UCRL- 95088, Rev. 1
PREPRINT

REDUCED STORAGE MATRIX METHODS IN STIFF ODE SYSTEMS

Peter N. Brown
Alan C. Hindmarsh

This paper was prepared for submission to the Journal of Applied Mathematics & Computation.

See vol. 31 (May 1989), pp. 40-91.

June 1987

Lawrence Livermore National Laboratory

Reduced Storage Matrix Methods in Stiff ODE Systems *

Peter N. Brown
Department of Mathematics
University of Houston
Houston, TX 77004

Alan C. Hindmarsh
Computing & Mathematics Research Division., L-316
Lawrence Livermore National Laboratory
Livermore, CA 94550

June 3, 1987

* 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. The first author was also supported in part by the National Science Foundation Grant DMS-8506651.

Abstract

Stiff initial-value ODE methods (e.g. BDF) normally require the Jacobian matrix as part of a Newton-like iteration within each implicit time step, and require it to be formed and stored explicitly, whether the linear algebraic method used is direct or iterative. But, a Krylov subspace iterative linear system method, which involves the system matrix only in operator form, can be made part of an inexact Newton method within such a stiff ODE method in a matrix-free manner, requiring no explicit Jacobian matrix storage. Such combinations, using BDF methods, have been implemented with Arnoldi iteration, GMRES (the Generalized Minimum RESidual method), and CG (the Conjugate Gradient method). Various practical matters (scaling, starting, stopping, etc.) are dealt with in the stiff ODE context. In the context of general nonlinear algebraic systems, we provide some theoretical foundation for the combined Newton-Krylov method by giving convergence results that include errors due to the difference quotient approximation to the linear operator. Earlier tests showed matrix-free methods to be quite effective, at least when the spectrum of the problem Jacobian is rather tightly clustered. To improve their robustness, we have added preconditioning, in an experimental solver called LSODPK, which we tested on ODE systems that arise from time-dependent PDE systems by the method of lines. Preconditioner matrices can be formed from the interaction or reaction terms, from the spatial transport terms, or both (as in operator splitting). The additional matrix storage can be reduced greatly by grouping the diagonal blocks in a natural way. The methods appear to be quite effective in improving both speed and storage economy over traditional stiff methods, and over matrix-free methods without preconditioning.

1. Introduction

In a previous paper [4], we considered the use of Krylov-subspace projection methods in solving large stiff systems of ordinary differential equations (ODE's). Typically, methods for solving stiff ODE systems are implicit, and so require the solution of a nonlinear algebraic system of equations at each integration step. Newton's method (or some modification of it) appears to be the best general approach to such systems, and this leads to solving several linear systems at each step. For large problems, most of the work required for the integration is in the linear algebra operations associated with these linear systems. In addition, when using direct methods to solve the linear systems, much of the core memory required is used for the storage of the coefficient matrix and its decomposition factors. Alternatively, the Krylov methods are iterative linear system solvers which do not require the storage of the coefficient matrix in any form, and hence require far less storage than direct methods. All that is required is the ability to perform coefficient matrix-vector multiplies. In [4], we referred to the combined stiff ODE method/Krylov method as a *matrix-free method*, and discussed both theoretical and computational aspects of the combined algorithm. In this paper, we will continue investigating these combined algorithms, with particular emphasis on the importance of preconditioning the linear systems solved by the Krylov methods.

To be more specific, we will consider here the numerical solution of the ODE Initial Value Problem

$$\dot{y} = f(t,y),\ y(t_o) = y_o\ \ (\cdot = d/dt\ , y \epsilon R^N). \tag{1.1}$$

We will assume that the ODE in (1.1) is stiff, meaning that one or more strongly damped modes are present, and will use the popular BDF (Backward Differentiation Formula) methods to solve it. These methods have the general form

$$y_n = \sum_{j=1}^{q} \alpha_j y_{n-j} + h\beta_o \dot{y}_n, \quad \dot{y}_n = f(t_n, y_n), \tag{1.2}$$

where q is the method order. Since the BDF methods are implicit, at each time step one must solve the algebraic system

$$y_n - h\beta_o f(t_n, y_n) - a_n = 0, \quad a_n \equiv \sum_{j=1}^{q} \alpha_j y_{n-j}, \quad \beta_o > 0, \tag{1.3}$$

for y_n. We will actually deal with an equivalent form of (1.3), namely

$$F_n(x_n) \equiv x_n - hf(t_n, a_n + \beta_o x_n) = 0, \tag{1.4}$$

in which x_n is defined by

$$x_n = h\dot{y}_n = (y_n - a_n)/\beta_o.$$

The Newton iteration then has the form:

Let $x_n(0)$ be an initial guess for x_n.

For $m = 0, 1, 2, \cdots$ until convergence:

$$Ps_n(m) = -F_n(x_n(m)) \tag{1.5}$$

$$x_n(m+1) = x_n(m) + s_n(m),$$

where the coefficient matrix P is some value of (or an approximation to a value of)

$$F_n'(x) = I - h\beta_o J(t,y) \quad (y = a_n + \beta_o x), \tag{1.6}$$

with $J(t,y) = \partial f/\partial y$, the system Jacobian matrix.

In [4], we considered two Krylov-subspace projection methods for approximately solving (1.5). These were *Arnoldi's Algorithm* and the *Incomplete Orthogonalization Method* (IOM), both due primarily to Saad [17,18]. The preliminary tests in [4] indicated the potential usefulness of these linear solvers on ODE problems for which there is some clustering of the spectrum of the matrix P, but also indicated the need for some form of preconditioning of the linear systems (1.5) in order for the combined BDF/Krylov method solver to be effective on a much wider class of problems. Preconditioning techniques must be chosen with the particular problem features in mind, and also with a view to keeping the storage requirements low. For problems arising from time-dependent partial differential equation (PDE) systems, choices based on successive overrelaxation (SOR), and on the interaction of the PDE variable at each spatial point are available.

Work that is closely related to ours includes that of Gear and Saad [9], who originally proposed using Arnoldi's Algorithm and IOM in stiff ODE solvers, and Miranker and Chern [15] who considered the use of the Conjugate Gradient Method in the solution of the model problem $dy/dt = Jy$ by BDF methods for which J is symmetric and positive definite. Additionally, Chan and Jackson [5] have considered the use of Preconditioned Krylov-subspace projection methods in ODE solvers. However, their methods differ from those considered here in several respects. First, the basic Krylov methods considered in [5] are the *Conjugate Residual Method* (CR) and *Orthomin(k)* (cf. [8]). We note that CR applied to (1.5) is only guaranteed to converge when P is symmetric and positive definite, while Orthomin(k) only requires that P is positive definite for convergence. Chan and Jackson argue that for symmetric problems (i.e. $J = \partial f/\partial y$ symmetric) the step size selection strategy of the ODE solver will normally choose h so that

$$1 - h\beta_o\lambda_i > 0 \qquad (1.7)$$

for all λ_i ($i = 1, ..., N$) an eigenvalue of J. Hence, in this case $P = I - h\beta_o J$ would be positive definite. When J is nonsymmetric, (1.7) holding for all λ_i does not imply P is positive definite, as Chan and Jackson note. Thus, the application of CR and Orthomin(k) to such linear systems may fail. Second, these methods are actually applied to the preconditioned linear systems, and so one must be careful when choosing the preconditionings to be used so that the resulting system has a matrix which is again positive definite, as Chan and Jackson also note. In our setting, Arnoldi's Algorithm and GMRES are guaranteed to converge when the matrix P is nonsingular (whether or not it is positive definite). This results in a wider class of available preconditionings when using Arnoldi and GMRES.

The rest of the paper is organized as follows: Section 2 summarizes the Newton and Newton-like iteration for the nonlinear system and the basic linear iterations to be considered (Arnoldi, GMRES, and CG), and includes a new result on an incomplete version of GMRES. Section 3 gives some local convergence results for the combined Newton/Krylov methods. Section 4 discusses scaled preconditioned methods in general, and Section 5 describes specific preconditioners suitable for ODE systems arising from certain PDE systems. Section 6 gives some numerical test results, using a modified version of the general purpose ODE solver LSODE [13,14].

2. Preliminaries

In this section we introduce a class of Newton-like iteration schemes known as *Inexact Newton Methods* and discuss their relevance here in solving the nonlinear system (1.4). We then introduce the Krylov subspace projection methods under consideration, and discuss some of their convergence properties.

(a) Newton Methods

Newton's method applied to a general nonlinear system

$$F(x) = 0, \ F : R^N \rightarrow R^N, \tag{2.1}$$

with solution x^* , results in the iteration scheme

```
    Choose $x(0)$ an initial guess for $x^*$.
    For $m = 0, 1, \ldots$ until convergence, do:
    Solve
```

$$Ps(m) = F(x(m)) \tag{2.2}$$

```
    Set $x(m+1) = x(m) + s(m)$,
```

where $P = F'(x(m))$ is the Newton matrix (F' denoting $\partial F/\partial x$). In the stiff ODE context, a system of the form (2.1) needs to be solved at every step, and so many ODE solvers attempt to save work by computing and storing P (and its decomposition factors if a direct method is used to solve (2.2)) once, and using it for all iterations on that step. Furthermore, P is also held fixed over several steps of the integration, only discarding the current P when it is determined to be sufficiently out of date. The resulting iteration is known as *modified Newton*, and typically gives a linear rate of convergence as opposed to the quadratic rate of convergence for the full Newton scheme.

When using an iterative method to solve (2.2) approximately, one has several options. First, in the case where P is formed and stored explicitly, and saved for use over several steps, then the basic iteration scheme is modified Newton with P some approximation to $F'(x(m))$. One then approximately solves the modified Newton equations (2.2) by some iterative scheme. One example of this approach is the ODE solver GEARBI [12], which uses Block-SOR as the basic iterative solver. Another is the solver developed by Chan and Jackson [5], which uses the Krylov methods CR and Orthomin(k) as the iterative solvers; we note that both of these ODE solvers require the forming and storing of the matrix P.

A second approach to solving (2.2) approximately is to work with the full Newton equations (2.2) where $P = F'(x(m))$. Since there is a significant cost associated with forming P, this approach only makes sense when P does not need to be formed explicitly. All of the Krylov subspace projection methods mentioned above only require the action of P times a vector v, not P itself. Hence, one possible way to approximate this action is by using a difference quotient

$$Pv = F'(x)v \approx (F(x + \sigma v) - F(x))/\sigma \quad (\sigma \epsilon R) \tag{2.3}$$

where v is a vector of unit length and x is the current iterate. This approach is taken by the authors in [4] and continued here. We refer to these methods as *matrix-free* due to the absence of required storage for P. With either approach there is an error associated with only approximately solving (2.2), and with the second approach there is an additional error resulting from the approximation (2.3). We next discuss the class of Inexact Newton methods, which deals with errors in (2.2), and then in Section 3 we discuss the combined errors associated with (2.2) and (2.3).

From Dembo, Eisenstat and Steihaug [6], an Inexact Newton Method for (2.1) has the following general form:

Choose $x(0)$ an initial guess for x^*.

For $m = 0, 1, \cdots$ until convergence, do:

Find (in some unspecified manner) a vector $s(m)$ satisfying

$$F'(x(m))s(m) = -F(x(m)) + r(m) \tag{2.4}$$

Set $x(m+1) = x(m) + s(m)$.

The residual $r(m)$ represents the amount by which $s(m)$ fails to satisfy the Newton equation (2.2). It is not generally known in advance, being the result of some inner algorithm which produces only an approximate solution to (2.2) (e.g. an iterative method). Typically, one must require an auxiliary condition on the size of the residual $r(m)$ for convergence. In [6], it is shown that if

$$\|r(m)\| \leq \eta \|F(x(m))\|, \quad m = 0, 1, 2 \cdots, \tag{2.5}$$

where $0 \leq \eta < 1$, then $x(m)$ converges to a true solution of $F(x) = 0$ at least linearly, as long as the initial guess $x(0)$ is close enough. Here, $\|\cdot\|$ is any norm on R^N.

For the present stiff ODE context, the condition (2.5) is overly restrictive in that actual convergence of the iterates is not necessary, and the cost of obtaining them is high. Here, the Newton iteration begins with an explicit prediction $y_n(0)$, and a corresponding prediction $x(0) = (y_n(0) - a_n)/\beta_o$ of $h\dot{y}_n$ Thus, the first linear system to be solved on the n^{th} time step is $Ps = b$ with

$$b = -F(x(0)) = hf(t_n, y_n(0)) - x(0),$$

$$P = F'(x(0)) = I - h\beta_o J(t_n, y_n(0)).$$

The stiffness of the problem can be expected to make b largest in the stiff components (i.e., in the subspace corresponding to the stiff eigenvalues). Since the prediction is normally sufficient in the nonstiff components, all one really needs in the corrector iteration is to damp out the errors associated with stiff components, for stability, not actual convergence.

Thus it is of interest to find out how much one can relax (2.5) and still obtain enough accuracy in the approximate solution $x(m)$ to x^*. In [4], it is shown that if (2.5) is replaced by the weaker condition

$$\|r(m)\| \leq \delta, \quad m = 0, 1, 2 \cdots, \tag{2.6}$$

then

$$\limsup_{m \to \infty} \|x(m) - x^*\| \leq \delta / K,$$

where K is a constant depending only upon F and x^* , assuming that $x(0)$ is close enough to x^* and δ is sufficiently small. Thus, one can obtain any degree of accuracy in $x(m)$ desired by simply choosing δ small enough. It is further argued in [4] that the constant $K \approx 1$ for the stiff ODE context. Therefore, if ϵ_1 is the desired tolerance in the error for the approximate $x(m)$, choosing $\delta \approx \epsilon_1$ is reasonable in (2.6).

For modified Newton iteration, under appropriate conditions on P and F that guarantee the local convergence of the iterates $x(m)$ to x^* with $P \neq F'(x^*)$, one has *linear* convergence in that as $m \to \infty$

$$\|x(m+1) - x^*\| / \|x(m) - x^*\| \to C,$$

where $0 < C < 1$, and again $\|\cdot\|$ is any norm on R^N. The estimate

$$C_m = \|x(m+1) - x(m)\| / \|x(m) - x(m-1)\| \tag{2.7}$$

of the asymptotic rate constant C can be easily found once $x(m+1)$ has been computed, and then used in subsequent stopping tests. Hence, a stopping condition on $x(m+1)$ of the form

$$\|x(m+1) - x^*\| \leq \epsilon$$

will be satisfied approximately if the (verifiable) condition

$$\bar{C}\|x(m+1)-x(m)\| \leq \epsilon$$

holds, provided that $\bar{C}$ well approximates $C/(1-C)$, or simply C if C is sufficiently small. LSODE uses this convergence acceleration idea in its stopping test for modified Newton iterations (along with some suitable fudge factors), and it is quite beneficial in reducing the average number of iterations per step. In [4], it is shown that if the Inexact Newton iterates $x(m)$ and residuals $r(m)$ satisfy the stronger condition

$$\|r(m)\| \leq \eta \; \|F(x(m))\|^2 \; for \; m = 0, 1, 2, \cdots,$$

where $0 \leq \eta < 1$, then for any $\epsilon' > o$

$$\|x(m+1)-x^*\| \leq C_m \|x(m+1)-x(m)\|(1+\epsilon^*) \tag{2.8}$$

for all sufficiently large m, with C_m given by (2.7). When only (2.6) holds, inequality (2.8) is no longer true in general. However, a heuristic argument is given in [4] which indicates that if δ is sufficiently small, then (2.8) does hold for all iterates of interest. Again if ϵ_1 is the prescribed tolerance for $x(m)$, then it is likely the case that δ needs to be much smaller than ϵ_1. The exact choices used for δ and ϵ_1 are given below and in [4].

(b) Krylov Subspace Projection Methods.

In this subsection we consider three iterative linear solvers. These are Arnoldi's Algorithm due to Saad [16,17], the Generalized Minimum Residual Method (GMRES) due to Saad and Schultz [19] and the Conjugate Gradient (CG) Method due to Hestenes and Stiefel [11]. All of these are algorithms for the approximate solution of the linear system

$$Ax = b, \tag{2.9}$$

where A is an $N \times N$ matrix, and x and b are N-vectors. Here, (2.9) represents the full Newton equations (2.2) with $A = F'(x(m)), b = -F(x(m))$ and the solution vector x represents the increment $s(m) = x(m+1) - x(m)$ giving the next Newton iterate $x(m+1)$. (To conform with normal usage, the letter x is also used to denote the solutions of linear systems; the particular meaning should be clear from the context, however.) We give here a brief development of Arnoldi and GMRES, along with incomplete versions of these algorithms. More details on Arnoldi are given in [16,17,18], and on GMRES in [19]. We then close with a statement of the CG Method for symmetric positive definite systems.

If x_o is an initial guess for the true solution of (2.9), then letting $x = x_o + z$, we get the equivalent system

$$Az = r_o, \tag{2.10}$$

where $r_o = b - Ax_o$ is the initial residual. Let K_l be the *Krylov subspace*

$$K_l = span\ (r_o, Ar_o, \cdots, A^{l-1}r_o).$$

By a *Krylov subspace projection method* on K_l we mean a method which finds an approximate solution

$$x_l = x_o + z_l,\ with\ z_l \epsilon K_l.$$

To uniquely specify x_l (or z_l) some additional requirements are necessary. These typically are one of two types: either require that

$$(b - Ax_l) \perp K_l\ (\ or\ (r_o - Az_l) \perp K_l) \tag{2.11}$$

or

$$\|b - Ax_l\|_2 = \min_{x \epsilon x_0 + K_l} \|b - Ax\|_2\ \ (= \min_{z \epsilon K_l} \|r_o - Az\|_2). \tag{2.12}$$

Here, orthogonality is meant in the usual Euclidean sense, and $\|\cdot\|_2$ denotes the Euclidean norm. The combinations of requirements (2.10) and (2.11) versus (2.10) and (2.12) give rise to different Krylov methods. Requiring that (2.10) and (2.11) hold leads to Arnoldi's Algorithm, while (2.10) and (2.12) lead to GMRES.

Arnoldi's Algorithm and GMRES both use an *Arnoldi process* [1] to construct an orthonormal basis of the Krylov subspace K_l. Briefly, an orthonormal basis $(v_1, \cdots, v_l)$ of K_l is constructed using the algorithm:

```
1. Compute $r_o = b - Ax_o$ and set $v_1 = r_o/\|r_o\|_2$ .
2. For $j = 1, \cdots, l$ do:
   $w_{j+l} = Av_j - \sum_{i=1}^{j} h_{ij} v_j$ , $h_{ij} = (Av_j, v_i)$
   $h_{j+1,j} = \|w_{j+1}\|_2$, $v_{j+1} = w_{j+1}/h_{j+1,j}$.
```

Here $(\cdot, \cdot)$ is the Euclidean inner product. If we let $V_l = [v_1, \cdots, v_l]$ denote the $N \times l$ matrix with columns v_i, and $H_l = (h_{ij})$ is the $l \times l$ upper Hessenberg matrix whose nonzero entries are given by the above h_{ij} , then Saad [17] has shown that

$$H_l = V_l^T A V_l \textit{ and } V_l^T V_l = I_l, \tag{2.13}$$

where I_l is the $l \times l$ identity matrix. It is assumed throughout the vectors $r_o, Ar_o, \cdots, A^{l-1} r_o$ are linearly independent so that the dimension of K_l is l.

To describe *Arnoldi's Algorithm* , first let $z_l = V_l y_l$ where $y_l \epsilon R^l$. Then condition (2.11) is equivalent to

$$V_l^T A V_l y_l - V_l^T r_o = 0.$$

If $H_l = V_l^T A V_l$ is nonsingular, then $y_l = H_l^{-1} V_l^T r_o$ and

$$x_l = x_o + z_l = x_o + V_l H_l^{-1} V_l^T r_o. \tag{2.14}$$

Since $V_l^T r_o = \beta e_1$, where $\beta = \|r_o\|_2$ and $e_1 = (1,0,\cdots,0)^T \epsilon R^l$, (2.14) reduces to

$$x_l = x_o + \beta V_l H_l^{-1} e_1. \tag{2.15}$$

An important practical consideration is the choice of l, which amounts to a stopping criterion. A very useful identity for this is the following equation for the residual norm:

$$\|b - Ax_l\|_2 = h_{l+1,l} |e_l^T y_l|, \tag{2.16}$$

where $e_l = (0,\cdots,0,1)^T \epsilon R^l$. The equation (2.16) follows from the relation

$$AV_l = V_l H_l + h_{l+1,l} v_{l+1} e_l^T,$$

which can be derived from the algorithm. An interesting feature of the relation (2.16) is that one does not have to form x_l or y_l in order to compute $\|b - Ax_l\|_2$. If we perform an LU factorization of H_l , writing $H_l = LU$, and assume that no pivoting was necessary, then it can be shown [17] that

$$h_{l+1,l} \left| e_l^T y_l \right| = h_{l+1,l} \beta \left| u_{ll}^{-1} \prod_{i=1}^{l-1} l_i \right|, \tag{2.17}$$

where the $l_i (i = 1,\cdots,l-1)$ are the successive multipliers (subdiagonal elements of L) . In general, a similar equation holds in which the product above is taken only over those i for which no pivoting is done. See [13] for more details.

The use of (2.16) to estimate the error $\|b - Ax_l\|_2$ then leads to the following algorithm, in which l_{max} and δ are given parameters:

Algorithm 2.1 (Arnoldi's Algorithm):

```
1. Compute $r_o = b - Ax_o$ and set $v_1 = r_o / \|r_o\|_2$.
2. For $l = 1,2,\cdots,l_{max}$ do:
    (a) $w_{l+1} = Av_l - \sum_{i=1}^{l} h_{il} v_i, h_{il} = (Av_l, v_i)$
```

$h_{l+1,l} = \|w_{l+1}\|_2$

$v_{l+1} = w_{l+1}/h_{l+1,l}$

(b) Update the LU factorization of H_l .

(c) Use (2.17) to compute $\rho_l = h_{l+1,l}|e_l^T y_l| = \|b - Ax_l\|_2$.

(d) If $\rho_l \leq \delta$, go to Step 3. Otherwise, go to (a).

3. Compute $x_l = x_o + \|r_o\|_2 V_l H_l^{-1} e_1$ and stop.

In the above algorithm, if the test on ρ_l fails, and if $l = l_{max}$ iterations have been performed, then one has the option of either accepting the final approximation x_l, or setting $x_o = x_l$ 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 $H_{l-1}(l > 1)$.

In Algorithm 2.1, as l gets large, a considerable amount of the work involved is in making the vector v_{l+1} orthogonal to all the previous vectors $v_1, \cdots, v_l$. Saad [17] has proposed a modification of Algorithm 2.1 in which the vector v_{l+1} is only required to be orthogonal to the previous p vectors, $v_{l-p+1}, \cdots, v_l$. Saad [17] has shown that equations (2.16) and (2.17) still hold in this case. This leads to an algorithm called the *Incomplete Orthogonalization Method,* denoted by *IOM.* It differs from Algorithm 2.1 only in that the sum in Step 2(a) begins at $i = i_o$ instead of at $i = 1$, where $i_o = max(1, l - p + 1)$. The remarks made after Algorithm 2.1 are also applicable to IOM. In [17], Saad compares the two algorithms on several test problems, and reports that IOM is sometimes preferred, based on total work required and run times.

When A is symmetric, the inner products $h_{i,l}$ theoretically vanish for $i < l - 1$, so that one can take $p = 2$. If A is also positive definite then IOM with $p = 2$ is equivalent to the Conjugate Gradient method [17, Sec. 3.3.1, Remark 4]. Thus a value of p less than l_{max} might be expected to be cost-effective when A is nearly symmetric.

Saad [17] and Gear and Saad [9] have given a convergence analysis of Algorithm 2.1 which shows that Arnoldi's Method converges in at most N iterations and suggests (but in general does not prove) that the convergence of the iterates $\{ x_l \}$ to the solution of (2.9) is fastest in the dominant subspace (that is, in those components corresponding to the eigenvalues in the outermost part of the spectrum of A), which would include the stiff components for the ODE context.

The possibility of a breakdown also exists when using Algorithm 2.1. This can happen in two different ways: either $w_{l+1} = 0$ so that v_{l+1} cannot be formed, or $H_{l_{max}}$ may be singular which means that the maximum number of Arnoldi steps has been taken, but the final iterate cannot be computed. The first case has been referred to as a happy breakdown, since $w_{l+1} = 0$ implies H_l is nonsingular and x_l is the exact solution of (2.9) (cf. Brown [2] or Saad [18]). The second case is more serious in that it causes a convergence failure. In the ODE setting, where $A = I - h\beta_o J$, a way to handle the second failure is to reduce the stepsize h and retry the step. We note, however, this second kind of breakdown cannot happen when A is positive definite. To see this, recall that $H_l = V_l^T A V_l$, and so for any $y \neq 0$

$$(y, H_l y) = (y, V_l^T A V_l y) = (V_l y, A V_l y) > 0$$

since $V_l y \neq 0$ by the fact that V_l has orthonormal columns, and since A is positive definite. Thus, H_l is positive definite and so nonsingular.

The GMRES method differs from Arnoldi's method only in the way the vector y_l is computed, where $x_l = x_o + V_l y_l$. Suppose we have taken l steps of the above Arnoldi process with $w_{l+1} \neq 0$. Then we have two matrices,

$$V_{l+1} = [v_1, \cdots, v_{l+1}] \epsilon R^{N \times (l+1)}$$

whose columns are orthonormal, and the matrix $\bar{H}_l \epsilon R^{(l+1) \times l}$ defined by

$$\bar{H}_l = \begin{bmatrix} H_l \\ r^T \end{bmatrix}, \; where \; r = (0, \cdots, 0, h_{l+1,l})^T \epsilon R^l.$$

It follows from the Arnoldi process that

$$AV_l = V_{l+1} \bar{H}_l. \tag{2.18}$$

The vector $z_l \epsilon K_l$ is chosen to satisfy (2.12), namely

$$\|r_o - Az_l\|_2 = \min_{z \epsilon K_l} \|r_o - Az\|_2. \tag{2.19}$$

Letting $z = V_l y$ and using (2.18) gives

$$J(y) = \|r_o - Az\|_2 = \|\beta v_1 - AV_l y\|_2$$

$$= \|V_{l+1}(\beta e_1 - \bar{H}_l y)\|_2 = \|\beta e_1 - \bar{H}_l y\|_2$$

where $\beta = \|r_o\|_2$ and $e_1 = (1, 0, \cdots, 0)^T \epsilon R^{l+1}$, since V_{l+1} has orthonormal columns. Thus, the solution of (2.12) or (2.19) is given by

$$x_l = x_o + V_l y_l,$$

where y_l minimizes $J(y)$ over $y \epsilon R_l$.

The minimization of $J(y)$ is accomplished by performing a QR factorization of $\bar{H}_l$ using Givens rotations. As Saad and Schultz [19] have noted, it is desirable to be able to update the factorization of $\bar{H}_l$ progressively as each column appears (i.e., at every step of the Arnoldi process). This allows one to compute the residual norm $\|b - Ax_l\|_2$ without computing x_l

at each step. To see this, let F_j be the rotation matrix which rotates e_j and e_{j+1} by the angle θ_j , namely

$$F_j = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & c_j & -s_j & & \\ & & & s_j & c_j & & \\ & & & & & 1 & \\ & & & & & & \ddots \\ & & & & & & & 1 \end{bmatrix} \begin{array}{l} \\ \\ \\ \\ \leftarrow row\ j+1 \\ \\ \\ \\ \end{array} \qquad (2.20)$$

where $c_j = cos(\theta_j)$ and $s_j = sin(\theta_j)$. Next, suppose that the rotations $F_1, \cdots, F_j$ have been applied to $\bar{H}_j$, giving

$$F_j F_{j-1} \cdots F_1 \bar{H}_j = R_j \epsilon R^{(j+1)\times j},$$

where R_j is upper triangular with its last row containing all zeros. At the next step of the Arnoldi process, the last row and column of $\bar{H}_{j+1}$ appear. Let $d = (d', h)^T$, where $d' \epsilon R^{j+1}$ and $h = h_{j+2,j+1}$, be the new column. To obtain R_{j+1} first form $\bar{d} = F_j F_{j-1} ... F_1 d$ and let its next-to-last component be denoted by r.. (Note that the last component h is the same in both $\bar{d}$ and d .) The rotation F_{j+1} is then chosen to eliminate h in $\bar{d}$. This gives

$$c_{j+1} \equiv r/\sqrt{r^2+h^2}$$

$$s_{j+1} \equiv -h/\sqrt{r^2+h^2}.$$

After l steps, one has the decomposition

$$Q_l^T \bar{H}_l = R_l,$$

where $Q_l^T = F_l F_{l-1} \cdots F_1 \epsilon R^{(l+1)\times(l+1)}$ and $R_l \epsilon R^{(l+1)\times l}$ is upper triangular with zeros in its last row. Thus, we have

$$J(y) = ||\beta e_1 - \bar{H}_l y||_2 = ||\beta e_1 - Q_l R_l y||_2 = ||\beta Q_l^T e_1 - R_l y||_2.$$

Let $R_l = \begin{pmatrix} \bar{R}_l \\ 0 \cdots 0 \end{pmatrix}$ and $g_l = \beta Q_l^T e_1 = (\bar{g}_l, g)^T$ with $\bar{g}_l \epsilon R^l$ and $g \epsilon R$. The value of y which minimizes $J(y)$ is then

$$y_l = (\bar{R}_l)^{-1} \bar{g}_l, \tag{2.21}$$

and

$$||b - Ax_l||_2 = ||\beta Q_l^T e_1 - R_l y_l||_2 = |g|.$$

An easy calculation gives $g = \beta \cdot s_1 s_2 \cdots s_l$, and thus we have

$$||b - Ax_l||_2 = \beta |s_1 \cdots s_l|, \tag{2.22}$$

which is an inexpensive way to calculate the norm of the residual associated with x_l. See Saad and Schultz [19] for more details.

The use of (2.22) leads to the following algorithm, in which l_{max} and δ are given parameters:

Algorithm 2.2 (GMRES):

```
1. Compute $r_o = b - Ax_o$ and set $v_1 = r_o/||r_o||_2$ .
2. For $l = 1, \cdots, l_{max}$ do:
   (a) $w_{l+1} = Av_l - \sum_{i=1}^{l} h_{il} v_i, h_{il} = (Av_l, v_i)$
       $h_{l+1,l} = ||w_{l+1}||_2$
       $v_{l+1} = w_{l+1}/h_{l+1,l}$ .
   (b) Update the QR factorization of $\bar{H}_l$ .
   (c) Use (2.22) to compute $\rho_l = ||r_o||_2 \cdot |s_1 \cdots s_l| = ||b - Ax_l||_2$ .
   (d) If $\rho_l \leq \delta$ , go to Step 3. Otherwise, go to (a).
```

3. Compute $x_l = x_o + V_l y_l$ with y_l given by (2.21), and stop.

The remarks on restarting after Algorithm 2.1 are also valid here.

As in Algorithm 2.1, it is also the case here that as l gets large much work is required to make v_{l+1} orthogonal to all the previous vectors $v_1, \cdots, v_l$. One can also propose an incomplete version of GMRES (denoted by IGMRES), which differs from Algorithm 2.2 only in that the sum in Step 2(a) begins at $i = i_o$ instead of at $i = 1$, where $i_o = max(1, l-p+1)$. One immediate problem with IGMRES comes from the fact that equality (2.22) no longer holds. To see this, note that from the incomplete Arnoldi process, as long as $w_{l+1} \neq 0$, we still have

$$AV_l = V_{l+1}\bar{H}_l,$$

where $\bar{H}_l \epsilon R^{(l+1)\times l}$ is now a banded upper Hessenberg matrix, and V_{l+1} has columns with unit norm but $V_{l+1}^T V_{l+1} \neq I_l$ in general. It follows that $\bar{H}_l$ will still have full column rank, and so let

$$\bar{H}_l = Q_l R_l$$

be its QR factorization. Here again, Q_l is an $(l+1) \times (l+1)$ orthogonal matrix, and R_l is an $(l+1) \times l$ upper triangular matrix whose last row contains all zeros. The approximate solution x_l is given by

$$x_l = x_o + V_l y_l$$

where y_l solves the minimization problem

$$\min_{y \epsilon R^l} \|\beta e_1 - \bar{H}_l y\|_2 \tag{2.23}$$

as before. The residual associated with x_l is

$$b - Ax_l = r_o - Az_l = r_o - AV_l y_l = V_{l+1}[\beta e_1 - \bar{H}_l y_l],$$

since $r_o = \beta V_{l+1} e_1$. Next,

$$\beta e_1 - \bar{H}_l y_l = \beta e_1 - Q_l R_l y_l = Q_l[\beta Q_l^T e_1 - R_l y_l],$$

and since y_l solves the minimization problem (2.23) we have

$$\beta e_1 - \bar{H}_l y_l = Q_l \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \beta q_{l+1}^T e_1 \end{bmatrix} = \beta q_{l+1} q_{l+1}^T e_1,$$

where $Q_l = [q_1, \cdots, q_{l+1}]$. Therefore,

$$b - A x_l = \beta V_{l+1} q_{l+1} q_{l+1}^T e_1,$$

which gives

$$\|b - A x_l\|_2 = \beta |q_{l+1}^T e_1| \cdot \|V_{l+1} q_{l+1}\|_2. \tag{2.24}$$

If V_{l+1} has orthonormal columns, then $\|V_{l+1} q_{l+1}\|_2 = \|q_{l+1}\|_2 = 1$ since Q_l is orthogonal. In this case (2.24) and (2.22) agree, and so we must have

$$|q_{l+1}^T e_1| = |s_1 s_2 \cdots s_l|. \tag{2.25}$$

To show that (2.25) still holds when V_{l+1} does not have orthonormal columns will require some further justification.

At each stage of the incomplete Arnoldi process the QR factorization of $\bar{H}_l$ will be updated. Let $F_{j,l} (j = 1, \cdots, l)$ be the Givens rotation matrices of dimension $(l+1) \times (l+1)$ defined by (2.20) so that

$$Q_l^T = F_{l,l} \cdot F_{l-1,l} \cdots F_{1,l},$$

with

$$\bar{H}_l = Q_l R_l.$$

The extra subscript l on the $F_{j,l}$ indicates only that they are viewed as $(l+1) \times (l+1)$ matrices when used to form Q_l . The components c_j and s_j are independent of l (where l denotes the current step of the Arnoldi process). Also, it is clear that some of the columns of Q_l change as l increases. Hence, we will write

$$Q_l = [q_{1,l},\ q_{2,l},\ \cdots,\ q_{l+1,l}].$$

Theorem 2.1: Let $V_{l+1} = [v_1, \cdots, v_{l+1}]$ be the vectors computed in taking l steps of the above incomplete Arnoldi process. Let $F_{j,l}(j = 1, ..., l)$ be the Givens rotations used to factor $\bar{H}_l$ into $Q_l R_l$. Then the last column of Q_l is

$$q_{l+1,l} = \left(\prod_{i=1}^{l} s_i,\ c_1 \prod_{i=2}^{l} s_i,\ c_2 \prod_{i=3}^{l} s_i,\ \cdots,\ c_{l-1} s_l,\ c_l \right)^T \tag{2.26}$$

and

$$V_{l+1} q_{l+1,l} = s_l V_l q_{l,l-1} + c_l v_{l+1}. \tag{2.27}$$

Proof: We show (2.26) by induction on l . For $l = 1$,

$$Q_l^T = F_{1,1} = \begin{pmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{pmatrix},$$

and so $q_{2,1} = (s_1, c_1)^T$. Assume (2.26) holds for l replaced by $l-1$. Then $q_{l,l-1}^T$ is the last row in the matrix

$$S = F_{l-1,l-1} F_{l-2,l-1} \cdots F_{1,l-1}.$$

Now, because $F_{j,l-1}$ and $F_{j,l}$ $(j = 1, \cdots, l-1)$ are related by

$$F_{j,l} = \left(\begin{array}{c|c} F_{j,l-1} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 0 \cdots 0 & 1 \end{array} \right),$$

we immediately have that

$$F_{l-1,l} \cdot F_{l-2,l} \cdots F_{1,l} = \left(\begin{array}{c|c} S & 0 \\ \hline 0 & 1 \end{array}\right).$$

Next, since

$$F_{l,l} = \left(\begin{array}{c|c} I_{l-1} & 0 \\ \hline 0 & C \end{array}\right), \ with\ C = \begin{pmatrix} c_l & -s_l \\ s_l & c_l \end{pmatrix},$$

and letting

$$S = \left(\begin{array}{c} \bar{S} \\ \hline q_{l,l-1}^T \end{array}\right), \quad \bar{S} \epsilon R^{(l-1)\times l},$$

we have

$$Q_l^T = \left(\begin{array}{c|c} & 0 \\ \bar{S} & \vdots \\ & 0 \\ \hline \multicolumn{2}{c}{CU} \end{array}\right),$$

where

$$U = \left(\begin{array}{c|c} q_{l,l-1}^T & 0 \\ \hline 0 \cdots 0 & 1 \end{array}\right) \epsilon R^{2\times(l+1)}$$

Therefore,

$$q_{l+1,l} = (s_l q_{l,l-1}^T, c_l)^T. \tag{2.28}$$

Relationship (2.26) now follows from (2.28). Finally, from (2.28),

$$V_{l+1} q_{l+1,l} = [V_l, v_{l+1}] q_{l+1,l} = s_l V_l q_{l,l-1} + c_l v_{l+1},$$

which is (2.27). Q.E.D.

From (2.26) it is clear that

$$q_{l+1,l}^T e_1 = s_1 \cdot s_2 \cdots s_l,$$

and so (2.25) continues to hold for the incomplete process. Next, (2.27) gives a relatively inexpensive way to form $V_{l+1}q_{l+1,l}$ in (2.24). If $d_l = V_l q_{l,l-1}$ has been saved from the last step, then

$$d_{l+1} = V_{l+1}q_{l+1,l} = s_l d_l + c_l v_{l+1}$$

can be formed at the cost of one scalar-vector multiply plus one vector add. The norm of d_{l+1} then needs to be computed. While not cheap, this is still much less expensive than the complete orthogonalization method (i.e. GMRES) when p (the number of vectors to reorthogonalize Av_l against) is even modestly smaller than l_{max} .

When the matrix A is symmetric, the inner products h_{il} theoretically vanish for $i < l-1$, so that one can take $p = 2$ as with IOM. If A is also positive definite, then GMRES or IGMRES with $p = 2$ is equivalent to the Conjugate Residual method, while if A is only positive definite, then GMRES is equivalent to the Generalized Conjugate Residual method (cf. Saad and Schultz [19] for more details).

Saad and Schultz [19] have given a convergence analysis of Algorithm 2.2 which shows that the GMRES iterates converge to the true solution of (2.9) in at most N iterations. We also note that Algorithm 2.2 may have breakdowns. If $w_{l+1} = 0$ in the Arnoldi process, then Saad and Schultz [19] have shown x_l is the exact solution of (2.9). This is also referred to as a happy breakdown. When $w_{l+1} \neq 0$, the matrix $\bar{H}_l$ has full column rank, and so the least squares problem (2.19)(or the minimization of $J(y)$) can always be solved via the above QR factorization. However, in some cases the approximation x_l may not be of much use. We give an example illustrating how GMRES (and also Arnoldi's method) can have a dramatic failure.

Example 2.1: Let A be the permutation matrix sending $e_1 \to e_2 \to \cdots \to e_N \to e_1$, where e_i is the i^{th} standard basis vector in R^N . Then

$$A = \begin{pmatrix} 0 & & & & 1 \\ 1 & \ddots & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{pmatrix}.$$

Consider solving $Ax = b$, where $b = e_1$ and $x_o = 0$. Then $x* = A^{-1}b = e_N$. We have $V_l = [e_1, \cdots, e_l]$ and H_l is given by

$$\bar{H}_l = \begin{pmatrix} 0 & & & & 0 \\ 1 & \ddots & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{pmatrix} \; for \; l < N,$$

with $H_N = A$. Hence, H_l is singular for $l < N$ and the Arnoldi iterates x_l do not exist for $l < N$, but $x_N = x_*$. Next, for GMRES

$$\bar{H}_l = \begin{pmatrix} H_l \\ 0 \cdots 0 \;\; 1 \end{pmatrix},$$

and $J(y)$ is a minimum for

$$y = y_l = (\bar{H}_l^T \bar{H}_l)^{-1} \bar{H}_l^T V_{l+1}^T b.$$

Since $V_{l+1}^T b = e_1$, and since $\bar{H}_l^T e_1 = 0 (l = 1, \cdots, N-1)$ because the first row of $\bar{H}_l$ is all zeros, the GMRES iterates satisfy

$$x_l = V_l y_l = 0 \; (l = 1, 2, \cdots, N-1), \quad x_N = x_*.$$

For this example, neither GMRES nor Arnoldi's method make any progress until $l = N$. Notice that restarting either algorithm when $l_{max} < N$ is of no avail here either.

Finally, when the matrix A is symmetric and positive definite, it is appropriate to use an iterative linear solver specifically designed for such systems. We will consider the

Conjugate Gradient Algorithm (CG) of Hestenes and Stiefel [11] in this case. As noted above, Arnoldi's Method and CG are theoretically equivalent when A is symmetric and positive definite. Thus CG also generates approximations x_l in the Krylov subspace K_l . For definiteness, we give a version of the algorithm below.

Algorithm 2.3 (Conjugate Gradient):

```
1. Compute $r_o = b - Ax_o$ and set $p_1 = r_o$ .
2. For $l = 1, 2, \cdots, l_{max}$ do:
    (a) $w_l = Ap_l$
        $\alpha_l = r_{l-1}^T r_{l-1} / p_l^T w_l$
        $x_l = x_{l-1} + \alpha_l p_l$
        $r_l = r_{l-1} - \alpha_l w_l$
    (b) If $\|r_l\|_2 \leq \delta$ , then stop. Otherwise, go to (c).
    (c) $\beta_{l+1} = r_l^T r_l / r_{l-1}^T r_{l-1}$
        $p_{l+1} = r_l + \beta_{l+1} p_l$ .
```

We note that the storage requirements for CG do not depend upon l_{max}, in contrast to Arnoldi's Method and GMRES. For a modern treatment of the CG method, see Golub and Van Loan [10].

3. Nonlinear Convergence Theory

For all of the algorithms considered in Sec. 2(b) the matrix A is not needed explicitly. All one needs is to be able to calculate matrix-vector products of the form Av, for any vector v. Since $A = F'(\bar{x})$ for $\bar{x}$ an approximation to a root of (2.1), the matrix-vector products

Av in the above algorithms can be replaced by difference quotients of the form

$$F'(\bar{x})v \approx [F(\bar{x}+\sigma v) - F(\bar{x})]/\sigma, \quad \sigma \textit{ a scalar}.$$

The resulting algorithms will be referred to as Finite-Difference Projection methods. In [2], Brown has given a convergence theory for the combined Inexact-Newton/Finite-Difference Projection methods which result when using finite-difference versions of Arnoldi's Algorithm and GMRES to approximately solve the Newton equations (2.2). We summarize these results in this section, and then present a similar theory for a finite-difference version of the Conjugate Gradient Method.

(a) Finite-Difference Arnoldi and GMRES

In this subsection we present a finite-difference version of Arnoldi's Algorithm. We show how to relate the results of this algorithm applied to $Ax = b$ to that of applying the regular Arnoldi method (Algorithm 2.1) to the perturbed problem

$$(A+E)x = b,$$

where $b = -F(\bar{x})$ and $A = F'(\bar{x})$. We then state a result relating the size of the residual for the finite-difference algorithm to the sizes of the σ 's in the difference quotients. For more details, the reader is referred to the paper by Brown [2].

A finite-difference version of Algorithm 2.1 can be given as:

Algorithm 3.1 (Finite-Difference Arnoldi):

```
1. For $\hat{x}_o$ an initial guess, form $q_o = [F(\bar{x}+\sigma_o\hat{x}_o) - F(\bar{x})]/\sigma_o$ .
   Set $\hat{r}_o = b - q_o$, $\hat{v}_1 = \hat{r}_o/\|\hat{r}_o\|_2$, $q_1 = \hat{v}_1$.
2. For $l = 1, \cdots$ do:
   (a) Form $q_{l+1} = [F(\bar{x}+\sigma_l\hat{v}_l) - F(\bar{x})]/\sigma_l$
```

(b) Set $\hat{w}_{l+1} = q_{l+1} - \sum_{i=1}^{l} \hat{h}_{il}\hat{v}_i,\ \hat{h}_{il} = q_{l+1}^T\hat{v}_i\ (i = 1, \cdots, l)$

$\hat{h}_{l+1,l} = \|\hat{w}_{l+1}\|_2,\ \hat{v}_{l+1} = \hat{w}_{l+1}/\hat{h}_{l+1,l}$.

After l steps of this algorithm have been taken, one can compute the approximation

$$\hat{x}_l = \hat{x}_o + \hat{V}_l\hat{H}_l^{-1}\hat{V}_l^T\hat{r}_o,$$

assuming $\hat{H}_l^{-1}$ exists. The hat's are used to distinguish the resulting vectors and scalars from those obtained using regular Arnoldi, Algorithm 2.1.

If l steps of Algorithm 3.1 are possible (that is, $\hat{w}_i \neq 0\ for\ i = 2, \cdots, l$) , then the vectors $q_1, \cdots, q_l$ are linearly independent and the vectors $\hat{v}_1, \cdots, \hat{v}_l$ are an orthonormal basis for the subspace $\hat{K}_l =\ span\ (q_1, \cdots, q_l)$. This follows from the fact that step 2(a) above is simply a Gram-Schmidt orthonormalization procedure. Next, let the errors in the finite-difference quotients be given by

$$\epsilon_i = q_{i+1} - A\hat{v}_i \quad (i = 1, \cdots, l). \tag{3.1}$$

If we let the $N \times N$ matrix E_l be given by

$$E_l = \epsilon^l\hat{V}_l^T, \tag{3.2}$$

where $\epsilon^l = [\epsilon_1,...,\epsilon_l]\epsilon R^{N\times l}$ and $\hat{V}_l = [\hat{v}_1,...,\hat{v}_l]$, then

$$(A + E_l)\hat{v}_i = q_{i+1}\ for\ i = 1, \cdots, l. \tag{3.3}$$

In [2], Brown has shown that applying Algorithm 2.1 (regular Arnoldi) to the perturbed problem

$$(A + E_l)z = \hat{r}_o, \tag{3.4}$$

with $z_o = 0$, is equivalent to applying Algorithm 3.1 to the unperturbed problem

$$Az = \hat{r}_o, \tag{3.5}$$

again with $\hat{z}_o = 0$. (Recall that $Ax = b$ is equivalent to the system $Az = r_o$, where $r_o = b - Ax_o$ and $x = x_o + z$.) It follows that if $\hat{z}_l$ is defined by

$$\hat{z}_l = \hat{V}_l \hat{H}_l^{-1} \hat{V}_l^T \hat{r}_o = \hat{\beta} \hat{V}_l \hat{H}_l^{-1} e_1,$$

with $\hat{\beta} = \|\hat{r}_o\|_2$, then $\hat{z}_l$ can be viewed as an approximate solution to the perturbed problem (3.4) generated by applying Algorithm 2.1 for l steps. In addition, equation (2.16) holds for the perturbed problem (3.4).

The residual associated with $\hat{x}_l = \hat{x}_o + \hat{z}_l$ viewed as an approximate solution of the linear system $Ax = b$ can be expressed as

$$\begin{aligned} \bar{r}_l &= b - A\hat{x}_l = b - A\hat{x}_o - A\hat{z}_l \\ &= [(b - A\hat{x}_o) - \hat{r}_o] + [\hat{r}_o - (A + E_l)\hat{z}_l] + E_l \hat{z}_l, \end{aligned}$$

or

$$\bar{r}_l = \epsilon_o + [\hat{r}_o - (A + E_l)\hat{z}_l] + E_l \hat{z}_l, \tag{3.6}$$

where $\epsilon_o = q_o - A\hat{x}_o = b - A\hat{x}_o - \hat{r}_o$. Thus, the residual $\bar{r}_l$ can be thought of as being composed of three types of errors: the first term on the right hand side of (3.6) representing the error in $\hat{r}_o$, the second term representing the errors in solving the perturbed system (3.4), and the last representing the errors in the difference quotients. For the implemented algorithms in the ODE context described later, we always assume $\hat{x}_o = 0$. This then gives $\epsilon_o = 0$. We now state a slightly modified version of a result in [2].

Theorem 3.1: Let $\bar{x}$ be an approximation to a solution x^* of (2.1), where $F'(\bar{x})$ is nonsingular with F' Lipschitz continuous with constant γ on a convex neighborhood D in R^N containing $\bar{x}$ and x^* . Consider the linear system

$$Ax = b,$$

where $A = F'(\bar{x})$ and $b = -F(\bar{x})$. Let $\hat{x}_o = 0$ and let $\delta > 0$ be given. Choose $\beta > 0$ small enough that

$$\beta\gamma\|A^{-1}\|_2 < 1, \quad \beta\gamma\|A^{-1}\|_2\|b\|_2 \leq \frac{\delta}{2}, \tag{3.7}$$

and

$$\bar{x} + v\epsilon D \textit{ for all } v\epsilon R^N \textit{ with } \|v\|_2 \leq \beta.$$

Let $\sigma^N = (\sigma_1,...,\sigma_N)^T \epsilon R^N$ be chosen so that $\|\sigma^N\|_2 \leq \beta$.

Then for at least one $l\epsilon\{1,\cdots,N\}$ the Finite-Difference Arnoldi iterate $\hat{x}_l$ given by

$$\hat{x}_l = \|\hat{r}_o\|_2 \hat{V}_l \hat{H}_l^{-1} e_1,$$

exists and satisfies

$$\|b - A\hat{x}_l\|_2 \leq \delta.$$

This result says that for $\sigma_i (i = 1,\cdots,N)$ chosen small enough, the approximation computed by Algorithm 3.1 can be made as accurate as desired. Unfortunately, nothing can be said about the size of l required for a given δ , as Example 2.1 shows. However, even when the residual norm exceeds δ for the final $\hat{x}_l$ computed ($l = l_{max}$) , it may still be possible to continue. This follows from the fact that the regular Arnoldi iterate x_l is always a descent direction for the full nonlinear problem. If we define the function

$$g(x) = \frac{1}{2}F(x)^T F(x),$$

then a *descent direction* for g at the current approximation $\bar{x}$ is any vector p such that

$$\nabla g(\bar{x})^T p < 0,$$

where $\nabla g(\bar{x}) = (\frac{\partial g}{\partial x_1}(\bar{x}),\cdots,\frac{\partial g}{\partial x_N}(\bar{x}))^T$. Since $\nabla g(\bar{x}) = F'(\bar{x})^T F(\bar{x}) = -A^T b, p$ is a descent direction for g at $\bar{x}$ if

$$-b^T A p < 0.$$

For such a direction it follows that $g(\bar{x} + \lambda p) < g(\bar{x})$ for all λ small enough. Brown [2] has shown, when $x_o = 0$ and the Arnoldi iterate x_l exists, that

$$-b^T A x_l = -b^T b < 0 \; for \; all \; l = 1, \cdots, N.$$

Hence, x_l is always a descent direction for g at $\bar{x}$. For Algorithm 3.1, when $\hat{x}_o = 0$ and $\hat{x}_l$ exists we have

$$-b^T A \hat{x}_l = -b^T b + b^T \epsilon^l \hat{y}_l.$$

Hence, if the ϵ_i's are small enough, then $\hat{x}_l$ will also be a descent direction for g at $\bar{x}$. See Brown [2] for more details.

In practice one would not try to enforce condition (3.7), which may be overly restrictive on some problems. To see this, note that $F' = I - h\beta_o J$ in the ODE setting, and so the Lipschitz constant γ for F' is simply $h\beta_o\gamma_f$, where γ_f is the Lipschitz constant for $\partial f/\partial y$, the ODE system Jacobian. For a typical stiff problem γ_f can be quite large, and so trying to force (3.7) to hold would result in too small a value of h. This is the price one pays for using norms in the analysis, as the Lipschitz constant γ measures the worst-case nonlinearity in F. A better approach is to choose the σ_i so that the errors in the finite-difference quotients are small. See Brown [2] for more details, and Dennis and Schnabel [7] for more general information on computing finite-difference derivative approximations. Since Arnoldi and GMRES both use the same Arnoldi process, a completely similar theory is true for the finite-difference version of GMRES. We refer the reader to [2] for more details.

(b) Finite-Difference Conjugate Gradient

In [2], Brown has presented a convergence theory for a finite-difference version of the Generalized Conjugate Residual method of Eisenstat, Elman and Schultz [8]. Here, we show how a modification of this theory will allow us to prove a result similar to Theorem 3.1 for a finite-difference version of the Conjugate Gradient Method. The main tool will again relate the results of the finite-difference algorithm to that of applying the regular algorithm to a perturbed problem.

If we replace the matrix-vector products in Algorithm 2.3, then we have the following:

Algorithm 3.2 (Finite-Difference Conjugate Gradient)

```
1. Set $\hat{w}_o = [F(\bar{x} + \sigma_o \hat{x}_o) - F(\bar{x})]/\sigma_o$ and let $\hat{r}_o = b - \hat{w}_o$ .
   Set $\hat{p}_1 = \hat{r}_o$ .
2. For $l = 1, 2, \cdots$ do:
   $\hat{w}_l = [F(\bar{x} + \sigma_l \hat{p}_l) - F(\bar{x})]/\sigma_l$
   $\hat{\alpha}_l = \hat{r}_{l-1}^T \hat{r}_{l-1} / \hat{p}_l^T \hat{w}_l$
   $\hat{x}_l = \hat{x}_{l-1} + \hat{\alpha}_l \hat{p}_l$
   $\hat{r}_l = \hat{r}_{l-1} - \hat{\alpha}_l \hat{w}_l$
   $\hat{\beta}_{l+1} = \hat{r}_l^T \hat{r}_l / \hat{r}_{l-1}^T \hat{r}_{l-1}$
   $\hat{p}_{l+1} = \hat{r}_l + \hat{\beta}_{l+1} \hat{p}_l.$
```

The hat's above are again used to distinguish the resulting vectors and scalars from those obtained using regular Conjugate Gradient, Algorithm 2.3. Note that once $\hat{x}_o$ has been chosen and $\hat{r}_o$ is defined, Algorithm 3.2 applied to $Ax = b$ is the same as applying it to

$$Az = \hat{r}_o, \text{ with } x = x_o + \hat{z} \text{ and } \hat{z}_o = 0. \tag{3.8}$$

Suppose l steps of Algorithm 3.2 are possible and the vectors $\hat{p}_1, ..., \hat{p}_l$ are linearly independent. Define $\hat{P}_l = [\hat{p}_1, ..., \hat{p}_l] \epsilon R^{N \times l}$ and note that $\hat{P}_l$ has full column rank. Let

$$\epsilon_i = \hat{w}_i - A\hat{p}_i \; (i = 1, \cdots, l)$$

be the errors in the difference-quotient approximations, and let the $N \times N$ matrix E_l be given by

$$E_l = \epsilon^l (\hat{P}_l^T \hat{P}_l)^{-1} \hat{P}_l^T, \tag{3.9}$$

where $\epsilon^l = [\epsilon_1, \cdots, \epsilon_l] \epsilon R^{N \times l}$. Then

$$\hat{w}_i = (A + E_l)\hat{p}_i \;\; (i = 1, \cdots, l).$$

We can then define the perturbed problem

$$(A + E_l)z = \hat{r}_o. \tag{3.10}$$

Note that E_l is not likely to be symmetric and at the moment $A + E_l$ is not necessarily even positive definite.

We next give a result which shows that applying Algorithm 3.2 to the reformulated problem (3.8) with $\hat{z}_o = 0$ is equivalent to applying Algorithm 2.3 to the perturbed problem (3.10) with $z_o = 0$.

Theorem 3.2: Assume that l steps of Algorithm 3.2 applied to (3.8) have been possible. Let $\hat{R}_l = [\hat{r}_o, ..., \hat{r}_l], \hat{W}_l = [\hat{w}_1, ..., \hat{w}_l], \hat{P}_{l+1} = [\hat{p}_1, ..., \hat{p}_{l+1}], \hat{\alpha}_1, ..., \hat{\alpha}_l$ and $\hat{\beta}_2, ..., \hat{\beta}_{l+1}$ be the resulting scalars and vectors, and let $\hat{z}_l$ be the approximate solution to (3.8). If $\hat{P}_l = [\hat{p}_1, ..., \hat{p}_l]$ has full column rank, then l steps of Algorithm 3.2 applied to (3.10) are possible. If $R_l, W_l, P_{l+1}, \alpha_1, ..., \alpha_l, \beta_2, ..., \beta_{l+1}$ and z_l are the resulting scalars and vectors, then

$$R_l = \hat{R}_l, W_l = \hat{W}_l, P_{l+1} = \hat{P}_{l+1}, \alpha_i = \hat{\alpha}_i (i = 1, \cdots, l), \beta_i = \hat{\beta}_i (i = 2, \cdots, l+1), \tag{3.11}$$

$$z_l = \hat{z}_l \; \textit{and}$$

$$\hat{r}_l = \hat{r}_o - (A + E_l)\hat{z}_l, \text{ with } E_l \text{ given by } (3.9)\ . \tag{3.12}$$

Proof: The results follow from the fact that $\hat{w}_i = (A + E_l)\hat{p}_i$ $(i = 1, \cdots, l)$, and simply from the structure of the algorithms. QED

From relations (3.6) and (3.12) we next have that

$$\bar{r}_l = \epsilon_o + \hat{r}_l + E_l\hat{z}_l,$$

and so

$$\bar{\rho}_l \equiv \|\bar{r}_l\|_2 \le \|\epsilon_o\|_2 + \|\hat{r}_l\|_2 + \|E_l\hat{z}_l\|_2. \tag{3.13}$$

From (3.9),

$$\|E_l\|_2 \le \|\epsilon^l\|_2 \cdot \|(\hat{P}_l^T\hat{P}_l)^{-1}\hat{P}_l^T\|_2,$$

and so we must find bounds for $\|\epsilon^l\|_2$ and $\|(\hat{P}_l^T\hat{P}_l)^{-1}\hat{P}_l^T\|_2$. To bound the norm of ϵ^l , assume that F' is Lipschitz continuous with constant γ in some neighborhood D in R^N of a solution of (2.1). Then by Lemma 4.1.12 in Dennis and Schnabel [7] for each $i = 1, \cdots, l$

$$\|F(\bar{x} + \sigma_i\hat{p}_i) - F(\bar{x}) - F'(\bar{x})\sigma_i\hat{p}_i\|_2 \le \frac{\gamma}{2}|\sigma_i|^2\|\hat{p}_i\|_2^2$$

as long as $\bar{x} + \sigma_i\bar{p}_i \epsilon D$. Thus,

$$\|\epsilon_i\|_2 \le \|\hat{w}_i - A\hat{p}_i\|_2 \le \frac{\gamma}{2}|\sigma_i| \cdot \|\hat{p}_i\|_2^2 \ (i = 1, \cdots, l). \tag{3.14}$$

Letting $\|\cdot\|_F$ denote the Frobenius norm of a matrix and $\sigma^l = (\sigma_1, ..., \sigma_l)^T \epsilon R^l$,

$$\|\epsilon^l\|_2 \le \|\epsilon^l\|_F = \left[\sum_{i=1}^{l}\|\epsilon_i\|_2^2\right]^{\frac{1}{2}} \le \frac{\gamma}{2}\left(\sum_{i=1}^{l}\sigma_i^2\|\hat{p}_i\|_2^4\right)^{\frac{1}{2}}.$$

Now, since $\hat{P} = [\hat{p}_1, ..., \hat{p}_l]$, we have $\|\hat{p}_i\|_2 \le \|\hat{P}_l\|_2$ for $i = 1, \cdots, l$. Hence,

$$\|\epsilon^l\|_2 \le \frac{\gamma}{2}\|\hat{P}_l\|_2^2\|\sigma^l\|_2,$$

and so from (3.13) we have

$$\bar{\rho}_l \leq \|\epsilon_o\|_2 + \|\hat{r}_l\|_2 + \frac{\gamma}{2}\|\hat{P}_l\|_2^2\|\sigma^l\|_2 \cdot \|(\hat{P}_l^T\hat{P}_l)^{-1}\hat{P}_l^T\|_2 \cdot \|\hat{z}_l\|_2.$$

Similarly, from the Lipschitz continuity of F',

$$\|\epsilon_o\|_2 \leq \frac{\gamma}{2}|\sigma_o| \cdot \|\hat{x}_o\|_2^2,$$

and we can write

$$\hat{\rho}_l \leq \frac{\gamma}{2}|\sigma_o| \cdot \|\hat{x}_o\|_2^2 + \|\hat{r}_l\|_2 + \frac{\gamma}{2}\|\hat{P}_l\|_2^2\|\sigma^l\|_2 \cdot \|(\hat{P}_l^T\hat{P}_l)^{-1}\hat{P}_l^T\|_2 \cdot \|\hat{z}_l\|_2. \tag{3.15}$$

We will use inequality (3.15) in the proof of Theorem 3.6 below.

To bound $\|(\hat{P}_l^T\hat{P}_l)^{-1}\hat{P}_l^T\|_2$, we begin with a review of some of the properties of the Conjugate Gradient Method, and then introduce a definition and prove some technical lemmas.

Theorem 3.3: Consider the linear system $Ax = b$, where A is symmetric and positive definite. Suppose that l steps of Algorithm 2.3 are possible with $r_l \neq 0$. Then the following are true:

$$r_i^T r_j = 0 \; for \; i \neq j, \;\; (i,j = 0,\cdots,l) \tag{3.16a}$$

$$r_l^T p_i = 0 \; for \; i = 1,\cdots,l \tag{3.16b}$$

$$p_l^T A p_i = 0 \; for \; i = 1,\cdots,l-1 \tag{3.16c}$$

$$span\;(p_1,\ldots,p_{l+1}) = \; span\;(r_o,\ldots,r_l) = \; span\;(r_o, Ar_o,\ldots, A^l r_o) \tag{3.16d}$$

$$r_l \neq 0 \; implies \; p_{l+1} \neq 0 \tag{3.16e}$$

The Conjugate Gradient iterates x_l converge to the exact solution $A^{-1}b$ in at most N iterations. (3.16f)

Proof: See Golub and Van Loan [10]. QED

Let the function $d_k : R^{N\times k} \rightarrow R$ be defined by

$$d_k(B) = \; min\;(\|Bq\|_2 : q\epsilon R^k \; with \; \|q\|_2 = 1),$$

for any $N \times k$ matrix B . Clearly, $d_k(B) > 0$ if and only if B has full column rank. Additionally, if $b \epsilon R^N$, then $d_k(B) \geq d_{k+1}([B,b])$ for $[B,b] \epsilon R^{N\times(k+1)}$. Furthermore, if B has full column rank, then

$$\|(B^T B)^{-1} B\|_2 = 1/d_k(B). \tag{3.17}$$

For a proof of this last fact, see Brown [2].

Lemma 3.4: Consider solving $Ax = b$ using Algorithm 2.3 where A is symmetric and positive definite. Assume that l steps have been taken with $r_l \neq 0$. Let $R_l = [r_o,...,r_l], W_l = [w_1,...,w_l]$ and $P_{l+1} = [p_1,...,p_{l+1}]$. Then

$$\|r_o\|_2 \geq d_{k+1}(R_k) \geq d_{l+1}(R_l) > 0 \ (k = 0, \cdots, l), \tag{3.18a}$$

$$\|w_1\|_2 \geq d_k(W_k) \geq d_l(W_l) > 0 (k = 1, \cdots, l), \ \textit{and} \tag{3.18b}$$

$$\|p_1\|_2 \geq d_k(P_k) \geq d_{l+1}(P_{l+1}) > 0 \ (k = 1, \cdots, l+1). \tag{3.18c}$$

Proof: If any $r_i = 0$ with $i < l$, then Algorithm 2.3 would have given x_i as the exact solution and the algorithm would have terminated at step $i < l$. Thus $r_i \neq 0$ for all $i = 0, 1, \cdots, l$. By (3.16a) all the r_i are orthogonal and so $d_{l+1}(R_l) > 0$. Since $d_1(R_o) = \|r_o\|_2$, we have

$$\|r_o\|_2 \geq d_2(R_1) \geq d_3(R_2) \cdots \geq d_{l+1}(R_l) > 0,$$

which gives (3.18a). By (3.16d), $d_{l+1}(P_{l+1}) > 0$ which immediately gives (3.18c). Since $W_l = AP_l$, if $W_l y = 0$ for some $y \epsilon R^l$, then $AP_l y = 0$ which implies $P_l y = 0$ since A is nonsingular. Next, $P_l y = 0$ implies $y = 0$ since P_l has full column rank. Thus, $d_l(W_l) > 0$ which gives (3.18b). QED

Lemma 3.5: Let $\bar{x}$ be an approximation to a root x^* of (2.1), where $F'(\bar{x})$ is symmetric and positive definite, with F' Lipschitz continuous with constant γ on a convex neighborhood D in R^N containing $\bar{x}$ and x^* . Consider solving

$$Ax = b, \textit{ where } A = F'(\bar{x}) \textit{ and } b \textit{ is arbitrary }, \tag{3.19}$$

using Algorithm 2.3 with initial guess $x_o = 0$. Assume that l steps have been taken with $r_l \neq 0$, and let $w_1 = Ap_1$ and $P_{l+1} = [p_1,...,p_{l+1}]$. Then for each ϵ in $(0, \epsilon')$, where $\epsilon' = d_{l+1}(P_{l+1})$, there exists a $\tau > 0$ such that

$$l+1 \textit{ steps of Algorithm } 3.2 \textit{ applied to } (3.19) \textit{ with } \hat{x}_o = 0 \textit{ are possible}, \tag{3.20}$$

$$d_{l+1}(\hat{P}_{l+1}) \geq d_{l+1}(P_{l+1}) - \epsilon > 0, \tag{3.21}$$

$$\|\hat{R}_{l+1} - R_{l+1}\|_2 < \epsilon, \quad \|\hat{P}_{l+1} - P_{l+1}\|_2 < \epsilon, \textit{ and} \tag{3.22}$$

$$A + E_{l+1} \textit{ is positive definite with} \tag{3.23}$$

$$\|E_{l+1}\|_2 \leq \frac{\tau\gamma}{2}(\epsilon + \|P_{l+1}\|_2)^2/(d_{l+1}(P_{l+1}) - \epsilon)$$

whenever $\|\sigma^{l+1}\|_2 \leq \tau$.

Proof: Let $\epsilon > 0$ be chosen so that

$$\epsilon < \epsilon'. \tag{3.24}$$

Next, let $\mu > 0$ be chosen so that $\bar{x} + v\epsilon D$ whenever $\|v\|_2 < \mu$. Then choose $\tau > 0$ so that

$$\tau < \mu/(\epsilon + \|P_{l+1}\|_2), \textit{ and} \tag{3.25}$$

$$\frac{\tau\gamma}{2}(\epsilon + \|P_{l+1}\|_2)^2/(d_{l+1}(P_{l+1}) - \epsilon) < \lambda_{min}(A) , \tag{3.26}$$

where $\lambda_{min}(A)$ is the smallest eigenvalue of A . Note that since x_o and $\hat{x}_o$ are both zero, $r_o = \hat{r}_o = p_1 = \hat{p}_1 = b$. Further, $b \neq 0$ since $r_l \neq 0$, and by Lemma 3.4, (3.18a) - (3.18c) hold. From Algorithm 3.2 and the Lipschitz continuity of F', $\hat{w}_1 = \hat{w}_1(\sigma_1)$ is a continuous function of σ_1 in a neighborhood of $\sigma_1 = 0$, and $\hat{w}_1 \to w_1$ as $\sigma_1 \to 0$. For $|\sigma_1| \leq \tau$ define

$$\epsilon_1 = \hat{w}_1 - A\hat{p}_1 \textit{ and } E_1 = \epsilon_1(\hat{p}_1^T\hat{p}_1)^{-1}\hat{p}_1^T,$$

so that

$$(A + E_1)\hat{p}_1 = \hat{w}_1.$$

Using (3.25), $\bar{x} + \sigma_1\hat{p}_1 \epsilon D$ since $\|\sigma_1\hat{p}_1\|_2 = |\sigma_1| \cdot \|p_1\|_2 \leq \tau(\epsilon + \|P_{l+1}\|_2) < \mu$.

Thus, by (3.14)

$$\|\epsilon_1\|_2 \leq \frac{\gamma}{2}|\sigma_1| \cdot \|\hat{p}_1\|_2^2 \quad \leq \frac{\gamma}{2}|\sigma_1| \cdot (\epsilon + \|P_{l+1}\|_2)^2.$$

Using (3.18c), we have

$$\|E_1\|_2 \leq \frac{\tau\gamma}{2}(\epsilon + \|P_{l+1}\|_2)^2/\|p_1\|_2$$

$$\leq \frac{\tau\gamma}{2}(\epsilon + \|P_{l+1}\|_2)^2/(d_{l+1}(P_{l+1}) - \epsilon),$$

whenever $|\sigma_1| \leq \tau$. By (3.26) $\|E_1\|_2 < \lambda_{min}(A)$, and so $A + E_1$ is positive definite. This then implies

$$\hat{p}_1^T\hat{w}_1 = \hat{p}_1^T(A + E_1)\hat{p}_1 > 0,$$

since $\hat{p}_1 \neq 0$. Thus, $\hat{\alpha}_1, \hat{x}_1, \hat{r}_1, \hat{\beta}_2$ and $\hat{p}_2$ can be formed, and it is clear that each of these is a continuous function of σ_1 for σ_1 near zero. Furthermore,

$$\alpha_1 \to \alpha_1,\ \hat{x}_1 \to x_1,\ \hat{r}_1 \to r_1,\ \hat{\beta}_2 \to \beta_2 \ \textit{ and } \hat{p}_2 \to p_2 \ \textit{ as } \sigma_1 \to 0.$$

Hence, by choosing τ smaller, if necessary, we have

$$\|\hat{R}_1 - R_1\|_2 < \epsilon \textit{ and } \|\hat{P}_2 - P_2\|_2 < \epsilon \textit{ whenever } |\sigma_1| \leq \tau. \tag{3.27}$$

Now, for any $\alpha \epsilon R^2$ with $\|\alpha\|_2 = 1$, we have

$$\|\hat{P}_2\alpha\|_2 = \|P_2\alpha + (\hat{P}_2 - P_2)\alpha\|_2$$

$$\geq \|P_2\alpha\|_2 - \|\hat{P}_2 - P_2\|_2$$

$$\geq d_2(P_2) - \epsilon \ \geq d_{l+1}(P_{l+1}) - \epsilon \ > 0$$

by (3.18c), (3.24) and (3.27). This gives $d_2(\hat{P}_2) \geq d_{l+1}(P_{l+1}) - \epsilon > 0$ and so $\hat{P}_2^T\hat{P}_2$ is nonsingular with $\hat{p}_2 \neq 0$. Next, form $\hat{w}_2$ and define

$$\epsilon_2 = \hat{w}_2 - A\hat{p}_2 \ and \ E_2 = \epsilon^2(\hat{P}_2^T\hat{P}_2)^{-1}\hat{P}_2^T, where \ \epsilon^2 = [\epsilon_1, \epsilon_2].$$

Then by (3.25), $\bar{x} + \sigma_2\hat{p}_2\epsilon D$ since

$$\|\sigma_2\hat{p}_2\|_2 \leq |\sigma_2| \cdot \|\hat{P}_2\|_2 \leq \tau(\epsilon + \|P_2\|_2) \leq \tau(\epsilon + \|P_{l+1}\|_2) < \mu.$$

Thus, by (3.14)

$$\|\epsilon_i\|_2 \leq \frac{\gamma}{2}|\sigma_i| \ \|\hat{p}_i\|_2^2 \ \leq \frac{\gamma}{2}|\sigma_i|(\epsilon + \|P_{l+1}\|_2)^2 \qquad (i = 1, 2)$$

whenever $\|\sigma^2\|_2 \leq \tau$, where $\sigma^2 = (\sigma_1, \sigma_2)^T \epsilon R^2$. The bound for $\|E_2\|_2$ is then

$$\|E_2\|_2 \leq \|\epsilon^2\|_2/d_2(\hat{P}_2) \ \leq \|\epsilon^2\|_F/d_2(\hat{P}_2)$$

$$\leq \frac{\gamma}{2}\|\sigma^2\|_2(\epsilon + \|P_{l+1}\|_2)^2/(d_{l+1}(P_{l+1}) - \epsilon)$$

$$\leq \frac{\tau\gamma}{2}(\epsilon + \|P_{l+1}\|_2)^2/(d_{l+1}(P_{l+1} - \epsilon)$$

whenever $\|\sigma^2\|_2 \leq \tau$. By (3.26), $A + E_2$ is positive definite, and so

$$\hat{p}_2^T\hat{w}_2 = \hat{p}_2^T(A + E_2)\hat{p}_2 > 0,$$

since $\hat{p}_2 \neq 0$. Thus $\hat{\alpha}_2, \hat{x}_2, \hat{r}_2, \hat{\beta}_3$ and $\hat{p}_3$ can all be formed, and again it is clear that each is a continuous function of the vector σ^2 for σ^2 near zero, with

$$\hat{\alpha}_2 \rightarrow \alpha_2, \ \hat{x}_2 \rightarrow x_2, \ \hat{r}_2 \rightarrow r_2, \ \hat{\beta}_3 \rightarrow \beta_3 \ \ and \ \hat{p}_3 \rightarrow p_3 \ \ as \ \sigma^2 \rightarrow 0.$$

By choosing τ smaller, if necessary, we have

$$\|\hat{R}_2 - R_2\|_2 < \epsilon \textit{ and } \|\hat{P}_3 - P_3\|_2 < \epsilon \textit{ whenever } \|\sigma^2\|_2 \leq \tau.$$

This last inequality implies $d_3(\hat{P}_3) \geq d_{l+1}(P_{l+1}) - \epsilon > 0$ in a way similar to that for $\hat{P}_2$, and so $\hat{P}_3^T\hat{P}_3$ is nonsingular with $\hat{p}_3 \neq 0$. Now form $\hat{w}_3$ and define

$$\epsilon_3 = \hat{w}_3 - A\hat{p}_3 \textit{ and } E_3 = \epsilon^3(\hat{P}_3^T\hat{P}_3)^{-1}\hat{P}_3^T, \textit{ where } \epsilon^3 = [\epsilon_1, \epsilon_2, \epsilon_3,].$$

Similarly, $\|E_3\|_2$ satisfies the same bound as that for $\|E_2\|_2$, and so $A + E_3$ is positive definite.

This process can be continued for $l-2$ more steps since $\hat{p}_i \neq 0$ for $i = 1, \cdots, l+1$, and choosing τ smaller if necessary at each stage. We note that since $r_l \neq 0$, it is possible to compute x_{l+1}, r_{l+1} and p_{l+2} using the Conjugate Gradient Method (Algorithm 2.3). Also, the fact that $\hat{p}_{l+1} \neq 0$ allows the computation of $\hat{w}_{l+1}$ and $E_{l+1} = \epsilon^{l+1}(\hat{P}_{l+1}^T\hat{P}_{l+1})^{-1}\hat{P}_{l+1}^T$ with

$$\hat{p}_{l+1}^T\hat{w}_{l+1} = \hat{p}_{l+1}^T(A + E_{l+1})\hat{p}_{l+1} > 0$$

for τ chosen smaller if necessary. Thus $\hat{r}_{l+1}$ exists and is a continuous function of σ^{l+1} near $\sigma^{l+1} = 0$ with $\hat{r}_{l+1} \to r_{l+1}$ as $\sigma^{l+1} \to 0$. Thus, by choosing τ small enough we will have

$$d_{l+1}(\hat{P}_{l+1}) \geq d_{l+1}(P_{l+1}) - \epsilon > 0$$

$$\|\hat{R}_{l+1} - R_{l+1}\|_2 < \epsilon, \ \|\hat{P}_{l+1} - P_{l+1}\|_2 < \epsilon \ ,$$

$$A + E_{l+1} \textit{ is positive definite, and}$$

$$\|E_{l+1}\|_2 \leq \frac{\tau\gamma}{2}(\epsilon + \|P_{l+1}\|_2)^2/(d_{l+1}(P_{l+1}) - \epsilon)$$

for all $\|\sigma^{l+1}\|_2 \leq \tau$. QED

The next theorem is the main result for the finite-difference CG method, and is the analogue for the CG method of Theorem 3.1 on Arnoldi's method.

Theorem 3.6: Let $\bar{x}$ be an approximation to a root x^* of (2.1), where $F'(\bar{x})$ is symmetric and positive definite, and F' is Lipschitz continuous with constant γ in a neighborhood D in R^N of x^* containing $\bar{x}$. Consider solving the linear system

$$Ax = b, \tag{3.28}$$

where $A = F'(\bar{x})$ and $b = -F(\bar{x})$. Let $\delta > 0$ be given. Then there exist constants $\alpha, \tau > 0$ and an integer $L \leq N$ such that the Finite-Difference CG iterates $\hat{x}_l$ exist for $l = 0, 1, \cdots, L$ and satisfy

$$\bar{\rho}_l = \|b - A\hat{x}_l\|_2 \leq \delta \tag{3.29}$$

for at least one $l \leq L$, whenever $\|\sigma^L\|_2 \leq \tau$, $\sigma^L = (\sigma_{1,\ldots,}\sigma_L)^T \epsilon R^L$, and $|\sigma_o| \leq \alpha$.

Proof: Let $\hat{x}_o$ be an initial guess for the solution $A^{-1}b$ and form $\hat{r}_o = b - \hat{w}_o$, where

$$\hat{w}_o = [F(\bar{x} + \sigma_o\hat{x}_o) - F(\bar{x})]/\sigma_o,$$

where $|\sigma_o| \leq \alpha$ and $\alpha > 0$ is chosen so that

$$\alpha\gamma|||\hat{x}_o\|_2^2 \leq \delta \textit{ and } \bar{x} + \sigma_o\hat{x}_o\epsilon D \textit{ whenever } |\sigma_o| \leq \alpha. \tag{3.30}$$

If $\hat{r}_o = 0$, then $b = \hat{w}_o$, and so

$$\bar{\rho}_o = \|b - A\hat{x}_o\|_2 \leq \frac{\gamma}{2}|\sigma_o|\|\hat{x}_o\|_2^2 \leq \delta/2$$

using (3.30) and the Lipschitz continuity of F' . Thus, if $\hat{r}_o = 0$ we are finished.

For $\hat{r}_o \neq 0$, let $x = \hat{x}_o + z$. Then applying Algorithm 3.2 to the problem

$$Az = \hat{r}_o, \ \hat{z}_o = 0, \tag{3.31}$$

is equivalent to applying Algorithm 3.2 to (3.28) with initial guess $\hat{x}_o$. Next, apply Algorithm 2.3 to (3.31) with $z_o = 0$. Let $L \leq N$ be the maximum value of l such that the CG iterates $\{ z_l \}$ exist for $l = 1, \cdots, L$ and satisfy

$$\|r_l\|_2 = \|\hat{r}_o - Az_l\|_2 > 0 \quad for\ l = 1, \cdots, L-1.$$

Since L is the maximum such value, $z_L = A^{-1}\hat{r}_o$ and $r_L = 0$. Lemma 3.5 applied to (3.31), with $b = \hat{r}_o$ and x replaced by z , then implies that for each ϵ in $(0, \epsilon')$, where $\epsilon' = d_L(P_L)$, there exists a $\tau > 0$ such that:

$$L\ steps\ of\ Algorithm\ 3.2\ applied\ to\ (3.31)\ with\ \hat{z}_o = 0\ are\ possible, \tag{3.32}$$

$$d_L(\hat{P}_L) \geq d_L(P_L) - \epsilon > 0, \tag{3.33}$$

$$\|\hat{R}_L - R_L\|_2 < \epsilon,\ \|\hat{P}_L - P_L\|_2 < \epsilon,\ and \tag{3.34}$$

$$A + E_L\ is\ positive\ definite\ with \tag{3.35}$$

$$\|E_L\|_2 \leq \frac{\tau\gamma}{2}(\epsilon + \|P_L\|_2)^2/(d_L(P_L) - \epsilon) \equiv \tau C$$

whenever $\|\sigma^L\|_2 \leq \tau$. By use of the Perturbation Lemma (see Theorem 3.1.4 in Dennis and Schnabel [7], and the proof of Theorem 3.3 in Brown [2]), τ can be chosen smaller if necessary so that

$$\|(A + E_L)^{-1}\|_2 \leq 2\|A^{-1}\|_2.$$

The residual for $\hat{x}_L = \hat{x}_o + \hat{z}_L$ as a solution to (3.28) is then

$$\bar{r}_L = \epsilon_o + \hat{r}_L + E_L\hat{z}_L.$$

Hence,

$$\bar{\rho}_L = \|\bar{r}_L\|_2 \leq \|\epsilon_o\|_2 + \|\hat{r}_L\|_2 + \|E_L\|_2\|\hat{z}_L\|_2.$$

Next, let z_L^* be the exact solution of the perturbed problem $(A + E_L)z = \hat{r}_o$. Then

$$\|z_L^*\|_2 = \|(A + E_L)^{-1}\hat{r}_o\|_2 \leq 2\|A^{-1}\|_2 \cdot \|\hat{r}_o\|_2.$$

Since

$$\hat{z}_L - z_L^* = (A + E_L)^{-1}[(A + E_L)\hat{z}_L - \hat{r}_o] = -(A + E_L)^{-1}\hat{r}_L,$$

we have

$$\|\hat{z}_L\|_2 \leq \|z_L^*\|_2 + \|(A + E_L)^{-1}\|_2 \cdot \|\hat{r}_L\|_2 \leq 2\|A^{-1}\|_2(\|\hat{r}_o\|_2 + \|\hat{r}_L\|_2).$$

Thus, using (3.15),

$$\begin{aligned} \bar{\rho}_L &\leq \frac{\alpha\gamma}{2}\|\hat{x}_o\|_2^2 + 2\|A^{-1}\|_2 \cdot \tau \cdot C(\|b - A\hat{x}_o\|_2 + \frac{\alpha\gamma}{2}\|\hat{x}_o\|_2^2) \\ &\quad + \|\hat{r}_L\|_2(1 + 2\tau C\|A^{-1}\|_2) \\ &\leq \frac{\delta}{2} + 2\|A^{-1}\|_2\tau C(\|b - A\hat{x}_o\|_2 + \frac{\delta}{2}) + \|\hat{r}_L\|_2(1 + 2\tau C\|A^{-1}\|_2). \end{aligned}$$

Since $\|\hat{r}_L\|_2 \to 0$ as $\|\sigma^L\|_2 \to 0$ we easily see that (3.29) holds by choosing τ smaller, if necessary. QED

As we noted earlier, Arnoldi's Agorithm is equivalent to the Conjugate Gradient Method when A is symmetric and positive definite. Therefore, when $x_o = 0$ the CG interates x_l satisfy

$$-b^T A x_l = -b^T b < 0 \quad for\ l = 1, \cdots, N.$$

Thus, x_l is always a descent direction for $g(x) = \frac{1}{2}F(x)^T F(x)$ at $x = \bar{x}$. Similarly, the finite-difference CG iterates $\hat{x}_l$ will be descent directions (when $\hat{x}_o = 0$) if the ϵ_i's are small enough.

4. Scaling and Preconditioning

The basic linear iteration methods considered in the previous sections are not of much practical value as they stand. As discussed in [4], realistic problems require the inclusion of scale factors, so that all vector norms become weighted norms in the problem variables. However, even the scaled iterative methods seem to be competitive only for a fairly narrow class of problems, characterized mainly by tight clustering in the spectrum of the system Jacobian. Thus for robustness, it is essential to enhance the methods further. As in other contexts involving linear systems, preconditioning of the linear iteration is a natural choice. In what follows, the use of scaling is reviewed, and then preconditioned scaled iteration methods and algorithms are studied in a general setting.

(a) Scaling

The user of an ODE solver must provide parameters that define error tolerances to be imposed on the computed solution. In LSODE, there are relative and absolute tolerances RTOL and ATOL such that the combination

$$w_i = RTOL_i \left| y^i_{n-1} \right| + ATOL_i$$

is applied as a scale factor for component y^i during the time step from t_{n-1} to t_n. Specifically, a weighted root-mean-square norm

$$\|x\|_{WRMS} = \left[N^{-1} \sum_1^N (x^i/w_i)^2 \right]^{1/2}$$

is used on all error-like vectors. Thus if we define a diagonal matrix

$$D = \sqrt{N} \; diag \; (w_1, \cdots, w_N),$$

we can relate this to an L_2 norm:

$$\|x\|_{WRMS} = \|D^{-1}x\|_2.$$

The linear systems $Ps \doteq r$ in (1.5) or (2.2) can be restated in scaled form in terms of $D^{-1}s = \tilde{s}$ and $D^{-1}r = \tilde{r}$. Likewise, the nonlinear system $F(x) = 0$ can be restated in a scaled form $\tilde{F}(\tilde{x}) = 0$. The scaled version of the IOM algorithm, denoted SIOM, is described in detail in [4].

(b) Preconditioned Krylov methods.

When a basic iteration fails to show acceptable convergence on a given problem, preconditioning is often beneficial, especially when the cause of the slow convergence can be identified with one or more parts of the problem which are (individually) easier to deal with than the whole problem. Generally, preconditioning in an iterative method for solving $Ax = b$ means applying the method instead to the equivalent system

$$(P_1^{-1}AP_2^{-1})(P_2x) = (P_1^{-1}b), \quad or \quad \bar{A}\bar{x} = \bar{b}, \tag{4.1}$$

where P_1 and P_2 are matrices chosen in advance. The preconditioned problem is easier to solve than the original problem provided that (1) linear systems $P_1x = c$ and $P_2x = c$ can be solved economically, and (2) the product P_1P_2 is in some way close to A. Condition (1) is essential because carrying out the method on $\bar{A}\bar{x} = \bar{b}$ clearly requires evaluating vectors of the form $P_k^{-1}c$, at the beginning of the iteration, during each iteration, and at the end. Condition (2) is less well-defined, but means simply that the convergence of the method for $\bar{A}\bar{x} = \bar{b}$ should be much better than for $Ax = b$, because $\bar{A}$ is somehow close to the identity matrix (for which convergence is immediate).

The system (4.1) is said to be preconditioned on both sides, with P_1 the left preconditioner and P_2 the right preconditioner. Either matrix could be the identity, and in that case one is preconditioning on the left only or right only, with a single matrix approximating A.

It is essential that the scaling of the linear system (discussed above) be retained in the preconditioned methods. Since the scaling matrix D is based on the tolerance inputs to the ODE solver, D^{-1} can be thought of as removing the physical units from the components of x so that the components of $D^{-1}x$ can be considered dimensionless and mutually comparable. On the other hand, the matrix $A = I - h\beta_o \partial f/\partial y$ is <u>not</u> similarly scaled, and so, because P_1 and P_2 are based on approximating A , the matrix

$$\bar{A} = P_1^{-1} A P_2^{-1}$$

is also not similarly (dimensionally) scaled. More precisely, it is easy to show that if the (i, j) elements of P_1 and P_2 each have the same physical dimension as that of A , i.e. the dimension of y_i/y_j, then so does the (i, j) element of $\bar{A}$. The same is true for the vectors $\bar{x}$ and $\bar{b}$: the *ith* component of each has the same physical dimension as that of y_i. It follows that the diagonal scaling D^{-1} should be applied to $\bar{x}$ and $\bar{b}$ in the same way that it was applied to x and b without preconditioning. Thus we change the system (4.1) again to the equivalent scaled preconditioned system

$$(D^{-1}\bar{A}D)(D^{-1}\bar{x}) = (D^{-1}\bar{b}), \quad \textit{or} \quad \tilde{A}\tilde{x} = \tilde{b}. \tag{4.2}$$

Combining the two transformations, we have

$$\tilde{A} = D^{-1}P_1^{-1}AP_2^{-1}D, \ \tilde{x} = D^{-1}P_2x, \ \tilde{b} = D^{-1}P_1^{-1}b. \tag{4.3}$$

An alternative point of view is to rescale the system first, to

$$(D^{-1}AD)(D^{-1}x) = (D^{-1}b)$$

and then apply preconditioners Q_1, Q_2 to $D^{-1}AD$, to get

$$(Q_1^{-1}D^{-1}ADQ_2^{-1})(Q_2D^{-1}x) = (Q_1^{-1}D^{-1}b).$$

But if the Q_k are unscaled to $P_k = DQ_kD^{-1}$, this system is identical to (4.2).

Consider now one of the Krylov subspace methods applied to the scaled preconditioned problem. These methods all have in common the generation of the Krylov sequence

$$\tilde{K} \equiv \{\tilde{A}^l\tilde{r}_o : l = 0, 1, \cdots\}$$

from the initial residual $\tilde{r}_o$. We can assume that $\tilde{r}_o = \tilde{b} - \tilde{A}\tilde{x}_o$ corresponds to a given initial guess x_o and residual $r_o = b - Ax_o$ for the original problem, by way of relations

$$\tilde{x}_o = D^{-1}P_2x_o, \; \tilde{r}_o = D^{-1}P_1^{-1}r_o.$$

The correction vector $\tilde{x} - \tilde{x}_o$ is chosen from the span of the sequence $\tilde{K}$, truncated to a given finite length (the number of linear iterations performed). But from the relation $\tilde{x} = D^{-1}P_2x$, the subspace in which the correction $x - x_o$ is chosen is instead the span of $P_2^{-1}D\tilde{K}$ (truncated), and this is the subspace we are ultimately interested in. Thus for a fixed matrix A and fixed initial residual r_o, define

$$K(P_1, P_2) = P_2^{-1}D\tilde{K} = \{P_2^{-1}D\tilde{A}^l\tilde{r}_o : l = 0, 1, \cdots\}. \tag{4.4}$$

This is the sequence whose first l vectors are used to get $x_l - x_o$. The following easy result helps to clarify the roles of left and right preconditioning.

Theorem 4.1: The transformed Krylov sequence given by Eq. (4.4) satisfies

$$K(P_1, P_2) = K(I, P_1P_2) = K(P_1P_2, I). \tag{4.5}$$

Proof: The general vector in the sequence $K(P_1, P_2)$ is

$$P_2^{-1}D(D^{-1}P_1^{-1}AP_2^{-1}D)^lD^{-1}P_1^{-1}r_o = P_2^{-1}(P_1^{-1}AP_2^{-1})^lP_1^{-1}r_o$$

$$= P_2^{-1}P_1^{-1}(AP_2^{-1}P_1^{-1})^l r_o = (P_2^{-1}P_1^{-1}A)^l P_2^{-1}P_1^{-1}r_o.$$

The last two expressions are identical to the corresponding vectors in $K(I, P_1P_2)$ and $K(P_1P_2, I\ )$ respectively. Notice that all D and D^{-1} factors also drop out. QED

The above result says that the subspace of interest is independent of the scaling matrix D, and independent of whether the two preconditioners P_1 and P_2 are applied on opposite sides or their product applied on one side or the other. Moreover, if P_1 and P_2 commute, the subspace is also unchanged if the preconditioners are interchanged: $K(P_1, P_2) = K(P_2, P_1)$. However, this independence does not hold in the actual algorithms. Dependence on D is evident every time a convergence test is applied to the norm of a vector. Dependence on the arrangement of the preconditioners (as well as on D) arises in the posing of orthogonality or minimality conditions on the residual. For example, in the Arnoldi method the residual $\tilde{A}\tilde{z} - \tilde{r}_o$ must be orthogonal to $\tilde{K}_l$ (the l-dimensional subspace spanned by the first l vectors in $\tilde{K}$), with $\tilde{z}\epsilon\tilde{K}_l$. This is equivalently rewritten as

$$D^{-1}P_1^{-1}(Az - r_o) \perp D^{-1}P_2K_l \ with \ z\epsilon K_l = P_2^{-1}D\tilde{K}_l$$

in the case of preconditioners (P_1, P_2). But for preconditioners (I, P_1P_2) this condition becomes

$$D^{-1}(Az - r_o) \perp D^{-1}P_1P_2K_l, \quad z\epsilon K_l,$$

and for preconditioners $(P_1P_2, I\ )$ it becomes

$$D^{-1}P_2^{-1}P_1^{-1}(Az - r_o) \perp D^{-1}K_l, \quad z\epsilon K_l,$$

and the three conditions are not equivalent in general, although K_l is the same in all three.

(c) Preconditioned Krylov Algorithms.

For given preconditioners P_1 and P_2, specific preconditioned algorithms result from applying the basic IOM and IGMRES algorithms of Sec. 2 to the transformed system $\tilde{A}\tilde{x} = \tilde{b}$ in (4.2). Most of the algorithmic issues that arise in the unpreconditioned case carry over directly to the preconditioned case, and we will adopt the same decisions that were made earlier (and described in [4]), as follows:

- We take $x_o = 0$, having no choice readily available that is clearly better.
- We will incorporate the scaling in an explicit sense, storing vectors $\tilde{v}_i$ that arise in the method as it stands rather than unscaled vectors $D\tilde{v}_i = v_i$.
- We use a difference quotient representation

  $$Jv \approx [f(t, y + \sigma v) - f(t, y)]/\sigma, \quad \sigma = (\|v\|_{WRMS})^{-1},$$

  although in the code described in [4], we also included a user option to supply Jv in closed form.
- We use the modified Gram-Schmidt procedure for orthogonalizing basis vectors.
- We handle breakdowns in the same same manner as in the basic algorithms.
- We will use the same constant $\delta = .05\epsilon_1$ as a bound on the residuals $\|b - Ax_l\|_{WRMS}$ (ϵ_1 is the tolerance on the nonlinear iteration).

The presence of preconditioning does have some side effects on the algorithm, however. One is that we must actually deal with the operator

$$Av = v - h\beta_o Jv,$$

as opposed to dealing with Jv and making corrections accordingly. This is because the span

$[\tilde{r}_o, \tilde{A}\tilde{r}_o, \cdots, \tilde{A}^l\tilde{r}_o]$ is not generally the same as $[\tilde{r}_o, \tilde{J}\tilde{r}_o, \cdots, \tilde{J}^l\tilde{r}_o]$ with $\tilde{J} = D^{-1}P_1^{-1}JP_2^{-1}D$, unless $P_1P_2 = I$.

Secondly, the residual quantity ρ_l computed during the algorithm is $\|\tilde{r}_l\|_2 = \|\tilde{b} - \tilde{A}\tilde{x}\|_2$, and in general this is not directly related to the quantity $\|r_l\|_{WRMS} = \|b - Ax_l\|_{WRMS}$ in which we are really interested. Instead, we have a relation $\tilde{r}_l = D^{-1}P_1^{-1}r_l$ and hence

$$\rho_l = \|\tilde{r}_l\|_2 = \|P_1^{-1}r_l\|_{WRMS}. \qquad (4.6)$$

In order to impose a convergence test that comes closer to the test $\|r_l\|_{WRMS} < \delta$, including the effect of P_1 when it is nontrivial but without going to the extra expense of computing $\|r_l\|_{WRMS}$, we use a test

$$\rho_l < \delta' \equiv \delta\|P_1^{-1}r_o\|_{WRMS}/\|r_o\|_{WRMS}.$$

That is, the factor by which P_1^{-1} reduces the norm of r_o is included as an approximation to the factor by which P_1^{-1} would reduce the norm of r_l. In particular, the convergence test made at $l = 0$ is precisely $\|r_o\|_{WRMS} < \delta$, independent of P_1. Because of Eq. (4.6), there is some theoretical advantage to doing all preconditioning on the right only, but whether this is a real advantage in practice is unclear.

We can now state our algorithms for scaled preconditioned versions of the IOM and GMRES method. These are given for arbitrary x_o, for the sake of generality, and denoted hereafter by SPIOM and SPIGMR, respectively.

Scaled Preconditioned IOM (SPIOM):

1. (a) $r_o = b - Ax_o$; stop if $\|r_o\|_{WRMS} < \delta$.

   (b) $\tilde{r}_o = D^{-1}P_1^{-1}r_o$, compute $\|\tilde{r}_o\|_2 = \|P_1^{-1}r_o\|_{WRMS}$,

   $\delta' = \delta\|P_1^{-1}r_o\|_{WRMS}/\|r_o\|_{WRMS}, \tilde{v}_1 = \tilde{r}_o/\|r_o/\|_2$.

```
2. For $l = 1, 2, \cdots, l_{max}$, do:
    (a) Compute $\tilde{A}\tilde{v}_l = D^{-1}P_1^{-1}AP_2^{-1}D\tilde{v}_l$.
    (b) $\tilde{w}_{l+1} = \tilde{A}\tilde{v}_l - \sum_{i=i_o}^{l} \tilde{h}_{il}\tilde{v}_i$, where $i_o = max(1, l-p+1)$, $\tilde{h}_{il} = (\tilde{A}\tilde{v}_l, \tilde{v}_i)$.
    (c) $\tilde{h}_{l+1,l} = \|\tilde{w}_{l+1}\|_2$, $\tilde{v}_{l+1} = \tilde{w}_{l+1}/\tilde{h}_{l+1,l}$.
    (d) Update the $LU$ factorization of $\tilde{H}_l = (\tilde{h}_{ij})$ (an $l \times l$ matrix).
    (e) Indirectly compute $\rho_l = \|\tilde{b} - \tilde{A}\tilde{x}_l\|$ by (2.16)-(2.17).
    (f) If $\rho_l < \delta'$, go to Step 3; otherwise go to (a).
3. Compute $\tilde{z} = \|\tilde{r}_o\|_2 \tilde{V}_l \tilde{H}_l^{-1} e_1$, $x_l = x_o + P_2^{-1}D\tilde{z}$.
```

Scaled Preconditioned Incomplete GMRES (SPIGMR):

```
1. (a) $r_o = b - Ax_o$; stop if $\|r_o\|_{WRMS} < \delta$.
   (b) $\tilde{r}_o = D^{-1}P_1^{-1}r_o$, compute $\|\tilde{r}_o\|_2 = \|P_1^{-1}r_o\|_{WRMS}$,
       $\delta' = \delta\|P_1^{-1}r_o\|_{WRMS}/\|r_o\|_{WRMS}$, $\tilde{v}_1 = \tilde{r}_o/\|\tilde{r}_o\|_2$.
2. For $l = 1, 2, \cdots, l_{max}$, do:
    (a) Compute $\tilde{A}\tilde{v}_l = D^{-1}P_1^{-1}AP_2^{-1}D\tilde{v}_l$.
    (b) $\tilde{w}_{l+1} = \tilde{A}\tilde{v}_l - \sum_{i=i_o}^{l} \tilde{h}_{il}\tilde{v}_i$, where $i_o = max\ (1, l-p+1)$, $\tilde{h}_{il} = (\tilde{A}\tilde{v}_l, \tilde{v}_i)$.
    (c) $\tilde{h}_{l+1,l} = \|\tilde{w}_{l+1}\|_2$, $\tilde{v}_{l+1} = \tilde{w}_{l+1}/\tilde{h}_{l+1,l}$.
    (d) Update QR factorization of $\tilde{H}_l = (\tilde{h}_{ij})$ ( *an* $(l+1) \times l$ *matrix* ).
    (e) Compute residual $\rho_l$ indirectly by (2.24)-(2.27).
    (f) If $\rho_l < \delta'$, go to Step 3; otherwise go to (a).
3. Compute $\|\tilde{r}_o\|_2 Q_l^T e_1 = (\bar{g}_l, g)^T$, $\tilde{z} = \tilde{V}_l \bar{R}_l^{-1} \bar{g}_l$, $x_l = x_o + P_2^{-1}D\tilde{z}$.
```

In the case of the CG method, further considerations are necessary because of the assumptions made there that A is symmetric positive definite. CG with preconditioning, or PCG, involves a single matrix M which is also assumed to be symmetric positive definite.

In the absence of scaling, the PCG algorithm is the equivalent of applying CG to the transformed system,

$$(M^{-1/2}AM^{-1/2})(M^{1/2}x) = (M^{-1/2}b),$$

but is expressed in such a way that no square roots are involved (see Golub and Van Loan [10]). In the presence of scaling, there seem to be two choices as to how to apply PCG, depending on whether we assume symmetry in the original or the scaled form of the system (they cannot both be symmetric for a nontrivial scaling matrix D). In either case, we certainly want to measure the scaled residual, i.e. test $\|b - Ax_l\|_{WRMS}$, in the convergence tests. We therefore arrive at two different algorithms, as follows:

Under the assumption that A itself, and a preconditioner M approximating A, are symmetric positive definite or nearly so, we can apply PCG to $Ax = b$ without use of scaling except that the norm of $r_l = b - Ax_l$ that we test is $\|r_l\|_{WRMS} = \|D^{-1}r_l\|_2$. We will refer to this simply as *PCG.*

Preconditioned Conjugate Gradient (PCG).

```
1. $r_o = b - Ax_o$; stop if $\|r_o\|_{WRMS} < \delta$.
2. For $l = 1, 2, \cdots, l_{max}$ , do:
    (a) Compute $z_{l-1} = M^{-1}r_{l-1}$.
    (b) $\beta_l = z_{l-1}^T r_{l-1} / z_{l-2}^T r_{l-2}$  $(\beta_1 = 0)$.
    (c) $p_l = z_{l-1} + \beta_l p_{l-1}$  $(p_1 = z_o)$ .
    (d) Compute $Ap_l$ .
    (e) $\alpha_l = z_{l-1}^T r_{l-1} / p_l^T A p_l$ .
    (f) $x_l = x_{l-1} + \alpha_l p_l$, $r_l = r_{l-1} - \alpha_l A p_l$.
    (g) If $\|r_l\|_{WRMS} < \delta$ , stop; otherwise go to (a).
```

Alternatively, it may be appropriate to assume instead that the scaled matrix $\bar{A} = D^{-1}AD$, and the corresponding scaled preconditioner $\bar{M} = D^{-1}MD$, are both symmetric positive definite, and to apply PCG to the scaled system $\bar{A}\bar{x} = \bar{b}$ ($\bar{x} = D^{-1}x$, $\bar{b} = D^{-1}b$) with preconditioner $\bar{M}$. We refer to this as scaled PCG, or *SPCG*. There appears to be a slight advantage in efficiency to making the scaling implicit by rewriting the algorithm in terms of unscaled quantities. The result is the same as the above PCG algorithm except in the calculation of the inner products in β_l and α_l , which are now

$$\beta_l = z_{l-1}^T D^{-2} r_{l-1} / z_{l-2}^T D^{-2} r_{l-2} \ (\beta_1 = 0), \ \alpha_l = z_{l-1}^T D^{-2} r_{l-1} / p_l^T D^{-2} A p_l.$$

This SPCG algorithm can also be obtained by applying CG to the transformed system (4.2) - (4.3), in which $P_1 = P_2 = M^{1/2}$, keeping in mind that $\bar{M} = D^{-1}MD$ is assumed to be symmetric, while M and $M^{1/2} = D\bar{M}^{1/2}D^{-1}$ are not (in general).

We have implemented both PCG and SPCG, filling in algorithm details in the same way as done for the other methods (using $x_o = 0$, a difference quotient Jv, $\delta = .05\epsilon_1$, etc.). Of course, each algorithm is subject to breakdown when applied to a problem for which the assumed symmetry or definiteness fails to hold. Thus the denominators in β_l and α_l are tested for being zero before the divide is done.

5. Preconditioners for Reaction-Diffusion Systems

The preconditoned Krylov subspace methods described so far are quite general in nature, with preconditioner matrices that are as yet unspecified. In this section some specific choices will be described, as motivated by ODE systems that arise from PDE systems by the method of lines (MOL). At this point, our approach attempts to compromise between totally general methods, in which problem structure is not exploited at all, and totally *ad hoc* solution schemes, in which the algorithm and problem features are so closely linked that flexibility as to problem scope is lost. Here the preconditioners will exploit problem structure, within a setting of general purpose methods (BDF, Newton, and Krylov). The general and special parts of the algorithm are logically well separated. To the extent that some storage for preconditioner matrices (and associated data) will be required, our methods are no longer truly matrix-free in some cases. However, since storage economy (as compared to traditional stiff system methods) is still a prime concern, any choice of preconditioners should be strongly influenced by its storage costs.

(a) Problem Structure.

The class of problems we shall concentrate on here is that of ODE systems in time that are the result of treating time-dependent PDE systems by some form of the method of lines. Assume that a vector $u = u(t,x)$ of length p is governed by a PDE system in time t and a space variable x (of any dimension) of the form

$$\partial u/\partial t = R(t,x,u) + S(t,x,u), \tag{5.1}$$

plus initial and boundary conditions, in which R represents reaction terms and S represents a spatial transport operator in x (diffusion, advection , etc.). Thus R is assumed to be a point function of u, while S contains partial derivatives of u with respect to x. The MOL treatment of such a system consists of representing the spatial variation of u in a discrete manner, and thereby obtaining a semi-discretized form of (5.1) in which S is replaced by a discrete function, while the time derivative remains continuous. To be more specific, but without digressing unduly into MOL techniques, consider traditional finite difference discretizations of (5.1). In this case, discrete values u_i represent $u(t, x_i)$ at q discrete mesh points x_i and suitable difference approximations to $S(t, x, u)$ at each x_i, and to the boundary conditions are formed. The vector

$$y = (u_1, \cdots, u_q)^T$$

of length $N = pq$ then satisfies an ODE system

$$\dot{y} = \bar{R}(t, y) + \bar{S}(t, y), \tag{5.2}$$

with given initial conditions, in which $\bar{R}$ and $\bar{S}$ are discrete forms of R and S.

Of course there are many variations on this approach (staggered grids, moving grids, etc.) and there are radically different discretization choices, notably the various finite element schemes. The latter generally result in ODE systems that are linearly implicit, with a square mass matrix multiplying $\dot{y}$. For the present, we will assume that, whatever the spatial discretization, the ODE system has been put into the explicit form (5.2) (possibly by multiplying by the inverse of a mass matrix), although it is a straightforward matter to extend our methods so as to treat linearly implicit systems as such. In order to reflect this greater generality in (5.2), we will denote the global vector y as

$$y = (y_1, \cdots, y_q)^T \tag{5.3}$$

in which each of the q blocks y_i (each of length p) is associated with a point $x = x_i$, but may or may not represent a discrete value of the original vector u in (5.1).

In (5.2), the essential feature of the additive splitting $\bar{R} + \bar{S}$ is that $\bar{R}$ does not involve any spatial coupling, while $\bar{S}$ involves little or no interaction between the components of y_i at any given point x_i. Thus the individual blocks of (5.2) can be written

$$\dot{y}_i = f_i(t, y) = \bar{R}_i(t, y_i) + \bar{S}_i(t, y_1, \cdots, y_q), \tag{5.4}$$

where $\bar{R}_i$ is a function of y_i but no other y_j, while the dominant feature of $\bar{S}_i$ is the coupling among various y_j arising from the discretization of spatial derivatives.

(b) Block-Diagonal Preconditioners.

To obtain a preconditioner matrix M that is intended to approximate the Newton matrix $A = I - h\beta_o J$, the most obvious approach is to consider a matrix L that approximates the system Jacobian J, and use $M = I - h\beta_o L$. For the blocked ODE system given by (5.2)-(5.4), a natural choice for L is one that is block-diagonal and hence can accurately reflect the interaction of the components at each spatial point, but not the spatial coupling. Thus consider a block-diagonal matrix

$$B = diag(B_1, \cdots, B_q) \tag{5.5}$$

in which each B_i is $p \times p$. We can get B to approximate J in one of two ways:

$$B_i = \partial f_i / \partial y_i \quad (i^{th}\ diagonal\ block\ of\ J) \tag{5.6}$$

or

$$B_i = \partial \bar{R}_i / \partial y_i. \tag{5.7}$$

We will refer to these choices as the <u>total block-diagonal</u> and the <u>interaction only</u> approximations, respectively. They differ by $\partial \bar{S}_i / \partial y_i$, the diagonal part of the discrete transport Jacobian.

In either case, the computation and processing of the matrix B and $M = I - h\beta_o B$ is certainly a nontrivial matter. It may be that the problem permits the B_i to be supplied in closed form fairly cheaply, but in most practical cases, we expect that this is not feasible.

Then a difference quotient approximation to B_i must be done. For this, we assume that a routine to compute the individual blocks $f_i(t, y)$ (in the total case) or $\bar{R}_i(t, y_i)$ (in the interaction only case) is supplied and is reasonably inexpensive. The cost of one such evaluation would be expected to be about $1/q$ times that of evaluating all of $f(t, y) = \dot{y}$ (somewhat more in the total case, somewhat less in the interaction only case). Then a difference quotient estimate of B_i will involve p such calls (assuming a base value of f_i or $\bar{R}_i$ is saved for use in the difference quotients, but without exploiting any sparsity structure within B_i), and the total cost of B will be about that of p evaluations of f.

Once B is evaluated, it is subjected to LU decomposition, and the LU factors of the blocks are saved. Then these are used as needed whenever a vector $B^{-1}v$ is called for. Periodic reevaluation of B is done by the same strategy used for traditional methods involving evaluation and use of the Jacobian J.

The block-diagonal structure of B allows for considerable potential speedup if the scheme is implemented on a multiprocessor. The various blocks, corresponding to the various spatial points, can be evaluated, then factored, and the factors applied to a given (blocked) vector, all in parallel with one another. As many as q processors can be occupied concurrently for a problem with q mesh points. Block-diagonal preconditioning can be expected to improve performance considerably when the interaction among the components of u at each point x is the dominant contributor to the stiffness of the system (5.2). The inclusion of part of the transport contribution, as done in (5.6), may or may not improve performance further.

The storage cost is clearly an additiional $p^2q = pN$ words, to hold the B_i and then their factors. This cost could well be greater than that of all of the remaining work space involved when p is sizable, although not nearly as big as the cost of storing all of the Jacobian and its LU factorization in a typical multi-dimensional system treated by a traditional method.

A natural way to reduce the storage cost is to do grouping of the diagonal blocks B_i. By this we mean that the spatial points are grouped, most likely by a simple decomposition of the original spatial domain, and for each group only one B_i is evaluated and used as an approximation to the others in the group. This is simply the analog in space of the strategy of periodic Jacobian evaluations in time - a strategy which is widely used and highly cost-effective. In both ideas, the motivation is that the Jacobian elements needed are likely to vary smoothly as functions of the relevant independent variable (space or time) and so need only be evaluated on some subset of the discrete values of that variable that are generated. The resulting crude approximation to Jacobian elements is acceptable within a Newton iteration, whereas the full set of discrete values is dictated by the accuracy of the final computed solution. If the q spatial points x_i are somehow grouped into g groups (not necessarily of equal size) then the cost of evaluating the grouped block-diagonal approximation B goes down to about $p(g/q)$ evaluations of f, and the storage is reduced to p^2g. Thus there is a potentially great reduction in storage costs with grouping, but the reduction in computing cost depends on whether significantly more linear and/or nonlinear iterations must be taken than were taken without grouping.

(c) Transport-Based Preconditioners.

Another natural choice presents itself for blocked problems of the type (5.2)- (5.4), namely a preconditioner that uses the discretized transport operator $\bar{S}_i$. Here the interaction of the components at each point is ignored, and for each component separately, a preconditioner arises upon treating the transport contributions of the problem by traditional iterative methods such as SOR (successive overrelaxation). The details of this preconditioner will depend heavily on the particular nature of the problem. In order to be somewhat specific here, suppose that the transport occurs linearly in $\bar{S}_i$, so that we can write

$$\bar{S}_i(t, y_1, \cdots, y_q) = \sum_{j<i} \bar{L}_{ij} y_j + \bar{B}_i y_i + \sum_{j>i} \bar{U}_{ij} y_j. \tag{5.8}$$

Assume further that the various coefficient blocks (all $p \times p$) in (5.8) are diagonal, reflecting the fact that component interaction is completely accounted for in the term $\bar{R}_i$ of (5.4). Then the blocked matrix

$$\begin{pmatrix} \bar{B}_1 & & (\bar{U}_{ij}) \\ & \ddots & \\ (\bar{L}_{ij}) & & \bar{B}_q \end{pmatrix} = \bar{L} + \bar{B} + \bar{U} \tag{5.9}$$

is an approximation to $\partial \bar{S}/\partial y$ on which a preconditioner can be based. (It may not be all of $\partial \bar{S}/\partial y$ because the coefficients might depend on y, and that dependence is being ignored here.) The corresponding approximation to $A = I - h\beta_o J$ is

$$B - L - U = (I - h\beta_o \bar{B}) - h\beta_o \bar{L} - h\beta_o \bar{U},$$

which also has diagonal $p \times p$ blocks.

Now consider, for example, the application of SOR to the system

$$(B - L - U)x = b. \tag{5.10}$$

We must suppose that the acceleration parameter ω is given, and if no better information about its choice is available, we may simply have $\omega = 1$ (and we are doing Gauss-Seidel iteration). The initial guess is some easily computable vector $x^o = M_o b$, where the matrix M_o might be 0, or B^{-1}, or $\omega(B - \omega L)^{-1}$, for example. (In our tests, we use the third choice for M_o.) The iteration is given by

$$x^{v+1} = \mathcal{L} x^v + \omega(B - \omega L)^{-1} b$$

with

$$\mathcal{L} = (B - \omega L)^{-1}[(1-\omega)B + \omega U].$$

Thus a given number, m, of iterations produces an iterate

$$x^m = \mathcal{L}^m x^o + (\mathcal{L}^{m-1} + \mathcal{L}^{m-2} + \cdots + \mathcal{L} + I\,)\omega(B - \omega L)^{-1}b = Mb,$$

where

$$M = \mathcal{L}^m M_o + \omega(\mathcal{L}^{m-1} + \cdots + I\,)(B - \omega L)^{-1}. \tag{5.11}$$

This matrix M represents a preconditioner for the system $Ax = b$. In practice, it would be applied by carrying out the SOR iteration in whatever manner is most appropriate for the particular problem, depending on the number of spatial dimensions, the boundary conditions, the uniformity or variation of the coefficients, etc. Keeping storage to a minimum would be an important criterion in the implementation.

An easy variation of SOR as a preconditioner is Symmetric SOR, or SSOR, where each iteration has a forward and a backward sweep:

$$x^{v+1/2} = \mathcal{L}x^v + (B - \omega L)^{-1}\omega b$$

$$x^{v+1} = \bar{\mathcal{L}}x^{v+1/2} + (B - \omega U)^{-1}\omega b$$

$$\textit{where } \bar{\mathcal{L}} = (B - \omega U)^{-1}[(1-\omega)B + \omega L].$$

Some given number of SSOR iterations, say m, has the same cost as $2m$ SOR iterations, but may be more effective because of the symmetrization.

In special cases, one might be able to do even better in approximating the solution of (5.10). For example, suppose that the dominant transport process is simple diffusion, that the mesh is uniform, and that for each PDE component the diffusion coeﬃcient is constant.

Then the product Jx represents a decoupled set of discrete diffusion operators (with source term), $d\nabla^2 u + r$ and the system $Ax = x - h\beta_o Jx = b$, after division by $h\beta_o d$ in each PDE component, corresponds to a collection of scalar Helmholtz equations

$$\lambda u + \nabla^2 u = s, \quad \lambda = -1/h\beta_o d,$$

discretized on a uniform mesh. For such a problem, fast Poisson solvers are available. To the extent that the diffusion operator dominates the physical transport process, the resulting preconditoner should be very effective.

It should be emphasized that in all of the above cases, the various PDE components, i.e. the components of the vector u in (5.1), are treated independently. Whichever of these preconditioners is used, it is applied separately to each PDE component. In particular, the implementation of the combined algorithm on a multiprocessor could readily take advantage of this fact by carrying out the preconditioning operations concurrently for the various components.

(d) Operator Splitting.

The preceding paragraphs give specific preconditioner matrices without saying just how they are to be applied. Clearly, if a single matrix is chosen as a preconditioner, it can be applied on the left as P_1 or on the right as P_2 in the SPIOM or SPIGMR algorithms, or as the matrix M in PCG or SPCG. But if two different matrices are available as preconditioners, then in SPIOM or SPIGMR they can be applied as P_1 and P_2, in either order, or their product (in either order) could be applied on one side only, or the product applied as M in PCG or SPCG. There seems to be no convincing argument for or against any particular choice of arrangements of the preconditioners for a given Krylov algorithm.

For the reaction-transport problems described above, choosing both a block-diagonal (presumably the interaction-only choice) and a transport-based preconditioner constitutes an operator splitting approach to preconditioning that closely resembles splitting methods used elsewhere. The block-diagonal matrix B is dominated by the interaction of the PDE components at each spatial point, while the transport-based matrix M attempts to account for the remainder of the problem. Each matrix can be dealt with reasonably efficiently, in contrast to the matrix $J = \partial f / \partial y$ which reflects both the interaction and the transport operators.

This splitting approach obviously generalizes in the same sense that traditional splitting methods do, to ODE systems with any additive splitting. Thus suppose the system is

$$\dot{y} = f = f_1 + f_2 + \cdots + f_s,$$

with the various terms f_k representing distinct physical processes (operators). Assume further that the individual Jacobian matrices $J_k = \partial f_k / \partial y$ are easily computed or approximated in such a way that a system $(I - h\beta_o J_k)x = b$ can be solved (or approximately solved) efficiently. Then each matrix $A_k = I - h\beta_o J_k$ (or a computable approximation to it) serves as a preconditioner and the product of $A_1, \cdots, A_s$ is a single preconditioner to the linear system $Ax = b, A = I - h\beta_o J$. Two products, of two subsets of these A_k, could be used as left and right preconditioners. The best choice of order in the products is unclear in general, and performance may not depend strongly on the order. Heuristically, the use of $P = A_1 A_2 \cdots A_s$ as a preconditioner would be expected to do well to the extent that P approximates A, i.e. to the extent that

$$(I - h\beta_o J_1) \cdots (I - h\beta_o J_s) \approx I - h\beta_o(J_1 + J_2 + \cdots + J_s).$$

The two matrices above agree only to first order in h. The errors in P associated with higher order terms would have to be dealt with by the Krylov method in order for the combination to be effective.

6. Numerical Tests

Tests of the methods described above were done using an experimental solver derived from the general purpose ODE solver LSODE, which uses BDF methods for stiff problems. Stiff test problems were generated by applying the method of lines to PDE systems in two space dimensions. What follows is a brief description of the solver, and a description of tests on three problems. All testing was done on a Cray-1 at LLNL.

(a) LSODPK, an Experimental Solver.

A solver package called LSODPK (Livermore Solver for ODE's with Preconditioned Krylov methods) was created from LSODE, in which to implement the preconditioned iteration methods. LSODPK resembles the solver LSODP described in [4], which uses (unpreconditioned) Arnoldi and IOM algorithms. LSODPK allows the user to select among Arnoldi/IOM, GMRES/IGMRES, PCG, and SPCG as the basic iteration method, and accepts user-supplied routines to accomplish the preconditioning operations. This is shown in a simplified form in Fig. 1. The driver routine, LSODPK, calls a single-step routine, STODPK, which calls PKSET to prepare for preconditioning, and SOLPK to carry out the linear system solution by calling one of four other routines, as shown. The user supplies F

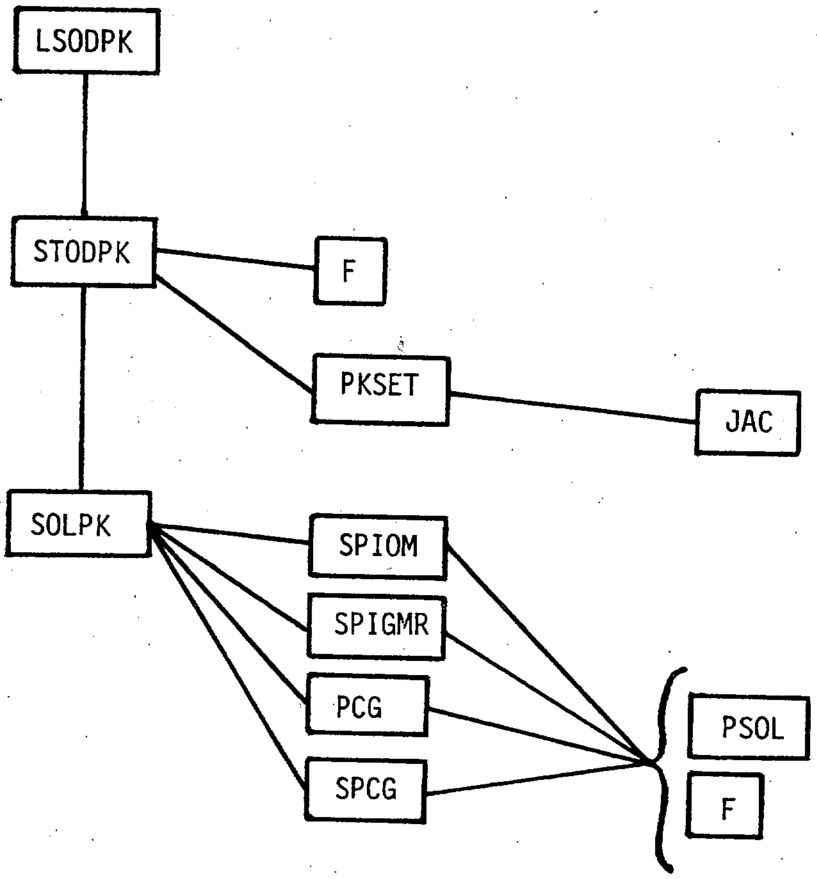

Fig. 1. Simplified Block Structure of LSODPK

(for $f(t, y)$) as in LSODE, but also two routines, JAC and PSOL, for preconditioning. (All three names are dummy names, passed by the user to LSODPK.) JAC must compute and preprocess any Jacobian-related data involved in the preconditioning. PSOL must carry out the preconditioning of a vector, i.e. replace a vector v with $P_1^{-1}v, P_2^{-1}v$, or $M^{-1}v$, depending on the choice of method and the nature of the preconditioning. Various integer flags are input by the user to specify these choices, and a flag is passed to PSOL to specify which $P_k^{-1}v$ is desired when both are possible. In all other respects, the overall algorithm in LSODPK is the same as LSODE and/or LSODP. Calls to PKSET (hence to JAC) are made as infrequently as judged suitable, by the same heuristic rules that govern J evaluation in LSODE. As in LSODP, the default value of the heuristic convergence test constant δ_1 is .05 (i.e. $\delta = .05\epsilon_1$), that of l_{max} is 5, and that of the parameter p is l_{max} (giving complete rather than incomplete methods).

Various cumulative performance statistics are made available by LSODPK. These include:

NST = number of time steps

NFE = number of f evaluations

NPE = number of preconditioner evaluations (JAC calls)

NNI = number of nonlinear iterations

NLI = number of linear iterations

NPS = number of preconditioner solves (PSOL calls)

NCFN = number of nonlinear convergence failures

NCFL = number of linear convergence failures

NFE is also equal to NNI + NLI + 1 (plus the number of internal restarts at order 1, if any). In addition, LSODPK monitors the average Krylov subspace dimension

$$\text{AVDIM} = \Delta\ \text{NLI} \ /\ \Delta\ \text{NNI},$$

and the convergence failure rates

$$\text{RCFN} = \Delta\ \text{NCFN}/\ \Delta\ \text{NST}$$

$$\text{RCFL} = \Delta\ \text{NCFL}/\ \Delta\ \text{NNI}$$

within an output interval. Warning messages are printed if these appear to be too large.

While the preconditioning routines JAC and PSOL are not part of the LSODPK solver as such, we have generated several pairs of routines for use on tests in PDE-based problems. One such pair generates the total block-diagonal part of the Jacobian, in either user-supplied or finite-difference form, and then uses LU factorizations of the blocks for linear system solutions. A second pair generates the interaction-only block-diagonal matrix and also performs a fixed number of SOR (actually Gauss-Seidel) iterations on the diffusion terms in the system (not coded very efficiently). A third uses the interaction-only matrix and treats the diffusion terms with a fast Poisson solver, namely HWSCRT [20]. Three other JAC/PSOL pairs are the analogs of the first three in which block-grouping is done by a static partitioning of the 2-D mesh. The partitioning is a simple Cartesian one in which a discrete $M_1 \times M_2$ mesh is divided into $N_1 N_2$ groups by partitioning each 1-D mesh of size M_j uniformly into N_j groups, N_j being an integer divisor of M_j . Other possibilities are easy to imagine.

(b) Test Problem 1.

We begin with a problem that was used in [4] to test LSODP, namely a system derived from a 2-D diurnal kinetics-transport PDE system with two species. The PDE's have the form

$$\frac{\partial c^i}{\partial t} = K_h \frac{\partial^2 c^i}{\partial x^2} + \frac{\partial}{\partial z}\left(K_v(z)\frac{\partial c^i}{\partial z}\right) + V\partial c^i/\partial x + R^i(c^1, c^2, t)\ (i = 1, 2)\ (V = .01) \qquad (6.1)$$

and are discretized by finite differencing on a 20x20 mesh, giving a system of size NEQ=800. The details are available in [4], and so are not repeated here.

This problem was run with LSODP (i.e. with unpreconditioned SIOM), and with LSODPK with a variety of method choices. The preconditioners tested included the total block-diagonal part of the Jacobian, applied on either the left or right, denoted BDL and BDR (respectively) in the table below. Also tested were the analogous preconditioners with block grouping, denoted BGL and BGR, using a simple partitioning of the 20x20 mesh into 4=2*2 or 25=5*5 groups. The diagonal blocks were generated by difference quotients in the results given here.

Preliminary runs with a 10x10 mesh, supported by a close look at the Jacobian of this problem, showed that the lack of symmetry makes PCG or SPCG a poor choice here. Thus in Table 1 we give the results of runs on the 20x20 mesh for the SPIOM and SPIGMR algorithms (in LSODPK), as well as for SIOM (in LSODP). (The LSODP results differ slightly from those in [4] because of minor changes to LSODP.) In the column headings, PRE denotes the preconditioner choice, NGR denotes the number of groups of diagonal blocks, AVDIM is the overall average subspace dimension (NLI/NNI), and RT is the CPU run time in sec. The tolerances used were RTOL=10^{-5} and ATOL = 10^{-3} .

TABLE 1. RESULTS FROM PROBLEM 1

Method	PRE	NGR	NST	NNI	NLI	AVDIM	RT
SIOM	-	-	2341	4525	10059	2.22	65.4
SPIOM	BDL	-	2350	2788	6397	2.29	83.5
SPIOM	BDR	-	2318	2748	6257	2.28	81.2
SPIGMR	BDL	-	2348	2793	6536	2.34	84.4
SPIGMR	BDR	-	2357	2820	6374	2.26	83.5
SPIOM	BGR	5*5	2258	2655	6254	2.36	76.7
SPIOM	BGR	2*2	2341	2801	6405	2.29	78.5

As is clear from Table 1, the LSODP solver, using unpreconditioned scaled IOM, is the fastest overall choice. It is also faster than LSODE (with banded treatment of J) and GEARBI (with block SOR), as documented in [4], where the good performance of LSODP is explained in terms of tight clustering in the spectrum. The addition of preconditioning has two opposing effects: It reduces the average number of nonlinear iterations per step (NNI/NST) from about 1.9 to about 1.2 without changing the total number of steps appreciably. But it roughly doubles the average cost per nonlinear iteration, not by taking more linear iterations, but by having to compute and process the preconditioner. The cost of computing and factoring the diagonal blocks is reduced by a factor of NGR/400 (= 1/100 to 1/16) with block grouping, but evidently the cost of the backsolve operations outweighs the cost savings from the reduced NNI. If the function f were more expensive, the opposite would probably be true, and a choice such as SPIOM-BGR would be the likely winner. Finally, the data here shows no significant preference between SPIOM and SPIGMR, nor between left and right preconditioning, and this is the reason only one such choice was made for the block grouping tests.

The total storage requirement is also minimal for LSODP. The total length of the real and integer work arrays is 12,882, or about 16.1*NEQ. For the LSODPK runs, it is about 20.1*NEQ for BDL or BDR, but 17.3*NEQ to 17.1*NEQ for BGR.

(c) Test Problem 2.

In order to obtain test problems easily with higher numbers of interacting components, we have used models of multi-species food webs [3], in which mutual competition and/or predator-prey relationships in a spatial domain are simulated. The general form of these models, for s species in two dimensions, is

$$\partial c^i/\partial t = f_i(x,y,t,c) + d_i(c^i_{xx} + c^i_{yy}) \quad (i = 1, 2, \cdots, s), \tag{6.2}$$

with

$$f_i(x,y,t,c) = c^i(b_i + \sum_{j=1}^{s} a_{ij}c^j). \tag{6.3}$$

The interaction and diffusion coefficients (a_{ij}, b_i, d_i) could be functions of (x, y, t) in general. The choices made for this and the next test problem are for a simple model of p prey and p predator species $(s = 2p)$, arranged in that order in the vector c. We take the various coefficients to be as follows:

$$\left.\begin{aligned} a_{ii} &= -1 \quad (\mathit{all}\ i) \\ a_{ij} &= -\tfrac{1}{2}\cdot 10^{-6} \ (i \le p, j > p) \\ a_{ij} &= 10^4 \ (i > p, j \le p) \end{aligned}\right\} \tag{6.4}$$

(all other $a_{ij} = 0$),

$$\left.\begin{aligned} b_i &= (1 + \alpha xy) \quad (i \le p) \\ b_i &= -(1 + \alpha xy) \quad (i > p) \end{aligned}\right\} \tag{6.5}$$

$$\left.\begin{aligned} d_i &= 1 \quad (i \le p) \\ d_i &= .05 \quad (i > p) \end{aligned}\right\} \tag{6.6}$$

The domain is the unit square $0 \leq x, y \leq 1$, and $0 \leq t \leq 10$. The boundary conditions are of Neumann type (zero normal derivatives) everywhere. The coefficients are such that a unique stable equilibrium is guaranteed to exist in the constant coefficient case $\alpha = 0$ [3], namely $c = -A^{-1}b$, and empirically the same appears to be true for $\alpha > 0$. In this problem we take $\alpha = 1$.

The initial conditions used for this problem are taken to be simple peaked functions that satisfy the boundary conditions, given by the polynomial function

$$c^i(x, y) = 10 + i[16x(1-x)y(1-y)]^2 \quad (1 \leq i \leq s),$$

which varies between 10 and $10 + i$.

The PDE system (6.2) (plus boundary conditions) was discretized with central differencing on an $m \times m$ mesh, much the same as for Problem 1 (see [4]). The resulting ODE system has size $NEQ = 2\ pm^2$. It is stiff, and an estimate of the spectrum is easily obtained from the interaction terms f_i , for which the dominant eigenvalues are about $-10^4 p(1 + \alpha xy)$ for the components at a mesh point (x, y) . However, the diffusion terms cause the profiles to flatten out at steady state, so that the equilibrium values of any species c^i are spread by a factor of only about 1.08 rather than $1 + \alpha = 2$ (though the spread factor exceeds 2 during the transient). The discrete diffusion terms contribute significantly to the stiffness also.

For the tests reported here, we take $m = 12$ (144 mesh points), and we take two cases for p ,

(a) $p = 5$ (10 species, NEQ = 1440),

(b) $p = 10$ (20 species, NEQ = 2880).

we report results for LSODPK with a variety of method choices. The tolerances used were

RTOL = 10^{-6} and ATOL = 10^{-8} . The PCG and SPCG methods were not tested because the Jacobian at equilibrium is highly nonsymmetric. The preconditioner pairs tested, with mnemonic names used in the tables, are as follows:

BDL:	total block-diagonal part of J on left;
BDR:	total block-diagonal part of J on right;
OSS:	operator splitting, using 5 SOR (Gauss-Seidel) iterations on the diffusion terms for P_1 and the interaction-only Jacobian for P_2 ;
OSF:	operator splitting, using HWSCRT on the diffusion for P_1 and the interaction-only Jacobian for P_2 .

Where a block-diagonal part of the Jacobian J is required, both a closed-form user-supplied option (USJ) and a difference quotient option form (DQJ) were tested. Further, we tested the analogous set of preconditioner pairs in which block grouping was done, with $NGR = n_g * n_g$ groups from a Cartesian product partition of the mesh with n_g groups in each dimension (n_g a divisor of 12). All the various possible combinations are too numerous to test completely. Nor did we test the other possible arrangements of the preconditioners (interchanging P_1 with P_2 , or using the product on one side only), as we do not expect those to yield major variations in performance. However, we did run tests using the individual preconditioners in OSS by themselves, i.e.

SOR:	left preconditioning with 5 SOR (actually Gauss-Seidel) iterations;
IOJ:	right preconditioning with interaction-only Jacobian.

Table 2a gives results for the 10-species case ($p = 5$) , beginning with those from SIOM (without preconditioning). We see that various preconditioned method combinations do better than SIOM, both in statistics and run time. The benefits of preconditioning in reducing the number of linear and nonlinear iterations per step, and also the number of failed steps and step size reductions forced by convergence failures, are clear for the SPIGMR method with OSS and OSF preconditioning. The BDL and BDR choices give competitive run times but with much higher average numbers of (cheaper) linear iterations per step — a tradeoff of dubious merit when more costly forms f are contemplated. The one-sided choices SOR and IOJ fail even more badly. The OSF choice is more expensive than OSS here because the 5 SOR iterations are evidently cheaper than the HWSCRT call and nearly as effective. For the same problem with more diffusion, namely $d_i = 10$ $(i \leq 5)$, the preference order is reversed, with run times of 54 for OSS (where 10 SOR iterations were found necessary) vs 38 for OSF. From the close agreement between SPIGMR and SPIOM performance, with either OSS or BDR preconditioning, no further testing of SPIOM seemed to be worth doing. The use of block grouping with OSS raised AVDIM only slightly, but reduced the cost slightly and reduced the required storage considerably — by a factor of about .63 in the case of 9 groups and .62 with 4 groups.

The last six entries in Table 2a deal with the case of difference quotient Jacobian elements. Again block grouping is effective, and it gives a greater overall cost reduction here because of the higher cost of computing the diagonal blocks with DQJ.

It should be mentioned here that the overall average AVDIM above is somewhat misleading because it includes the nonstiff transient (roughly $0 \leq t \leq 10^{-3}$), which contributes nearly half the total cost. The local averages Δ NLI/ Δ NNI for intervals beyond the transient are much higher, uniformly exceeding 1.5 for $t \geq .1$ and uniformly exceeding 4 for $t \geq 5$. Thus many of the options tested would have performed better with a value of l_{max} higher than 5, but then their storage requirements would be higher accordingly. We can also see here the potential benefits of a stiff/nonstiff switching algorithm in combination with these algebraic methods.

TABLE 2a. RESULTS FOR PROBLEM 2 (p = 5)

Method	PRE	NGR	NST	NNI	NLI	AVDIM	RT
SIOM	-	-	678	1125	4018	3.57	39.2
SPIGMR	OSS(USJ)	-	354	404	596	1.48	27.3
SPIGMR	OSF(USJ)	-	354	401	567	1.41	34.0
SPIGMR	BDL(USJ)	-	393	502	1434	2.86	29.0
SPIGMR	BDR(USJ)	-	395	485	1283	2.65	27.1
SPIGMR	SOR	-	566	1063	3061	2.88	76.3
SPIGMR	IOJ(USJ)	-	448	577	1744	3.02	35.0
SPIGMR	OSS(USJ)	9	351	400	640	1.60	27.1
SPIOM	OSS(USJ)	-	355	403	588	1.46	27.4
SPIOM	BDR(USJ)	-	409	501	1368	2.73	29.1
SPIGMR	OSS(DQJ)	-	354	403	597	1.48	29.3
SPIGMR	OSS(DQJ)	9	351	400	640	1.60	27.5
SPIGMR	OSS(DQJ)	4	350	396	649	1.64	27.3
SPIGMR	BDR(DQJ)	-	408	512	1408	2.75	33.1
SPIGMR	BDL(DQJ)	-	390	480	1324	2.76	30.2
SPIGMR	BDR(DQJ)	9	401	497	1381	2.78	27.3

Knowing the relative effectiveness of the various method choices for the $p = 5$ case, we ran tests on the $p = 10$ case with a more restricted set of choices. On the grounds that realistic problems are usually too complex to allow for closed form Jacobian element calculation, we consider only the DQJ option. Table 2b shows the results for SPIGMR, with OSS and BDR as preconditioners. Here OSS is the better choice, as BDR suffers from repeated convergence failures in the linear iteration. The use of block grouping is cost-effective in both the SPIGMR-OSS and SPIGMR-BDR cases, and also for SPIOM-OSS, and the total storage required is reduced by a factor of .48 in the 9 group case and .45 in the 1-group case. That is, the relative flatness of the equilibrium solution is being exploited in the preconditioner, so as to reduce the total work space from about 38 NEQ to about 17 NEQ, with a speedup of 11% in addition.

TABLE 2b. RESULTS FOR PROBLEM 2 (p = 10)

Method	**PRE**	**NGR**	**NST**	**NNI**	**NLI**	**AVDIM**	**RT**
SPIGMR	OSS(DQJ)	-	374	420	626	1.49	63.9
SPIGMR	OSS(DQJ)	9	374	421	696	1.65	56.6
SPIGMR	OSS(DQJ)	1	375	424	704	1.66	56.6
SPIGMR	BDR(DQJ)	-	416	512	1370	2.68	73.0
SPIGMR	BDR(DQJ)	9	429	529	1488	2.81	58.6
SPIOM	OSS(DQJ)	1	375	422	691	1.64	56.2

(d) Test Problem 3.

The parameters in Problem 2 are such that the solution is rather flat spatially over the stiff part of the problem. In order to obtain a more realistic situation, the problem can be made much more inhomogeneous spatially by increasing the parameter α in the interaction coefficients b_i in (6.5). For Problem 3, we take $\alpha = 50$, leaving all other problem parameters unchanged from Problem 2. We consider the same two cases, $p = 5$ and $p = 10$, with the same initial conditions and discretization (on a 12x12 mesh). The solution now shows a spread in each c^i by a factor of up to 5.2 at equilibrium, and equilibrium is reached somewhat sooner.

Table 3a contains results for this problem in the 10-species case (p=5). The runs with SIOM (no preconditioning) and with SPIGMR-SOR failed to finish after 1 minute and so were stopped. Clearly this problem requires a preconditioning that includes the interaction Jacobian. All cases in the table are for user-supplied Jacobian elements; all our data suggests that the DQJ results would show a similar preference order. The OSS preconditioner was faster than OSF, as in Problem 2, but here both were slower than the IOJ (interaction-only) preconditioner. The diffusion apparently contributes less (relatively) to the stiffness, so that dropping the SOR preconditioning at the cost of a somewhat higher average Krylov dimension seems to be a good tradeoff in terms of total cost. The BDL and BDR (block-diagonal) preconditioners perform much like IOJ, with BDR slightly faster. Thus the inclusion of the diagonal transport coefficient has little effect on either NLI or the run time, when there is no block grouping. Block grouping is effective with both IOJ and BDR, but less so for IOJ; the reasons for the higher NLI there are not clear. For BDR, there is also evidence that using fewer than 16 groups is probably unwise. The considerable inhomogeneity of the equilibrium solution and the moderate amount of diffusion (though not dominant) are such that the optimal choice of preconditioner here is total block-diagonal on the right with some block grouping. The SPIOM runs with IOJ and BDR agree well with the SPIGMR results.

TABLE 3a. RESULTS FOR PROBLEM 3 (p = 5)

Method	PRE	NGR	NST	NNI	NLI	AVDIM	RT
SIOM	-	-	(failed to finish)				>60
SPIGMR	OSS	-	300	344	418	1.22	21.0
SPIGMR	SOR	-	(failed to finish)				>60
SPIGMR	IOJ	-	299	346	626	1.81	15.4
SPIGMR	BDL	-	300	346	670	1.94	16.2
SPIGMR	BDR	-	299	344	605	1.76	15.2
SPIGMR	OSF	-	306	351	437	1.25	27.9
SPIGMR	BDR	36	299	345	644	1.87	14.9
SPIGMR	BDR	16	299	342	635	1.86	14.5
SPIGMR	BDR	9	298	343	655	1.91	14.7
SPIGMR	IOJ	36	306	355	687	1.94	15.6
SPIGMR	IOJ	16	311	366	755	2.06	16.5
SPIOM	IOJ	-	306	359	706	1.97	16.9
SPIOM	BDR	-	299	342	601	1.76	15.0

For the larger case $p = 10$ $(s = 20)$, we again restrict our testing to the SPIGMR method and the DQJ option. The results are given in Table 3b. As before, both IOJ and BDR have lower run times than OSS as preconditoners, despite a higher average Krylov dimension. Also as before, block grouping is cost-effective, but by a much wider margin, because of the larger number of species and the DQJ option. With IOJ there is a clearly optimal choice of 36 groups, whereas BDR is less sensitive to the number of groups, with a minimal run time at 16 groups. The total work space storage for SPIGMR-BDR (16 groups) is about 19.4 NEQ, or about half of that for SPIGMR-BDR without block grouping.

TABLE 3b. RESULTS FOR PROBLEM 3 (p = 10)

Method	PRE	NGR	NST	NNI	NLI	AVDIM	RT
SPIGMR	OSS	-	322	367	466	1.27	54.1
SPIGMR	IOJ	-	318	363	658	1.81	41.9
SPIGMR	IOJ	36	330	380	776	2.04	37.1
SPIGMR	IOJ	16	365	432	1044	2.42	45.0
SPIGMR	BDR	-	331	380	738	1.94	46.8
SPIGMR	BDR	36	323	371	715	1.93	35.3
SPIGMR	BDR	16	324	378	754	1.99	34.8
SPIGMR	BDR	9	349	407	933	2.29	40.2

7. Conclusion

Krylov subspace iteration methods for linear systems can be combined with Newton iteration and the BDF method to obtain effective algorithms for stiff ODE systems. We have shown how the Arnoldi/IOM, GMRES/IGMRES, and CG methods fit into such combinations, and provided some theoretical support for the combined linear/nonlinear iteration when a finite-difference approximation is used for the linear operator. We have also given algorithms for these methods that include both scaling and preconditioning, and implemented these in an experimental solver called LSODPK.

For some problems, mainly those with much spectral clustering, the basic iteration methods with only scaling added are sufficient. But for most problems, preconditioning is necessary to achieve robustness. We have constructed several preconditioner combinations for reaction-diffusion problems, using the reaction and transport operators individually, or in combination as in operator splitting, and using grouping of blocks of Jacobian elements involved to achieve both storage and computational economies.

The tests we have reported show that these methods are effective on at least a small set of test problems. The optimal choices of method combinations depend on the nature of the problem, and may not be clear without experimentation. Preconditioning based on operator splitting does well on most of PDE-based problems we tested, but not all. For some cases, where diffusion and interaction terms are both equally dominant in the stiff part of the problem, we found that neither the splitting approach nor one using either operator alone seemed to do well. In general, we found that SPIGMR is slightly preferable to SPIOM, but that on problems where a symmetry assumption is true or approximately true, PCG or SPCG can be more efficient. The same is probably true for the GCR (Generalized Conjugate Residual) method, although we have not included that method in our testing.

There are other types of preconditioners in use elsewhere that we have not yet studied, but which might prove useful in connection with the methods used here. These include block-SOR (which has had considerable success by itself within BDF solvers), incomplete LU or modified forms thereof (MILU), polynomial preconditioners (e.g. Chebyshev), and Quasi-Newton (based on rank-one matrix updates). The block-diagonal preconditioners would benefit greatly from a dynamic group selection scheme.

Finally, we hope to see these methods extended to other types of stiff ODE methods (e.g. implicit RK), and to implicit forms of ODE's (e.g. $A\dot{y} = g$, or fully implicit systems), and to differential-algebraic equations.

References

[1] W. E. Arnoldi, The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem, *Quart. J. Appl. Math.*, 9 (1951), pp. 17-29.

[2] P. N. Brown, A Local Convergence Theory for Combined Inexact-Newton/Finite-Difference Projection Methods, *SIAM J. Num. Anal.*, 24 (1987), pp. 407-434.

[3] P. N. Brown, Decay to Uniform States in Food Webs, *SIAM J. Appl. Math.*, 46 (1986), pp. 376-392.

[4] P. N. Brown and A. C. Hindmarsh, Matrix-Free Methods for Stiff Systems of ODEs, *SIAM J. Num. Anal.*, 23 (1986), pp. 610-638.

[5] T. F. Chan and K. R. Jackson, The Use of Iterative Linear Equation Solvers in Codes for Large Systems of Stiff IVPs for ODEs, *SIAM J. Sci. Stat. Comp.*, 7 (1986), pp. 378-417.

[6] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, Inexact Newton Methods, *SIAM J. Num. Anal.*, 19 (1982), pp. 400-408.

[7] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[8] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, Variational Methods for Nonsymmetric Systems of Linear Equations, *SIAM J. Num. Anal.*, 20 (1983), pp. 345-357.

[9] C. W. Gear and Y. Saad, Iterative Solution of Linear Equations in ODE Codes, *SIAM J. Sci. Stat. Comp.*, 4 (1983), pp. 583-601.

[10] G. H. Golub and C. F. Van Loan, *Matrix Computations,* John Hopkins University Press, Baltimore, 1983.

[11] M. R. Hestenes and E. Stiefel, Methods of Conjugate Gradient for Solving Linear Systems, *J. of Res. of the Nat'l Bur. of Stan.,* 49 (1952), pp. 409-436.

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

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

[14] A. C. Hindmarsh, ODEPACK, A Systematized Collection of ODE Solvers, *Scientific Computing,* R. S. Stepleman et al. (eds.), North-Holland, Amsterdam, 1983, pp. 55-64.

[15] W. L. Miranker and I-L. Chern, Dichotomy and Conjugate Gradients in the Stiff Initial Value Problem, *J. Lin. Alg. Appl.,* 36 (1981), pp. 57-77.

[16] Y. Saad, Variations on Arnoldi's Method for Computing Eigenelements of Large Unsymmetric Matrices, *J. Lin. Alg. Appl.,* 34 (1980), pp. 269-295.

[17] Y. Saad, Krylov Subspace Methods for Solving Large Unsymmetric Linear Systems, *Math. Comp.,* 37 (1981), pp. 105-126.

[18] Y. Saad, Practical Use of Some Krylov Subspace Methods for Solving Indefinite and Nonsymmetric Linear Systems, *SIAM J. Sci. Stat. Comp.,* 5 (1984), pp. 203-228.

[19] Y. Saad and M. H. Schultz, GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM J. Sci. Stat. Comp.*, 7 (1986), pp. 856-869.

[20] P. N. Swartztrauber and R. A. Sweet, Algorithm 541: Efficient Fortran Subprograms for the Solution of Separable Elliptic Partial Differential Equations, *ACM Trans. Math. Softw.*, 5 (1979), pp. 352-364.

Technical Information Department • Lawrence Livermore National Laboratory
University of California • Livermore, California 94550