|$ .
(d)  If $\rho_\ell < \delta$, go to Step 3.  Otherwise, go to (a).

3.  Compute $x_\ell = x_0 + \|r_0\| V_\ell H_\ell^{-1} e_1$ and stop.
```

In the above algorithm, if the test on ρ_ℓ fails, and if $\ell = \ell_{max}$ steps have been performed, then one has the option of either accepting the final approximation x_ℓ, or setting $x_0 = x_\ell$ and then going back to step 1 of the algorithm. This last procedure has the effect of "restarting" the algorithm. We also note that due to the upper Hessenberg form of H_ℓ, there is a convenient way to perform an LU factorization of H_ℓ by using the LU factors of $H_{\ell-1}$.

In the solution of the stiff ODE problem, the nonlinear problem (2.3) has the form

(3.7) $\quad F(y) \equiv y - h\beta_0 f(t,y) - a = 0$,

where a is a constant N-vector, and so

$$F'(y) = I - h\beta_0 J(t,y) \ , \ J(t,y) = \frac{\partial f}{\partial y}(t,y) \ .$$

Since the problem (2.1) is stiff, the Jacobian J has at least one eigenvalue with large negative real part. In the nonlinear iteration for solving (3.7), we then want to consider solving

$$F'(y(m))\, s(m) = -F(y(m))$$

by the IOM algorithm, and then set

$$y(m+1) = y(m) + s(m) \ .$$

The predicted value y(0) is obtained using an explicit method and hence will have its largest errors in the stiff components. In the solution of a linear system $Ax = b$ using Algorithm 3.1, Saad [6] has shown that convergence of the iterates x_ℓ is fastest in the dominant subspace. Thus, the convergence of the iterates $x_\ell(m)$ to $s(m)$ is fastest in the stiff components. This result also suggests that a relatively small value of ℓ_{max} may suffice in Algorithm 3.1. In the testing reported below, we used $\ell_{max} = 5$.

In Algorithm 3.1, for ℓ close to ℓ_{max}, a considerable amount of the work involved is in making the vector $v_{\ell+1}$ orthogonal to all the previous vectors $v_1,\dots,v_\ell$. Saad [6] has proposed a modification cf Algorithm 3.1 in which the vector $v_{\ell+1}$ is only required to be orthogonal to the previous p vectors, $v_{\ell-p+1}, \dots, v_\ell$. Eqns. (3.5) and (3.6) then do not hold excactly, but are still useful as estimates of the residual norm. This leads to the following algorithm, denoted <u>IOM</u>:

<u>Algorithm 3.2 (Incomplete Orthogonalization Method)</u>:

```
1.  Compute $r_0 = b - Ax_0$ and set $v_1 = r_0/\|r_0\|$ .
2.  For $\ell = 1,2,\dots\ell_{max}$ do:
    (a)  $w_{\ell+1} = Av_\ell - \sum_{i=i_0}^{\ell} h_{i\ell}v_i$ , $h_{i\ell} = (Av_\ell, v_i)$ ,
             $i_0 = \max(1, \ell-p+1)$ ,
         $h_{\ell+1,\ell} = \|w_{\ell+1}\|$ ,
         $v_{\ell+1} = w_{\ell+1}/h_{\ell+1,\ell}$ .
    (b)  Update the LU factorization of $H_\ell$.
    (c)  Use (3.6) to compute $\rho_\ell = h_{\ell+1,\ell}|e_\ell^T y_\ell| \simeq \|b - Ax_\ell\|$ .
    (d)  If $\rho_\ell < \delta$, go to Step 3. Otherwise, go to (a).
3.  Compute $x_\ell = x_0 + \|r_0\|V_\ell H_\ell^{-1} e_1$ and stop.
```

The remarks made after Algorithm 3.1 are also applicable here. In [6], Saad compares Algorithms 3.1 and 3.2 on several test problems, and reports that Algorithm 3.2 is sometimes preferred, based on total work required and run times.

In the testing described below, we used <u>scaled</u> versions of Algorithms 3.1 and 3.2. Suppose that instead of the linear system $Ax = b$ we want to solve an equivalent problem that is more well-scaled,

$$\tilde{A}\,\tilde{x} = \tilde{b}$$

where $\tilde{A} = D^{-1}AD$, $\tilde{x} = D^{-1}x$, $\tilde{b} = D^{-1}b$, and D is a diagonal scaling matrix related to the tolerance parameters in the ODE problem (2.1). In Algorithm 3.2 we then have

$$\tilde{w}_{\ell+1} = \tilde{A}\tilde{v}_\ell - \sum_{i=i_0}^{\ell} \tilde{h}_{i\ell}\,\tilde{v}_i \ , \quad \tilde{h}_{i\ell} = (\tilde{A}\tilde{v}_\ell, \tilde{v}_i)$$

$$\tilde{h}_{\ell+1,\ell} = \|\tilde{w}_{\ell+1}\|$$

$$\tilde{v}_{\ell+1} = \tilde{w}_{\ell+1}/\,\tilde{h}_{\ell+1,\ell} \quad .$$

If we define the vectors v_i and w_i by

$$v_i = D\tilde{v}_i \quad \text{and} \quad w_i = D\tilde{w}_i \ ,$$

then we have

$$D^{-1}w_{\ell+1} = D^{-1}ADD^{-1}v_\ell - \sum_{i=i_0}^{\ell} \tilde{h}_{i\ell}\,D^{-1}v_i$$

or

$$w_{\ell+1} = Av_\ell - \sum_{i=i_0}^{\ell} \tilde{h}_{i\ell}v_i \ ,$$

where $\tilde{h}_{i\ell} = (D^{-1}Av_\ell,\ D^{-1}v_i)$, and

$$\tilde{h}_{\ell+1,\ell} = \|D^{-1}\tilde{w}_{\ell+1}\| \ , \ v_{\ell+1} = w_{\ell+1}/\tilde{h}_{\ell+1,\ell} \quad .$$

The matrix H_ℓ is now replaced by $\tilde{H}_\ell = (\tilde{h}_{i,j})$. Hence, the effect of the scaling matrix D is to introduce the use of a weighted inner product and associated norm into the algorithm. The result is the following algorithm, denoted <u>SIOM</u>:

<u>Algorithm 3.3 (Scaled IOM)</u>:

```
1.  Compute $r_0 = b - Ax_0$ and set $v_1 = r_0/\|D^{-1}r_0\|$ .
2.  For $\ell = 1,2,\ldots,\ell_{max}$ do:
    (a)  $w_{\ell+1} = Av_\ell - \sum_{i=i_0}^{\ell} \tilde{h}_{i\ell}v_i$ , $\tilde{h}_{i\ell} = (D^{-1}Av_\ell, D^{-1}v_i)$ ,
         $i_0 = \max(1,\ell-p+1)$
         $\tilde{h}_{\ell+1,\ell} = \|D^{-1}w_{\ell+1}\|$ ,
         $v_{\ell+1} = w_{\ell+1}/\tilde{h}_{\ell+1,\ell}$ .
    (b)  Update the LU factorization of $\tilde{H}_\ell$ .
    (c)  Use (3.6) to compute $\rho_\ell \simeq \|D^{-1}(b - Ax_\ell)\|$ .
    (d)  If $\rho_\ell < \delta$, go to Step 3. Otherwise, go to (a).
3.  Compute $x_\ell = x_0 + \|D^{-1}r_0\| V_\ell \tilde{H}_\ell^{-1}e_1$ , and stop.
```

We remark that the scaling matrix D is determined automatically by LSODE from user-supplied tolerances, and that it can change from one time step to the next. As indicated earlier, the vector $\tilde{H}_\ell^{-1}e_1$ is computed by generating an LU decomposition of $\tilde{H}_\ell$ (by successive updating as ℓ

varies), followed by back-substitution. Here $\tilde{H}_\ell$ is treated as a _general_ Hessenberg matrix, even though it is actually banded (with a lower half-bandwidth of p-1), because its size is too small to gain any efficiency advantage from the band structure.

4. Algorithmic Implementation

The LSODE package [3] was modified to perform the solution of the linear system (2.5) using the IOM algorithm. 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.

4.1 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 which approximates $I-h\beta_0 J$, and subsequently calls SOLSY to solve the linear system (2.4). Both of these routines call LINPACK routines [2] to do the matrix operations.

Within STODE, the basic algorithm for time 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 $m \geq 1$.

(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 Step (5).

   (c) If $m = 3$ and J is current, set $h \leftarrow h/4$ and go to Step (1) (redo time step).

   (d) If $m = 3$ and J is not current, set flag to reevaluate J and go to Step (3) (redo time 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 $m \geq 1$, to form the ratio 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 Sec. 3.) The convergence test in step (8) requires the product $\|s_n(m)\| \cdot \min(1, 1.5C)$ to be less than a constant which depends only on q. This is based on linear convergence, with

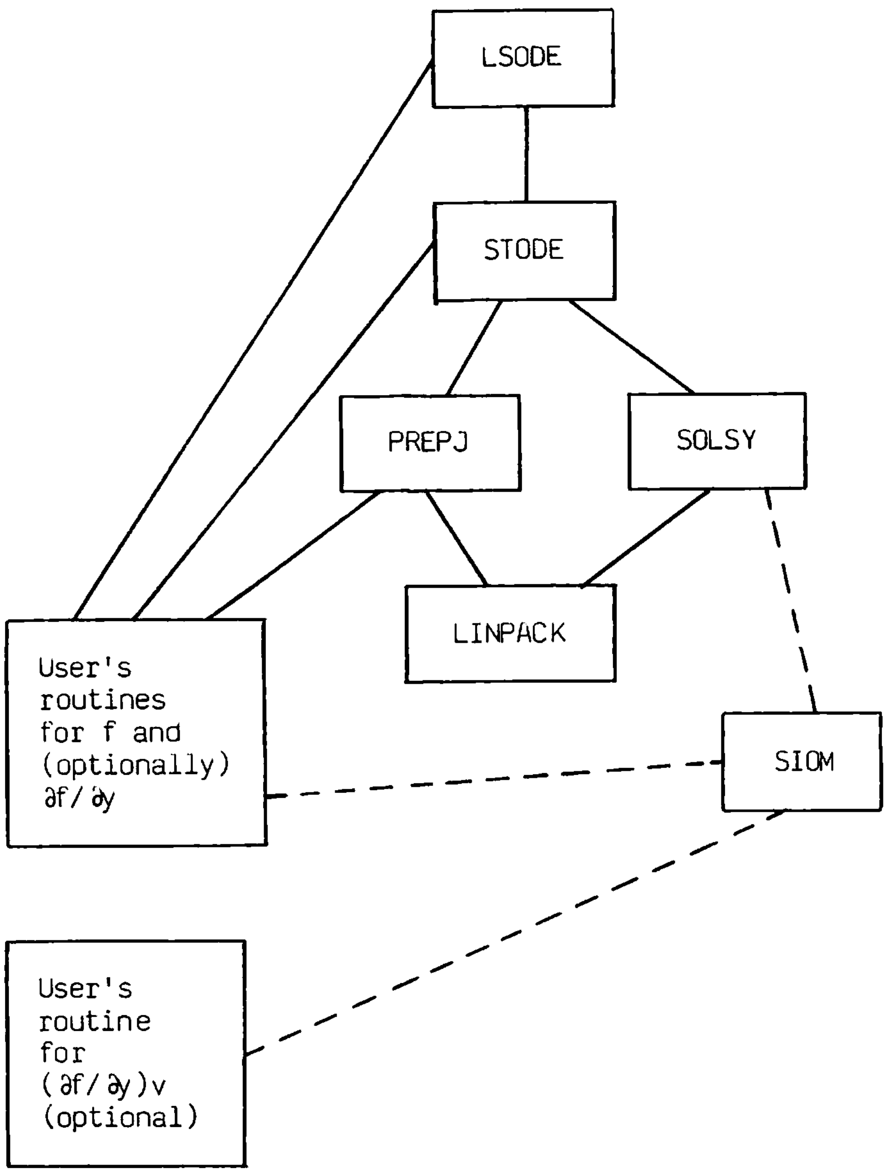
LSODE
STODE
PREPJ
SOLSY
LINPACK
SIOM
User's routines for f and (optionally) ∂f/∂y
User's routine for (∂f/∂y)v (optional)

Fig. 1. Simplified overall structure of the LSODE package.

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.

4.2 The Modified Algorithm

In the modified algorithm, subroutine SOLSY calls a driver routine SIOM which performs the solution of the linear system (2.5) using the scaled version of the IOM algorithm given in Algorithm 3.3. Subroutine SIOM then calls user-supplied routines for evaluating f and the operation of the Jacobian J times a vector v, as indicated by the dashed lines in Fig. 1. It also performs the looping and convergence test for the SIOM iterations. We describe below the essential features of our implementation of the SIOM algorithm into the LSODE package.

First, we note that in Algorithm 3.3 the matrix A is not needed explicitly, and only the action of A times a vector v is necessary. Since

$$A = F_n'(y_n(m)) = I - h\beta_0 J \text{ with } J = \frac{\partial f}{\partial y}(t_n, y_n(m)),$$

we can approximate Av by first using the difference quotient

(4.1) $\quad Jv \simeq w = [f(t_n, y+\sigma v) - f(t_n, y)]/\sigma$

(where y denotes $y_n(m)$) for a suitably chosen scalar σ, and then setting

$$Av \simeq v - h\beta_0 w \ .$$

Note that if $f(t_n, y)$ has been saved, then this only requires one additional f evaluation.

The choice of σ is limited by roundoff error if σ is too small, in that the two values f in (4.1) may be numerically equal in some components while the true components of Jv may not be zero. Also, σ is limited by truncation error if σ is too large, in that f may be nonlinear between y and $y + \sigma v$, and the difference quotient (4.1) may be inaccurate as a result. In

Algorithm 3.3, Jv is needed for vectors v which are normalized according to

$$\sqrt{N}\,\|v\|_{WRMS} = \|D^{-1}v\| = 1 ,$$

where $\|\cdot\|_{WRMS}$ is the weighted root-mean-square norm, $\|\cdot\|$ is the Euclidean norm, and $D = \mathrm{diag}(d_1,\ldots,d_N)$ is the scaling matrix given by

$$d_i = |y_{n-1,i}|\cdot RTOL_i + ATOL_i \quad (i=1,\ldots,N) .$$

$RTOL_i$ and $ATOL_i$ are user-defined tolerances. In the scaled space, we have $\tilde{y} = D^{-1}y$ and $\tilde{v} = D^{-1}v$, and we can form the equivalent problem in terms of

$$\tilde{f}(t_n,\tilde{y}) = D^{-1}f(t_n,D\tilde{y}) \text{ and } \tilde{J} = \frac{\partial\tilde{f}}{\partial\tilde{y}} = D^{-1}JD .$$

The corresponding value for $\tilde{J}\tilde{v}$ is

$$\begin{aligned}\tilde{w} &= [\tilde{f}(t_n,\tilde{y}+\sigma\tilde{v}) - \tilde{f}(t_n,\tilde{y})]/\sigma \\ &= [D^{-1}f(t_n,y+\sigma v) - D^{-1}f(t_n,y)]/\sigma \\ &= D^{-1}w .\end{aligned}$$

Note that $\|\tilde{v}\| = \|D^{-1}v\| = 1$ here.

The test on local error in LSODE requires that the local error vector e satisfy $\|e\|_{WRMS} \leq 1$, and we can expect $\|e\|_{WRMS} \simeq 1$ for the step sizes selected. This means that $\tilde{e} = D^{-1}e$ satisfies

$$\|\tilde{e}\| = \sqrt{N}\,\|\tilde{e}\|_{RMS} = \sqrt{N}\,\|e\|_{WRMS} \simeq \sqrt{N} .$$

The vector $\tilde{e}$ can be regarded as a small correction to $\tilde{y}$ and so its size is about at the user's tolerance level. So it is likely that f is reasonably linear between $\tilde{y}$ and $\tilde{y} + \tilde{e}$, but unlikely that $\tilde{e}$ is so small as to make $\tilde{f}$ hard to resolve due to roundoff between $\tilde{y}$ and $\tilde{y} + \tilde{e}$. Therefore, a reasonable criterion on σ is to make $\sigma\tilde{v}$ and $\tilde{e}$ have the same norm (Euclidean norm).

This leads to

$$\sigma\|\tilde{v}\| = \|e\| \approx \sqrt{N} ,$$

or

$$\sigma \approx \sqrt{N} .$$

As an alternative to using (4.1), it is easy to give the user the option of supplying his own routine for the computation of Jv. This has been done in the experimental version of LSODE containing SIOM.

We elected to implement Algorithm 3.3 without the restart feature. This was due mostly to storage considerations and simplicity. The convergence parameter $\delta = .05$ was chosen so as to ensure that the errors in the solution $s_n(m)$ of (2.5) (as measured in the norm $\|D^{-1}\cdot\|$) were much below the requested accuracy in the solution y_n. (To be precise, since we are not using the restart feature of the IOM algorithm, we cannot always guarantee that the errors in $s_n(m)$ are insignificant. However, in the test results given below, these errors were apparently of no consequence.) This choice of δ also ensures essentially linear convergence of the Newton iteration, as was shown in Section 2.

Finally, we note that no nonzero guess to the solution of the linear system (2.5) is readily available, and so in Algorithm 3.3, we take $x_0 = 0$. This means that the convergence test $D^{-1}(b - Ax_\ell) \leq \delta$ is easily applied also for $\ell = 0$. This test is made in Step 1 of Algorithm 3.3, and if it passes, $x_0 = 0$ is accepted as the approximate solution.

The modified version of the basic algorithm in STODE is then as follows:

(1) Predict $y_n(0)$

(2) Compute $f(t_n,y_n(0))$; set $m = 0$, and reset the convergence rate constant C to 0.7.

(3) Form $F_n(y_n(m))$.

(4) Call SOLSY and correct to get $y_n(m+1)$ (by Algorithm 3.3).

(5) Update estimate of convergence rate constant C, if $m \geq 1$.

(6) Test for convergence.

(7) If the convergence test failed:

    (a) set $m \leftarrow m+1$.

    (b) If $m < 3$, compute $f(t_n,y_n(m))$ and go to Step (3).

    (c) If $m = 3$, set $h \leftarrow h/4$ and go to Step (1) (redo time step).

(8) If convergence test passed, update history, do error test, etc.

The comments regarding steps (7) and (10) after the unmodified algorithm are still relevant for steps (6) and (8) here, respectively. The user has the option to specify the size of the parameters p and ℓ_{max} in Algorithm 3.3 (which affects the size of the core memory required). In the limited testing reported below, these parameters were both taken equal to 5.

4.3 Algorithmic Variations

A number of variations on the above algorithm using SIOM are possible, and several are given in [5] and [6]. We tested some such variations, where it seemed they might result in improvements in performance. Although each has a plausible argument in its favor, none of them resulted in any actual gains in efficiency. Nevertheless, for the sake of completeness, we describe here the variations that were tried.

First, it is well known that the Gram-Schmidt orthogonalization process, represented by Step 2(a) in Algorithm 3.1, can potentially produce bad answers because of roundoff error. To avoid this without a major redesign of that part of the algorithm, we simply substituted double precision for single precision (on the Cray-1 computer) in the computation of the scaled inner products $(D^{-1}Av_{\ell}, D^{-1}Av_i)$ in Algorithm 3.3 and also in the computation of the $w_{\ell+1}$ there. In the tests done, the numerical results were no different, suggesting strongly that roundoff had in fact not been a problem in single precision, while the run times were much larger, by as much as a factor of 3.

Once a set of basis vectors v_i is computed on any given Newton iteration, within any given time step, various possibilities come to mind for reusing this basis in later Newton iterations. The considerable effort required to produce the vectors and the corresponding Hessenberg matrix $\tilde{H}_{\ell}$ would then be avoided some of the time. The most immediate idea is to save the basis from the first Newton iteration on a given step and reuse it on the subsequent Newton iterations in the same step. However, it is not hard to see that this will be of little or no benefit, for in the limiting case that $f(t,y)$ is linear in y, this strategy (with $x_0 = 0$) produces a correction of <u>zero</u> on the second and subsequent iterations, because the residual vector is already orthogonal to the basis vectors.

An idea that does make sense is to save a basis computed on one time step and reuse it on subsequent steps. This is motivated by the same fact that explains the success of modified Newton iteration, namely that $A = I - h\beta_0 J$ is subject only to gradual changes from step to step, arising from $J = \partial f/\partial y$, at least as long as the scalar $h\beta_0$ is fixed. The basis $\{v_i\}$ computed in the <u>first</u> Newton correction on step n, producing $y_n(1)$ from $y_n(0)$, is likely to be much more useful on subsequent steps than that from any of the later Newton corrections in step n (if any), because it reflects the error in the prediction $y_n(0)$, which presumably has a larger error than any other $y_n(m)$. This all suggests (and tests confirmed) that the reuse of a basis generated on step n should be considered on step n+1 only when the following conditions all hold:

(a) step n only required one Newton iteration;

(b) the SIOM algorithm on step n produced a basis of $\ell \geq 1$ vectors (i.e. it did not terminate with $x_0 = 0$); and

(c) the relative change in $h\beta_0$ from step n to step n+1 does not exceed a small input parameter η.

In mosts of the tests done, η was set to 0.3. When these conditions hold, a decision is made to use the basis $\{v_i\}$ produced on step n for the first Newton iteration on step n+1, but no others, for the reasons given above. If a second Newton iteration is needed in step n+1, it is done with the SIOM algorithm (giving new basis vectors), as are the Newton iterations in step n+2. The new basis vectors are then overwritten on the old ones, for storage economy. But if step n+1 requires only one Newton correction (with the old basis), and if condition (c) above holds with respect to steps n+1 and n+2, then the first Newton iteration on step n+2 is also done with the saved basis, and so on. We refer to this as an Approximate SIOM algorithm. (In [5], Gear and Saad give a quite different modified IOM algorithm for reusing the v_i, which we did not test.)

In all of our tests with this algorithm, the run times were actually slightly larger than for the original SIOM algorithm. The results suggest that the changes in the matrix A from step to step were sufficiently large that the reduction in basis vector calculations was outweighed by additional Newton iterations.

A second form of this approximate SIOM is one given in [5], and corrects for changes in $h\beta_0$. Here, on any step satisfying the conditions given above for reuse of an old basis, instead of using the old values of the inner products $\tilde{h}_{i\ell}$ in Algorithm 3.3, we adjust these as a function of the relative change in $h\beta_0$, denoted by α. As shown in [5], this change is correctly accounted for by adding $\alpha - 1$ to the diagonal elements, refactoring the adjusted Hessenberg matrix $\tilde{H}_\ell$, and scaling the final solution vector x by α. The tests with this <u>Adjusted Approximate SIOM</u> algorithm also showed no gain in efficiency. The added costs appeared to outweigh the benefits.

5. Numerical Tests

The SIOM algorithm described above, and implemented in a modified version of the LSODE package, was tested on various ODE test problems. In this section we give, for each of three problems, a description of the problem, numerical results obtained, and some discussion. All three test problems are based on time-dependent partial differential equation (PDE) systems, solved by the method of lines. All of the tests were done on a Cray-1 computer with the CFT compiler.

The algorithms tested are the unaltered LSODE package (as discussed in Sec. 4.1), and the version modified to use the SIOM algorithm (the scaled version of the IOM algorithm, as described in Sec. 4.2) to solve the linear systems (2.5) . In all the tests, we used the parameter values $p = \ell_{max} = 5$ and $\delta = 0.05$.

In what follows, we will use the following abbreviations for the various algorithms:

LSODE: unaltered LSODE package

LSIOM: LSODE with the scaled IOM algorithm first method

Unless otherwise stated, the algorithms are as described in detail in Sections 3 and 4.

For each problem and each algorithm, a test run was made and yielded various statistics. Those of interest include the following:

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)
NSIOM	= number of calls to routine SIOM (= number of corrector iterations)
FPSIOM	= (NFE - 1 - NSIOM)/NSIOM = average number of f evaluations per SIOM call in LSIOM

Of course the counter NJE is relevant only to LSODE, while NSIOM and FPSIOM are relevant only to LSIOM. The number FPSIOM is significant in that it indicates on the average how hard the SIOM algorithm must work to achieve convergence. If on a particular problem FPSIOM is very close to the value of the parameter ℓ_{max}, then it may be wise to increase ℓ_{max} to improve the overall accuracy and efficiency for that problem. We will also tabulate the work space, which is the total length of the real and integer work arrays required. The J evaluations in LSODE were in all cases done by a user-supplied subroutine, while the products Jv in the SIOM algorithm were generated using the difference quotient (4.1).

5.1. 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 [4] 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, with vary with altitude z and horizontal position x, both in km, with $0 \leq x \leq 20$, $30 \leq z \leq 50$, and with time t in sec, $0 \leq t \leq 86400$ (one day). These obey a pair of coupled reaction-diffusion equations:

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

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

$$R^1(c^1,c^2,t) = -k_1c^1 - k_2c^1c^2 + k_3(t) \cdot 7.4 \cdot 10^{16} + k_4(t)c^2$$

$$R^2(c^1,c^2,t) = k_1c^1 - k_2c^1c^2 - k_4(t)c^2$$

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

$$k_3(t) = \left\{\begin{array}{ll} \exp[-22.62/\sin(\pi t/43200)] & \text{for } t < 43200 \\ 0 \ \ \text{otherwise} & \end{array}\right\} \ ,$$

$$k_4(t) = \left\{\begin{array}{ll} \exp[-7.601/\sin(\pi t/43200)] & \text{for } t < 43200 \\ 0 \ \ \text{otherwise} & \end{array}\right\} \ ,$$

Homogeneous Neumann boundary conditions are posed:

$\partial c^i/\partial x = 0$ on $x = 0$, $x = 20$,

$\partial c^i/\partial z = 0$ on $z = 30$, $z = 50$.

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

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

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

$$\beta(z) \equiv 1 - (.1z-4)^2 + (.1z-4)^4/2.$$

The PDE's are treated by central differencing, on a rectangular grid with uniform spacings, $\Delta x = 20/(J-1)$ and $\Delta z = 20/(K-1)$. If c^i_{jk} denotes the approximation to $c^i(x_j,z_k,t)$, where $x_j = (j-1)\Delta x$, $z_k = 30 + (k-1)\Delta z$, $1 \le j \le J$, $1 \le k \le K$, then we obtain the following ODE's

$$\dot{c}^i_{jk} = (K_h/\Delta x^2)\ (c^i_{j+1,k} - 2c^i_{jk} + c^i_{j-1,k})$$

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

$$+ R^i(c^1_{jk},c^2_{jk},t)\ .$$

The boundary conditions are simulated by taking

$$c^i_{0,k} = c^i_{2,k},\ c^i_{J+1,k} = c^i_{J-1,k} \quad \text{(all } k\text{), and}$$

$$c^i_{j,0} = c^i_{j,2},\ c^i_{j,K+1} = c^i_{j,K-1} \quad \text{(all } j\text{)}\ .$$

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 $c^i_{jk} = y_m$ with $m = i + 2(j-1) + 2J(k-1)$.

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 = $2J = 40$. A mixed relative/absolute error tolerance was chosen, with RTOL = 10^{-5} and ATOL = 10^{-3}.

The results of testing the two algorithms on this problem are shown in Table 1. In this case, LSIOM shows a dramatic improvement over LSODE, trading 90 Jacobian evaluations (and hence 90 banded LU decompositions) for 680 SIOM calls (and a net increase in f evaluations), resulting in a savings of roughly 14 sec. (67%) of the CPU time. We note also that the work space required was reduced by approximately 87%, and the average number of IOM iterates computed per SIOM call was approximately 1.13. The latter value indicates that the SIOM is not having any difficulty at all achieving convergence here.

TABLE 1. Test results for Problem 1.

Algorithm	R.T.	NST	NFE	NJE	NSIOM	Work Space (words)	FPSIOM
LSODE	21.5	462	659	90	0	106,442	0
LSIOM	7.1	355	1446	0	680	13,675	1.13

5.2 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 variables, $c^1(x,z,t)$ and $c^2(x,z,t)$, representing species densities over the spatial habitat $\Omega = \{(x,z)\colon 0 \le x \le 1,\ 0 \le z \le 1.8\}$ and time t in sec, $0 \le t \le 10$. The equations are

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

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

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

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

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

Homogeneous Neumann boundary conditions are imposed:

$\partial c^i/\partial x = 0$ on $x = 0,\ x = 1;\quad \partial c^i/\partial z = 0$ on $x = 0,\ x = 1.8\ .$

The initial conditions involve products of cosines and are chosen to be 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

$c^1 = c^1_* \equiv 1 - 10^{-6},\quad c^2 = c^2_* \equiv 10^{-6}.$

The two partial differential equations are again treated by central differencing on a rectangular grid with uniform spacings, $\Delta x = 1/(J-1)$ and $\Delta z = 1.8/(K-1)$, and with boundary conditions treated as before. For this test, we chose J = 20 and K = 20, making the system size N = 2JK = 800. The problem is stiff, 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. The savings here are not as dramatic as in the first test problem. However, there is a tradeoff of 66 Jacobian evaluations (and LU decompositions) for 1202 SIOM calls, which does result in a savings of 4.9 sec. (26%) of CPU time. Notice also that for this problem the total number of steps for LSIOM is slightly greater than that for LSODE, and the average number of iterates per SIOM call is approximately 1.53, slightly higher than before. Again, however, the storage requirement is reduced by 87% with LSIOM.

TABLE 2. Test results for Problem 2.

Algorithm	R.T.	NST	NFE	NJE	NSIOM	Work Space (words)	FPSIOM
LSODE	18.6	582	665	66	0	106,442	0
LSIOM	13.7	617	3040	0	1202	13,675	1.53

5.3 Test Problem 3

Like Problem 2, this problem is based on a two-dimensional reaction-diffusion system comprising a Lotka-Volterra predator-prey model. The two variables, c^1 and c^2, represent the prey and predator species densities, and vary over the habitat $\Omega = \{(x,z): 0 \le x \le 1,\ 0 \le z \le 1\}$ and $0 \le t \le 3$. The equations are as in Problem 2 except for the terms

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

and the initial conditions,

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

As $t \to \infty$, the solution becomes spatially homogeneous and tends to a time-periodic solution of the Lotka-Volterra ODE system $dc^i/dt = f^i$ $(i = 1,2)$. This ODE system is alternately stiff and nonstiff depending on the position of the solution in (c^1,c^2) phase space.

The two PDE's are discretized on a square J by K grid in the same way as in Problem 2, giving an ODE system of size N = 2JK, except that we vary the mesh dimensions from J = K = 10 (N = 200) to J = K = 50 (N = 5000). The Jacobian has half-bandwidths ML = MU = 2J. Tolerance parameters RTOL = 10^{-6}

and ATOL = 10^{-4} were used.

The test results on this problem are given in Table 3. Here, in all cases, LSIOM gives superior run times, with a 52% savings in run time and a 95% savings in required work space for the 50x50 grid problem. However, note the steady rise in FPSIOM with grid size, indicating that SIOM might require a larger value of ℓ_{max} if the grid were further refined.

TABLE 3. Test results for Problem 3.

Algorithm	R.T.	NST	NFE	NJE	NSIOM	Work Space (words)	FPSIOM
			<u>10 x 10 Grid</u>				
LSODE	6.97	1248	1635	129	0	14,642	0
LSIOM	6.18	1280	5042	0	2609	3,497	.93
			<u>20 x 20 Grid</u>				
LSODE	52.0	1346	1795	197	0	106,422	0
LSIOM	28.5	1206	6408	0	2441	13,675	1.62
			<u>30 x 30 Grid</u>				
LSODE	131.3	1197	1569	144	0	347,444	0
LSIOM	78.7	1141	8154	0	2287	30,675	2.56
			<u>50 x 50 Grid</u>				
LSODE	661.4	1,145	1,493	139	0	1,565,042	0
LSIOM	318.7	1,163	12,198	0	2371	85,097	4.14

6. Conclusions and Future Work

We have presented here an essentially matrix-free Newton-like iteration scheme and its implementation into the LSODE package for the solution of stiff systems of ODE's. The focus has been on those problems for which the cost of performing the linear algebra associated with solving the system (2.5) by matrix methods far outweighs that of a function evaluation. Hence, one can use a matrix-free method such as the IOM algorithm to perform the Newton iterations involved in the integration of the problem, at a greatly reduced cost in run time, overall work and storage. The inclusion of scaling associated with error tolerances, in the SIOM form of the algorithm, is crucial for robustness, we feel. Although our testing has been somewhat limited, initial results seem very promising.

There are several aspects of the method which need investigation, however. For example, a deeper understanding of the convergence of the IOM algorithm may indicate a class of problems on which IOM will work very well, while at the same time eliminating other classes. Currently, it is impossible to predict how IOM will perform.

The corrector loop strategy given in the modified algorithm of Section 4.2 may show improvement by employing some of the techniques from Dembo, Eisenstat and Steihaug [1] for Inexact Newton Methods. In [5] and [6], Gear and Saad have suggested ways of reusing the vectors $v_1,\dots,v_\ell$ and the matrix $\tilde{H}_\ell$. Our initial testing of these ideas indicated that they are not beneficial, but much more testing needs to be done to make a final decision as to their usefulness.

In the test results of Section 5, we have exclusively used the parameter values $\ell_{max} = p = 5$. This effectively reduces Algorithm 3.3 to a scaled version of Arnoldi's algorithm, instead of the IOM algorithm. Some of the test cases were also run with p values between 2 and 5, and some of the resulting run times were lower (from fewer function evaluations), but not by very much. This may be due partly to the fact that both p and ℓ_{max} are relatively small. In the course of further testing (and possibly even in a dynamic manner), ℓ_{max} may need to be made larger, thus making IOM more competitive.

In conclusion, the IOM algorithm certainly shows promise of being a very competitive method for use in the numerical solution of large stiff systems of ODE's, and may also prove to be competitive for use in a Newton-like method for the solution of nonlinear systems of equations in general.

References

[1] R. S. Dembo, S. C. Eisenstat and T. Steihaug, "Inexact Newton Methods," SIAM J. Numer. Anal. Vol. 19, No. 2, April 1982, pp 400-408.

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

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

[4] A. C. Hindmarsh, ODEPACK, a Systematized Collection of ODE Solvers, in R.S. Stepleman (ed.), Numerical Methods for Scientific Computation, North-Holland Publ. Co., 1983.

[5] C. W. Gear and Y. Saad, "Iterative Solution of Linear Equations in ODE Codes," UIUCDCS-R-81-1054, Dept. of Comp. Sci., University of Illinois, Champaign-Urbana, 1981; also SIAM J. on Sci. & Stat. Comp., to appear.

[6] Y. Saad, "Krylov Subspace Methods for Solving Large Unsymmetric Linear Systems," Mathematics of Computation, Vol. 37, No. 155, July 1981, pp 105-126.

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161
Price: Printed Copy $; Microfiche $4.50

Page Range	Domestic Price	Page Range	Domestic Price
001-025	$ 7.00	326-350	$ 26.50
026-050	8.50	351-375	28.00
051-075	10.00	376-400	29.50
076-100	11.50	401-426	31.00
101-125	13.00	427-450	32.50
126-150	14.50	451-475	34.00
151-175	16.00	476-500	35.50
176-200	17.50	501-525	37.00
201-225	19.00	526-550	38.50
226-250	20.50	551-575	40.00
251-275	22.00	576-600	41.50
276-300	23.50	601-up[1]	
301-325	25.00		

[1]Add 1.50 for each additional 25 page increment, or portion thereof from 601 pages up.