The Use of Iterative Linear-Equation Solvers in Codes for Large Systems of Stiff IVPs for ODEs §

Tony F. Chan †and Kenneth R. Jackson ‡

Research Report YALEU/DCS/RR-312
April 1984

§This work was supported in part by U.S. Air Force Grants AFOSR-81-0193 and AFOSR-83-0097, U.S. Department of Energy Grant DE-AC02-81ER10996, and Natural Sciences and Engineering Council of Canada Grants U0133 and A2521.

†Department of Computer Science, Yale Univ., Box 2158, Yale Station, New Haven, Conn. 06520.

‡Department of Computer Science, University of Toronto, Ontario, Canada, M5S 1A4.

Abstract

Systems of linear algebraic equations must be solved at each integration step in all commonly used methods for the numerical solution of systems of stiff IVPs for ODEs. Frequently, a substantial portion of the total computational-work and storage required to solve stiff IVPs is devoted to solving these linear algebraic systems, particularly if the systems are large. Over the past decade, several efficient iterative methods have been developed to solve large sparse (nonsymmetric) systems of linear algebraic equations. We study the use of a class of these iterative methods in codes for stiff IVPs. Our theoretical estimates and preliminary numerical results show that the use of iterative linear-equation solvers in stiff-ODE codes improves the efficiency - in terms of both computational-work and storage - with which a significant class of stiff IVPs having large sparse Jacobians can be solved.

1. Introduction.

As Gear [35, 36] and many others have noted, a major open problems in scientific computing is the efficient solution of large systems of stiff initial-value problems (IVPs) for ordinary differential equations (ODEs) of the form

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0. \tag{1.1}$$

These problems arise either directly in models of physical systems (such as chemical kinetics or electrical networks) or indirectly as a step in the solution of another problem (such as the application of the method-of-lines to a system of parabolic partial differential equations [59]). Consequently, the efficient solution of large systems of stiff IVPs is of great practical importance.

Although several authors have investigated techniques for avoiding implicitness in the numerical solution of stiff IVPs, most workers in the field still agree with Stetter's comment [79] that "all reasonable methods for stiff systems of ODEs have to be implicit", except, possibly, for special classes of problems. That is, a system of linear or nonlinear algebraic equations must be solved at each step of the numerical integration. Moreover, it seems that a Newton-like method must be used to solve the nonlinear systems to avoid a severe restriction on the stepsize. Consequently, large systems of linear equations must be solved in this case as well.

As we explain in more detail in §4, frequently a substantial portion of the total computational-work and storage required to solve large systems of stiff IVPs is devoted to solving systems of linear algebraic equations. Therefore, any improvement in the efficiency with which these linear systems are solved will directly improve the performance of the integrator. Fortunately, the linear algebraic systems that arise in large systems of stiff IVPs are usually sparse and this property can be exploited to great advantage.

Over the past decade, several efficient iterative methods have been developed to solve large sparse systems of linear algebraic equations. The Krylov subspace methods, of which the conjugate gradient method [43] is a well-know example, have proven to be particularly

effective for solving the linear systems that arise in the numerical solution of elliptic and parabolic partial differential equations. (See, for example, [2, 10, 13, 14, 16, 17, 24, 42, 54, 55, 57, 60, 61, 68, 69, 70, 81, 83, 86] and the references therein.) Therefore, it is natural to consider the use of iterative linear-equation solvers in codes for large systems of stiff IVPs for ODEs. Not only are iterative methods faster than direct solvers for many systems of linear algebraic equations, but also they require significantly less storage than direct solvers in most cases. In addition, the use of iterative methods will ease some of the restrictions on the stepsize- and order-selection strategies used in stiff-ODE codes; we believe that this may improve the efficiency of these codes as well.

The outline of the remainder of this paper is as follows. In §2, we review the numerical solution of the implicit formulas used in many of the most popular stiff-ODE codes, emphasizing the relationship between the user specified error tolerance for the IVP and the accuracy with which the implicit formulas must be solved. We also show that a large class of stiff IVPs have properties that make the associated systems of linear algebraic equations amenable to solution by iterative methods. We then review the "inexact Newton methods" in which the systems of linear equations that arise in Newton's method are solved approximately, rather than exactly. Again, we emphasize the relationship between the accuracy with which the implicit formulas and associated linear algebraic systems must be solved.

In §3, we review iterative linear-equation solvers with particular emphasis on two Krylov subspace methods: the preconditioned conjugate residual method for symmetric positive-definite systems and preconditioned Orthomin(k) for nonsymmetric positive-real systems.[1] We also point out how these iterative linear-equation solvers can be used in a stiff-ODE code that does not explicitly compute or store the Jacobian associated with the IVP, and, in particular, how the linear systems can be preconditioned in this case.

In §4, we develop theoretical estimates of the computational work and storage required

---

1 A real square matrix A is positive-real with respect to a real inner-product $(\cdot,\cdot)$ if and only if $(x,Ax) > 0$ for all nonzero real vectors x. Typically, the usual Euclidean inner-product, $(x,y) = \sum x_i y_i$, is used.

to solve the spatially-discretized two- and three-dimensional Heat Equation using a stiff-ODE solver that employs either direct of iterative linear-equation solvers. In §5, we present numerical results for the solution of the spatially-discretized two- and three-dimensional Heat and Convection-Diffusion Equations as well as the 30 Stiff Detest Problems [27, 29] using stiff-ODE solvers based upon either direct or iterative linear-equation solvers. Both the theoretical and numerical results look quite promising.

Finally, in §6, we present our conclusions.

This paper complements the work of Miranker and Chern [64], Gear and Saad [37], and Brown and Hindmarsh [3], who also studied the use of iterative linear-equation solvers in stiff-ODE codes. We believe our development of the properties of the linear algebraic systems that arise in stiff-ODE solvers that makes these linear systems amenable to solution by iterative linear-equation solvers is new, as is our analysis of the relationship between the three tolerances required in a stiff-ODE code employing an iterative linear-equation solver. In addition, our theoretical estimates and numerical results extend the work of the authors referenced above, and, in particular, show the importance of preconditioning in the solution of some large systems of stiff IVPs.

Although their point-of-view is distinctly different, the predictor-corrector methods developed and analyzed by van der Houwen and Sommeijer [51] are related to the stiff-ODE methods discussed in this paper and those referenced in the preceding paragraph.

2. Implicit Formulas.

Many numerical methods have been developed during the past few decades for the solution of systems of stiff IVPs for ODEs, and this continues to be an active area of research. Most of these methods can be classified as being in one of three families: linear multistep (multiderivative) methods, implicit Runge-Kutta methods, and extrapolation methods. Of these, the linear multistep methods have so far proven to be the most successful [27, 29], with the most widely used codes being DIFSUB [33, 34], GEAR [44], EPISODE [5], and LSODE

[48], each of which is based upon the Backward Differentiation Formulas (BDFs) popularized by Gear [34]. Therefore, in our discussion of implicit formulas below, we concentrate on the BDFs, although much of what we say applies to stiff methods in general.

A k-step BDF for the solution of (1.1) can be written in the form

$$y_n = \alpha_n^1 y_{n-1} + \cdots + \alpha_n^k y_{n-k} + h_n \beta_n f(t_n, y_n). \tag{2.0.1}$$

Tables of coefficients for these formulas may be found in [34]. To advance the numerical solution from t_{n-1} to $t_n = t_{n-1} + h_n$, (2.0.1) is solved for the approximation y_n to $y(t_n)$ using the previously computed approximations $\{y_{n-i}\}$. Because (2.0.1) is implicit in y_n, an equation of the form

$$F(y_n) = y_n - h_n \beta_n f(t_n, y_n) + c_n = 0 \tag{2.0.2}$$

must be solved at each step of the integration, where c_n contains the terms in (2.0.1) that do not depend upon y_n.

2.1. Accuracy Requirement for (2.0.2).

In general, (2.0.2) is nonlinear and cannot be solved exactly. Shampine [73, 74] discusses accuracy requirements for this equation. He notes that most stiff-ODE codes attempt to compute an approximate solution, $\tilde{y}_n$, to (2.0.2) satisfying

$$\|y_n - \tilde{y}_n\| \leq c_1 \cdot TOL, \tag{2.1.1}$$

where TOL is the user specified error tolerance for the IVP and c_1 is a positive constant (usually less than 1). Shampine [74] presents a convincing argument that, for a stiff-ODE solver, a more appropriate criterion is to accept $\tilde{y}_n$ if

$$\|F(\tilde{y}_n)\| \leq c_1 \cdot TOL. \tag{2.1.2}$$

Not only is this criterion more easily related to the accuracy requirement for the IVP, but also it is simpler to implement. Furthermore, Shampine gives an intuitive argument that suggests that, for most stiff problems,

$$\|y_n - \tilde{y}_n\| \leq \|F(\tilde{y}_n)\|. \tag{2.1.3}$$

However, Houbak, Norsett, and Thomsen [50] demonstrate that it often takes more computational work to satisfy (2.1.2) than (2.1.1) with little or no gain in the accuracy of the numerical

solution of the associated IVP. Although we do not address the interesting question of which of these stopping criterion is more appropriate in a stiff-ODE solver, we do develop a bound on $\|y_n - \tilde{y}_n\|$ in terms of $\|F(\tilde{y}_n)\|$ similar to one given by Williams [84], but using a somewhat different (and possibly simpler) derivation. This bound and some of the relations used in its derivation are important to our discussion of inexact Newton methods and iterative linear-equation solvers below.

The validity of (2.1.3) is intimately related to the stability of the associated IVP. Assume that the IVP satisfies

$$(f(t,y)-f(t,z),y-z) \leq \gamma(y-z,y-z) \tag{2.1.4}$$

for all (t,y) and (t,z) in the domain of interest, where y and z are real vectors, γ is a real (possibly negative) constant, and $(\cdot,\cdot)$ is a real inner-product. This assumption is frequently made in studying the nonlinear stability of formulas for stiff IVPs as it ensures the stability of the IVP (1.1) in the following sense. Let y(t) be a solution of (1.1) and let z(t) satisfy the same differential equation but have a different initial value, $z(t_0)$. If (2.1.4) is satisfied in a domain containing both y(t) and z(t), then

$$\|y(t)-z(t)\| \leq e^{\gamma(t-t_0)}\|y(t_0)-z(t_0)\|, \tag{2.1.5}$$

where $\|\cdot\|$ is the norm associated with the inner-product in (2.1.4). We say that the IVP is dissipative if and only if $\gamma < 0$. In this case, the IVP is asymptotically stable in the sense that the distance between y(t) and any neighbouring solution of the differential equation, z(t), decreases exponentially with t.

Inequality (2.1.4) can also be used to bound $\|y_n - \tilde{y}_n\|$ in terms of $\|F(\tilde{y}_n)\|$. Assume that (2.1.4) holds at $t = t_n$ in a domain containing both y_n and $\tilde{y}_n$. By (2.0.2) and (2.1.4),

$$(1-h_n\beta_n\gamma)(y-z,y-z) \leq (F(y)-F(z),y-z). \tag{2.1.6}$$

Hence, if $1-h_n\beta_n\gamma > 0$, then, by (2.1.6) and the Cauchy-Schwartz inequality,

$$\|y-z\| \leq \frac{1}{1-h_n\beta_n\gamma}\|F(y)-F(z)\|.$$

Taking $y = y_n$ and $z = \tilde{y}_n$, we get

$$\|y_n - \tilde{y}_n\| \leq \frac{1}{1-h_n\beta_n\gamma} \|F(\tilde{y}_n)\|, \tag{2.1.7}$$

from which it follows that, if $\gamma \leq 0$, then (2.1.3) holds for any $h_n > 0$, since $\beta_n > 0$ for the BDFs. Note also that, if $1-h_n\beta_n\gamma > 0$, then (2.1.7) ensures that any solution of $F(y_n) = 0$ is unique in the domain for which (2.1.4) holds. Moreover, if (2.1.4) holds at $t = t_n$ for all real vectors y and z and if $1-h_n\beta_n\gamma > 0$, then, by the Uniform Monotonicity Theorem [67], the unique solution of $F(y_n) = 0$ exists.

Now we consider in more detail for which class of stiff IVPs we can expect the condition $1-h_n\beta_n\gamma > 0$ to hold throughout the course of the numerical integration. First, note that, if the Jacobian $f_y(t,y)$ exists and is continuous, then the algebraically smallest γ for which (2.1.4) holds is

$$\gamma = \max \frac{(f_y(t,y)v,v)}{(v,v)}, \tag{2.1.8}$$

where the maximum is taken over all nonzero real vectors v and all (t,y) in the domain of interest. Hence, $F_y(y_n) = I - h_n\beta_n f_y(t_n,y_n)$ is positive-real if $1-h_n\beta_n\gamma > 0$.

It is easy to show that

$$\max \{ Re(\lambda) : \lambda \textit{ an eigenvalue of } f_y(t,y) \} \leq \gamma,$$

where $Re(\lambda)$ is the real part of λ. If $f_y(t,y)$ is symmetric, then equality holds in the last inequality, but, if $f_y(t,y)$ is nonsymmetric, then the inequality may be strict, as the example below demonstrates. However, the proof of Theorem 1 of [39] can be adapted easily to show that, for any fixed (t,y) and any $\epsilon > 0$, there exists a real inner-product and an associated γ satisfying (2.1.8) for which

$$\gamma - \epsilon \leq \max \{ Re(\lambda) : \lambda \textit{ an eigenvalue of } f_y(t,y) \} \leq \gamma. \tag{2.1.9}$$

For IVPs having a symmetric Jacobian, it is quite reasonable to expect $1-h_n\beta_n\gamma > 0$. In fact, for a large subclass of these problems, all the eigenvalues of the Jacobian $f_y(t,y)$ are nonpositive, from which it follows that $\gamma \leq 0$, whence $1-h_n\beta_n\gamma \geq 1$, since $h_n\beta_n > 0$. Thus, (2.1.3) holds. On the other hand, if $\gamma > 0$, then the Jacobian must have a positive eigenvalue arbitrarily close to γ in the domain on interest. Consequently, the differential equation has

solutions whose components may grow like $e^{\gamma t}$ in a neighbourhood of the solution of the IVP. Hence, it is reasonable to expect a stiff-ODE solver to choose a stepsize h_n for which $1-h_n\beta_n\gamma > 0$ to control the accuracy in the potentially growing components of the solution. In fact, if $\gamma > 0$, one would expect $1-h_n\beta_n\gamma$ to be close to 1, at least if the error tolerance is sufficiently stringent. Moreover, the numerical solution of (2.0.2) requires that $F_y(y_n)$ be "numerically" nonsingular. This effectively ensures that $1-h_n\beta_n\gamma > 0$, provided that $h_n\beta_n$ changes by small increments, since this inequality holds initially for h_n sufficiently small and, as $1-h_n\beta_n\gamma$ is essentially the algebraically smallest eigenvalue of $F_y(y_n) = I - h_n\beta_n f_y(t_n, y_n)$, $1-h_n\beta_n\gamma$ can never approach or pass through zero. Thus, for IVPs having a symmetric Jacobian, it is reasonable to expect that $F_y(y_n)$ will be positive-definite and (2.1.7) will hold in the event that (2.1.3) does not.

If the Jacobian $f_y(t,y)$ is nonsymmetric, then the assumption that $1-h_n\beta_n\gamma > 0$ is somewhat more problematic, since, for a given inner-product, the associated γ given by (2.1.8) may be much larger than the real part of any of the eigenvalues of $f_y(t,y)$. For example, consider the differential equation $y' = Ay$, where

$$A = \begin{bmatrix} -1 & 2a \\ 0 & -1 \end{bmatrix}.$$

The eigenvalues of A are both -1, but, for the usual Euclidean inner-product,

$$\gamma = \max \frac{(x, Ax)}{(x, x)} = -1 + |a|$$

can be arbitrarily large even though the eigenvalues of A are fixed. In particular, if $|a| > 1$, then A is not negative-real. Moreover, if a stiff-ODE solver is used to integrate $y' = Ay$ over a long time interval with absolute error control, the numerical solution will decay exponentially outside of an initial transient region and h_n will become large. Hence, for large t, it is reasonable to expect that $1-h_n\beta_n\gamma << 0$ and $F_y(y_n) = I - h_n\beta_n A$ will not be positive-real. Furthermore, for the usual Euclidean norm, the smallest constant c that ensures that

$$\|y-z\| \le c\,\|F(y)-F(z)\|$$

is $\|(I-h_n\beta_n A)^{-1}\|$, which may be larger than $\sqrt{8a/27}$. Thus, for large a, the residual is not a good estimate of the error for this problem in the usual Euclidean norm. This is not to say

that, for the usual Euclidean inner-product and all IVPs having nonsymmetric Jacobians, the condition $1-h_n\beta_n\gamma > 0$ will be violated and $F_y(y_n)$ will not be positive-real or that the residual will be a poor estimate of the error in (2.0.2), but this is the case for some problems.

On the other hand, even if the Jacobian $f_y(t_n,y_n)$ is nonsymmetric, by an argument similar to the one used in the symmetric case, it follows that it is reasonable to expect that the stepsize, h_n, in a stiff-ODE solver will be restricted by accuracy considerations to the extent that $1-h_n\beta_n\mathrm{Re}(\lambda) > 0$ for all eigenvalues λ of $f_y(t_n,y_n)$. In fact, for many stiff IVPs, $\mathrm{Re}(\lambda) \le 0$ for all the eigenvalues of $f_y(t_n,y_n)$, whence $1-h_n\beta_n\mathrm{Re}(\lambda) \ge 1$ without any restriction on the stepsize h_n. In any event, if $1-h_n\beta_n\mathrm{Re}(\lambda) > 0$ for all eigenvalues λ of $f_y(t_n,y_n)$, then, by (2.1.9), there exists a real inner-product with respect to which $F_y(y_n)$ is positive-real, although this inner-product may depend upon (t_n,y_n). For example, for the matrix A above, if we use the real inner-product $(x,y)_T = (x,Ty)$, where $T = diag(\delta^2,1)$ and $(\cdot,\cdot)$ is usual Euclidean inner-product, then

$$\gamma = \max \frac{(x,Ax)_T}{(x,x)_T} = -1 + |a\delta|$$

which, for $\delta = \epsilon/a$, is within ϵ of -1. Hence, for $0 < \delta < \frac{1}{|a|}$, A is negative-real, whence $I-h_n\beta_nA$ is positive-real with respect to the $(\cdot,\cdot)_T$ inner-product for any $h_n > 0$. Although these observations may not be of any practical importance in the selection of an error control strategy for a stiff-ODE solver, we believe that they may be of significance for the implementation of iterative linear-equation solvers, as will become evident in §3. Moreover, as we explain in that section, the iterative solvers that we consider are guaranteed to converge if $F_y(y_n)$ is positive-real, but may break-down otherwise. Hence, their break-down gives a warning that inequality (2.1.7) is violated.

2.2. Numerical Solution of (2.0.2).

For nonstiff-ODE solvers, it is common to use functional iteration to solve (2.0.2) or to employ an implicit formula, such as (2.0.1), as the corrector in a predictor-corrector method. However, for stiff-ODE solvers, the use of either of these techniques severely restricts the

stepsize and it is exactly this type of restriction that must be avoided for stiff problems. Therefore, in most stiff-ODE solvers, a chord-Newton method is used to solve (2.0.2) at each step in the integration. That is, given an initial approximation y_n^0 to the solution y_n of (2.0.2), the system of linear equations

$$W_n^k(y_n^{k+1} - y_n^k) + F(y_n^k) = 0 \tag{2.2.1}$$

is solved repeatedly until an acceptable approximation y_n^k is computed, where W_n^k is an approximation to the Newton iteration matrix

$$F_y(y_n^k) = I - h_n \beta_n f_y(t_n, y_n^k). \tag{2.2.2}$$

Frequently, W_n^k is just the Newton iteration matrix retained from an earlier iteration on the current or a previous step. Of course, if (1.1) is linear and the exact Newton iteration matrix (2.2.2) is used at each step, then (2.2.1) gives the solution to (2.0.2) in one iteration.

With the exception of GEARBI [47], all the "production" codes for stiff IVPs known to the authors employ direct methods to solve the system of linear algebraic equations (2.2.1). For example, GEAR, EPISODE, and LSODE each use Gaussian Elimination (GE) with partial pivoting, while DIFSUB computes the inverse of W_n^k explicitly. For large systems of stiff IVPs, great savings in both time and storage can be achieved by taking advantage of the sparsity of the Jacobian. This observation lead to the development of codes that employ either banded GE (such as GEARB [45], GEARIB [46], and LSODE) or sparse GE (such as GEARS [77] and LSODES [49]).

Furthermore, much of the consideration in choosing the formulas, strategies, and heuristics in a stiff-ODE solver is directed towards solving (2.0.2) as efficiently as possible. To this end, most stiff methods evaluate the Jacobian and refactor W_n^k as seldom as possible, since, as explained in more detail in § 4, the cost of these two operations may dominate the computation. Hence, in most stiff-ODE solvers, W_n^k remains unchanged for several consecutive integration steps.

The desire to avoid refactoring W_n^k also affects the choice of stepsize- and order-selection strategies in a stiff-ODE solver. If the stepsize or order is changed from one step to the next,

then at least one of the terms h_n or β_n occurring in the Newton iteration matrix (2.2.2) is changed as well. Therefore, unless W_n^k is updated and refactored, it may be a poor approximation to (2.2.2). As a result, the chord-Newton iteration (2.2.1) may fail to converge or converge too slowly. (Note that this observation applies to linear as well as nonlinear IVPs.) Consequently, the stepsize- and order-selection strategies in most current stiff-ODE solvers are restricted by this consideration. For example, EPISODE changes stepsize and/or order only after a failed step or when it estimates that it can increase its stepsize on the next step by a factor of at least 1.3. In addition to forcing the method to take more steps and function evaluations to integrate a problem than might otherwise be required, this constraint on the order- and stepsize-selection strategies reduces the "smoothness" of the dependence of the actual error committed by the code in solving a problem on the user specified error tolerance; it is generally agreed [36] that such "smoothness" is a very desirable property for an ODE solver to possess.

The choice of variable-stepsize implementation of a multistep formula is also affected by the consideration of how this choice will effect the efficiency of the Newton iteration. The two commonly used implementations are the fixed-coefficient implementation (FCI) of Nordsieck [65], which is used in DIFSUB, GEAR, and LSODE, and the variable-coefficient implementation (VCI), which is used in EPISODE. For nonstiff problems, both theoretical considerations and numerical testing have shown VCI to be superior to FCI for the Adams formulas. (See, for example, [28, 38, 52, 75] and the references therein.)

However, this clear superiority of one implementation over the other for Adams codes does not extend to stiff methods based upon the BDFs. The reason for this seems to be that, when VCI is used with a k-step BDF, the coefficient β_n in (2.2.2) continues to change on each of the k-1 steps following a stepsize change. Therefore, unless W_n^k is updated and refactored on each of these steps, it may be a poor approximation to the Newton iteration matrix (2.2.2). On the other hand, FCI does not share this disadvantage, since, for this implementation, β_n is a constant that depends only upon the formula being used. We believe that it is primarily for

this reason that the numerical results in [27, 29] indicate that GEAR is more efficient than EPISODE. On the other hand, the numerical results in [6, 7] suggest that EPISODE is more robust than GEAR. This empirical observation is supported by the theoretical results in [38], which show that VCI is more stable than FCI for the BDFs. (See [53] for a more detailed discussion of this topic.)

The cost of solving the implicit equation (2.0.2) also affects the choice of formulas used in a stiff-ODE solver. For example, although A-stable for arbitrarily high orders, the classical implicit Runge-Kutta formulas (IRKFs) [4] suffer the major disadvantage that the implicit system of equations associated with an S-stage formula is S times as large as the corresponding system (2.0.2) for the BDFs.

There has been a considerable effort during the past decade to alleviate some of the difficulties discussed above associated with solving an implicit equation of the form (2.0.2) at each step of the integration of a stiff ODE. However, one approach that has has only recently begun to be investigated actively is the use of iterative methods to solve (2.2.1) [3, 37, 64].

For parabolic PDEs, iterative methods have been popular since the early days of computing: SOR and ADI have been used effectively for several decades [80]. More recently, the conjugate gradient method [2, 16, 17, 55] has received a considerable amount of attention. We believe that the use of iterative methods in stiff-ODE codes should be investigated as well. It appears that these methods offer a great potential for reducing the cost - in terms of both time and storage - of solving large systems of stiff IVPs having sparse Jacobians. Furthermore, as discussed in more detail below, the use of iterative methods may alleviate some of the constraints on the stepsize- and order-selection strategies discussed above.

2.3. Inexact Newton Methods.

To begin, note that the linear equation (2.2.1) is solved only to obtain an approximate solution to the nonlinear equation (2.0.2): there is no reason why a direct linear-equation solver must be used in a stiff-ODE code to solve (2.2.1). Moreover, Sherman [76] and Dembo,

Eisenstat, and Steihaug [15] show that it is only necessary to approximate the solution of these linear equations "sufficiently accurately" to obtain a quadratic rate of convergence for the Newton iteration.

More specifically, consider the class of inexact Newton methods [15]. Given an initial guess y_n^0, any such method computes a sequence of values $\{y_n^k\}$ satisfying the recursion

$$\|F_y(y_n^k)(y_n^{k+1}-y_n^k) + F(y_n^k)\| \leq \eta_k \|F(y_n^k)\|, \tag{2.3.1}$$

where $\eta_k \leq \eta_{max} < 1$. In the next section, we discuss the use of iterative methods to compute $(y_n^{k+1}-y_n^k)$ satisfying (2.3.1), but, independently of how y_n^{k+1} is determined, Dembo, Eisenstat, and Steihaug [15] prove that, if

(1) $F(y_n) = 0$,

(2) F is continuously differentiable in a neighbourhood of y_n,

(3) $F_y(y_n)$ is nonsingular, and

(4) $\|y_n^0 - y_n\|$ is sufficiently small,

then $y_n^k \to y_n$ with a rate of convergence that is at least linear. In addition, they show that

(a) $y_n^k \to y_n$ superlinearly if $\eta_k \to 0$,

(b) $y_n^k \to y_n$ with strong order at least 1+p, $0 < p \leq 1$, if $\eta_k = O(\|F(y_n^k)\|^p)$ and F_y is Holder continuous with exponent p at y_n,[2] and

(c) $y_n^k \to y_n$ with weak order at least 1+p, $0 < p \leq 1$, if F_y is Holder continuous with exponent p at y_n and $\eta_k \to 0$ with weak order at least 1+p.

Taking p = 1 in (b), we get that an inexact Newton method may retain the quadratic rate of convergence characteristic of Newton's method.

Even though it is not necessary to factor or invert $F_y(y_n^k)$ in an inexact Newton method, it is necessary to evaluate the Jacobian of the IVP, $f_y(t_n, y_n^k)$, to compute $F_y(y_n^k)$ on each iteration. For large problems, the evaluation of the Jacobian may be very expensive, and,

---

2 A function g is Holder continuous with exponent p at y if there exists a constant L and a neighbourhood N of y such that $\|g(y)-g(x)\| \leq L\|y-x\|^p$ for all $x \in N$.

consequently, should be avoided whenever possible. Therefore, we consider the class of inexact chord-Newton methods for which (2.3.1) is replaced by

$$\|W_n^k(y_n^{k+1}-y_n^k) + F(y_n^k)\| \le \eta_k \|F(y_n^k)\|, \tag{2.3.2}$$

where, as in the previous subsection, W_n^k is an approximation to $F_y(y_n^k)$. But in this case, if an iterative method is used to solve (2.3.2), there is little additional cost associated with using the current value of the scalar $h_n\beta_n$ in W_n^k, although the Jacobian may remain unchanged from one inexact chord-Newton iteration to the next. In any case, the proof of Theorem 2.3 in [15] can be adapted easily to show that $y_n^k \to y_n$ linearly for an inexact chord-Newton method if, in addition to (1)-(4) above, we assume that W_n^k is a good approximation to $F_y(y_n)$ in the sense that

$$\|W_n^k - F_y(y_n)\| \le \gamma \quad \text{and} \quad \|(W_n^k)^{-1} - F_y(y_n)^{-1}\| \le \gamma,$$

where γ is the constant appearing in the similar inequalities (2.3) and (2.4), respectively, of [15].

Like a chord-Newton method, the rate of convergence of an inexact chord-Newton method is not superlinear in general. This together with the convergence results quoted above suggest that an appropriate choice for η_k is a constant $\eta < 1$, since (in theory at least) there is no benefit in allowing $\eta_k \to 0$, as there is for an inexact Newton method, while allowing $\eta_k \to 0$ makes the acceptance criterion (2.3.2) more stringent and, consequently, more expensive to satisfy for an iterative linear-equation solver.

In choosing a value for η, it is useful to note that, in many stiff ODE solvers such as GEAR, EPISODE, and LSODE, y_n^0 is normally a very good initial approximation to y_n in the sense that both $\|y_n^0 - y_n\|$ and $\|F(y_n^0)\|$ are close to TOL, the user specified error tolerance for the IVP, since y_n^0 is computed by an explicit formula of the same order as the implicit corrector. As a result, usually only one or two iterations of (2.2.1) are required to compute y_n^k satisfying either (2.1.1) or (2.1.2). To avoid an excessive number of evaluations of $F(y_n^k)$ when using an inexact chord-Newton method to solve (2.0.2), we also require that, on most steps, only one or two iterations of (2.3.2) be used to compute an acceptable y_n^k. Furthermore, note

that

$$F(y_n^{k+1}) \approx F_y(y_n^k)(y_n^{k+1}-y_n^k) + F(y_n^k) \approx W_n^k(y_n^{k+1}-y_n^k) + F(y_n^k).$$

Hence, if $\|F(y_n^0)\| \approx TOL$ and we want y_n^1 to satisfy (2.1.2), then a reasonable value for η is $r \cdot c_1$, where $r < 1$ is a positive constant and c_1 is the constant appearing in (2.1.2). An alternative is to replace (2.3.2) by

$$\|W_n^k(y_n^{k+1}-y_n^k) + F(y_n^k)\| \le r \cdot c_1 \cdot TOL, \tag{2.3.3}$$

since we require only that y_n^k satisfy the acceptance criterion (2.1.2) and not that y_n^k ultimately converges to y_n.

Based upon the relationship between $\|y_n^k-y_n\|$ and $\|F(y_n^k)\|$ developed in § 2.1, it also seems appropriate to use either (2.3.2) with $\eta = r \cdot c_1$ or (2.3.3) as the acceptance criterion for an inexact chord-Newton method when the acceptance criterion for the implicit equation (2.0.2) is (2.1.1) rather than (2.1.2), although the justification is more tenuous in this case. However, our numerical tests reported in §5, based upon a modified version of LSODE which employs LSODE's acceptance criterion of the form (2.1.1) for (2.0.2) and the acceptance criterion (2.3.3) for the inexact chord-Newton method, show that this heuristic works quite well in practice.

A stopping criterion of the form (2.3.3) for the inexact Newton method is also used by Brown and Hindmarsh [3] in their modified version of LSODE. In addition, they prove a result about the iterates y_n^k, which, although apparently not tight, suggests that the stopping criterion (2.3.3) is appropriate for stiff-ODE solvers.

Finally, we note that the accuracy of the approximation y_n^k to y_n affects not only the accuracy and stability of the underlying implicit ODE formula [58] but also other formulas, strategies, and heuristics used in the ODE solver. For example, in our preliminary numerical tests with a modified version of LSODE, we found that $y_n^1 = y_n^0$ often satisfied (2.3.3), particularly on the initial steps of the integration. However, accepting $y_n^1 = y_n^0$ has a deleterious effect upon the code, since the error estimate in LSODE is based upon the difference between y_n^0 and the accepted y_n^k and, moreover, the stepsize- and order-selection strategies are based

upon the magnitude of the error estimate. Hence, the error may be grossly underestimated and too large a stepsize selected for the next step. We were able to avoid this difficulty in part by taking $-F(y_n^k)$, rather than 0, as the initial guess for $y_n^{k+1}-y_n^k$ in the iterative solution of (2.3.3). Moreover, as this initial guess corresponds to the usual corrector in a predictor-corrector method, it produces a good initial approximation to the nonstiff components of (2.3.3). The choice of a good initial guess for $y_n^{k+1}-y_n^k$ is discussed in more detail for linear systems of IVPs in [64], where, in our notation, they consider initial guesses for $y_n^{k+1}-y_n^k$ of the form $-(I+A+\cdots+A^j)F(y_n^k)$ where $A = I-W_n^k$ and $j \geq 0$. However, the effect of the accuracy of the approximation y_n^k to y_n on the formulas, strategies, and heuristics of an ODE solver clearly requires much more study, not only for methods employing inexact Newton methods, but also for all methods based upon implicit formulas.

3. Iterative Linear-Equations Solvers.

In this section, we discuss the choice of iterative methods for solving the systems of linear equations that arise in inexact chord-Newton methods (2.3.2). Because these iterative methods must function as a component of a general purpose stiff-ODE solver, it is essential that they perform effectively for general sparse systems of linear equations and are not dependent upon any special matrix properties such as those, for example, associated with the five-point operator for the two-dimensional Laplacian. This consideration immediately eliminates PDE-related methods such as ADI or multi-grid. Moreover, even for the application of the method-of-lines to parabolic problems, many of these PDE-related methods are unsuitable because they require specific information about the PDE itself (e.g., grid structure or operator splittings) which is not usually available to a general-purpose stiff-ODE solver.

Although the classical iterative methods, such as Jacobi, Gauss-Seidel, and SOR, are not restricted to PDE-related problems, they may not converge if the linear system is not symmetric positive-definite. Moreover, these methods are often slow when used on their own and are, therefore, frequently coupled with an acceleration technique to improve their convergence rate. For example, the Symmetric SOR (SSOR) method [85] may be accelerated by

either the Chebyshev semi-iteration method or Richardson's second-order method [40]. One undesirable feature of these acceleration techniques is the need to estimate parameters to make them effective. Typically, these parameters depend upon the eigenvalues of the coefficient matrix, which are generally not known to the user a priori. However, adaptive Chebyshev methods, which automatically estimate these parameters, have been developed recently [60, 61] for both symmetric and nonsymmetric problems. These methods may be particularly effective for time dependent problems, since the coefficient matrix W_n^k of (2.3.2) (and, hence, the associated optimal Chebyshev parameters also) change slowly from step to step throughout the numerical integration of (1.1). Moreover, the Chebyshev iteration is guaranteed to converge if the required parameters are chosen "correctly" and if the real part of each of the eigenvalues of W_n^k is positive. As we argued in the last section, if this last condition is not satisfied, then the stepsize h_n is almost surely too large and should be reduced until this condition is satisfied to ensure a reliable numerical integration. However, to date we have not investigated in depth the use of adaptive Chebyshev methods in stiff-ODE solvers.

The Conjugate Gradient (CG) method is possibly the most well-known example of another class of iterative methods that has received considerable attention recently. CG was originally proposed by Hestenes and Stiefel [43] as a direct method, but it was re-introduced by Reid [68] as an iterative method for large sparse systems of linear equations. It has proven to be very effective in the latter role for a wide range of problems arising from, for example, geophysical applications [71], elliptic PDEs [10, 11, 13, 14, 72], and time-dependent PDEs [2, 16, 17, 55]. We believe that this class of iterative methods is also suitable for solving the linear systems that arise in stiff-ODE solvers. In particular, we consider the Preconditioned Conjugate Residual method [10, 24] and one of its generalizations for nonsymmetric problems, Preconditioned Orthomin [20, 24, 81]. In the remainder of this section, we give a brief description of these methods.

3.1. The Preconditioned Conjugate Residual Method.

Throughout this subsection, let A be a symmetric positive-definite matrix. To solve the system of linear equations

$$Ax = b, \tag{3.1.1}$$

the Conjugate Residual (CR) method, like CG, requires only that the user supply a routine to compute the matrix-vector product Av for any given vector v. Thus, CR can take full advantage of the sparsity of A. However, the effectiveness of CR can often be improved dramatically by applying CR to the equivalent preconditioned system

$$\hat{A}\hat{x} = \hat{b} \tag{3.1.2}$$

instead of (3.1.1), where $\hat{A} = S^{-1}AS^{-t}$ is a symmetric positive-definite matrix since A is, $\hat{x} = S^t x$, $\hat{b} = S^{-1}b$, and $Q = SS^t$ is "close" to A (in a sense to be made more precise below), but substantially less "expensive" than A to invert. We refer to CR applied to (3.1.2) as the Preconditioned Conjugate Residual (PCR) method and Q as the preconditioner.

One of the several equivalent forms of PCR is given in Figure 3.1.1. Although any inner-product can be used with PCR, the usual Euclidean inner-product is most often used in practice.

If $Q = I$, then PCR reduces to CR. Both methods require the same amount of storage, but, for $Q \neq I$, PCR requires one additional solve of the form $Qu = v$ per iteration. Note, though, that the matrix S associated with (3.1.2) is not required explicitly. Also, if $Q \neq I$, only the residual $\bar{r}_i = Q^{-1}(b - Ax_i) = Q^{-1}r_i$ is available in this implementation of PCR; if the residual r_i for (3.1.1) is required also, then either one additional matrix-vector product of the form Qu must be computed per iteration or one additional vector must be stored.

It is well-known [1, 10] that PCR is an optimal polynomial-based method in the sense that the i^{th} iterate x_i computed by PCR minimizes

$$\|\hat{r}_i\| = (\hat{r}_i, \hat{r}_i)^{1/2} = (r_i, Q^{-1}r_i)^{1/2} = \|r_i\|_{Q^{-1}} \tag{3.1.3}$$

over the translated preconditioned Krylov subspace

$$x_0 + \langle Q^{-1}r_0, (Q^{-1}A)Q^{-1}r_0, \ldots, (Q^{-1}A)^{i-1}Q^{-1}r_0 \rangle \tag{3.1.4}$$

Choose x_0.

Set $r_0 = b - Ax_0$.

Solve $Q\tilde{r}_0 = r_0$.

Set $p_0 = \tilde{r}_0$.

FOR i=0 STEP 1 UNTIL convergence DO

Solve $Qq_i = Ap_i$.

$a_i = (\tilde{r}_i, A\tilde{r}_i) / (Ap_i, q_i)$

$x_{i+1} = x_i + a_i p_i$

$\tilde{r}_{i+1} = \tilde{r}_i - a_i q_i$

$b_i = (\tilde{r}_{i+1}, A\tilde{r}_{i+1}) / (\tilde{r}_i, A\tilde{r}_i)$

$p_{i+1} = \tilde{r}_{i+1} + b_i p_i$

END FOR

Figure 3.1.1: The Preconditioned Conjugate Residual (PCR) Method.

where, $r_i = b - Ax_i$ is the residual for (3.1.1) associated with x_i, for $\hat{x}_i = Sx_i$, $\hat{r}_i = \hat{b} - \hat{A}\hat{x}_i = S^{-1}r_i$ is the corresponding residual for (3.1.2), x_0 is the initial guess for the solution of (3.1.1), and $r_0 = b - Ax_0$ is the associated residual.

The Preconditioned Conjugate Gradient (PCG) method can be implemented in a similar way, but we believe that, for our application, PCR is more appropriate than PCG. First, note that the inexact chord-Newton method requires that the residual of (2.2.1) satisfy (2.3.2). Therefore, for this problem, CR is the optimal unpreconditioned Krylov subspace method in

the sense that it minimizes the norm of the residual over the Krylov subspace (3.1.4) with $Q = I$. On the other hand, CG minimizes the A-norm of the error

$$\|x - x_i\|_A = (x - x_i, A(x - x_i))^{1/2} = (r_i, A^{-1} r_i)^{1/2} = \|r_i\|_{A^{-1}}, \tag{3.1.5}$$

rather than the residual itself, over the same space (3.1.4). However, this advantage is partially lost for the preconditioned methods, since PCR minimizes $\|r_i\|_{Q^{-1}}$ while PCG minimizes $\|r_i\|_{A^{-1}}$ over (3.1.4). Second, PCR can be generalized more easily than PCG for nonsymmetric problems, partly because (3.1.3) defines a norm for any nonsingular matrix A, provided Q is positive-real, while (3.1.5) does not. Moreover, the preconditioned Krylov subspace methods discussed in the next subsection which extend PCR to nonsymmetric systems are capable of minimizing the residual associated with (3.1.1) provided the preconditioning is applied "on the right". Therefore, we consider PCR only throughout the remainder of this subsection, although similar results hold for PCG.

Since x_i is a member of the affine space (3.1.4), the residual $\hat{r}_i$ associated with (3.1.2) satisfies

$$\hat{r}_i = (I - \hat{A} P_{i-1}(\hat{A}))\hat{r}_0 = R_i(\hat{A})\hat{r}_0, \tag{3.1.6}$$

where P_{i-1} is a polynomial of degree i-1 and R_i is a polynomial of degree i that satisfies $R_i(0) = 1$. If $\hat{A}$ has k distinct eigenvalues (which are all positive since $\hat{A}$ is symmetric positive-definite), we can choose a polynomial R_k of degree k such that $R_k(0) = 1$ and $R_k(\hat{\lambda}) = 0$ for each eigenvalue $\hat{\lambda}$ of $\hat{A}$. Since PCR minimizes $\|\hat{r}_i\|$ over (3.1.4) and this choice of R_k makes $\|\hat{r}_i\|$ zero, it follows that PCR, like PCG, converges to the exact solution of (3.1.1) in at most k steps. This is a slightly sharper version of the well-known result that PCG solves (3.1.1) in at most M steps (assuming exact arithmetic is used in the computation), where M is the dimension of the system (3.1.1). Note, though, that A and $\hat{A}$ may not have the same number of distinct eigenvalues. In particular, A may have only a few distinct eigenvalues, while $\hat{A}$ may have M. Consequently, in preconditioning, one must take care not to destroy an advantageous eigenvalue distribution.

More generally, one can derive from (3.1.3) and (3.1.6) the bound [10]

$$\|\hat{r}_i\| \le \left[\min_{R \in \Pi_i} \max_{1 \le j \le M} |R(\hat{\lambda}_j)| \right] \|\hat{r}_0\| \tag{3.1.7}$$

where Π_i is the set of polynomials of degree i or less that satisfy R(0)=1 and $\{\hat{\lambda}_j\}$ are the eigenvalues of $\hat{A}$, which are also the eigenvalues of $Q^{-1}A$ since these two matrices are similar. Using the i^{th} Chebyshev polynomial as a particular choice for R, one can derive the following bound [1, 10]

$$\|\hat{r}_i\| \le 2 \left[\frac{1 - 1/\sqrt{K(\hat{A})}}{1 + 1/\sqrt{K(\hat{A})}} \right]^i \|\hat{r}_0\| \tag{3.1.8}$$

where $K(\hat{A}) = \lambda_{max}(\hat{A})/\lambda_{min}(\hat{A})$ is the spectral condition-number of $\hat{A}$. Again, since $\hat{A}$ and $Q^{-1}A$ are similar, $K(\hat{A}) = K(Q^{-1}A)$. Also, since $\|\hat{r}_i\| = \|r_i\|_{Q^{-1}}$, inequalities (3.1.7) and (3.1.8) hold with $\|\hat{r}_i\|$ and $\|\hat{r}_0\|$ replaced by $\|r_i\|_{Q^{-1}}$ and $\|r_0\|_{Q^{-1}}$, respectively. Similarly, it is well-known [1, 10] that both (3.1.7) and (3.1.8) hold for PCG with $\|\hat{r}_i\|$ and $\|\hat{r}_0\|$ replaced by $\|r_i\|_{A^{-1}}$ and $\|r_0\|_{A^{-1}}$, respectively.

If A is well-conditioned or the eigenvalues of A are clustered, then CR reduces the error in the initial approximation very rapidly. Therefore, this method can be expected to perform very effectively on the linear equations that arise in mildly stiff IVPs or in large IVPs for which the eigenvalues of the associated Jacobian form a few clusters. In particular, CR is well-suited for problems with a few stiff components only. (See [30, 82] and the references therein for a more detailed discussion of this latter class of problems.) On the other hand, if A is ill-conditioned with its eigenvalues spread throughout a very large interval, then these bounds suggest that CR may require a great many more iterations than PCR to generate an acceptable approximation to the solution of (3.1.1). Since such linear algebraic systems arise during the numerical solution of many large systems of stiff IVPs (in particularly, those that arise from the spatial discretization of time-dependent PDEs), we believe that it is necessary to consider effective preconditionings for use with iterative linear-equation solvers in codes for stiff ODEs. The importance of preconditioning is demonstrated in the next two sections.

Among the more popular preconditionings for symmetric positive-definite systems are SSOR [42, 85], the Incomplete Cholesky (IC) factorization [63], and the Modified Incomplete

LU (MILU) factorization [41], a generalization of the Dupont-Kendall-Rachford (DKR) factorization [18]. Each of these preconditionings can be written in the form

$$Q = L L^t = A + E,$$

where L is a lower triangular matrix having the same sparsity structure as A, and E is an error matrix. These preconditionings do not require more storage than the original matrix A and, if implemented carefully [19], may require substantially less. Furthermore, to solve a system $Qu = v$ or to compute Qu for any of these preconditionings does not require more computational work than multiplying a vector by A and, when embedded in PCR, may require substantially less [19].

As explained in §2.1, if the Jacobian $f_y(t,y)$ is symmetric, then it is reasonable to expect that the chord-Newton iteration matrix W_n^k (2.2.1) will be symmetric positive-definite. If this is not the case, then the stepsize h_n is almost surely too large for the IVP and should be reduced until W_n^k is positive-definite to ensure a reliable numerical integration. For W_n^k symmetric positive-definite, the SSOR preconditioning is well-defined and both the IC and MILU incomplete factorizations can usually be formed [56, 62, 63].

PCR based upon these preconditionings has proven to be very effective for solving the linear equations associated with self-adjoint elliptic PDEs [10]. In the next two sections, we present some theoretical and empirical results for the spatially-discretized Heat Equation which show that PCR is very effective for this model problem also.

3.2. Preconditioned Orthomin.

Although both PCG and PCR have proven to be a very effective methods for solving symmetric positive-definite systems of linear algebraic equations, only recently have they been extended to solve more general systems effectively. As explained in the previous subsection, the solution of symmetric indefinite systems is not of great importance for stiff-ODE solvers. Therefore, we only consider the solution of nonsymmetric systems in this subsection.

An obvious way to extend either PCG or PCR to solve a nonsymmetric system $Ax = b$ is

to apply either of these methods to the symmetric positive-definite normal equations $A^tAx = A^tb$ or to the related system $AA^ty = b, x = A^ty$. In either case, though, for the badly conditioned systems that arise in stiff-ODE solvers, this approach is not attractive because it frequently leads to a slow rate of convergence.

Recently, several effective Krylov subspace methods have been developed which extend PCG and/or PCR to nonsymmetric systems. For example, Concus and Golub [12] and Widlund [83] developed a technique known as the Generalized Conjugate Gradient (GCG) method which uses the symmetric part of A, $S = \frac{1}{2}(A + A^t)$, as a preconditioning. GCG is particularly effective if a "fast" solver exists for S. Although this may be the case for many parabolic problems, this method is not well-suited for use in a general-purpose stiff-ODE solver, since there is no guarantee that systems of the form $Sx = b$ can be solved cheaply.

We chose to base our investigation of the use of iterative linear-equation solvers in codes for stiff IVPs upon the Preconditioned Orthomin(k) (POR(k)) method [20, 24, 81], an extension of PCR to nonsymmetric systems, partly because Elman's codes [21, 25] were available to us and partly because, like PCR, POR(k) minimizes the residual associated with the preconditioned system over a subspace described in more detail below. One of several other alternative Krylov subspace methods is discussed by Gear and Saad [37] and Brown and Hindmarsh [3].

In this subsection, we briefly outline POR(k) and the related Preconditioned Generalized Conjugate Residual (PGCR) method from which it is derived; a more detailed discussion of these methods can be found in [20, 24].

Like PCR, the effectiveness of POR(k) can often be improved dramatically by an appropriate choice of preconditioning. However, since POR(k) is applicable to nonsymmetric systems, there is more flexibility in the choice of preconditioning for POR(k) than there is for PCR. More specifically, the preconditioned system associated with (3.1.1) may be of the form

$$\hat{A}\hat{x} = \hat{b} \tag{3.2.1}$$

where $\hat{A} = Q_1^{-1}AQ_2^{-1}$, $\hat{x} = Q_2x$, $\hat{b} = Q_1^{-1}b$, Q_1 and Q_2 are substantially less "expensive" to

invert than A, and the preconditioning matrix $Q = Q_1 Q_2$ is "close" to A (in a sense to be made more precise below). In this formulation, the preconditioning (3.1.2) used with PCG or PCR is equivalent to a *symmetric positive-definite split preconditioning* having $Q_2 = Q_1^t$. Two other particular forms of preconditioning (3.2.1) are worth noting: *preconditioning on the left only* with $Q_2 = I$ and *preconditioning on the right only* with $Q_1 = I$.

The prototype of the PGCR family of methods from which POR(k) is derived is shown in Figure 3.2.1. The expression for a_i used there is mathematically equivalent to the expression given in Figure 3.1.1, but Elman [24] believes that the former is less sensitive to roundoff error for nonsymmetric problems.

Choose x_0.

Set $r_0 = b - Ax_0$.

Compute $\hat{r}_0 = Q_1^{-1} r_0$.

Compute $p_0 = Q_2^{-1} \hat{r}_0$.

FOR i=0 STEP 1 UNTIL convergence DO

$a_i = (\hat{r}_i, Q_1^{-1} A p_i) / (Q_1^{-1} A p_i, Q_1^{-1} A p_i)$

$x_{i+1} = x_i + a_i p_i$

$\hat{r}_{i+1} = \hat{r}_i - a_i Q_1^{-1} A p_i$

Compute p_{i+1}.

END FOR

Figure 3.2.1: The Prototype of the Preconditioned Generalized Conjugate Residual (PGCR) Family of Methods.

The two-term recurrence

$$p_{i+1} = \hat{r}_{i+1} + b_i p_i \tag{3.2.2}$$

used in PCR generates an $\hat{A}^t \hat{A}$-orthogonal sequence of search directions $\{p_i\}$ provided $\hat{A}$ is symmetric positive-definite. However, to obtain such a sequence for $\hat{A}$ nonsymmetric, it appears to be necessary to explicitly orthogonalize p_{i+1} against all previous search directions p_j in general. The recurrence recommended by Elman [20, 24] for PGCR is

$$p_{i+1} = Q_2^{-1}\hat{r}_{i+1} + \sum_{j=0}^{i} b_j^i p_j, \quad b_j^i = -\frac{(Q_1^{-1}AQ_2^{-1}\hat{r}_{i+1}, Q_1^{-1}Ap_j)}{(Q_1^{-1}Ap_j, Q_1^{-1}Ap_j)}. \tag{3.2.3}$$

PGCR consists of the prototype method given in Figure 3.2.1 together with these last two equations to compute p_{i+1}.

The recurrence (3.2.3) requires the storage of all past search directions p_j as well as far more computation than (3.2.2). This may be prohibitively expensive for large problems. In POR(k), the truncated recurrence

$$p_{i+1} = Q_2^{-1}\hat{r}_{i+1} + \sum_{j=j_i}^{i} b_j^i p_j, \quad j_i = \max(0, i-k+1), \tag{3.2.4}$$

is used instead, where b_j^i is computed as in (3.2.3). That is, p_{i+1} is orthogonalized against the past k search directions only. Hence, POR(k) requires the storage of at most k past search directions and the recurrence (3.2.4) is significantly cheaper to compute than (3.2.3).

The work per iteration for these preconditioned Krylov subspace methods is the same as for the unpreconditioned versions except that $Q_1^{-1}AQ_2^{-1}\hat{r}_{i+1}$ must be computed in place of Ar_{i+1}. In computing the former product, the intermediate result $Q_2^{-1}\hat{r}_{i+1}$ can be used to compute p_{i+1}, and $Q_1^{-1}Ap_{i+1}$ can be computed without any additional matrix-vector multiplies provided that $Q_1^{-1}Ap_j$ is saved instead of p_j. Moreover, for SSOR and several of the incomplete factorizations [41, 63], $Q_1^{-1}AQ_2^{-1}\hat{r}_{i+1}$ can be computed very efficiently using Eisenstat's technique [19].

If $Ax = b$ is preconditioning on the left only ($Q_2 = I$), then each of the PGCR family of methods requires the same amount of storage as its corresponding unpreconditioned version. Otherwise, each preconditioned method requires one more vector of storage than its corresponding unpreconditioned version. However, the residual $\hat{r}_i$ calculated in this implementation of the PGCR family of methods is the residual associated with the preconditioned system (3.2.1). If the residual $b - Ax_i = r_i = Q_1\hat{r}_i$ associated with (3.1.1) is required, then the storage advantage of preconditioning on the left only is lost: in this case, each preconditioned method requires one more vector of storage than its corresponding unpreconditioned version.

If $\hat{A}$ is positive-real, then both PGCR and POR(k), $k \geq 0$, are convergent descent methods in the sense that $\|\hat{r}_i\| \to 0$ as $i \to \infty$ and $\|\hat{r}_{i+1}\| < \|\hat{r}_i\|$ for $\hat{r}_i \neq 0$ [20, 24]. More specifically, PGCR, like PCR, minimizes $\|\hat{r}_i\|$ over the translated Krylov subspace (3.1.4). Hence, in this case also, the residual $\hat{r}_i$ at the i^{th} PGCR iteration satisfies

$$\|\hat{r}_i\| \leq \min_{R \in \Pi_i} \|R(\hat{A})\| \, \|\hat{r}_0\|, \tag{3.2.5}$$

where Π_i is the set of polynomials of degree i or less satisfying $R(0) = 1$. Using (3.2.5), one can prove [20, 24] that

$$\|\hat{r}_i\| \leq \left[1 - \frac{\lambda_{min}(\hat{S})^2}{\lambda_{max}(\hat{A}^t \hat{A})}\right]^{i/2} \|\hat{r}_0\|, \tag{3.2.6}$$

where $\hat{S} = \frac{1}{2}(\hat{A} + \hat{A}^t)$, the symmetric part of $\hat{A}$, is positive-definite since $\hat{A}$ is positive-real by assumption.[3] This bound, though, is not nearly as strong as (3.1.8) even though PGCR and PCR compute identical iterates x_i if A is symmetric positive-definite. If A has a complete set of eigenvectors, then

$$\|\hat{r}_i\| \leq K(\hat{T}) \hat{M}_i \, \|\hat{r}_0\|, \tag{3.2.7}$$

where $K(\hat{T}) = \|\hat{T}\| \, \|\hat{T}^{-1}\|$ is the condition number of the matrix $\hat{T}$ that diagonalizes $\hat{A}$,

$$\hat{M}_i = \min_{R \in \Pi_i} \max_{1 \leq j \leq M} |R(\hat{\lambda}_j)|, \tag{3.2.8}$$

and $\{\hat{\lambda}_j\}$ are the eigenvalues of $\hat{A}$. Note, if $\hat{A}$ is normal, then $K(\hat{T}) = 1$.

For $i > k$, the i^{th} iterate x_i computed by POR(k) minimizes $\|\hat{r}_i\|$ over the affine space

$$x_{i-k+1} + \langle p_{i-k+1}, \cdots, p_{i-1} \rangle$$

rather than the full translated Krylov subspace (3.1.4). However, in this case also, (3.2.6) holds for any k.

In all of the bounds listed above, $\hat{r}_i$ may be replaced by $Q_1^{-1} r_i$, since these two vectors are equal. Thus, one advantage of preconditioning on the right only ($Q_1 = I$) is that, in this case, the PGCR family of methods minimizes the residual r_i associated with the unpreconditioned problem $Ax = b$ at each iteration, since $\hat{r}_i = r_i$. As explained in the previous subsec-

---

3 $0 < (x, \hat{A}x) = (x, \hat{S}x)$ for all real non-zero vectors x, since $\hat{A} = \hat{S} + \hat{N}$ is positive-real and $(x, \hat{N}x) = 0$ for all real x because $\hat{N} = \frac{1}{2}(A - A^t)$ is skew-symmetric.

tion, this seems to be the most appropriate measure of the error to be minimized by an iterative method embedded in an inexact Newton iteration.

Provided that the associated chord-Newton iteration matrix W_n^k is positive-real, these bounds indicate that, like CR, the (unpreconditioned) GCR family of methods is very effective for mildly stiff IVPs and for stiff IVPs for which the eigenvalues of the associated Jacobian form a few clusters. On the other hand, if the eigenvalues of W_n^k are spread throughout a large domain, then, as is demonstrated in the next two sections, the effectiveness of POR(k) may be improved dramatically by an appropriate choice of preconditioning. However, care must be taken in choosing a preconditioning since, for the more general preconditionings considered in this subsection, $\hat{A}$ may fail to be positive-real even though A is. One advantage of using the symmetric positive-definite split preconditioning ($Q_2 = Q_1^t$) is that $\hat{A}$ is positive-real if and only if A is. Furthermore, if A is "nearly" symmetric, then so is $Q_1^{-1}AQ_1^{-t}$, and, for symmetric problems, the iterates x_i computed by POR(k), $k \geq 1$, are identical to the iterates computed by PCR and PGCR. Intuitively, if $Q_1^{-1}AQ_1^{-t}$ is "nearly" symmetric, then we expect the convergence rate of POR(k) to be close to that of PGCR. On the other hand, even if A and $Q = Q_1Q_2$ are both "nearly" symmetric, $Q_1^{-1}AQ_2^{-1}$ need not be, and the convergence rate of POR(k) may be significantly slower than that of PGCR.

Some popular preconditionings for nonsymmetric systems are SSOR [42, 85], the Incomplete LU (ILU) factorization [63], and the Modified Incomplete LU (MILU) factorization [41]. Each of these preconditionings can be written in the form

$$Q = LU = A + E,$$

where L and U, respectively, are lower and upper triangular matrices having the same sparsity pattern as A. With these factorizations, it is possible to precondition on the left or right or to use a split preconditioning with $Q_1 = L$ and $Q_2 = U$.

POR(k) with these preconditionings has proven to be very effective for solving the systems of linear algebraic equations associated with discretized non-self-adjoint elliptic PDEs. Obviously, the smaller k is the more efficient these methods are in terms of storage required.

Elman [24] also found that these methods are most efficient in terms of computational work for $k \leq 5$, with $k = 1$ often requiring the least amount of work.

As mentioned in §2.1, W_n^k is positive-real with respect to a given inner-product for any $h_n > 0$ for a large class of stiff IVPs, including all problems that are dissipative with respect to that inner-product. But, for any given inner-product, there are stiff IVPs for which W_n^k is not positive-real with respect to that inner-product for a reasonable choice of stepsize, h_n. In the latter case, any member of the PGCR family of methods based upon that inner-product may either compute an acceptable numerical solution or may "break-down" during the computation.

On the other hand, if all the eigenvalues of the Jacobian $f_y(t,y)$ lie either in the left-half complex plane or on the imaginary axis, then, without any restriction on the stepsize h_n, all the eigenvalues of W_n^k lie strictly in the right-half complex plane. Moreover, as discussed in §2.1, even if some of the eigenvalues of $f_y(t,y)$ lie in the right-half complex plane, it is reasonable to expect the stepsize h_n to be constrained by the accuracy requirements to the extent that all the eigenvalues of W_n^k will lie strictly in the right-half complex plane. In either case, it follows from (2.1.9) that there exists a real inner-product with respect to which W_n^k is positive-real. We hope to find a computationally effective way to utilize this result to dynamically choose an appropriate inner-product whenever W_n^k is not positive-real with respect to the usual Euclidean inner-product.

3.3. Jacobian-Free Stiff-ODE Solvers.

As several authors have noted, it is possible to avoid explicitly computing and storing the Newton iteration matrix W_n^k when solving nonlinear equations by an inexact Newton method coupled with a Krylov subspace method. To implement such a Newton-Krylov method, it is only necessary to be able to compute Jv for any given vector v, where J is an approximation to the Jacobian $f_y(t_n, y_n^k)$. In many stiff-ODE solvers, divided differences are used to form J. But, since J is not needed explicitly, the directional difference

$$[f(t_n, y_n^k + \delta v) - f(t_n, y_n^k)] / \delta$$

can be used to calculate an approximation to $f_y(t_n, y_n^k)v$ directly, where δ is a scalar constant.

Garg and Tapia [32] and O'Leary [66] recently investigated a similar idea for the application of CG to minimization problems. O'Leary shows that, in addition to saving storage, the Newton-CG method employing directional differences requires less computational work than the traditional discrete-Newton method for large problems.

Furthermore, the test results of Brown and Hindmarsh [3] based on the code developed by Gear and Saad [37] demonstrate that the use of directional derivatives to approximate matrix-vector products in a Newton-Krylov iteration is very effective for the spatially-discretized nonlinear parabolic problems that they considered.

However, all of the preconditionings referenced above require an explicit representation of the matrix J. Chan and Jackson [8] though, recently developed a class of nonlinear preconditionings, including a variant of SSOR, that does not require J explicitly and so can be used with Newton-Krylov methods employing directional differences. Moreover, for their test problems, the nonlinear SSOR preconditioning was as effective as the standard explicit SSOR preconditioning.

Since computing and storing Jacobians is a major source of expense in solving large stiff IVPs, the possibility of avoiding this computation seems very attractive, particularly for those problems for which we can expect a Krylov subspace method to converge very rapidly, such as those IVPs for which the eigenvalues of the associated Jacobian form a few clusters.

4. Theoretical Results for the Heat Equation.

The theoretical results in the last section can be adapted easily to show that the use of Krylov subspace methods in stiff-ODE solvers is very effective for a large class of IVPs. As a particular example, in this section, we compare the computational-work and storage required to solve the spatially-discretized Heat Equation by five stiff-ODE solvers each based upon the BDFs but using one of the following methods to solve the systems of linear algebraic equations that arise at each step in the numerical integration: (1) full Gaussian Elimination (GE),

(2) band GE, (3) sparse GE, (4) the Conjugate Residual (CR) method, or (5) the Preconditioned Conjugate Residual (PCR) method with either the SSOR [42, 85] or MILU [41] preconditioning. Although we do not advocate using these methods to solve the Heat Equation in practice, the spatially-discretized Heat Equation is a good test problem from a theoretical point-of-view because it is representative of a class of large stiff IVPs with sparse Jacobians and it can be analyzed thoroughly.

Consider the Heat Equation in one dimension (1-D) with homogeneous Dirichlet boundary conditions:

$$\begin{aligned} u_t(t,x) &= u_{xx}(t,x) \quad for\ (t,x) \in (t_0,t_f] \times (0,1), \\ u(t,0) &= u(t,1) = 0 \quad for\ t \in (t_0,t_f], \\ u(t_0,x) &= u_0(x) \quad for\ x \in [0,1]. \end{aligned} \tag{4.1}$$

Applying the method of lines with the usual centered-difference approximation with stepsize $\Delta = \frac{1}{m+1}$ to the spatial derivative of (4.1) gives the linear system of $M = m$ ODEs $y' = A_1 y$ for $t \in (t_0,t_f]$ with initial conditions $y_i(t_0) = u(t_0,i\Delta)$ for $i=1,...,m$, where $y_i(t)$ is an approximation to $u(t,i\Delta)$ and $A_1 = \Delta^{-2} diag(1,-2,1)$. It is well-known that the eigenvalues of A_1 are

$$\{\, 2\Delta^{-2}[\cos(i\Delta\pi) - 1] : i = 1,...,m \,\}. \tag{4.2}$$

All the eigenvalues are negative and are distributed throughout an interval from approximately $-\pi^2$ to approximately $-4\Delta^{-2}$. As the spatial discretization becomes finer, the resulting system of ODEs becomes both larger and stiffer, but the eigenvalues of the associated matrix A_1 do not cluster.

Also consider a similar spatial discretization of the Heat Equation in two dimensions (2-D) and three dimensions (3-D), each with homogeneous Dirichlet boundary conditions. For the 2-D problem, the matrix A_2 associated with the resulting linear system of $M = m^2$ ODEs $y' = A_2 y$ is $A_2 = \Delta^{-2} diag(I_1, T_1, I_1)$, where I_1 is the $m \times m$ identity matrix and $T_1 = diag(1,-4,1)$. Hence, the eigenvalues of A_2 are

$$\{\, \lambda_i + \lambda_j : i = 1,...,m,\ j = 1,...,m \,\},$$

where λ_i and λ_j are eigenvalues (4.2) of the 1-D problem. Similarly, for the 3-D problem, the

matrix A_3 associated with the resulting linear system of $M = m^3$ ODEs $y' = A_3 y$ is $A_3 = \Delta^{-2} diag(I_2, B_2, I_2)$, where I_2 is the $m^2 \times m^2$ identity matrix, $B_2 = diag(I_1, T_2, I_1)$, and $T_2 = diag(1, -6, 1)$. Hence, the eigenvalues of A_3 are

$$\{ \lambda_i + \lambda_j + \lambda_k : i = 1,...,m, \; j = 1,...,m, \; k = 1,...,m \},$$

where λ_i, λ_j, and λ_k are eigenvalues (4.2) of the 1-D problem.

We compare the computational-work per step required by each of the five stiff-ODE solvers considered above to integrate the spatially-discretized 1-D, 2-D, and 3-D Heat Problems. The numerical results presented in the next section show that, for any given problem in this class, each solver requires essentially the same number of steps throughout the numerical integration. Thus, for each solver, the computational-work per step is representative of the total computational-work required. Moreover, implicit in our comparison is the assumption that each stiff-ODE solver requires the same number of Newton iterations per step. The validity of this assumption is supported by the numerical results also.

The computational-work per step can be divided into three components: (1) the work to factor W_n^k for the GE variants or to compute a preconditioning for PCR (if W_n^k is refactored or the preconditioning is recomputed on that step), (2) the work to solve (2.2.1) using either the LU factorization for the GE variants or the (P)CR method, and (3) all the remaining work per step, which is termed the computational-work overhead. We measure the computational-work for each operation in terms of the number of arithmetic operations required to perform it.

Similarly, the storage required by each solver can be divided into two components: (1) the storage required to solve the system of linear algebraic equations and (2) all the remaining storage, which is termed the storage overhead.

For each of the five stiff-ODE solvers considered, both the computational-work and storage overheads are proportional to M, the size of the system of ODEs solved. Moreover, in both cases, the overhead is identical for each solver.

In determining the computational-work and storage required for full GE, we assume that no advantage is made of the sparsity of the matrices A_1, A_2, and A_3. Thus, in each case, to factor the Newton iteration matrix $I - h_n \beta_n A_i$, $i = 1,2,3$, requires computational-work asymptotically proportional to M^3, which is m^3, m^6, and m^9, for the 1-D, 2-D, and 3-D problems, respectively. In each case, both the computational-work required to solve the associated system of linear equations, given the LU factorization, and the storage needed for either the Newton iteration matrix or its LU factorization are asymptotically proportional to M^2, which is m^2, m^4, and m^6, for the 1-D, 2-D, and 3-D problems, respectively.

The half-band widths for A_1, A_2, and A_3 are 1, m, and m^2, respectively. Thus, in each case, to factor the associated Newton iteration matrix using band GE takes computational-work asymptotically proportional to m, m^4, and m^7, respectively. Also, in each case, both the computational-work to solve the associated system of linear equations, given the factorization, as well as the storage required for either the matrix or its factorization are proportional to m, m^3, and m^5, respectively.

In determining the computational-work and storage required for sparse GE, we assume that the asymptotically optimal factorization is used, although, frequently, this is not the case in practice for the 2-D and 3-D problems (cf. [22, 23]). Thus, the computational-work to factor the Newton iteration matrix $I - h_n \beta_n A_i$, $i = 1,2,3$, is asymptotically proportional to m, m^3, or m^6, respectively. In each case, both the computational-work to solve the associated system of linear equations, given the factorization, as well as the storage required for either the Newton iteration matrix or its factorization are proportional to m, $m^2 \log m$, and m^4, respectively.

As stated in §2.3, to compute a sufficiently accurate solution for the inexact chord-Newton method, it is generally necessary to reduce the initial residual associated with (2.3.3) by a constant factor η only, where η is typically about .1. From (4.2), the spectral condition-number of the Newton iteration matrix $I - h_n \beta_n A_i$, $i = 1,2,3$, increases with h_n from 1 at $h_n = 0$ to $4\pi^{-2}\Delta^{-2}$ as $h_n \to \infty$. Hence, from (3.1.8), the number of CR iterations required to reduce the initial residual by a factor of η is at most $\left\lceil \log(2/\eta)\pi^{-1}\Delta^{-1} \right\rceil$. In addition, because

of the sparsity of A_1, A_2, and A_3, the number of arithmetic operations required for each CR iteration is proportional to M, the dimension of the matrix. Thus, for each of these matrices, the computational-work required to compute a sufficiently accurate solution to (2.3.3) is at most asymptotically proportional to m^2, m^3, and m^4, respectively, and the storage required is asymptotically proportional to m, m^2, and m^3, respectively.

For either the SSOR [42, 85] or MILU [41] preconditioning (with the appropriate choice of scalar parameters), the spectral condition-number of the preconditioned Newton iteration matrix increases with h_n from 1 at $h_n = 0$ to $c\,\Delta^{-1}$ (for some constant c) as $h_n \to \infty$ [10]. Hence, the number of PCR iterations required to reduce the initial residual by a factor of η is at most asymptotically proportional to $\Delta^{-\frac{1}{2}}$. In addition, because of the sparsity of each Newton iteration matrix and its associated preconditioning, the number of arithmetic operations required for each PCR iteration is proportional to M, the dimension of the system. Thus, in each case, the computational-work required by PCR to compute a sufficiently accurate solution to (2.3.3) is at most proportional to $m^{1\frac{1}{2}}$, $m^{2\frac{1}{2}}$, and $m^{3\frac{1}{2}}$, respectively,‡ and the storage required remains proportional to m, m^2, and m^3, respectively. Moreover, in each case, the work required to compute the MILU factorization is proportional to the number of nonzeros in the matrix, m, m^2, and m^3, respectively, while no work at all is required to "compute" the traditional form of the SSOR "factorization".

The computational-work estimates given above are biased in favour of the GE variants. During the initial transient for each problem, the stepsize h_n is "small", and, consequently, the condition number of the Newton iteration matrix $I - h_n \beta_n A_i$, $i = 1,2,3$, or the associated preconditioned matrix is "close" to 1. As a result, the computational work per step for CR and PCR is much smaller during the initial transient than the estimates given above indicate: these estimates are accurate for h_n "large" only. On the other hand, the computational-work required by the GE variants to factor and solve the Newton systems is independent of the

‡ For the 1-D problem, PCR preconditioned by MILU (with the associated MILU parameter $\alpha = 0$) converges in one iteration with computational-work proportional to m, since, in this case, the MILU factorization is actually the exact LU factorization of $I - h_n \beta_n A_1$. This result holds for several other incomplete factorizations as well.

stepsize h_n. In addition, even for h_n "large", the computational-work estimates for PCR do not appear to be optimal, whereas the estimates for the three GE variants discussed above are optimal. For example, Chan, Jackson, and Zhu [9] show that there is strong evidence that, if the "AD-DKR" preconditioning is used for the 2-D problem, then the condition number of the preconditioned Newton iteration matrix is asymptotically proportional to $\Delta^{-2/3}$ and that the number of PCR iterations required to reduce the initial residual by a constant factor of η is asymptotically proportional to $\Delta^{-1/3}$. Again, because of the sparsity of A_2 and the associated AD-DKR preconditioning, the number of arithmetic operations required for each PCR iteration is proportional to M, the dimension of the system. Thus, there is strong evidence that, for the 2-D problem, the computational-work required by PCR to compute a sufficiently accurate solution to (2.3.3) is at most asymptotically proportional to $m^{2\frac{1}{3}}$, rather than $m^{2\frac{1}{2}}$. Moreover, both the storage required for PCR and the computational-work needed to compute the AD-DKR incomplete factorization remain asymptotically proportional to m^2.

The computational-work and storage required for each of the five stiff-ODE solvers is summarized in Table 4.1. These estimates show that, for this class of problems, the user should take advantage of sparsity: full GE is not competitive with either band or sparse GE. For the 1-D problem, band (sparse) GE is the most effective method. On the other hand, for the 2-D problem, both CR and PCR require asymptotically less storage than any of the GE variants and are asymptotically faster than either full or band GE. However, it is not clear which of PCR or sparse GE is asymptotically faster. The answer to this question depends on how frequently the linear systems must be refactored as the stepsize increases during the course of the numerical integration when sparse GE is used as well as the proportion of steps taken in the transient region where h_n is "small" and PCR requires less computational-work per step than the estimates in Table 4.1 indicate. For the 3-D problem, though, PCR is asymptotically faster than any of the other methods and requires significantly less storage than any of the GE variants.

		full GE	band GE	sparse GE	CR	PCR
1-D	factor	m^3	m	m	-	m
	solve	m^2	m	m	m^2	$m^{1\frac{1}{2}}$
	storage	m^2	m	m	m	m
	overhead	m	m	m	m	m
2-D	factor	m^6	m^4	m^3	-	m^2
	solve	m^4	m^3	$m^2 \log m$	m^3	$m^{2\frac{1}{2}}$
	storage	m^4	m^3	$m^2 \log m$	m^2	m^2
	overhead	m^2	m^2	m^2	m^2	m^2
3-D	factor	m^9	m^7	m^6	-	m^3
	solve	m^6	m^5	m^4	m^4	$m^{3\frac{1}{2}}$
	storage	m^6	m^5	m^4	m^3	m^3
	overhead	m^3	m^3	m^3	m^3	m^3

Table 4.1: The principal asymptotic term for the storage for and the computational-work per step required to factor and solve the linear algebraic systems that arise during the numerical integration of the spatially-discretized Heat Problem, as well as the overhead of all the remaining storage and computational-work per step required by the stiff-ODE solver.

5. Numerical Results.

We have replaced the direct linear-equation solvers in LSODE [48] by PCGPACK, a collection of preconditioned Krylov subspace methods implemented by Elman [21, 25]. We refer to the resulting experimental code as LSODCG. In this section, we report some preliminary numerical experiments with LSODCG to test the effectiveness of iterative linear-equation solvers in codes for large systems of stiff IVPs for ODEs. In particular, we compare the performance of LSODCG and LSODES[4] [49] on two pairs of spatially-discretized two- and three-dimensional linear parabolic problems as well as the performance of LSODCG and LSODE on the thirty Stiff Detest Problems [27, 29]. Although most of the Stiff Detest Problems are not large, they do test the robustness of the inexact chord-Newton method and the associated iterative linear-equation solvers used in LSODCG. These preliminary test results look quite promising.

4 LSODES is a variant of LSODE incorporating the Yale Sparse Matrix Package [22, 23] to solve the systems of linear algebraic equations by a sparse direct method.

5.1. LSODE, LSODES, and LSODCG.

We developed two variants of LSODCG: LSODCG.V1 and LSODCG.V2. In the former, we did not make any modifications to the formulas, strategies, or heuristics used in LSODE other than those modifications that were necessary to interface LSODE and PCGPACK, such as changing the data structure for storing matrices in LSODCG to the sparse "IA-JA-A" representation used in PCGPACK and many other sparse linear-equation solvers. In LSODCG.V2, we made one additional modification to LSODE in hope of reducing the number of inexact chord-Newton iterations and associated function evaluations throughout the course of the numerical integration: each time $h_n \beta_n$ is changed in LSODCG.V2, this scalar factor is updated in the Newton iteration matrix $I - h_n \beta_n J$ without re-evaluating J, and, if a preconditioner is being used in PCGPACK, it is recomputed. Since the Jacobian approximation J is not re-evaluated, these updates are relatively cheap compared to solving the associated linear algebraic equations.[5] In LSODE, LSODES, and LSODCG.V1, on the other hand, the Newton iteration matrix is updated only when the magnitude of the relative change in $h_n \beta_n$ is greater than CCMAX, a constant set to .3 in each of these three codes. Whenever the Newton iteration matrix is updated, either it is refactored in LSODES or, if a preconditioner is being used in PCGPACK, the preconditioner is recomputed in LSODCG.V1. In LSODE and LSODCG.V1, the Jacobian approximation J is re-evaluated whenever the Newton iteration matrix is updated; in LSODES, the Jacobian is re-evaluated only when it is estimated to be a poor approximation to the current Jacobian.

In all four codes, the acceptance criterion for the Newton iteration is of the form (2.1.1) with $c_1 = CONIT = \frac{5}{NQ+2}$, where NQ is the order of the BDF in use. In both variants of LSODCG, we use a stopping criterion for the iterative linear-equation solver in the inexact chord-Newton method of the form (2.3.3). Our numerical experiments show that any r in the

---

5 Updating the Newton iteration matrix would be even cheaper if LSODCG.V2 stored $\frac{1}{h_n \beta_n} I - J$ rather than $I - h_n \beta_n J$. This change is easy to implement.

range [.1,.5] is quite satisfactory: smaller values of r in this range lead to more PCGPACK iterations per inexact chord-Newton iteration, but frequently lead to fewer inexact chord-Newton iterations resulting in fewer function evaluations. Some numerical results along this line are reported in the third subsection. Also, as mentioned in §2.3, we take $-F(y_n^k)$, rather than 0, as an initial guess for $y_n^{k+1}-y_n^k$ in (2.3.3) for both variants of LSODCG.

5.2. Spatially-Discretized Linear Parabolic Problems.

Consider the Heat Equation in two dimensions (2-D)

$$u_t = u_{xx} + u_{yy} \tag{5.2.1}$$

and three dimensions (3-D)

$$u_t = u_{xx} + u_{yy} + u_{zz} \tag{5.2.2}$$

and the Convection-Diffusion Equation in 2-D

$$u_t = u_{xx} + u_x + u_{yy} + u_y \tag{5.2.3}$$

and 3-D

$$u_t = u_{xx} + u_x + u_{yy} + u_y + u_{zz} + u_z \tag{5.2.4}$$

each with homogeneous Dirichlet boundary-conditions either on the unit square [0,1]×[0,1] for the 2-D problems or on the unit cube [0,1]×[0,1]×[0,1] for the 3-D problems and initial conditions for $t \in [0,10.24]$

$$u(0,x,y) = 16x(1-x)y(1-y)$$

for the 2-D problems and

$$u(0,x,y,z) = 64x(1-x)y(1-y)z(1-z)$$

for the 3-D problems. As described in §4, applying the method of lines to the Heat Equation with m+1 evenly spaced grid points in each dimension ordered in the usual left-to-right bottom-to-top manner and using the usual three-point second-order centered-difference approximation to the second-order spatial derivatives with stepsize $\Delta = \frac{1}{m+1}$ yields a system of stiff ODEs of the form

$$y'(t) = A\,y(t), \tag{5.2.5}$$

where A is a constant symmetric negative-definite matrix with $A = A_2$ of dimension $M = m^2$ for the 2-D problem and $A = A_3$ of dimension $M = m^3$ for the 3-D problem. Applying the method of lines to the Convection-Diffusion Equation in a similar way, but with the addition of the two-point second-order centered-difference approximation to the first-order spatial derivatives, also yields a system of stiff ODEs of the form (5.2.5), where again A is a constant matrix of dimension $M = m^2$ for the 2-D problem and of dimension $M = m^3$ for the 3-D problem. In this case, though, A is a nonsymmetric negative-real matrix for both the 2-D and 3-D problems.

The eigenvalues and eigenvectors of the matrix A associated with the spatially-discretized Heat Equation (5.2.5) are well-known. Therefore, the exact solution of the associated IVP can be calculated easily for any t. For the Convection-Diffusion Problem, we used EISPACK [31, 78] in double precision on an IBM 3033 to calculate the eigenvalues and eigenvectors of the matrix A associated with the spatially-discretized 1-D problem of the form (5.2.5). Since the solution of the spatially-discretized 2-D and 3-D problems can be writted as the tensor product of solutions of the associated 1-D problems, the exact solution of the spatially-discretized 2-D and 3-D Convection-Diffusion Problems can be computed easily for any t also.

We used LSODES, LSODCG.V1, and LSODCG.V2 on an IBM 3033 computer in double precision to compute numerical solutions of the 2-D problems for m = 5, 10, 15, 20, 25, 30 and the 3-D problems for m = 3, 5, 7, 9. In each case, we used the BDFs with exact Jacobians (MF=21) and an absolute local error tolerance of $ATOL = 10^{-3}$ (ITOL=1 and RTOL=0). We integrated from the initial point t=0 to the output points $T = 2^i/100$, for i=0,1,2,...,10, using the continuation option (ISTATE=2) to integrate from one intermediate output point to the next. Because we did not require the output points to be hit exactly (ITASK=1), the solution vector is computed by interpolation and, on occasion, more than one solution vector is computed per integration step, as can be seen in some of the numerical results presented below. No optional input (IOPT=0) was used.

We used the PCGPACK implementation of the Preconditioned Conjugate Residual (PCR) method [25] and the Preconditioned Orthomin(k) (POR(k)) method [21, 25] for k=1,3,5 to solve the linear algebraic equations in LSODCG. For each of these methods, we used one of the three PCGPACK preconditionings:

1. NOPRE - no preconditioning,
2. TCSSOR - the two-cyclic implementation [19] of the SSOR preconditioning, or
3. TCDKR - the two-cyclic implementation [19] of the DKR [18] incomplete factorization, more generally referred to as the Modified Incomplete LU (MILU) factorization [41].

For the TCSSOR preconditioning, we used $\omega=2/[1+\sin(\pi\Delta/2)]$, where $\Delta=\frac{1}{m+1}$ is the spatial stepsize. This value of ω is "near optimal" [85] for the spatially-discretized 2-D and 3-D Heat Equations. Although this value of ω may not be "near optimal" for the spatially-discretized Convection-Diffusion Equation, it is appropriate in this case as well, since, in practice, an optimal value of ω for the problem to be solved is typically not known. For the TCDKR preconditioning, we used $\alpha=0$ for all problems, as recommended by Chandra [10]. In Orthomin(k), the preconditioning was applied on the right as described in §3.2. In both variants of LSODCG, we used a stopping criterion of the form (2.3.3) with r=.5 for each iterative linear-equation solver. However, we also set the maximum PCGPACK iterations permitted to solve any one linear system to max(100,10m).

5.2.1. Detailed Numerical Results for One Problem.

Detailed results for the numerical solution of the spatially-discretized 2-D Convection-Diffusion Problem with m=30 using LSODES, LSODCG.V1, and LSODCG.V2, respectively, are given in Tables 5.2.1.1, 5.2.1.2, and 5.2.1.3. The linear-equation solver used in LSODCG is POR(1) preconditioned by TCDKR. These numerical results are representative of the performance of these three codes on the problems considered in this subsection.

In each table,

- T is the output point,
- ERROR is the root-mean-square norm[6] of the difference between the numerical and exact solutions to the problem at T,
- HU and NQU, respectively, are the stepsize and order used by the BDF in the last step taken to reach T, and
- NST, NFE, and NJE, respectively, are the total number of steps, function evaluations, and Jacobian evaluations used from the initial point t=0 to the current output point T.

Note also that NFE−1 is the number of Newton iterations used from the initial point t=0 to the current output point T, since all but the first function evaluation is associated with a Newton iteration. For LSODES, NLU, MLTFAC, and MLTSLV, respectively, are the total number of

- LU factorizations used,
- multiplies used in the LU factorizations, and
- multiplies used in forward and backward substitutions

by the Yale Sparse Matrix Package to solve the linear equations that arise in the numerical integration from the initial point t=0 to the output point T. For LSODCG, NPRE and ITSTOT, respectively, are the total number of

- preconditionings computed, and
- iterations used by the linear-equation solvers

to integrate from the initial point t=0 to the output point T. ITSMAX is the maximum number of iterations used to solve any one system of linear equations in integrating from the initial point t=0 to the output point T. For each ODE solver, MLTTOT is the total number of multiplies used to solve the linear equations from the initial point t=0 to the output point T; for LSODES, MLTTOT = MLTFAC + MLTSLV.

---

6 The root-mean-square norm on an n-vector x is $\|x\| = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$.

Also shown in these tables is the storage required by each of the three ODE solvers. In each case, STRMAT is the number of nonzeros in the matrix A associated with the ODE (5.2.5). For LSODES, STRFAC is the number of nonzeros in the LU factorization computed by YSMP. For LSODCG, STRPRE and STRMTH, respectively, are the number of nonzeros required to store the preconditioning (M for TCSSOR or TCDKR and 1 for NOPRE) and the additional storage used in the iterative method ([4+2k]M + 2k for POR(k) and 4M for PCR). For both LSODES and LSODCG, STRTOT is the total number of storage locations required for the linear equation solvers.[7] For LSODES, STRTOT = 2·STRMAT + 2·STRFAC + 11·M + 2, while, for LSODCG, STRTOT = 2·STRMAT + STRMTH + STRPRE + M + 1.

The values of ERROR, HU, NQU, and NST are very similar for all three codes. From this we deduce that, for this class of problems at least, the error-control, stepsize-selection, and order-selection strategies in LSODE are not significantly affected by the use of an iterative linear-equation solver. Although NFE also is similar for all three codes, it is 7-10% smaller for LSODCG.V2 than for either of the other two codes indicating that the use of the current value of $h_n \beta_n$ in the Newton iteration matrix $I - h_n \beta_n J$ reduces slightly the total number of Newton iterations required throughout the integration.

The difference in NJE for LSODES and LSODCG.V1 demonstrates the superiority, for this class of problems at least, of the strategy used in LSODES over the one used in LSODCG.V1 (taken without modification from LSODE) for determining when a Jacobian re-evaluation is required. LSODCG.V2 uses two, rather than one, Jacobian evaluations because we did not alter LSODE's strategy that forces a Jacobian re-evaluation every MSBP (=20) steps. If this requirement were removed from LSODCG.V2, then it too would use only one Jacobian evaluation throughout the course of the integration, as it should for this class of problems.

---

7 We count each double precision and integer variable as one storage location although, on the IBM 3033, each double precision variable requires two words of storage whereas each integer variable requires only one. However, this makes little difference in the comparison of the storage required by iterative and direct linear-equation solvers since, for a given problem, both techniques use approximately the same proportion of integer to double precision variables.

T	ERROR	HU	NQU	NST	NFE	NJE	NLU	MLTFAC	MLTSLV	MLTTOT
0.010	0.826D-03	0.278D-02	2	8	11	1	3	611382	203220	814602
0.020	0.570D-03	0.482D-02	3	11	15	1	4	815176	284508	1099684
0.040	0.454D-03	0.738D-02	3	14	18	1	5	1018970	345474	1364444
0.080	0.413D-03	0.137D-01	3	19	24	1	6	1222764	467406	1690170
0.160	0.101D-03	0.186D-01	4	24	29	1	6	1222764	569016	1791780
0.320	0.166D-03	0.301D-01	3	30	36	1	7	1426558	711270	2137828
0.640	0.483D-05	0.131D+00	1	35	41	1	9	1834146	812880	2647026
1.280	0.127D-05	0.123D+01	1	36	42	1	10	2037940	833202	2871142
2.560	0.455D-07	0.123D+01	1	37	43	1	10	2037940	853524	2891464
5.120	0.280D-08	0.123D+02	1	38	44	1	11	2241734	873846	3115580
10.240	0.143D-08	0.123D+02	1	38	44	1	11	2241734	873846	3115580

Storage required by YSMP: STRMAT = 4380, STRFAC = 20322, STRTOT = 59306.

Table 5.2.1.1: LSODES solution of the spatially-discretized 2-D Convection-Diffusion Problem on an mxm grid with m = 30.

T	ERROR	HU	NQU	NST	NFE	NJE	NPRE	ITSTOT	ITSMAX	MLTTOT
0.010	0.830D-03	0.279D-02	2	8	11	3	3	16	3	257483
0.020	0.568D-03	0.482D-02	3	11	15	4	4	24	3	381455
0.040	0.463D-03	0.742D-02	3	14	19	5	5	33	3	519649
0.080	0.414D-03	0.137D-01	3	19	25	6	6	52	5	803421
0.160	0.106D-03	0.188D-01	4	24	30	6	6	71	5	1082034
0.320	0.115D-03	0.275D-01	3	30	37	7	7	115	10	1723035
0.640	0.462D-05	0.120D+00	1	35	42	9	9	157	10	2335714
1.280	0.139D-05	0.108D+01	1	36	43	10	10	175	18	2596869
2.560	0.418D-07	0.108D+01	1	37	44	10	10	188	18	2783434
5.120	0.113D-07	0.108D+02	1	38	45	11	11	211	23	3115699
10.240	0.509D-08	0.108D+02	1	38	45	11	11	211	23	3115699

Storage required by PCGPACK: STRMAT = 4380, STRPRE = 900, STRTOT = 15963.

Table 5.2.1.2: LSODCG.V1 solution of the spatially-discretized 2-D Convection-Diffusion Problem on an mxm grid with m = 30.

T	ERROR	HU	NQU	NST	NFE	NJE	NPRE	ITSTOT	ITSMAX	MLTTOT
0.010	0.835D-03	0.277D-02	2	8	10	1	4	15	3	245062
0.020	0.575D-03	0.488D-02	3	11	13	1	5	20	3	324689
0.040	0.467D-03	0.738D-02	3	14	16	1	6	28	3	446982
0.080	0.414D-03	0.138D-01	3	19	21	1	7	44	4	686409
0.160	0.872D-04	0.187D-01	4	24	27	2	9	67	5	1030549
0.320	0.177D-03	0.304D-01	3	30	33	2	10	97	6	1470763
0.640	0.402D-05	0.134D+00	1	35	38	2	12	135	11	2026554
1.280	0.110D-05	0.134D+01	1	36	39	2	13	155	20	2316153
2.560	0.483D-07	0.134D+01	1	37	40	2	13	169	20	2516940
5.120	0.342D-07	0.134D+02	1	38	41	2	14	194	25	2877649
10.240	0.188D-07	0.134D+02	1	38	41	2	14	194	25	2877649

Storage required by PCGPACK: STRMAT = 4380, STRPRE = 900, STRTOT = 15963.

Table 5.2.1.3: LSODCG.V2 solution of the spatially-discretized 2-D Convection-Diffusion Problem on an mxm grid with m = 30.

The sequence of values of NLU for LSODES and both NJE and NPRE for LSODCG.V1 are identical indicating that both codes had the same number of "significant" changes in $h_n\beta_n$ between each pair of output points, where by a "significant" change we mean that the magnitude of the relative change in $h_n\beta_n$ is greater than CCMAX (=.3). The values of NPRE for LSODCG.V2 are slightly larger than those for LSODCG.V1. Thus, assuming that the stepsize sequences in all three codes were similar, there were some changes of $h_n\beta_n$ in LSODES and LSODCG.V1 that were not "significant". Consequently, on some steps in LSODES and LSODCG.V1, the factor $h_n\beta_n$ in the Newton iteration matrix $I - h_n\beta_n J$ was not equal to the value of $h_n\beta_n$ used in the BDF on that step. On the other hand, LSODCG.V2 updates the factor $h_n\beta_n$ in the Newton iteration matrix whenever $h_n\beta_n$ changes. This may explain why LSODCG.V2 used fewer Newton iterations (NFE-1) than either of the other two codes. As a result, MLTTOT is smaller for LSODCG.V2 than LSODCG.V1 at each output point even though LSODCG.V2 re-computed the TCDKR preconditioner more frequently than LSODCG.V1 did: the reduction in the number of Newton iterations and associated linear-system solves more than offset the additional preconditioner computations.

The final value of MLTTOT is approximately the same for all three codes. However, during the initial stages of the integration, MLTTOT for the two variants of LSODCG is significantly less than for LSODES. For these steps, h_n is small and the spectrum of $I - h_n\beta_n J$ is clustered around 1. Consequently, only a few POR iterations (ITSMAX) are required to solve each linear system. However, as the integration proceeds and h_n grows, the spectrum of $I - h_n\beta_n J$ expands and more iterations are required to solve each linear system. However, for $h_n > 1$, ITSMAX does not grow significantly with h_n, since, as a rule of thumb, it is the relative size of the eigenvalues to one another, rather than the absolute size of the eigenvalues, that determines the rate of convergence of most Krylov subspace methods, and the relative size of the eigenvalues does not change significantly with h_n for $h_n > 1$. For LSODES, MLTFAC is approximately two-thirds of MLTTOT, and this factor grows as the grids become finer.

Although, for this problem, each code requires approximately the same amount of computational-work, STRTOT for LSODES is about four times the value of STRTOT for either variant of LSODCG. Moreover, this factor grows exponentially as the grids become finer. In addition, note that, for LSODES, STRFAC is about five times as large as STRMAT. On the other hand, for LSODCG with POR(1) preconditioned by TCDKR (or TCSSOR), STRPRE is about one eighteenth of STRTOT, since the TCDKR (or TCSSOR) preconditioner requires only one M-vector of storage.

5.2.2. A Summary of the Numerical Results for All the Test Problems.

We present below a summary of the numerical results for LSODES and LSODCG.V2 using the PCGPACK linear-equation solvers PCR and POR(k), k=1,3,5, preconditioned by NOPRE, TCSSOR, and TCDKR for the four spatially-discretized parabolic problems on $m \times m$ grids, m=5,10,15,...,30, for the 2-D problems, and $m \times m \times m$ grids, m=3,5,7,9, for the 3-D problems. Since the numerical results for LSODCG.V2 are similar to, but generally better than, those for LSODCG.V1, we have not included a summary of the numerical results for the latter code.

The total number of PCGPACK iterations, ITSTOT, and the maximum number of iterations for any one solve, ITSMAX, used by LSODCG.V2 throughout the integration are listed in Tables 5.2.2.1 and 5.2.2.2 and Tables 5.2.2.3 and 5.2.2.4, respectively. Graphs of m against ITSMAX on a log-log scale for POR(1) preconditioned by NOPRE, TCSSOR, and TCDKR are given in Plots 5.2.2.1 and 5.2.2.2 for the two 2-D problems.

The total number of multiplies, MLTTOT, used by LSODES and LSODCG.V2 to solve the linear algebraic systems throughout the integration are listed in Tables 5.2.2.5 and 5.2.2.6. Graphs of m against MLTTOT on a log-log scale for LSODES and LSODCG.V2 with POR(1) preconditioned by NOPRE and TCDKR are given in Plots 5.2.2.3 to 5.2.2.6 for all four problems.

The total storage, STRTOT, required by the linear-equation solvers in LSODES and

LSODCG.V2 for the 2-D and 3-D problems is given in Table 5.2.2.7. (Each linear equation solver requires the same amount of storage for both of the 2-D problems as well as the same amount of storage for both of the 3-D problems.) Graphs of m against STRTOT on a log-log scale for LSODES and LSODCG.V2 with POR(1) preconditioned by NOPRE and TCDKR (or TCSSOR) are given in Plots 5.2.2.7 and 5.2.2.8 for the 2-D and 3-D problems.

An entry of "*" in place of a number in these tables indicates that, during the course of the integration, the associated iterative linear-equation solver failed to converge in the maximum number of iterations allowed, max(100,10m). Only PCR with no preconditioning failed to converge, and it failed on the spatially-discretized 2-D Convection-Diffusion Problem with m=10 and 15 only. It is in fact surprising that PCR did not fail on more of the Convection-Diffusion Problems, since the linear systems associated with these problems are nonsymmetric and PCR is not (in theory at least) applicable to such systems.

Consider the results for LSODCG.V2 first. For these test problems, POR(1) is the most effective of the four basic PCGPACK methods considered. For a given problem and preconditioning, MLTTOT for POR(k), k=1,3,5, generally increases with k even though ITSTOT often decreases with k: the reduction in the number of iterations is more than offset by the additional work required per iteration as k increases. As mentioned above, PCR failed on two problems and is not guaranteed to converge for any nonsymmetric linear system. Furthermore, for the symmetric Heat Problems, PCR is not significantly more efficient than POR(1). On the contrary, when preconditioned, PCR frequently requires more multiplies than POR(1) since, even though PCR may require fewer iterations, fewer multiplies are required per iteration to precondition POR(1) on the right than to precondition PCR symmetrically, as is required for the latter method.

Of the three preconditionings, TCDKR is nearly always the most effective in terms of both multiplies and iterations required. The effectiveness of preconditioning is much more pronounced for the nonsymmetric Convection-Diffusion Problems than for the symmetric Heat Problems. In fact, for the latter class of problems, MLTTOT for TCSSOR is frequently

Method	2-D Problem						3-D Problem			
	m						m			
	5	10	15	20	25	30	3	5	7	9
PCR NOPRE	96	273	269	417	535	652	42	85	132	197
PCR TCSSOR	64	138	198	247	300	358	47	66	103	125
PCR TCDKR	49	92	110	140	166	197	34	49	71	82
POR K=1 NOPRE	96	275	269	417	535	652	42	85	132	197
POR K=1 TCSSOR	67	118	201	252	306	360	45	64	103	123
POR K=1 TCDKR	50	98	114	146	177	209	34	52	73	85
POR K=3 NOPRE	96	275	269	411	535	652	42	85	132	197
POR K=3 TCSSOR	65	128	189	243	296	359	44	65	102	122
POR K=3 TCDKR	49	95	111	143	169	203	34	51	71	84
POR K=5 NOPRE	96	275	269	411	535	652	42	85	132	197
POR K=5 TCSSOR	64	127	191	243	291	351	44	65	102	122
POR K=5 TCDKR	49	94	109	139	165	198	34	51	71	83

Table 5.2.2.1: The total number of PCGPACK iterations, ITSTOT, used by LSODCG.V2 throughout the numerical integration of the spatially-discretized 2-D and 3-D Heat Problem on a mxm and mxmxm grid, respectively.

Method	2-D Problem						3-D Problem			
	m						m			
	5	10	15	20	25	30	3	5	7	9
PCR NOPRE	137	*	*	552	775	855	67	131	188	225
PCR TCSSOR	63	134	203	269	329	366	46	62	102	114
PCR TCDKR	47	90	147	150	178	192	36	49	66	83
POR K=1 NOPRE	121	293	390	556	609	983	69	121	167	240
POR K=1 TCSSOR	70	139	183	260	316	356	45	64	99	125
POR K=1 TCDKR	49	92	150	139	177	194	36	49	67	81
POR K=3 NOPRE	131	279	412	559	768	860	64	111	139	231
POR K=3 TCSSOR	69	124	190	251	322	397	44	63	100	111
POR K=3 TCDKR	49	88	147	136	174	207	35	48	67	80
POR K=5 NOPRE	124	282	394	563	773	846	59	107	138	230
POR K=5 TCSSOR	69	123	187	275	306	364	44	63	99	110
POR K=5 TCDKR	48	86	146	134	172	204	35	48	67	80

Table 5.2.2.2: The total number of PCGPACK iterations, ITSTOT, used by LSODCG.V2 throughout the numerical integration of the spatially-discretized 2-D and 3-D Convection-Diffusion Problem on a mxm and mxmxm grid, respectively.

Method	2-D Problem						3-D Problem			
	m						m			
	5	10	15	20	25	30	3	5	7	9
PCR NOPRE	11	17	25	33	42	50	5	7	14	20
PCR TCSSOR	8	11	14	18	21	24	6	8	11	13
PCR TCDKR	6	10	12	15	18	20	4	7	9	10
POR K=1 NOPRE	11	20	25	33	42	50	5	7	14	20
POR K=1 TCSSOR	10	13	16	17	23	26	6	8	10	12
POR K=1 TCDKR	7	11	14	17	23	23	4	7	10	12
POR K=3 NOPRE	11	20	25	33	42	50	5	7	14	20
POR K=3 TCSSOR	8	11	16	19	22	23	6	8	11	12
POR K=3 TCDKR	6	9	14	17	20	22	4	7	9	11
POR K=5 NOPRE	11	20	25	33	42	50	5	7	14	20
POR K=5 TCSSOR	8	11	14	18	21	25	6	8	11	12
POR K=5 TCDKR	6	9	12	15	18	21	4	7	9	12

Table 5.2.2.3: The maximum number of PCGPACK iterations, ITSMAX, used by LSODCG.V2 throughout the numerical integration of the spatially-discretized 2-D and 3-D Heat Problem on a mxm and mxmxm grid, respectively.

Method	2-D Problem m						3-D Problem m			
	5	10	15	20	25	30	3	5	7	9
PCR NOPRE	26	*	*	76	119	105	14	36	49	34
PCR TCSSOR	8	11	14	17	19	24	7	8	11	11
PCR TCDKR	6	11	14	16	18	22	4	7	9	11
POR K=1 NOPRE	17	43	52	87	80	236	12	23	30	42
POR K=1 TCSSOR	7	14	17	19	21	25	7	9	10	12
POR K=1 TCDKR	6	11	15	18	18	25	4	7	9	12
POR K=3 NOPRE	18	42	63	86	116	128	12	21	26	35
POR K=3 TCSSOR	7	11	15	17	23	28	6	8	11	12
POR K=3 TCDKR	6	10	15	16	18	21	4	6	9	11
POR K=5 NOPRE	16	44	52	75	113	116	9	18	26	33
POR K=5 TCSSOR	7	10	14	21	20	23	6	8	10	11
POR K=5 TCDKR	5	9	14	15	16	19	4	6	9	11

Table 5.2.2.4: The maximum number of PCGPACK iterations, ITSMAX, used by LSODCG.V2 throughout the numerical integration of the spatially-discretized 2-D and 3-D Convection-Diffusion Problem on a mxm and mxmxm grid, respectively.

Method	2-D Problem m						3-D Problem m			
	5	10	15	20	25	30	3	5	7	9
LSODES	10356	104966	347916	865202	1680568	3115580	16554	271491	2412900	14236235
PCR NOPRE	27405	303220	675184	1849272	3700203	6486182	15883	144115	605332	1889686
PCR TCSSOR	40036	334918	1069556	2355255	4448810	7609396	37247	255252	1072048	2763856
PCR TCDKR	32838	237926	636751	1412001	2587342	4366975	29664	205745	795430	1951422
POR K=1 NOPRE	28880	328721	726934	1999272	4008328	7035182	16396	150990	639289	2009242
POR K=1 TCSSOR	26988	190201	725462	1619344	3071298	5199680	22582	154048	672012	1717008
POR K=1 TCDKR	21455	163218	428364	968413	1823364	3090979	18891	135052	508784	1256204
POR K=3 NOPRE	33668	429857	968942	2719163	5634820	10003581	17708	173550	781429	2545302
POR K=3 TCSSOR	28752	235738	814011	1921186	3725054	6585355	23542	169736	742154	1927060
POR K=3 TCDKR	22375	177254	472584	1102094	2061592	3624176	19383	141172	540918	1365608
POR K=5 NOPRE	35644	498485	1140646	3307653	6985540	12558727	17790	183326	868979	2901946
POR K=5 TCSSOR	29140	245935	893818	2118869	4135414	7418851	23788	174624	777174	2025520
POR K=5 TCDKR	22679	183237	491248	1147825	2179768	3885289	19383	143428	562548	1418124

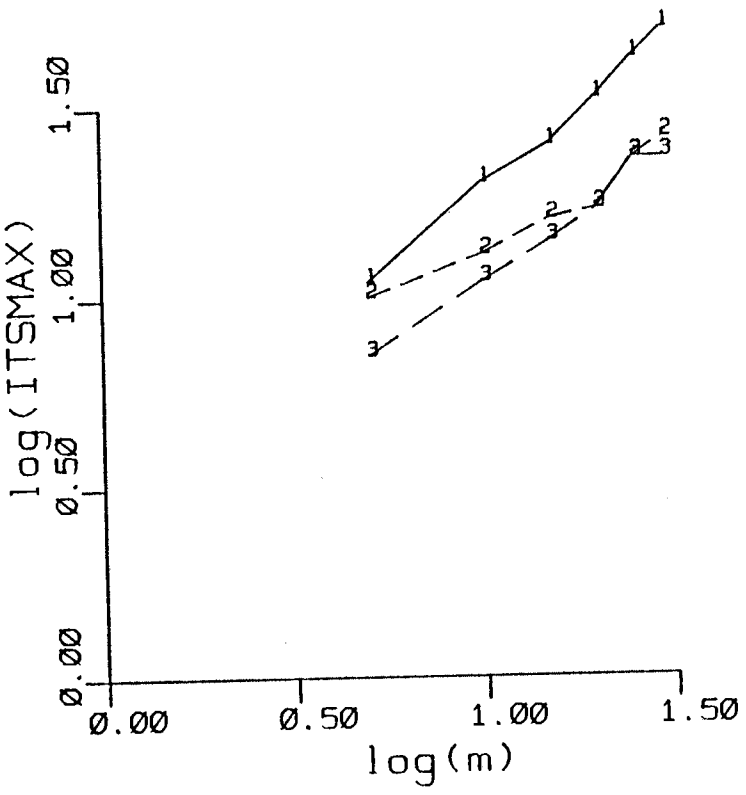
Table 5.2.2.5: The total number of multiplies, MLTTOT, used by LSODES and LSODCG.V2 to solve the linear algebraic systems throughout the numerical integration of the spatially-discretized 2-D and 3-D Heat Problem on a mxm and mxmxm grid, respectively.

Method	2-D Problem m						3-D Problem m			
	5	10	15	20	25	30	3	5	7	9
LSODES	10356	104966	347916	865202	1680568	3115580	17622	254595	2400687	14236235
PCR NOPRE	37942	*	*	2432742	5328457	8474507	23438	212782	838684	2141490
PCR TCSSOR	39519	325371	1094541	2559297	4865492	7772132	36517	241945	1062981	2538817
PCR TCDKR	31804	233563	833065	1508499	2767114	4265265	31636	205146	756072	1971107
POR K=1 NOPRE	35901	348365	1049339	2656349	4556076	10572603	25386	209387	797139	2427288
POR K=1 TCSSOR	28191	223160	662082	1670319	3170240	5142792	22664	154424	648730	1745685
POR K=1 TCDKR	21083	154627	558646	924439	1827538	2877649	19918	128596	474423	1205896
POR K=3 NOPRE	48706	447681	1519823	3768229	8209009	13349053	27938	236983	831492	3026822
POR K=3 TCSSOR	30099	228667	820678	1982488	4058544	7308315	23624	165056	729998	1738695
POR K=3 TCDKR	22451	164412	632610	1043708	2120654	3679020	19894	133588	512533	1304360
POR K=5 NOPRE	51444	543574	1749427	4754404	10467203	16729712	27046	255015	935132	3522528
POR K=5 TCSSOR	30783	237660	873010	2448637	4345524	7674620	23952	169192	753790	1795587
POR K=5 TCDKR	22079	167047	673038	1099601	2273542	3979381	19894	135844	531073	1367812

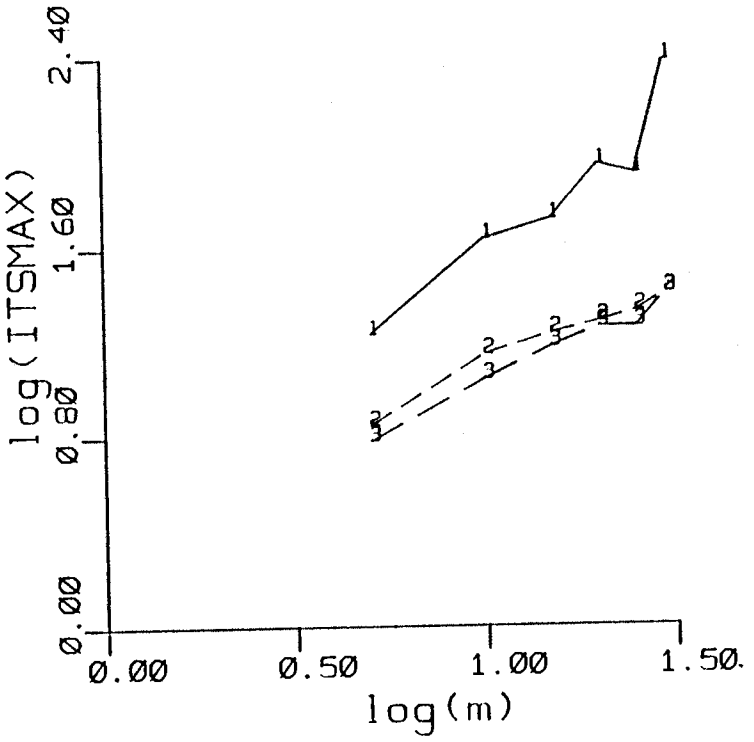
Table 5.2.2.6: The total number of multiplies, MLTTOT, used by LSODES and LSODCG.V2 to solve the linear algebraic systems throughout the numerical integration of the spatially-discretized 2-D and 3-D Convection-Diffusion Problem on a mxm and mxmxm grid, respectively.

Method	2-D Problem m						3-D Problem m			
	5	10	15	20	25	30	3	5	7	9
LSODES	845	4450	11549	22606	37609	59306	1103	7909	32415	97425
PCR NOPRE	337	1422	3257	5842	9177	13262	407	2077	5931	12881
PCR TCDKR	361	1521	3481	6241	9801	14161	433	2201	6273	13609
POR K=1 NOPRE	389	1624	3709	6644	10429	15064	463	2329	6619	14341
POR K=1 TCDKR	413	1723	3933	7043	11053	15963	489	2453	6961	15069
POR K=3 NOPRE	493	2028	4613	8248	12933	18668	575	2833	7995	17261
POR K=3 TCDKR	517	2127	4837	8647	13557	19567	601	2957	8337	17989
POR K=5 NOPRE	597	2432	5517	9852	15437	22272	687	3337	9371	20181
POR K=5 TCDKR	621	2531	5741	10251	16061	23171	713	3461	9713	20909

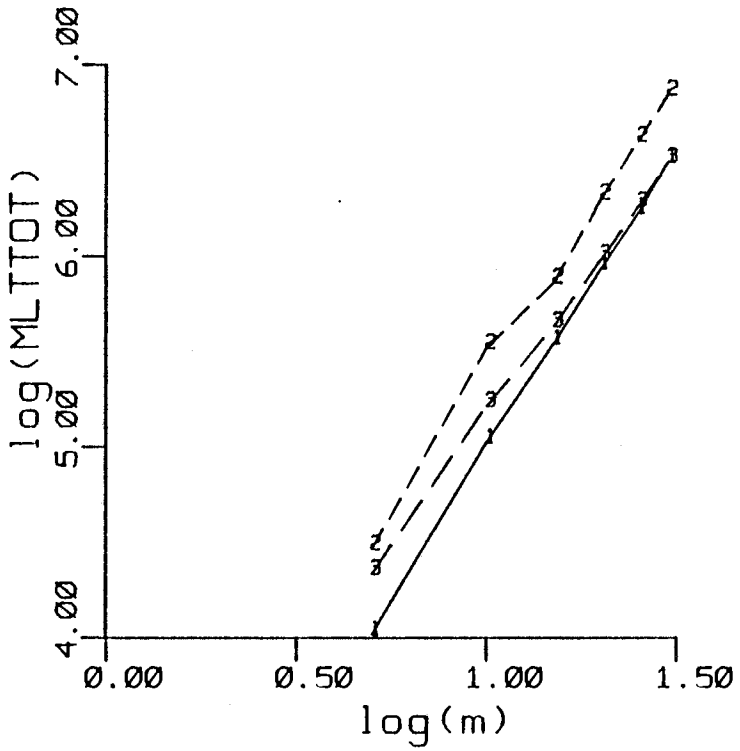
Table 5.2.2.7: Total storage, STRTOT, required by the linear-equation solvers in LSODES and LSODCG.V2 for the 2-D and 3-D problems on a mxm and mxmxm grid, respectively.



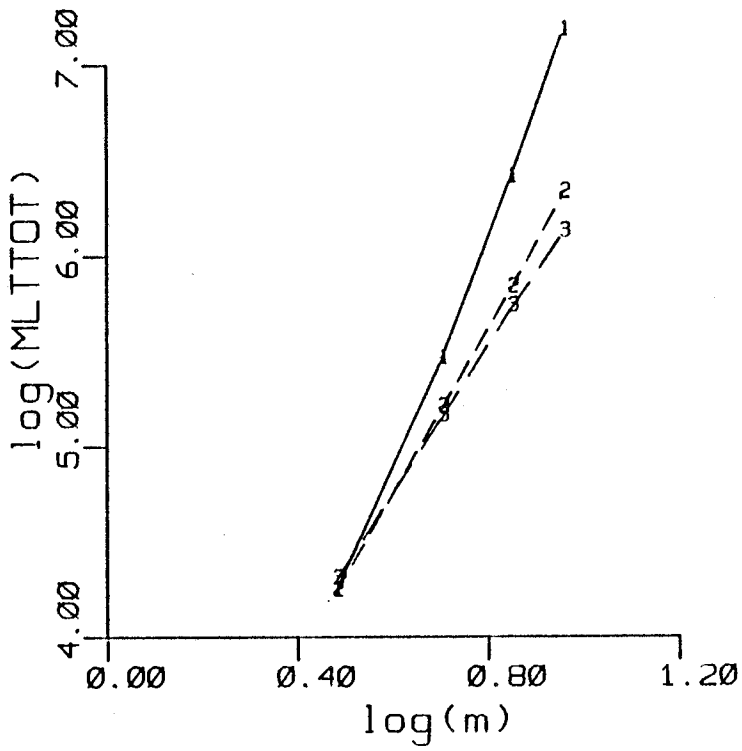
Plot 5.2.2.1: Graphs of m against ITSMAX for LSODCG.V2 with POR(1) preconditioned by (1) NOPRE, (2) TCSSOR, and (3) TCDKR for the spatially-discretized 2-D Heat Problem.



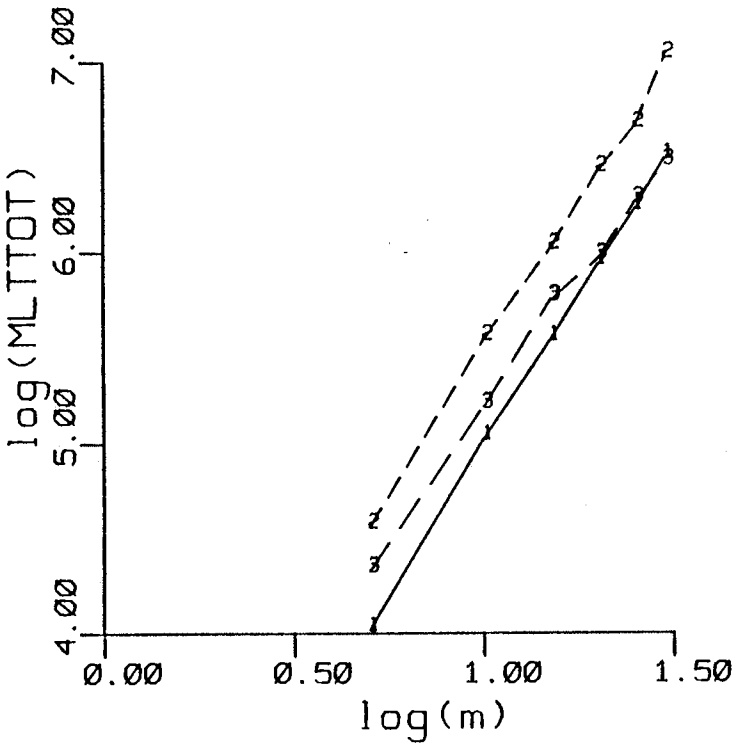
Plot 5.2.2.2: Graphs of m against ITSMAX for LSODCG.V2 with FOR(1) preconditioned by (1) NOPRE, (2) TCSSOR, and (3) TCDKR for the spatially-discretized 2-D Convection-Diffusion Problem.



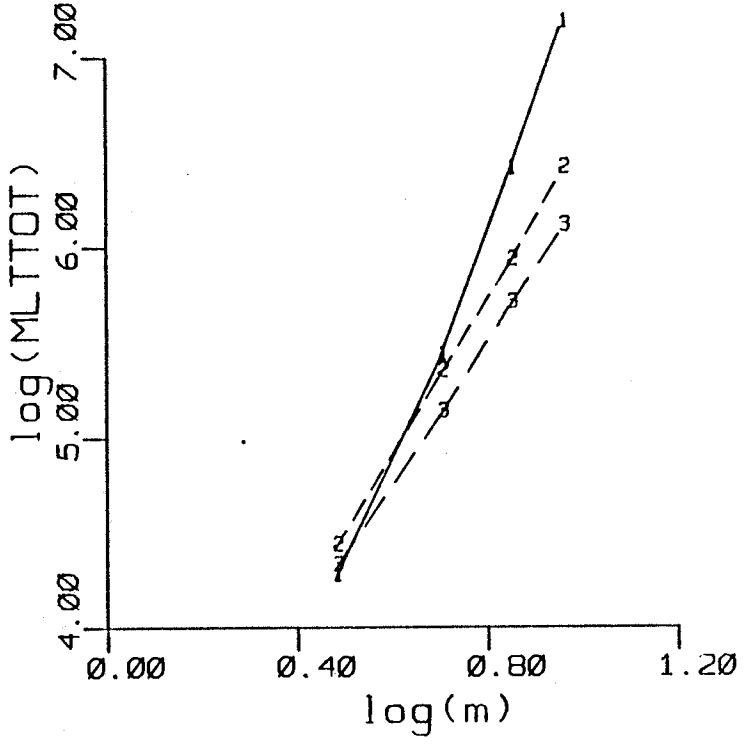
Plot 5.2.2.3: Graphs of m against MLTTOT for (1) LSODES and LSODCG.V2 with POR(1) preconditioned by (2) NOPRE and (3) TCDKR for the spatially-discretized 2-D Heat Problem.



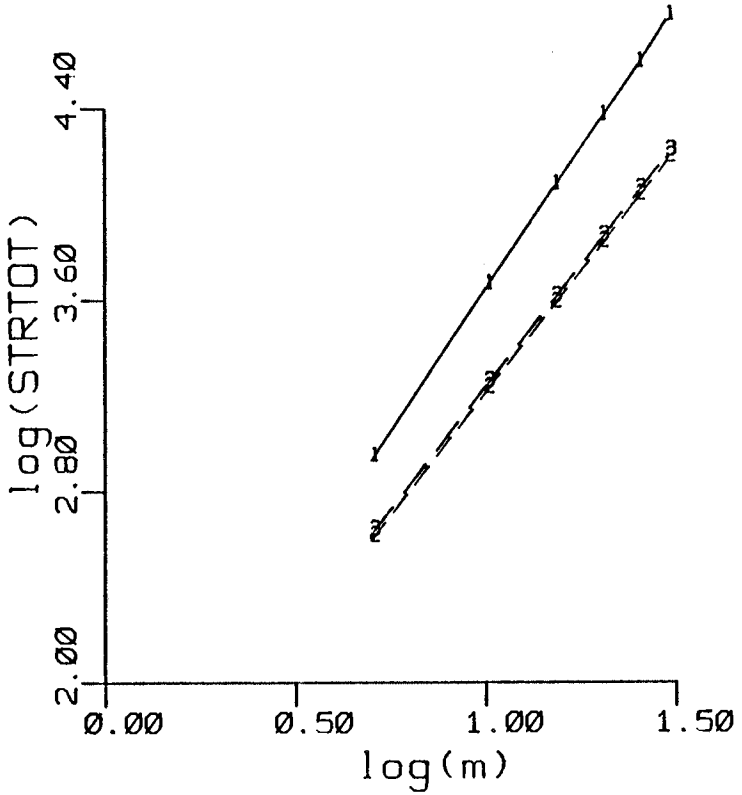
Plot 5.2.2.4: Graphs of m against MLTTOT for (1) LSODES and LSODCG.V2 with POR(1) preconditioned by (2) NOPRE and (3) TCDKR for the spatially-discretized 3-D Heat Problem.



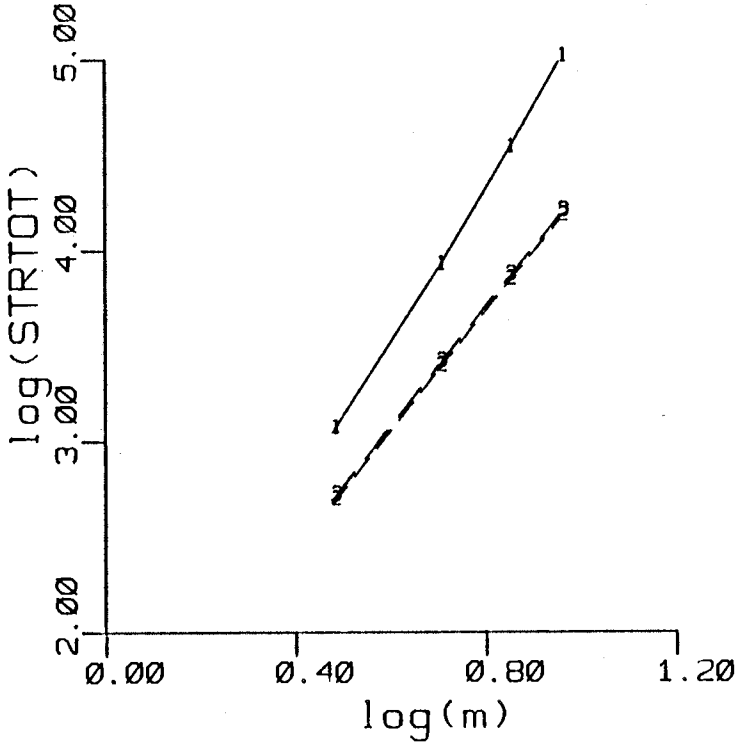
Plot 5.2.2.5: Graphs of m against MLTTOT for (1) LSODES and LSODCG.V2 with POR(1) preconditioned by (2) NOPRE and (3) TCDKR for the spatially-discretized 2-D Convection-Diffusion Problem.



Plot 5.2.2.6: Graphs of m against MLTTOT for (1) LSODES and LSODCG.V2 with POR(1) preconditioned by (2) NOPRE and (3) TCDKR for the spatially-discretized 3-D Convection-Diffusion Problem.



Plot 5.2.2.7: Graphs of m against STRTOT for (1) LSODES and LSODCG.V2 with POR(1) preconditioned by (2) NOPRE and (3) TCDKR (or TCSSOR) for the two spatially-discretized 2-D problems.



Plot 5.2.2.8: Graphs of m against STRTOT for (1) LSODES and LSODCG.V2 with POR(1) preconditioned by (2) NOPRE and (3) TCDKR (or TCSSOR) for the two spatially-discretized 3-D problems.

larger than for NOPRE for the same basic PCGPACK method, since the additional work required to precondition is not offset by a sufficient reduction in the number of PCGPACK iterations used throughout the integration. This is not the case for TCDKR. Although NOPRE required fewer multiplies than TCDKR on some coarse grid problems, the difference is never significant. On the other hand, TCDKR is frequently substantially more effective than NOPRE in terms of both multiplies and iterations required by PCGPACK. Although ITSMAX for TCDKR and TCSSOR are frequently close, ITSTOT for TCDKR is usually significantly less than for TCSSOR, indicating that TCDKR is substantially more effective than TCSSOR on the large number of linear algebraic systems for which h_n is small and the spectrum of $I - h_n \beta_n J$ is clustered around 1.

From the graphs of ITSMAX in Plots 5.2.2.1 and 5.2.2.2, it can be seen that not only do the preconditioned POR(1) methods require fewer PCGPACK iterations than POR(1) with no preconditioning but also the difference grows exponentially with m. Although not shown here, graphs for ITSTOT are similar, but, in this case, TCDKR can be seen to be substantially more effective than TCSSOR. Graphs of MLTTOT for POR(1) preconditioned by NOPRE and TCDKR are shown in Plots 5.2.2.3 to 5.2.2.6. Note that not only is MLTTOT significantly smaller for TCDKR than NOPRE, except on the coarsest grids, but also the difference between MLTTOT for these two preconditionings grows exponentially with m.

Now compare LSODES to LSODCG.V2 with POR(1) preconditioned by TCDKR. Tables 5.2.2.5 and 5.2.2.6 and Plots 5.2.2.3 to 5.2.2.6 reveal that LSODES requires fewer multiplies than LSODCG.V2 for the 2-D problems, except on the finest grid (m=30). However, the difference decreases with m and an extrapolation of the graphs in Plots 5.2.2.3 and 5.2.2.5 suggests that LSODCG.V2 with POR(1) preconditioned by TCDKR would become increasingly more efficient than LSODES for these two 2-D problems on finer grids. For the two 3-D problems with m=9, LSODES requires more than ten times as many multiplies as LSODCG.V2 to solve the linear algebraic equations throughout the integration. Moreover, from Plots 5.2.2.4 and 5.2.2.6, it is clear that this factor grows exponentially with m.

This supports and extends our earlier observation based on theoretical estimates of the computational-work in §4 that iterative methods are significantly more efficient than direct methods for solving the spatially-discretized 3-D Heat Problem.

Table 5.2.2.7 and Plots 5.2.2.7 and 5.2.2.8 show that, for both the 2-D and 3-D problems, STRTOT is significantly larger for LSODES than LSODCG.V2: for the 2-D problems with m=30, LSODES requires approximately 3.7 times as much storage as LSODCG.V2 and, for the 3-D problems with m=9, LSODES requires approximately 6.4 times as much storage as LSODCG.V2. Moreover, for both the 2-D and 3-D problems, this factor grows exponentially with m.

5.3. Stiff Detest Problems.

We used LSODE, LSODES, LSODCG.V1, and LSODCG.V2 on an IBM 3033 computer in double precision to solve the 30 Stiff Detest Problems [27, 29]. Although these problems are not large, they do test the robustness of the inexact chord-Newton method and the associated iterative linear-equation solvers in the two variants of LSODCG.

For each of the four codes, we solved the Stiff Detest Problems using the BDFs with exact Jacobians (MF=21) to an absolute local error tolerance of $ATOL = 10^{-2}, 10^{-4}, 10^{-6}$, and 10^{-8} (RTOL=0 and ITOL=1).

For LSODCG.V1 and LSODCG.V2, we used POR(5), the PCGPACK [21, 25] implementation of Orthomin(5) [20, 24], to solve the linear algebraic systems of equations that arise in the inexact chord-Newton method. We did not precondition POR(5) because, for many of the Stiff Detest Problems, an incomplete factorization would actually yield the exact factorization of the associated Newton iteration matrix and, consequently, POR(5) would generate the exact solution to the linear algebraic equations in one iteration.

We used a stopping criterion of the form (2.3.3) with r = .1, .25, and .5 for the solution of the linear algebraic systems arising in the inexact chord-Newton method. Since the Stiff Detest Problems are small and the tolerance for the linear algebraic systems is lax, we allowed

a maximum of 50 POR(5) iterations to solve each linear algebraic system.

We present our results for LSODE and LSODCG.V2 only. As in the previous section, the results for LSODCG.V2 are generally better than those for LSODCG.V1, and the strategies used in LSODCG.V2 are closer to those in LSODE than those in LSODES.

In Tables 5.3.1 and 5.3.2, respectively, we present the "normalized" number of function evaluations and Jacobian evaluations required by LSODE and LSODCG.V2 with r = .1, .25, and .5 to solve each of the 30 Stiff Detest Problems to an absolute global error tolerance of $Tol = 10^{-2}, 10^{-4}$, and 10^{-6} at the end-point of the integration; in Table 5.3.3, we present the "normalized" total number of POR(5) iterations required by LSODCG.V2 throughout the integration. These normalized statistics were calculated by a new version of the Stiff Detest Program which, as described in [26], first performs a least squares fit to

$$\sum_{i=1}^{NTOL} [\log(global\ error_i) - \log(C) - E \cdot \log(ATOL_i)]^2$$

for C and E, where, in this case, $ATOL_i = 10^{-2}, 10^{-4}, 10^{-6}$, and 10^{-8} and NTOL = 4. The Stiff Detest Program then performs a piecewise linear interpolation on the actual recorded values of the costs to solve the IVP at $ATOL_i$ versus the corresponding expected global accuracy $Tol = C \cdot ATOL^E$ to arrive at the normalized costs for an absolute global error tolerance of Tol. (A consequence of this procedure is that the normalized function and Jacobian evaluations are negative for one problem.)

A "-", "*", or "x" may occur as an entry in place of a number in these tables. A "-" indicates that Stiff Detest could not calculate the normalized statistics for this problem and tolerance based upon the actual global errors incurred. A "*" indicates that the method being tested (LSODE or LSODCG.V2) could not solve the problem at that tolerance, and a "x" indicates that Stiff Detest could not solve the problem at that tolerance. In addition, the 12 problems marked with a "#" have a Jacobian that is not negative-real over some subinterval of the range of integration.

From the tables, we see that the number of function and Jacobian evaluations typically

Problem	Tol = 1.0D-2				Tol = 1.0D-4				Tol = 1.0D-6			
	LSODE	LSODCG r=.1	LSODCG r=.25	LSODCG r=.5	LSODE	LSODCG r=.1	LSODCG r=.25	LSODCG r=.5	LSODE	LSODCG r=.1	LSODCG r=.25	LSODCG r=.5
A1	53	44	59	71	115	112	129	141	233	218	231	289
A2 #	64	*	*	*	150	148	145	*	307	276	290	-
A3	86	98	94	108	194	207	206	243	399	372	401	464
A4	90	77	89	106	192	182	210	234	374	356	392	453
B1 #	544	651	1417	1691	1017	2151	2533	2747	-	-	-	-
B2	49	49	53	59	106	104	121	133	191	201	219	250
B3	53	49	62	65	121	117	128	156	228	225	267	304
B4	84	80	82	99	190	173	201	218	582	608	441	495
B5	24	1933	327	314	2878	1802	584	666	3502	3439	1430	1489
C1	88	60	64	69	189	141	157	168	271	271	284	418
C2	50	52	61	83	126	139	144	178	265	247	267	410
C3	48	47	50	71	132	124	143	155	254	242	298	378
C4 #	210	513	623	777	362	813	1061	1343	635	-	1524	-
C5 #	361	1270	1368	1541	639	1564	1769	2552	-	-	-	-
D1	97	98	110	110	218	197	218	237	481	375	380	447
D2	85	93	89	94	222	177	186	204	444	341	347	356
D3 #	77	*x	*x	*x	180	160	176	194	348	317	350	378
D4 #	9	22	14	14	17	30	20	21	41	48	39	50
D5	57	60	67	61	120	125	123	138	241	247	250	283
D6	18	25	29	29	46	56	54	60	111	106	98	102
E1 #	-12	138	49	133	41	120	181	184	112	156	217	229
E2	325	335	352	404	575	595	660	704	-	-	-	-
E3	84	101	92	99	244	204	196	205	499	342	365	352
E4 #	248	*x	*x	*x	508	433	530	660	977	890	-	-
E5 #	-	x	x	*x	14	x	x	*x	29	30	33	35
F1 #	305	364	393	449	611	702	730	1050	1156	1347	1289	2091
F2 #	24	35	32	44	66	83	80	76	145	159	161	156
F3	x	x	x	x	x	x	x	x	21	34	36	37
F4	-	-	-	-	-	-	-	-	x	x	x	x
F5 #	x	*	*	*	72	54	69	83	113	117	119	135

Table 5.3.1: Normalized Function Evaluations for the Stiff Detest Problems.

Problem	Tol = 1.0D-2				Tol = 1.0D-4				Tol = 1.0D-6			
	LSODE	LSODCG r=.1	LSODCG r=.25	LSODCG r=.5	LSODE	LSODCG r=.1	LSODCG r=.25	LSODCG r=.5	LSODE	LSODCG r=.1	LSODCG r=.25	LSODCG r=.5
A1	13	2	2	3	17	5	6	6	25	9	11	13
A2 #	14	*	*	*	22	7	6	*	32	12	13	-
A3	16	4	4	4	25	8	9	10	40	17	17	20
A4	20	3	4	5	28	8	9	10	37	16	18	20
B1 #	46	65	159	187	72	108	178	208	-	-	-	-
B2	10	2	2	2	15	4	5	6	18	9	9	11
B3	11	2	2	3	16	5	5	7	21	10	11	13
B4	12	3	4	4	19	7	8	9	36	27	20	22
B5	136	87	14	12	146	83	27	27	180	156	65	66
C1	13	3	3	3	19	7	7	7	26	12	13	18
C2	12	3	3	3	20	6	7	8	30	11	12	18
C3	11	3	3	3	21	6	6	7	24	11	13	16
C4 #	26	21	41	74	35	35	80	84	43	-	87	-
C5 #	36	124	153	178	56	133	152	155	-	-	-	-
D1	12	4	4	3	29	7	8	9	56	15	15	18
D2	13	6	5	5	29	7	7	7	46	14	15	14
D3 #	18	*x	*x	*x	28	6	8	8	40	12	15	17
D4 #	5	1	1	1	5	3	1	1	10	2	2	2
D5	14	2	3	4	19	5	5	6	30	10	10	10
D6	8	2	2	2	8	2	3	3	16	4	3	4
E1 #	-1	15	10	20	12	8	25	26	15	9	19	28
E2	28	12	13	15	38	25	27	29	-	-	-	-
E3	13	3	3	3	30	8	8	8	55	14	16	15
E4 #	33	*x	*x	x	52	19	23	27	72	40	-	-
E5 #	-	x	x	*x	5	x	x	*x	7	2	2	2
F1 #	45	19	21	24	68	29	32	44	115	54	52	84
F2 #	7	2	2	2	10	3	3	3	17	6	6	6
F3	x	x	x	x	x	x	x	x	10	2	2	2
F4	-	-	-	-	-	-	-	-	x	x	x	x
F5 #	x	*	*	*	18	4	4	4	22	5	5	6

Table 5.3.2: Normalized Jacobian Evaluations for the Stiff Detest Problems.

Problem	Tol = 1.0D-2			Tol = 1.0D-4			Tol = 1.0D-6		
	LSODCG r=.1	LSODCG r=.25	LSODCG r=.5	LSODCG r=.1	LSODCG r=.25	LSODCG r=.5	LSODCG r=.1	LSODCG r=.25	LSODCG r=.5
A1	70	60	56	111	103	103	175	160	178
A2 #	*	*	*	322	307	*	511	491	-
A3	139	117	122	148	216	222	398	382	366
A4	307	318	322	607	609	632	1019	1043	1104
B1 #	1196	1521	1588	2214	2340	2281	-	-	-
B2	60	57	50	101	79	73	133	107	101
B3	82	74	67	134	107	98	184	161	137
B4	133	99	87	217	151	145	723	291	223
B5	2954	316	181	2309	318	271	3817	577	500
C1	100	85	70	172	145	139	257	231	240
C2	90	83	103	173	143	148	248	234	237
C3	84	78	81	162	147	154	257	252	266
C4 #	569	602	737	747	892	868	-	1025	-
C5 #	1199	1194	1197	1346	1266	1044	-	-	-
D1	142	147	139	243	240	239	386	355	368
D2	88	83	69	131	126	135	232	208	217
D3 #	*x	*x	*x	97	90	84	123	123	125
D4 #	29	11	11	37	17	17	41	33	40
D5	54	49	42	90	75	75	161	143	144
D6	34	37	29	66	56	62	113	101	101
E1 #	179	77	139	108	194	173	110	143	181
E2	111	64	41	90	35	22	-	-	-
E3	99	80	76	166	142	135	253	247	227
E4 #	*x	*x	x	589	544	595	1009	-	-
E5 #	x	x	*x	x	x	*x	47	42	39
F1 #	875	905	949	1658	1633	2050	2990	2761	3852
F2 #	23	18	18	39	29	27	66	48	44
F3	x	x	x	x	x	x	83	85	83
F4	-	-	-	-	-	-	x	x	x
F5 #	*	*	*	92	114	119	153	148	158

Table 5.3.3: Normalized PCGPACK iterations for the Stiff Detest Problems.

increases with r. For the Jacobian evaluations, the increase is generally not significant, but, for the function evaluations, the increase is frequently 10% or more from one value of r to the next. On the other hand, the number of POR(5) iterations typically decreases with r by a factor of 10% or more from one value of r to the next. Hence, if a POR(5) iteration is less expensive than a function evaluation, then, based upon these results, r=.1 would usually be the most cost effective of the three values considered. On the other hand, if a POR(5) iteration is substantially more expensive than a function evaluation (as is the case for the problems in the previous subsection), then, based upon these results, r=.5 would usually be the most cost effective of the three values considered. Thus, the choice of r is dependent upon the class of IVPs solved.

Except for problem C5, which has a Jacobian that is not negative-real, LSODCG.V2, with each of the values of r considered, used fewer Jacobian evaluations than LSODE on all

problems that were solved successfully by both codes. Moreover, for those IVPs having a negative-real Jacobian, the number of Jacobian evaluations required differs by a factor of 2 to 5. This superiority of LSODCG.V2 over LSODE is a result of the strategy used in LSODCG.V2 described above that permits it to update the scalar factor $h_n \beta_n$ in the Newton iteration matrix $I - h_n \beta_n J$ whenever $h_n \beta_n$ changes without re-evaluating the Jacobian, J. If we had also removed from LSODCG.V2 the requirement inherited from LSODE that the Jacobian be re-evaluated at least once every MSBP (=20) steps, then LSODCG.V2 would have used even fewer Jacobian evaluations.

Now consider the function evaluations required by LSODE and LSODCG.V2 with r=.1 to solve the Stiff Detest Problems.

LSODCG.V2 failed to solve 4 of the Stiff Detest Problems (A2, D3, E4, F5) at $Tol = 10^{-2}$. Each of these problems has a Jacobian that is not negative-real over some subinterval of the range of integration. However, except for problem F5 at $Tol = 10^{-6}$, LSODCG.V2 required fewer function evaluations than LSODE for these problems at $Tol = 10^{-4}$ and 10^{-6}.

Of the remaining problems, LSODCG.V2 with r=.1 used substantially fewer function evaluations than LSODE for 7 of the Stiff Detest Problems (A1, A4, C1, C3, D1, D2, E3). Again, this may be due to LSODCG.V2's updating the scalar factor $h_n \beta_n$ in the Newton iteration matrix whenever $h_n \beta_n$ changes resulting in a more accurate Newton iteration matrix and a more rapid convergence of the Newton iteration.

LSODE and LSODCG.V2 used approximately the same number of function evaluations on 11 of the Stiff Detest Problems (A3, B2, B3, B4, B5, C2, D5, D6, F2, F3, F5). It is worth noting that the class B problems are of the form $y' = A\,y$, where A is a constant matrix with complex eigenvalues and, consequently, A is far from being symmetric.

LSODE used substantially fewer function evaluations than LSODCG.V2 on 7 problems (B1, C4, C5, D4, E1, F1, F2) each of which has a Jacobian that is not negative-real over some subinterval of the range of integration.

Therefore, except for those problems having a Jacobian that is not negative-real, the use of an iterative linear-equation solver did not cause the performance of LSODCG.V2 to deteriorate relative to the unmodified code LSODE which incorporates a direct linear-equation solver. In fact, LSODCG.V2 performs as well as or better than LSODE on all the Stiff Detest Problems for which LSODCG.V2 is applicable.

One final point is worth noting. From Tables 5.3.1 and 5.3.3, we see that, for many of the Stiff Detest Problems, particularly at the more stringent tolerances, an average of less than one PCGPACK iteration is required per inexact chord-Newton iteration. That is, for many of the inexact chord-Newton iterations, the initial guess $-F(y_n^k)$ for $y_n^{k+1}-y_n^k$ satisfies (2.3.3) and no further PCGPACK iterations are required. Hence, when using an iterative linear-equation solver in a stiff-ODE code in this way, we automatically obtain the benefit of the use of an inexpensive predictor-corrector iteration when a more expensive Newton iteration is not required. Moreover, this appears to have no deleterious effect upon the overall performance of the stiff-ODE solver.

6. Conclusions.

Both the theoretical and numerical results presented in the preceding two sections show that the use of iterative linear-equation solvers in stiff-ODE codes has the potential to improve the efficiency - in terms of both computational-work and storage - with which a significant class of stiff IVPs having large sparse Jacobians can be solved. Moreover, these results demonstrate the importance of preconditioning for Krylov subspace methods used in stiff-ODE solvers.

The numerical results for both the linear and nonlinear IVPs show that the stopping criterion (2.3.3) for the inexact chord-Newton iteration works well in practice for $r \in [.1,.5]$. This supports the claim that the linear equations that arise in stiff-ODE solvers need not be solved very accurately. Moreover, the initial guess $-F(y_n^k)$ for the solution $y_n^{k+1}-y_n^k$ of the linear system proved to be quite effective in practice, particularly during the initial transient where the IVP is at most mildly stiff.

Updating the scalar factor $h_n \beta_n$ in the Newton iteration matrix $I - h_n \beta_n J$ whenever $h_n \beta_n$ changes without re-evaluating J, the approximation to the Jacobian, reduces the number of Newton iterations and associated function evaluations required throughout the course of the numerical integration with little added cost in a stiff-ODE code incorporating an iterative linear-equation solver. Furthermore, this strategy of updating $h_n \beta_n$ whenever it changes facilitates the decision when to re-evaluate the Jacobian and, thus, helps to avoid wasted computational-work. More generally, as mentioned in §2.2, the removal of the constraint imposed by the necessity to avoid refactoring $I - h_n \beta_n J$ in a stiff-ODE code employing a direct linear-equation solver may lead to other benefits in the choice of formulas, strategies, and heuristics for a stiff-ODE code incorporating an iterative linear-equation solver.

Most importantly, the numerical results demonstrate that stiff-ODE codes incorporating iterative linear-equation solvers do not suffer a loss of robustness on those IVPs for which the Newton iteration matrix W_n^k is positive-real throughout the course of the numerical integration. Note, though, that this restriction on W_n^k is imposed by the iterative technique we chose to solve the linear systems: the restriction is not characteristic of all stiff-ODE codes incorporating iterative linear-equation solvers. In particular, as mentioned earlier, there exist iterative linear-equation solvers that are guaranteed to converge to the solution of the Newton system (2.2.1) if all the eigenvalues of W_n^k lie in the right-half complex plane. As we argued in §2.1, if W_n^k does not satisfy this last restriction, then the stepsize is almost surely too large and should be reduced until this last restriction is satisfied to ensure a reliable numerical integration.

Hence, it appears possible to develop a stiff-ODE code incorporating an iterative linear-equation solver that, for a broad class of IVPs, is as robust as a similar stiff-ODE code incorporating a direct linear-equation solver, but more efficient than the latter code for a significant subclass of problems having large sparse Jacobians. We plan to continue to pursue this investigate in the future.

References

[1] O. Axelsson, *Solution of linear systems of equations: iterative methods*, in Sparse Matrix Techniques, V.A. Barker (ed), Lecture Notes in Mathematics, v. 572, Springer-Verlag, New York, NY, 1977.

[2] J.H. Bramble, *Multistep methods for quasilinear parabolic equations*, in Computational Methods in Nonlinear Mechanics, J.T. Oden (ed), North-Holland, Amsterdam, Netherlands, 1980, pp. 177-183.

[3] P.N. Brown and A.C. Hindmarsh, *Matrix-Free Methods in the solution of stiff systems of ODEs*, Technical Report UCID-19937, Lawrence Livermore Laboratory, University of California, Livermore, CA, 1972.

[4] J.C. Butcher, *Implicit Runge-Kutta processes*, Math. Comp., 18 (1964), pp. 50-64.

[5] G.D. Byrne and A.C. Hindmarsh, *A polyalgorithm for the numerical solution of ordinary differential equations*, ACM TOMS, 1 (1975), pp. 71-96.

[6] G.D. Byrne, A.C. Hindmarsh, K.R. Jackson, and H.G. Brown, *Comparative test results for two ODE solvers - EPISODE and GEAR*, Technical Report ANL-77-19, Argonne National Laboratory, Argonne, IL, 1977.

[7] G.D. Byrne, A.C. Hindmarsh, K.R. Jackson, and H.G. Brown, *A comparison of two ODE codes: GEAR and EPISODE*, Computers and Chemical Eng., 1 (1977), pp. 133-147.

[8] T.F. Chan and K.R. Jackson, *Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms*, Technical Report 259, Department of Computer Science, Yale University, New Haven, CT, 1983; accepted for publication by SIAM J. Sci. Stat. Comput.

[9] T.F. Chan, K.R. Jackson, and B. Zhu, *Alternating-direction incomplete factorizations*, SIAM J. Numer. Anal., 20 (1983), pp. 239-257.

[10] R. Chandra, *Conjugate gradient methods for partial differential equations*, Ph.D. Thesis, Technical Report 129, Department of Computer Science, Yale University, New Haven, CT, 1978.

[11] R. Chandra, S.C. Eisenstat, and M.H. Schultz, *The modified conjugate residual method for partial differential equations*, Technical Report 107, Department of Computer Science, Yale University, New Haven, CT, 1977.

[12] P. Concus and G.H. Golub, *A generalized conjugate gradient method for nonsymmetric systems of linear equations*, Proceedings of the Second International Symposium on Computing Methods in Applied Sciences and Engineering, R. Glowinski and J.L. Lyons (eds), Notes in Economics and Mathematical Systems, v. 134, Springer-Verlag, New York, NY, 1976.

[13] P. Concus, G.H. Golub, and D.P. O'Leary, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J.R. Bunch and D.J. Rose (eds), Academic Press, New York, 1976, pp. 309-332.

[14] P. Concus, G.H. Golub, and D.P. O'Leary, *Numerical solution of nonlinear elliptic partial differential equations by a generalized conjugate gradient method*, Technical Report STAN-CS-76-585, Department of Computer Science, Stanford University, Stanford, CA, 1976.

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

[16] J. Douglas Jr. and T. Dupont, *Preconditioned conjugate gradient iteration applied to Galerkin methods for a mildly nonlinear Dirichlet problem*, in Sparse Matrix Computations, J.R. Bunch and D.J. Rose (eds), Academic Press, New York, 1976, pp. 333-348.

[17] J. Douglas Jr., T. Dupont, and R.E. Ewing, *Incomplete iteration for time-stepping a Galerkin method for a quasilinear parabolic problem*, SIAM J. Numer. Anal., 16 (1979), pp. 503-522.

[18] T.Dupont, R.P. Kendall, and H.H. Rachford, *An approximate factorization procedure for solving self-adjoint elliptic difference equations*, SIAM J. Numer. Anal., 6 (1968), pp. 753-782.

[19] S.C. Eisenstat, *Efficient implementation of a class of preconditioned conjugate gradient methods*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 1-4.

[20] S.C. Eisenstat, H.C. Elman, and M.H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 345-357.

[21] S.C. Eisenstat, H.C. Elman, M.H. Schultz, and A.H. Sherman, *The (new) Yale sparse matrix package*, Technical Report YALEU/DCS/RR-265, Department of Computer Science, Yale University, New Haven, CT, 1977.

[22] S.C. Eisenstat, M.C. Gursky, M.H. Schultz, and A.H. Sherman, *Yale sparse matrix package I: The symmetric codes*, Technical Report 112, Department of Computer Science, Yale University, New Haven, CT, 1977.

[23] S.C. Eisenstat, M.C. Gursky, M.H. Schultz, and A.H. Sherman, *Yale sparse matrix package II: The nonsymmetric codes*, Technical Report 114, Department of Computer Science, Yale University, New Haven, CT, 1977.

[24] H.C. Elman, *Iterative methods for large, sparse, nonsymmetric systems of linear equations*, Ph.D. Thesis, Technical Report 229, Department of Computer Science, New Haven, CT, 1982.

[25] H.C. Elman, *PCGPACK, a collection of preconditioned Krylov subspace methods for the solution of systems of linear algebraic equations*, private communication.

[26] W.H. Enright, *Using a testing package for the automatic assessment of numerical methods for ODEs*, in Performance Evaluation of Numerical Software, L. Fosdick (ed), North-Holland, Amsterdam, Netherlands, 1979, pp. 199-213.

[27] W.H. Enright and T.E. Hull, *Comparing numerical methods for the solution of stiff systems of ODEs arising in chemistry*, in Methods for Differential Systems, L. Lapidus and W.E. Schiesser (eds), Academic Press, New York, 1976, pp. 45-65.

[28] W.H. Enright and T.E. Hull, *Test results in initial value methods for non-stiff ordinary differential equations*, SIAM J. Numer. Anal., 13 (1976), pp. 944-961.

[29] W.H. Enright, T.E. Hull, and B. Lindberg, *Comparing numerical methods for stiff systems of ODEs*, BIT, 15 (1975), pp. 10-48.

[30] W.H. Enright and M.S. Kamel, *Automatic partitioning of stiff systems and exploiting the resulting structure*, ACM TOMS, 5 (1979), pp. 374-385.

[31] B.S. Garbow, J.M. Boyle, J.J. Dongarra, and C.B. Moler, *Matrix Eigensystem Routines - EISPACK Guide Extension*, Lecture Notes in Computer Science, v. 51, Springer-Verlag, New York, NY, 1977.

[32] N.K. Garg and R.A. Tapia, *QDN: a variable storage algorithm for unconstrained optimization*, Technical Report, Department of Mathematical Sciences, Rice University, Houston,

TX, 1980.

[33] C.W. Gear, *Algorithm 407, DIFSUB for the solution of ordinary differential equations*, CACM, 14 (1971), pp. 185-190.

[34] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[35] C.W. Gear, *Numerical solution of ordinary differential equations: Is there anything left to do?* SIAM Review, 23 (1981), pp. 10-24.

[36] C.W. Gear, *Stiff Software: What do we have and what do we need?* Technical Report UIUCDCS-R-82-1109, Department of Computer Science, University of Illinois, Urbana, IL, 1982.

[37] C.W. Gear and Y. Saad, *Iterative solution of linear equations in ODE codes*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 583-601.

[38] C.W. Gear, K.W. Tu, and D.S. Watanabe, *The stability of automatic programs for numerical problems*, in Differential Systems, R.A. Willoughby (ed), Plenum Press, New York, 1974, pp. 111-121.

[39] W. Givens, *Fields of values of a matrix*, Proc. American Math. Soc., 3 (1952), pp. 206-209.

[40] G.H. Golub and R.S. Varga, *Chebyshev semi-iterative methods, successive over relaxation iterative methods, and second order Richardson iterative methods*, Numer. Math., 3 (1961), pp. 147-168.

[41] I. Gustafsson, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142-156.

[42] L.A. Hageman and D.M. Young, *Applied Iterative Methods*, Academic Press, New York, NY, 1981.

[43] M.R. Hestenes and E. Stiefel, *Methods of conjugate gradient for solving linear systems*, Journal of Research of the National Bureau of Standards, 49 (1952), pp. 409-436.

[44] A.C. Hindmarsh, *GEAR: ordinary differential equation system solver*, Technical Report UCID-30001, Rev. 2, Lawrence Livermore Laboratory, University of California, Livermore, CA, 1972.

[45] A.C. Hindmarsh, *GEARB: solution of ordinary differential equations having banded Jacobian*, Technical Report UCID-30059, Lawrence Livermore Laboratory, University of California, Livermore, CA, 1973.

[46] A.C. Hindmarsh, *Preliminary report of GEARIB: solution of implicit systems of ordinary differential equations with banded Jacobian*, Technical Report UCID-30130, Lawrence Livermore Laboratory, University of California, Livermore, CA, 1976.

[47] A.C. Hindmarsh, *Preliminary documentation of GEARBI: solution of ODE systems with block iterative treatment of the Jacobian*, Technical Report UCID-30149, Lawrence Livermore Laboratory, University of California, Livermore, CA, 1976.

[48] A.C. Hindmarsh, *LSODE and LSODI, two new initial value ordinary differential equation solvers*, ACM SIGNUM Newsletter, 15 (1980), pp. 10-11.

[49] A.C. Hindmarsh and A.H. Sherman, *LSODES*, private communication, 1982.

[50] N. Houbak, S.P Norsett, and P.G. Thomsen, *Displacement or residual test in the application of implicit methods for stiff problems*, manuscript.

[51] P.J. van der Houwen and B.P. Sommeijer, *Predictor-corrector methods with improved absolute stability regions*, Technical Report, Mathematisch Centrum, Amsterdam, Netherlands, 1982.

[52] K.R. Jackson, *Variable stepsize, variable order integrand approximation methods for the numerical solution of ordinary differential equations*, Ph.D. Thesis, Technical Report 129, Department of Computer Science, University of Toronto, Toronto, Canada, 1978.

[53] K.R. Jackson and R. Sacks-Davis, *An alternative implementation of variable stepsize multistep formulas for stiff ODEs*, ACM TOMS, 6 (1980), pp. 295-318.

[54] K.C. Jea, *Generalized conjugate gradient acceleration of iterative methods*, Ph.D. Thesis, Technical Report CNA-176, Center for Numerical Analysis, University of Texas, Austin, TX, 1982.

[55] D.S. Kershaw, *The incomplete Cholesky - conjugate gradient method for the iterative solution of systems of linear equations*, J. of Comput. Physics, 26 (1978), pp. 43-65.

[56] D.S. Kershaw, *On the problem of unstable pivots in the incomplete LU-conjugate gradient method*, J. Comput. Physics, 31 (1980), pp. 114-123.

[57] D.R. Kincaid, J.R. Respess, D.M. Young, and R.G. Grimes, *Algorithm 586. ITPACK 2C: a FORTRAN package for solving large sparse linear systems by adaptive accelerated iterative methods*, ACM TOMS, 8 (1982), pp. 302-322.

[58] F.T. Krogh and K. Stewart, *Asymptotic ($h \to \infty$) absolute stability for BDF's applied to stiff differential equations*, Computing Memorandum 478, Jet Propulsion Laboratory, California Institute of Technology, Pasedena, CA, (revised) 1983.

[59] J.D. Lambert, *Computational Methods in Ordinary Differential Equations*, John Wiley and Sons, New York, 1976.

[60] T.A. Manteuffel, *The Tchebychev iteration for non-symmetric linear systems*, Numer. Math, 28 (1977), pp. 307-327.

[61] T.A. Manteuffel, *Adaptive procedure for estimating parameters for the nonsymmetric Tchebychev iteration*, Numer. Math., 31 (1978), pp. 183-208.

[62] T.A. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 473-497.

[63] J.A. Meijerink and H.A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148-162.

[64] W.L. Miranker and I-L. Chern, *Dichotomy and conjugate gradients in the stiff initial value problem*, Technical Report RC 8032, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1980.

[65] A. Nordsieck, *On the numerical integration of ordinary differential equations*, Math. Comp., 16 (1962), pp. 22-49.

[66] D.P. O'Leary, *A discrete Newton algorithm for minimizing a function of many variables*, Math. Progr., 23 (1982), pp. 20-33.

[67] J.M. Ortega and W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, NY, 1960.

[68] J.K. Reid, *On the method of conjugate gradients for the solution of large sparse systems of linear equations*, in Large Sparse Sets of Linear Equations, J.K. Reid (ed), Academic Press, New York, NY, 1971, pp. 231-254.

[69] Y. Saad, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comp., 37 (1981), pp. 105-126.

[70] Y. Saad, *The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems*, SIAM J. Numer. Anal., 19 (1982), pp. 485-506.

[71] M.A. Saunders, *Sparse least squares by conjugate gradients: a comparison of preconditioning methods*, Technical Report SOL 79-5 Systems Optimization Laboratory, Stanford University, Stanford, CA, 1979.

[72] H.R. Schwarz, *The method of conjugate gradients in finite element applications*, J. of Applied Math. and Phys., 30 (1979), pp. 342-353.

[73] L.F. Shampine, *Evaluation of implicit formulas for the solution of ODEs*, Technical Report SAND79-1370, Sandia Laboratories, Albuquerque, NM, 1979.

[74] L.F. Shampine, *Implementation of implicit formulas for the solution of ODEs*, SIAM J. Sci. Stat. Comput., 1 (1980), pp. 103-118.

[75] L.F. Shampine, H.A. Watts, and S.M. Davenport, *Solving non-stiff ordinary differential equations - the state of the art*, SIAM Review, 18 (1975), pp. 376-411.

[76] A.H. Sherman, *On Newton-iterative methods for the solution of systems of nonlinear equations*, SIAM J. Numer. Anal., 15 (1978), pp. 755-771.

[77] A.H. Sherman and A.C. Hindmarsh, *GEARS: a package for the solution of sparse, stiff ordinary differential equations*, Electrical Power Problems: the Mathematical Challenge, A. Erisman, K. Neves, and M.H. Dwarakanath (eds), SIAM Publications, Philadelphia, PA, 1981, pp. 190-200.

[78] B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler, *Matrix Eigensystem Routines - EISPACK Guide*, Lecture Notes in Computer Science, v. 6, Springer-Verlag, New York, NY, 1976.

[79] H.J. Stetter, *Stability of discretizations on infinite intervals*, Conference on Applications of Numerical Analysis, Dundee, Scotland, J.L. Morris (ed), Lecture Notes in Mathematics, v. 228, Springer-Verlag, New York, NY, 1971, pp. 207-222.

[80] R.S. Varga, *Matrix Iterative Analysis*, Prentice-Hall, New York, NY, 1962.

[81] P.K.W. Vinsome, *Orthomin, an iterative method for solving sparse sets of simultaneous linear equations*, Paper SPE 5729, Proceedings of the Fourth Symposium on Reservoir Simulation, Society of Petroleum Engineers of AIME, 1976, pp. 149-159.

[82] D.S. Watkins and R.W. HansonSmith, *The numerical solution of separably stiff systems by precise partitioning*, ACM TOMS, 9 (1983), pp. 293-301.

[83] O. Widlund, *A Lanczos method for a class of nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 15 (1978), pp. 801-812,

[84] J. Williams, *The problem of implicit formulas in numerical methods for stiff differential equations*, Technical Report 40, Department of Mathematics, University of Manchester, Manchester, England, 1979.

[85] D.M. Young, *Iterative Solution of Large Linear Systems*, Academic Press, New York, NY, 1971.

[86] D.M. Young and K.C. Jea, *Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods*, Linear Algebra and Its Appl., 34 (1980), pp. 159-194.