SIAM J. SCI. COMPUT.
Vol. 19, No. 1, pp. 302–318, January 1998

021

NITSOL: A NEWTON ITERATIVE SOLVER FOR NONLINEAR SYSTEMS*

MICHAEL PERNICE† AND HOMER F. WALKER‡

Abstract. We introduce a well-developed Newton iterative (truncated Newton) algorithm for solving large-scale nonlinear systems. The framework is an inexact Newton method globalized by backtracking. Trial steps are obtained using one of several Krylov subspace methods. The algorithm is implemented in a Fortran solver called NITSOL that is robust yet easy to use and provides a number of useful options and features. The structure offers the user great flexibility in addressing problem specificity through preconditioning and other means and allows easy adaptation to parallel environments. Features and capabilities are illustrated in numerical experiments.

Key words. Newton iterative methods, truncated Newton methods, Newton–Krylov methods, inexact Newton methods, Newton's method, forcing terms, iterative linear algebra methods, Krylov subspace methods, GMRES, BiCGSTAB, TFQMR

AMS subject classifications. 65H10, 65F10, 65N22

PII. S1064827596303843

1. Introduction.

Our interest is in methods for solving a system of nonlinear equations

$$F(x) = 0, \qquad F:\mathbb{R}^n \to \mathbb{R}^n, \tag{1.1}$$

where F is assumed to be continuously differentiable everywhere in $\mathbb{R}^n$. Newton's method for solving (1.1) requires, at the kth step, the solution of the linear *Newton equation*

$$F'(x_k)\, s_k = -F(x_k), \tag{1.2}$$

where x_k is the current approximate solution. A *Newton iterative method*, or *truncated Newton method*, is an implementation of Newton's method in which an iterative linear solver is used to determine an approximate solution of (1.2). Newton iterative methods are especially well suited for large-scale problems and have been used very successfully in many scientific and industrial applications.

Our purpose here is to introduce a well-developed Newton iterative algorithm and to describe its implementation in a Fortran solver called NITSOL. In brief, the underlying nonlinear algorithm is an inexact Newton method with a backtracking globalization[1], previously formulated as Algorithm INB of [9, section 6]. In this, the Newton equation (1.2) is relaxed to an *inexact Newton condition* (cf. [8])

$$\|F(x_k) + F'(x_k)\, s_k\| \le \eta_k \|F(x_k)\|, \tag{1.3}$$

*Received by the editors May 17, 1996; accepted for publication (in revised form) March 10, 1997. http://www.siam.org/journals/sisc/19-1/30384.html

†Center for High Performance Computing, University of Utah, Salt Lake City, UT 84112 (usimap@sneffels.usi.utah.edu).

‡Department of Mathematics and Statistics, Utah State University, Logan, UT 84322-3900 (walker@math.usu.edu). Current address: Mathematical Sciences Department, Worcester Polytechnic Institute, Worcester, MA, 01609. The research of this author was supported in part by U.S. Department of Energy grant DE-FG03-94ER25221 and National Science Foundation grant DMS-9400217, both with Utah State University.

[1]We use the term "globalization" in its traditional sense, i.e., to refer to strategies for improving the likelihood of convergence from initial approximations that may not be near a solution.

in which the "forcing term" $\eta_k \in [0, 1)$ can be specified in several ways as in [10] to enhance efficiency and convergence. An initial s_k satisfying (1.3) is determined by using a Krylov subspace method to solve (1.2) approximately; thus, our algorithm is a *Newton–Krylov method*. The Krylov subspace methods implemented in NITSOL are GMRES(m) [25], [33], BiCGSTAB [31], and TFQMR [12]. Once an initial s_k has been determined, it is tested and, if necessary, reduced in length through safeguarded backtracking until an acceptable step is obtained.

With NITSOL, we have hoped to provide a robust, theoretically well-founded solver that is simple and easy to use but which offers enough options and flexibility to allow the user to implement sophisticated solution strategies that address specific problem needs and particular machine capabilities. No preconditioners are provided by NITSOL, but the structure allows the user to implement a preferred preconditioner with minimal difficulty. The user is also allowed to specify or supply an inner product and associated norm that are used throughout the algorithm; this capability not only provides a means of addressing problem scaling but also allows easy adaptation to parallel environments. There are options for producing a variety of diagnostic information and also for passing to user-supplied routines information that can be helpful in determining when to re-evaluate Jacobians or preconditioners.

Several Newton iterative solvers have been introduced previously by others. The NKSOL Fortran code [3] implements a Newton iterative method with either backtracking or trust region globalization; the Krylov solver options are GMRES and the full orthogonalization (Arnoldi) method [23]. The Scalable Nonlinear Equations Solver (SNES) package [15], which is part of the very extensive C library PETSc [16], offers a Newton iterative solver with either backtracking or trust region globalization and provides various Krylov solvers and preconditioners. SNES and PETSc achieve parallelism through message passing, using the Message-Passing Interface standard (MPI) [28] for interprocessor communication. The more specialized parallel reactive flow code MPSalsa [26], [27], also written in C, includes a nonlinear solver that, like NITSOL, is based on Algorithm INB of [9]; various Krylov solvers and incomplete factorization preconditioners are provided via the Aztec parallel iterative linear algebra library [17]. Less closely related are the truncated Newton solvers LANCELOT [6] and TN [21], [22], which employ the conjugate gradient method and are intended primarily for optimization.

In section 2 below, we introduce the inexact Newton backtracking method, discuss its implementation as a Newton iterative method, and outline the Krylov solver options. In section 3, we discuss the structure, features, and usage of NITSOL. In section 4, we give illustrative examples of NITSOL applications and usage.

2. The algorithm. The inexact Newton backtracking method that we implement is the following from [9], which offers strong global convergence properties combined with potentially fast local convergence. In this, the inexact Newton condition (1.3) is augmented with a condition of sufficient reduction of the norm of F.

ALGORITHM INB (Inexact Newton Backtracking method [9]).

LET x_0, $\eta_{\max} \in [0, 1)$, $t \in (0, 1)$, AND $0 < \theta_{\min} < \theta_{\max} < 1$ BE GIVEN.
FOR $k = 0, 1, \ldots$ (UNTIL CONVERGENCE) DO:
  CHOOSE INITIAL $\eta_k \in [0, \eta_{\max}]$ AND s_k SUCH THAT

$$\|F(x_k) + F'(x_k)\, s_k\| \le \eta_k \|F(x_k)\|.$$

  WHILE $\|F(x_k + s_k)\| > [1 - t(1 - \eta_k)]\, \|F(x_k)\|$ DO:
    CHOOSE $\theta \in [\theta_{\min}, \theta_{\max}]$.

UPDATE $s_k \longleftarrow \theta s_k$ AND $\eta_k \longleftarrow 1 - \theta(1 - \eta_k)$.
SET $x_{k+1} = x_k + s_k$.

Note that, given an initial η_k, a satisfactory initial s_k exists if the Newton equation (1.2) is consistent, in particular, if $F'(x_k)$ is invertible. If a satisfactory initial s_k can be found, then we have from remarks in [9, section 6] that in exact arithmetic Algorithm INB does not break down in the while-loop, i.e., an acceptable s_k is determined after, at most, a finite number of step reductions. Furthermore, it is easy to see that an inexact Newton condition (1.3) holds for each s_k and η_k determined by the while-loop and, in particular, for the final s_k and η_k.

The principal theoretical result for Algorithm INB in exact arithmetic is the following.

THEOREM 2.1 (see [9]). *Assume that F is continuously differentiable. If $\{x_k\}$ produced by Algorithm* INB *has a limit point x_* such that $F'(x_*)$ is invertible, then $F(x_*) = 0$ and $x_k \to x_*$. Furthermore, the initial s_k and η_k are accepted without modification in the while-loop for all sufficiently large k.*

It follows that if Algorithm INB does not terminate and produces $\{x_k\}$, then exactly one of the following must hold:

- $\|x_k\| \to \infty$, i.e., $\{x_k\}$ has no limit points.
- $\{x_k\}$ has one or more limit points, and F' is singular at each of them.
- $\{x_k\}$ converges to a solution x_* at which F' is invertible.

Note that in the case of the (desirable) third alternative, the initial s_k and η_k are accepted without modification for all sufficiently large k and, consequently, asymptotic convergence to the solution is determined by the initial η_k's as in the local convergence analysis of [8].

To implement Algorithm INB as a Newton iterative method, one must first specify various details in the algorithm and then choose a suitable iterative linear solver for determining initial inexact Newton steps. In our implementation, relatively minor details are specified as follows: as in [10, section 3.1], we use backtracking safeguard values $\theta_{\min} = 0.1$ and $\theta_{\max} = 0.5$. We take the norm to be an inner-product norm and choose $\theta \in [\theta_{\min}, \theta_{\max}]$ to minimize a quadratic that interpolates $\|F\|$ in the direction of the inexact Newton step. The value $t = 10^{-4}$ is used to judge sufficient reduction, and we use the upper bound $\eta_{\max} = 0.9$. Convergence is declared if either $\|F(x_k)\| \leq$ `FTOL` or $\|s_k\| \leq$ `STPTOL` $\cdot \|x_k\|$, where `FTOL` and `STPTOL` are user-supplied tolerances.

In the remainder of this section, we first address a major detail: choosing the initial forcing terms. We then discuss the Krylov solvers that are available in our implementation.

2.1. Choosing the forcing terms. As noted above, if the iterates produced by Algorithm INB converge to a solution of (1.1) at which F' is invertible, then the ultimate speed of convergence is determined by the initial forcing terms η_k. In particular, by the analysis of [8], desirably fast local convergence can be obtained by making suitably small choices of the initial forcing terms near a solution.

The initial forcing terms can also significantly affect the performance of the algorithm away from a solution. Indeed, choosing an initial η_k too small can lead to *oversolving* the Newton equation, i.e., imposing an accuracy on an approximate solution s_k of (1.2) that leads to significant disagreement between $F(x_k + s_k)$ and its local linear model $F(x_k) + F'(x_k)s_k$. Oversolving may result in little or no progress toward a solution. Moreover, in a Newton iterative implementation, it may involve pointless expense; a less accurate solution of (1.2) may be both cheaper and more effective in reducing the norm of F.

Here, we implement two choices of the initial forcing terms from [10] that give desirably fast local convergence and also tend to minimize oversolving. These are as follows:

Choice 1: Select any $\eta_0 \in [0,1)$ and choose

$$\eta_k = \frac{\left| \|F(x_k)\| - \|F(x_{k-1}) + F'(x_{k-1})\, s_{k-1}\| \right|}{\|F(x_{k-1})\|}, \qquad k = 1, 2, \ldots. \tag{2.1}$$

Choice 2: Given $\gamma \in [0,1]$ and $\alpha \in (1,2]$, select any $\eta_0 \in [0,1)$ and choose

$$\eta_k = \gamma \left(\frac{\|F(x_k)\|}{\|F(x_{k-1})\|} \right)^{\alpha}, \qquad k = 1, 2, \ldots. \tag{2.2}$$

Note that, for implementation in Algorithm INB, it is necessary to follow (2.1) and (2.2) with the safeguard

$$\eta_k \longleftarrow \min\{\eta_k, \eta_{\max}\}. \tag{2.3}$$

Also, in our implementation, we use the initial value $\eta_0 = 0.5$ with both Choice 1 and Choice 2.

Theorems 2.2 and 2.3 in [10] give results concerning the speed of local convergence to a solution of (1.1) when (2.1) or (2.2) is used to determine inexact Newton iterates. Augmenting Theorem 2.1 above with those results immediately yields the following extensions.

THEOREM 2.2. *Assume that F is continuously differentiable and that each η_k in Algorithm* INB *is given by* (2.1) *followed by* (2.3). *If $\{x_k\}$ produced by Algorithm* INB *has a limit point x_* such that $F'(x_*)$ is invertible, then $F(x_*) = 0$ and $x_k \to x_*$. Furthermore, if F' is Lipschitz continuous at x_*, then*

$$\|x_{k+1} - x_*\| \le \beta \|x_k - x_*\| \|x_{k-1} - x_*\|, \qquad k = 1, 2, \ldots, \tag{2.4}$$

for a constant β independent of k.

Remark. As noted in [10], it follows immediately from (2.4) that the convergence is q-superlinear, two-step q-quadratic, and of r-order $(1+\sqrt{5})/2$.

THEOREM 2.3. *Assume that F is continuously differentiable and that each η_k in Algorithm* INB *is given by* (2.2) *followed by* (2.3). *If $\{x_k\}$ produced by Algorithm* INB *has a limit point x_* such that $F'(x_*)$ is invertible, then $F(x_*) = 0$ and $x_k \to x_*$. Furthermore, if F' is Lipschitz continuous at x_*, then the following hold: if $\gamma < 1$, then the convergence is of q-order α; if $\gamma = 1$, then the convergence is of r-order α and of q-order p for every $p \in [1, \alpha)$.*

It is observed in [10] that although the forcing term choices given above are usually effective in avoiding oversolving, they occasionally become too small far away from a solution. Accordingly, we implement the following safeguards suggested in [10]:

Choice 1 *safeguard:* Modify η_k by $\eta_k \leftarrow \max\{\eta_k, \eta_{k-1}^{(1+\sqrt{5})/2}\}$ if $\eta_{k-1}^{(1+\sqrt{5})/2} > 0.1$.

Choice 2 *safeguard:* Modify η_k by $\eta_k \leftarrow \max\{\eta_k, \gamma \eta_{k-1}^{\alpha}\}$ if $\gamma \eta_{k-1}^{\alpha} > 0.1$.

These are applied after η_k has been determined by (2.1) or (2.2) and before applying the safeguard (2.3). Note that if $\{x_k\}$ converges to a solution of (1.1) at which F' is invertible and Lipschitz continuous, then, with either (2.1) or (2.2), we have $\eta_k \to 0$. It follows that the above safeguards eventually become inactive and do not affect the asymptotic convergence given in Theorems 2.2 and 2.3.

We employ one further safeguard with the above choices that plays a role near a solution of (1.1) when a stopping criterion of the form $\|F(x_k)\| < \epsilon$ is used. (Here, ϵ represents either an absolute or a relative tolerance.) Maintaining agreement between F and its local linear model means that

$$\|F(x_{k+1})\| \approx \|F(x_k) + F'(x_k)s_k\| \leq \eta_k\|F(x_k)\|.$$

Thus, if $\eta_k\|F(x_k)\| \ll \epsilon$, then it is likely that the stopping criterion can be satisfied with less effort by using a larger η_k. On the other hand, if $\eta_k\|F(x_k)\|$ is slightly larger than ϵ, then it is possible that the stopping criterion can be satisfied without an additional inexact Newton step by using a slightly smaller η_k. Therefore, we impose the following final safeguard: if $\eta_k \leq 2\epsilon/\|F(x_k)\|$, then $\eta_k \longleftarrow 0.8\epsilon/\|F(x_k)\|$.

In addition to the above choices, we also allow the user to specify a constant choice $\eta_k = \eta_*$, where $\eta_* \in [0, 1)$ is independent of k. Such a choice is occasionally useful, e.g., when only modest accuracy is desired in approximately solving (1.2). In this case, augmenting Theorem 2.1 above with Theorem 2.3 in [8] immediately gives the following convergence result for this choice.

THEOREM 2.4. *Assume that F is continuously differentiable and that each η_k in Algorithm* INB *is given by $\eta_k = \eta_*$ for some $\eta_* \in [0, 1)$ independent of k. If $\{x_k\}$ produced by Algorithm* INB *has a limit point x_* such that $F'(x_*)$ is invertible, then $F(x_*) = 0$ and $x_k \to x_*$. Furthermore, for every $\eta \in (\eta_*, 1)$, we have*

$$\|x_{k+1} - x_*\|_* \leq \eta\|x_k - x_*\|_* \tag{2.5}$$

for all sufficiently large k, where $\|v\|_* \equiv \|F'(x_*)v\|$ for $v \in \mathbb{R}^n$.

2.2. The Krylov subspace methods. In our Newton iterative implementation of Algorithm INB, a Krylov subspace method is used to obtain each initial s_k. Krylov subspace methods comprise a very broad and successful class of iterative linear algebra methods; see [13] for a recent survey.

As indicated in section 1, the Krylov solver options in our algorithm are GMRES(m), BiCGSTAB, and TFQMR. These methods are well known, widely used, and applicable to general linear systems. They are "transpose free", i.e., in an implementation, they require only products of the coefficient matrix with vectors and do not require products involving its transpose. Being "transpose free" can be very advantageous, particularly in the Newton iterative context, in which the coefficient matrix is $F'(x_k)$. If analytic evaluation of products of $F'(x_k)$ with vectors is undesirable or infeasible, then these products can be approximated with finite differences of F-values, as discussed further below.

At each step, GMRES(m) minimizes the residual norm over all corrections in the current Krylov subspace and, ipso facto, enjoys a certain optimality among all Krylov subspace methods. Because iterates are based on this norm minimization property, the method does not break down in exact arithmetic, i.e., a new iterate can always be constructed if the current iterate is not the solution, provided the coefficient matrix is nonsingular. However, if the restart value m is not sufficiently large, then the method may fail through *stagnation*, i.e., insufficient residual norm reduction over a cycle of m steps to justify continuing. Since GMRES(m) requires $O(mn)$ storage and $O(m^2n)$ arithmetic operations per cycle of m steps, it may not be feasible to take m large enough to overcome stagnation.

BiCGSTAB and TFQMR do not enjoy the residual norm minimization property of GMRES(m). However, they are based on the nonsymmetric Lanczos process and, as

a result, use short recurrences (see [13]). Because these recurrences are the result of a Galerkin condition, BiCGSTAB and TFQMR may fail to produce a new iterate and terminate prematurely. In the absence of this type of breakdown, they can proceed indefinitely with only a fixed, very modest amount of storage and arithmetic per iteration. Residual norm sequences produced by these methods usually decrease fairly smoothly, if not monotonically. However, these methods are more sensitive than GMRES(m) to roundoff error.

Our implementations of BiCGSTAB and TFQMR are standard; see [31] and [12], respectively. Our implementation of GMRES(m) is not that of [25] but rather the "simpler" Gram–Schmidt implementation of [33]. In this, the GMRES least-squares problem emerges in upper triangular form, rather than upper Hessenberg form, so no Givens rotations are necessary. Furthermore, the residual vector is maintained during the iterations and, in particular, is returned at no cost for use in the backtracking routine.

In our algorithm, products of $F'(x_k)$ with vectors that are required by the Krylov solvers can be either evaluated analytically by a user-supplied subroutine or approximated using finite-difference formulas provided by the algorithm. The available finite-difference formulas are of orders one, two, and four and are given, respectively, by

$$F'(x_k)v \approx \frac{1}{\delta}[F(x_k + \delta v) - F(x_k)], \tag{2.6}$$

$$F'(x_k)v \approx \frac{1}{2\delta}[F(x_k + \delta v) - F(x_k - \delta v)], \tag{2.7}$$

$$F'(x_k)v \approx \frac{1}{6\delta}\left[8F\left(x_k + \frac{\delta}{2}v\right) - 8F\left(x_k - \frac{\delta}{2}v\right) - F(x_k + \delta v) + F(x_k - \delta v)\right]. \tag{2.8}$$

For each of these finite-difference approximations, the difference step δ is chosen based on standard arguments to achieve accuracy by balancing estimates of floating point error and truncation error.

Since $F(x_k)$ is already available, each of (2.6)–(2.8) requires a number of new F-evaluations equal to its order. Thus, the greater accuracy of a higher-order formula carries an increased cost. We wish to point out carefully just what this cost is for the different Krylov solvers: in the case of BiCGSTAB and TFQMR, choosing a particular higher-order formula results in that formula being used *at each Krylov iteration*, and the increase in cost is likely to be significant. With GMRES(m), however, choosing a particular higher-order formula results in that formula being used *once at each restart only* to calculate the initial residual; the first-order formula (2.6) is always used within each cycle of m steps. Thus, the increase in cost of using a higher-order formula with GMRES(m) is likely to be very modest. Such selective higher-order differencing has been observed in [30] to result in essentially the same accuracy as if higher-order differencing were used throughout but at much less cost. Some practical consequences of the use of higher-order differencing are illustrated in section 4.

3. NITSOL usage. We have striven to make NITSOL as simple to use as possible while still providing a large number of options and the flexibility needed to implement sophisticated strategies. The simplest use of NITSOL requires only a minimal description of the problem to be solved: the size of the problem, an array that provides an initial approximate solution on entry and contains the final approximate solution on return, stopping tolerances, work space for the Krylov iterative method,

and the name of a user-supplied routine for evaluating the nonlinear function F. The name of a user-supplied routine for forming Jacobian-vector products is also required, but this can be a dummy routine if finite-difference approximations are preferred. In addition, the user is required to provide names of routines for calculating norms and inner products; if the usual Euclidean norm and inner product are desired, then these can be the `DNRM2` and `DDOT` BLAS routines [1] (see section 3.3 below). The names of all of these user-supplied subroutines must be declared `EXTERNAL` in the calling program.

In our experience, effective preconditioning of the linearized systems is the most crucial ingredient for obtaining good performance from the package. Since this is highly problem dependent, no preconditioner is provided with NITSOL. Instead, the code is structured to allow preconditioning to be implemented with minimal difficulty in the user-supplied routines mentioned above; in particular, no separate preconditioning routines are required. Left preconditioning, if desired, can be implemented by simply preconditioning the problem, i.e., applying the preconditioner on the left in the user's routine for evaluating F and, if Jacobian-vector products are evaluated analytically, in the routine for evaluating these as well. NITSOL explicitly accommodates right preconditioning; if this is desired, then the preconditioner is applied in the user's routine for evaluating Jacobian-vector products. See section 3.2 below for more details.

The argument list for NITSOL also includes arguments for specifying various options in the algorithm and for providing workspace, parameters, or other information to the user-supplied routines. The subsections that follow provide a more detailed description of NITSOL usage.

3.1. Specification of the nonlinear problem. A user-supplied subroutine is required to calculate F-values. Its signature is given by

```
SUBROUTINE F( N, XCUR, FCUR, RPAR, IPAR, ITRMF ),
```

where `N` is the size of the problem and `XCUR` is an array that contains the independent variable. The routine must return with the value of F at `XCUR` in the array `FCUR`. The arguments `RPAR` and `IPAR` are, respectively, real and integer arrays that may be used to provide problem-specific information or workspace to `F`. For example, with a discretized PDE problem, grid dimensions may be provided in `IPAR`, and workspace or parameters describing the problem (such as the relative strength of diffusion or mesh sizes in a nonuniform grid) may be provided in `RPAR`. These arrays are also passed to the optional Jacobian-vector product routine, described in section 3.2 below. Thus, these arrays can, if desired, be used to share information between `F` and the Jacobian-vector product routine. NITSOL places no restrictions on the length of these arrays and does not modify their contents. `ITRMF` is an integer flag that should be set to indicate whether F has been successfully evaluated. It is possible, for example, for the approximate solution to take on a value that will cause evaluation of F to produce a `NaN`. NITSOL has no mechanism to detect this and other error conditions that might arise in the evaluation of F and relies on the user-supplied routines to signal their occurrence.

3.2. Specifying Jacobian-vector products and preconditioners. A user-supplied subroutine is required that optionally calculates Jacobian-vector products and applies a preconditioner to a vector. If only one of these operations is desired, then the other need not be supplied; if neither is desired, then the routine may simply execute a `RETURN`. Its signature is given by

```
IF ( IJOB .EQ. 0 ) THEN     ! Return a Jacobian-vector product.
     ... decide whether to update the Jacobian ...
     ... evaluate the Jacobian-vector product ...
ELSE IF ( IJOB .EQ. 1 ) THEN     ! Apply the preconditioner.
     ... decide whether to update the preconditioner ...
     ... apply the preconditioning operation ...
END IF
```

FIG. 3.1. *General structure for user-defined* JACV.

```
SUBROUTINE JACV( N, XCUR, FCUR, IJOB, V, Z, RPAR, IPAR, ITRMJV ).
```

The arguments N, XCUR, FCUR, RPAR, and IPAR are as in section 3.1 above. IJOB is an input integer flag that specifies whether to form a Jacobian-vector product (IJOB=0) or apply the preconditioner (IJOB=1). JACV is called with IJOB=1 only if right preconditioning is used. The array V contains the input data for the requested operation, and the results are placed in the array Z. ITRMJV is an error flag that should be set within the routine to indicate success or failure of the requested operation.

In general, JACV must test the value of IJOB and then perform one of two operations as indicated. Structuring JACV in this way may seem burdensome to the user, but available alternatives are less desirable. One alternative is to require separate routines for evaluation of Jacobian-vector products, calculation of a preconditioner, and application of a preconditioner, all of which would be called directly by NITSOL; however, this would make NITSOL's argument list considerably longer and require the user to supply additional, possibly unnecessary routines. Another alternative is to employ reverse communication, but this would significantly complicate the interface and structure of the algorithm and can be error-prone.

We explicitly note that, if desired, the RPAR and IPAR arrays can be used to provide storage for the Jacobian of F or for any arrays needed by the preconditioner, such as matrix factors associated with incomplete factorization preconditioners. In practice, evaluating the Jacobian or preconditioner arrays may be very expensive but also may need to be done only infrequently. NITSOL provides information on which the user can base a strategy for occasionally updating the Jacobian or preconditioner arrays. This is accessed through a common block NITINFO that contains information on the progress of the algorithm, such as the nonlinear iteration count, the current F-norm, and the status (success or failure) and average rate of convergence of the previous Krylov iterations. NITINFO also has a flag that indicates the start of a new nonlinear iteration. By monitoring this information users can implement a variety of strategies for determining when to update the Jacobian or preconditioner.

A possible overall structure for JACV is illustrated by the code fragment in Figure 3.1. Of course, one can vary this structure to allow the use of a Jacobian-free method while still providing desired right preconditioning. This is done by choosing the option for finite-difference approximation of Jacobian-vector products, then selecting right preconditioning, and finally setting ITRMJV to indicate failure when IJOB .EQ. 0 is true in Figure 3.1. (This should never actually occur: if all the options to NITSOL are set correctly, then JACV is only called with IJOB=1 in this case.)

3.3. Customizing the algorithm. As indicated in section 2, numerous options are available in NITSOL. Most of these are obtained by appropriately setting values in an input integer array called INPUT; for ease of use, setting an INPUT value to zero results in a default choice in each case. Options that can be specified with the INPUT

array include a choice of three different Krylov solvers, selection of finite-difference or analytic Jacobian-vector products, several options for choosing the initial forcing terms, specification of maximum iteration counts for the nonlinear and linear iterations, specification of the restart value m when GMRES(m) is used, selection of right preconditioning, and specification of the maximum number of backtracks allowed at each nonlinear iteration. Note that the last option allows the user to turn off backtracking by setting the maximum number of backtracks to zero. Turning off backtracking can sometimes be useful, e.g., to allow the iterates to escape from a non-zero local minimum of $\|F\|$. As indicated in section 2, finite-difference approximations of Jacobian-vector products can be based on first-, second-, or fourth-order differencing formulas.

As mentioned in section 3, user-defined norms and inner products can also be specified. These can be particularly useful when the problem calls for a weighted norm and inner product. Their signatures are the same as those of the corresponding BLAS operations `DNRM2` and `DDOT`:

```
DOUBLE PRECISION FUNCTION USRNRM( N, X, INCX ),
DOUBLE PRECISION FUNCTION USRNPR( N, X, INCX, Y, INCY ),
```

where `N` is the vector length, `X` and `Y` are vectors, and `INCX` and `INCY` are strides. They can also be useful in adapting the code to parallel environments, as illustrated in section 4.2 below.

3.4. Monitoring progress of the algorithm. Several different levels of diagnostic information about the nonlinear iterations may be requested by the user. This information includes the current value of $\|F\|$, the value of the initial forcing term, step length and backtracking information, and convergence histories of the linear iterations. When GMRES is used, an estimate of the condition number of the GMRES least-squares problem is also provided, as described in [4]; this provides a lower bound on the condition number of the preconditioned Jacobian. The user may also specify an output destination for this information. On return, NITSOL provides a termination flag that indicates success or failure of the nonlinear iteration and an array that contains performance statistics.

4. Examples. In this section we present several different examples to illustrate the capabilities and flexibility of NITSOL and also to show the effects of selecting different options in the algorithm.

4.1. The generalized Bratu problem. The generalized Bratu problem is

$$\begin{aligned} \Delta u + du_x + \lambda e^u = 0 \ \text{ in } \ \Omega \equiv [0,1] \times [0,1], \\ u = 0 \ \text{ on } \ \partial\Omega, \end{aligned} \tag{4.1}$$

where d and λ are constants. The actual Bratu problem has $d = 0$ and appears in the asymptotic theory of thermal self-ignition of an enclosed chemically reactive mixture [11]. The solution u represents the difference in temperature between points interior to Ω and points on $\partial\Omega$. There is a $\lambda^{\mathsf{crit}} > 0$ such that solutions of (4.1) exist for $\lambda \leq \lambda^{\mathsf{crit}}$. Computing a solution becomes more difficult as λ increases to λ^{crit}.

To apply NITSOL to this problem, we discretized (4.1) using second-order centered finite differences on a uniform 128×128 grid. A fast Poisson solver [29] was used for right preconditioning. The arrays `IPAR` and `RPAR` were used to provide information and storage needed by the fast Poisson solver: the grid dimensions, the mesh size, d, λ, and two work arrays. We conducted experiments to compare how different choices

TABLE 4.1
Execution times (in seconds) on an SGI Power Challenge.

Forcing term	GMRES(50)	BiCGSTAB	TFQMR
Choice 1	4.94	6.52	6.41
Choice 2	5.37	6.16	6.43
$\eta = 10^{-1}$	6.81	8.67	7.32
$\eta = 10^{-4}$	7.94	9.81	8.46

of the forcing terms affect the various Krylov solvers. Choice 2 used the default values $\alpha = 2$ and $\gamma = 1.0$. The initial approximation was taken to be $u_0 = 0$, and the iterations were halted when $\|F(u_k)\| < 10^{-6}\|F(u_0)\|$. Results for parameter choices $d = 32$, $\lambda = 16$ appear in Table 4.1.

More insight into the behavior of the method may be gleaned by examining convergence histories for these experiments. Figure 4.1 shows plots of the nonlinear residual norm obtained using GMRES(50). The first plot shows that the fixed choice $\eta_k = 10^{-4}$ reduced $\|F\|$ fastest as a function of the number of nonlinear iterations. However, the second plot shows that dynamically selecting the forcing term using Choices 1 and 2 required less CPU time to reduce $\|F\|$ to a given level than either constant choice. Plots of $\|F\|$ as a function of the number of Jacobian-vector products are similar for this example. The convergence histories for BiCGSTAB and TFQMR are similar. The time for solving this problem is dominated by the time spent forming Jacobian-vector products. Problems that are dominated by the cost of nonlinear function evaluations may benefit more from a strategy that reduces the number of nonlinear iterations.

4.2. Flow in a driven cavity. The streamfunction formulation of flow in a driven cavity is

$$\begin{aligned} \frac{1}{Re}\Delta^2 u - (u_y \Delta u_x - u_x \Delta u_y) &= 0 \ \text{ in } \ \Omega \equiv [0,1] \times [0,1], \\ u &= 0 \quad \text{on } \ \partial\Omega, \\ \frac{\partial u}{\partial n} &= \begin{cases} 1 & \text{if } y = 1, \\ 0 & \text{otherwise} \end{cases} \quad \text{on } \ \partial\Omega, \end{aligned} \tag{4.2}$$

where Re is the Reynolds number. This equation models the motion of a fluid in a square container whose lid (the boundary $y = 1$) moves from left to right. To apply NITSOL, we discretized (4.2) using piecewise-linear finite elements on a uniform grid [3], [14], based on a code provided by P. Brown. A fast biharmonic solver [2] was used for right preconditioning. As in section 4.1, the arrays `IPAR` and `RPAR` were used to provide information and storage (grid dimensions, Reynolds number, mesh size, and two work arrays) needed by the preconditioner.

As mentioned in section 2.2, NITSOL provides first-, second-, and fourth-order differencing formulas for approximating Jacobian-vector products as an alternative to analytic evaluation. In experiments, we examined the sensitivity of the performance of the Krylov solvers to the choice of the finite-difference formula. For these, we discretized the problem on a 63×63 grid, resulting in a system of 3969 nonlinear equations, and used $Re = 500$. We used Choice 1 for the forcing terms, an initial approximation $u_0 = 0$, and halted the iterations when $\|F(u_k)\| < 10^{-7}\|F(u_0)\|$. No backtracking occurred. The results are summarized in Table 4.2.

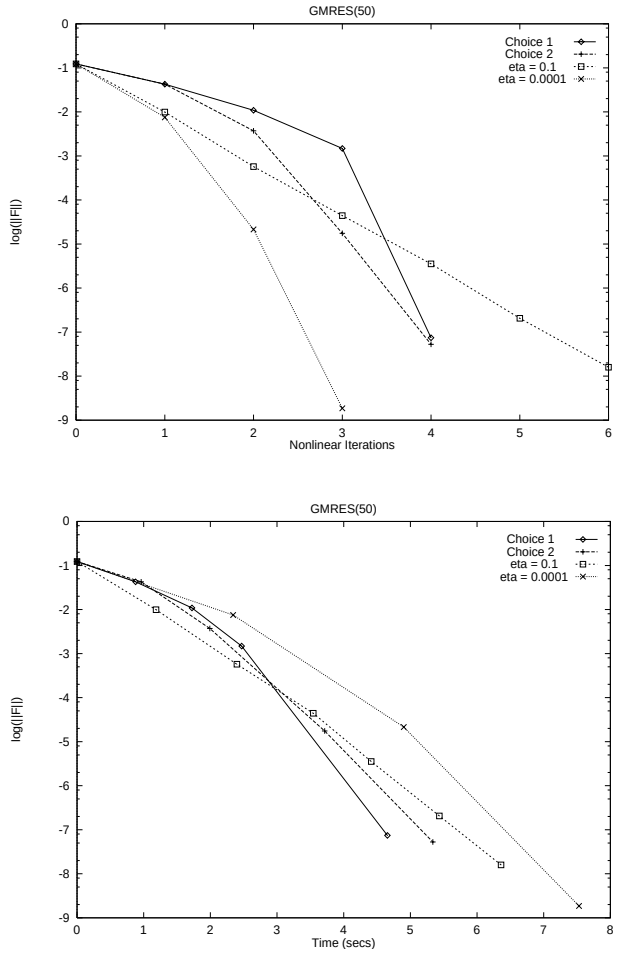

FIG. 4.1. *Convergence histories for the generalized Bratu problem using different choices of the forcing terms and GMRES as the linear solver.*

One sees from Table 4.2 that the performance of GMRES(50) was not affected at all by varying the finite-difference formula. In fact, in this example, GMRES(50) always terminated successfully in fewer than 50 iterations and never restarted. Since, in our implementation, higher-order differencing is used in GMRES(m) only to calculate the initial residual on restarts (see section 2.2), it follows that the higher-order formulas were never invoked. With TFQMR, the numbers of nonlinear and linear iterations were essentially unaffected by varying the finite-difference formula. However, since TFQMR uses higher-order differencing at each iteration, the number of function evaluations increased significantly with higher-order differencing and, as a result, so did the time. BiCGSTAB was by far the most sensitive to varying the finite-difference formula. Although the number of nonlinear iterations remained stable with BiCGSTAB, the number of linear iterations dropped sharply in going from the first-order formula to the second-order formula. However, there was a net increase in the number of function evaluations and, consequently, a slight increase in time. In going from the second-order formula to the fourth-order formula, there was

TABLE 4.2
Effect of the order of finite differences for approximating Jacobian-vector products. NNI*: number of nonlinear iterations.* NLI*: total number of linear iterations.* NFE*: number of function evaluations. Times are in seconds and were obtained on an SGI Power Challenge.*

Krylov method	Order	NNI	NLI	NFE	Time
GMRES(50)	1	10	135	146	7.85
GMRES(50)	2	10	135	146	7.84
GMRES(50)	4	10	135	146	7.81
BiCGSTAB	1	10	277	545	29.4
BiCGSTAB	2	9	207	822	30.5
BiCGSTAB	4	10	216	1699	46.6
TFQMR	1	9	94	210	11.2
TFQMR	2	10	95	419	15.1
TFQMR	4	10	95	827	22.5

a slight increase in the number of linear iterations and large increases in the number of function evaluations and in the run time.

4.3. Natural convection in an enclosed cavity. Natural convection in an enclosed cavity is a standard benchmark problem that is frequently used to test different numerical schemes and solution methods [7]. We use this example to illustrate how NITSOL can accommodate strategies that are essential for more complex, large-scale problems. The governing equations consist of the incompressible Navier–Stokes equations coupled to an energy transport equation:

$$\begin{aligned} (u^2)_x + (uv)_y + p_x - \tfrac{1}{Re}\Delta u &= 0, \\ (uv)_x + (v^2)_y + p_y - \tfrac{1}{Re}\Delta v - \tfrac{Gr}{Re^2}T &= 0, \\ u_x + v_y &= 0, \\ (uT)_x + (vT)_y - \tfrac{1}{RePr}\Delta T &= 0, \end{aligned} \tag{4.3}$$

where Gr is the Grashof number, Re is the Reynolds number, and Pr is the Prandtl number. Following [20], Re is fixed at 1, Pr is fixed at 0.71, and the Rayleigh number $Ra \equiv GrPr$ is varied. The problem is defined on the unit square $\Omega = [0,1] \times [0,1]$ with boundary conditions as depicted in Figure 4.2.

To apply NITSOL, we discretized (4.3) using finite volumes on a staggered grid. The computational domain was divided into a grid of $m \times m$ cells. The horizontal components of the velocity field were located at the midpoints of the vertical faces of the cells; the vertical components of the velocity field were located at the midpoints of the horizontal faces of the cells; and the scalar fields (pressure and temperature) were located at the centers of the cells. Our choice of fluxes at the faces of the finite-volume cells resulted in a second-order discretization [19]. Because of the conditional stability of the resulting scheme, we report results only for a modest value of Ra.

The preconditioner was a simple adaptation of ILUT(τ, `fill`) [24]. This is an incomplete factorization that retains entries in the factors based on their size up to a prescribed number of elements per row. Nonzero elements are dropped if they are smaller than a prescribed tolerance τ. In addition, the number of nonzeros per row of each factor is limited to the number of nonzero values in the unfactored row plus a prescribed number `fill`. Because the locations of fill elements in the factors cannot in general be predicted, ILUT uses the compressed sparse row (CSR) format for the

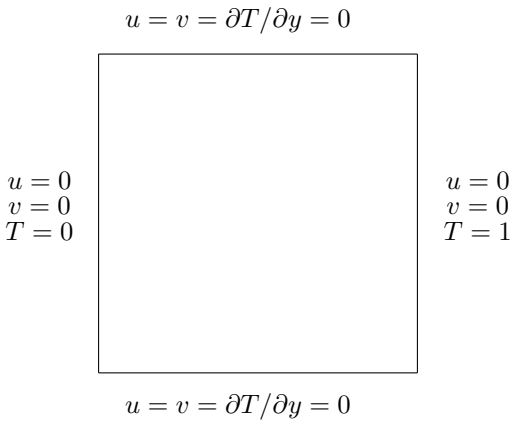

FIG. 4.2. *Boundary conditions for the natural convection problem.*

matrix and its factors; in the usual implementation, the entire matrix is stored. For large-scale problems of the type of interest here, it may be impractical or impossible to store the entire Jacobian F'. In our adaptation, we eliminated the need for this by modifying ILUT to generate the incomplete factors using one row of F' at a time. Since the ith row of each incomplete factor depends only on the previously factored rows and the ith row of F', all that is required for this is a procedure that provides the factorization routine with a CSR description of each row of F' as it is needed. This "on-the-fly" computation makes it possible to use ILUT in a fully Jacobian-free manner, in which only the incomplete factors are stored, and not F'. This can substantially reduce the required storage; see, for example, [18].

The ILUT factors can be expensive to compute, especially when a large value of `fill` is needed to obtain an effective preconditioner. In addition, there are applications for which evaluating the Jacobian is very expensive. Consequently, it may be advantageous to update the preconditioner only infrequently and to use the existing factors for as long as they are effective. For a 32×32 grid and $Ra = 1000$, we fixed $\tau = 0$ and `fill` $= 16$ and observed how sensitive the various Krylov solvers were to the frequency of updating the preconditioner. In this case the arrays `IPAR` and `RPAR` were used to provide grid dimensions, mesh size, and Rayleigh number, as well as work arrays for ILUT and for storing the CSR description of the factors. Information available in the `NITINFO` common block was used to decide when to update the preconditioner. We used Choice 1 for the forcing terms, an initial approximation of $u_0 = v_0 = p_0 = T_0 = 1$, and first-order differencing for approximating Jacobian-vector products. The iterations were halted when $\|F\|$ was reduced by a factor of 10^{-6}.

The results appear in Table 4.3. (No backtracking occurred for this problem.) BiCGSTAB and TFQMR failed to converge when the preconditioner was based only on the original Jacobian and never updated, while in this case GMRES shows its best performance, as measured by run time. The Lanczos-based algorithms failed in this case by reaching the maximum allowable number of linear iterations (which was set to 100 by using the `INPUT` array) without sufficiently reducing the linear residual norm to justify continuing. We note that increasing the maximum allowable number of linear iterations might not have helped. Indeed, in the cases in which the Lanczos-based algorithms succeeded, we also observed that these methods failed for the same reason if the maximum number of linear iterations was set at a somewhat higher

TABLE 4.3

Results of varying the frequency of updating the preconditioner in the natural convection problem. NNI*: number of nonlinear iterations.* NLI*: total number of linear iterations.* NFE*: number of function evaluations. Times are in seconds and were obtained on an SGI Power Challenge.*

Never update preconditioner

Krylov method	NNI	NLI	NFE	Time
GMRES(100)	8	272	281	86.6
BiCGSTAB	–	–	–	–
TFQMR	–	–	–	–

Update preconditioner every fifth Newton step

Krylov method	NNI	NLI	NFE	Time
GMRES(100)	8	256	265	163
BiCGSTAB	10	202	409	162
TFQMR	9	557	2242	178

Update preconditioner every other Newton step

Krylov method	NNI	NLI	NFE	Time
GMRES(100)	9	231	240	318
BiCGSTAB	9	177	357	394
TFQMR	8	450	1809	329

level. This is because, on this problem, the linear residual norms produced by the Lanczos-based algorithms initially decrease but subsequently increase beyond their original values; thus, early termination can lead to an adequate inexact Newton step while later termination may not.

Note that, overall, as the frequency of preconditioner updating is increased, the number of linear iterations and the number of nonlinear function evaluations both decrease. However, for this problem, these gains are not enough to compensate for the expense of updating the preconditioner more frequently, and the result is longer overall execution time.

4.4. Flow in a porous medium. User-defined inner products and norms in NITSOL allow the user to employ weighted inner products and norms that are best suited to the problem being solved. They can also be used to adapt NITSOL for execution on distributed memory parallel computers. All that is needed to create a parallel code is to provide distributed versions of these operations (which are easy to construct using, for example, MPI) and appropriate user-supplied routines for function evaluations, Jacobian-vector products, and preconditioning. In particular, *no internal changes to NITSOL are needed.* This approach to parallelism adheres to the general philosophy of NITSOL, which is to provide implementations of algorithms that are relatively well understood and theoretically well founded, and to leave the implementation of problem- or machine-specific strategies (in this case, data partition and communication strategies) to the user.

In this example, we show how the above approach to parallelism can be carried out. The problem we consider is

TABLE 4.4
Parallel performance of NITSOL as subdomain overlap is varied. NLI*: total number of linear iterations. Wall clock times are in seconds and were obtained on four processors of an IBM SP*2*.*

	ovlp=1		ovlp=2		ovlp=3	
Krylov method	NLI	Time	NLI	Time	NLI	Time
GMRES(50)	226	64.1	242	70.4	254	71.9
BiCGSTAB	193	63.8	203	73.3	192	67.2
TFQMR	294	81.7	272	96.2	279	88.9

$$\Delta(u^2) + d\tfrac{\partial}{\partial x}(u^3) + f = 0 \ \text{ in } \ \Omega \equiv [0,1] \times [0,1], \qquad u = \begin{cases} 1 & \text{if } x = 0 \text{ or } y = 0, \\ 0 & \text{if } x = 1 \text{ or } y = 1 \end{cases} \quad \text{on } \partial\Omega, \tag{4.4}$$

which is a steady-state special case of a general equation that models the influence of capillary pressure and gravity on flow in a homogeneous porous medium [32]. To apply NITSOL, we discretized (4.4) using centered differences on a uniform grid. The function f was a point source of magnitude 50 at the lower-left grid point, and we took $d = 50$. The initial approximate solution was $u_0(x, y) = 1 - xy$ on the interior grid points.

For parallel implementation, we partitioned the discrete domain into a $p \times q$ grid of rectangular subdomains of equal size and assigned one domain to each of $P = pq$ processors. The distributed norm and inner product were simple to implement: the respective BLAS operations `DNRM2` and `DDOT` were used to compute the local contributions to the global norm and inner product, and a global sum operation provided by MPI was then used to accumulate the local contributions. Because of the local dependency of the discretization scheme, parallel code for evaluating F requires only the exchange of a grid line between neighboring subdomains. By using finite-difference evaluation of Jacobian-vector products, no parallel code for this operation was needed.

For a parallel preconditioner, we used a single-level version of the additive Schwarz method [5], using ILUT with $\tau = 0$ and `fill` $= 4$ to approximately solve each local problem. In this example, the preconditioner was updated at the start of every nonlinear iteration. `IPAR` and `RPAR` were used to provide problem parameters (mesh size, d, f, boundary conditions) and storage needed for the incomplete factorization. In addition, `IPAR` was used to provide information regarding the structure of the parallel implementation: the rank of each process, a list of neighbors of each process, and the amount of overlap between neighboring subdomains. We also used the "on-the-fly" ILUT strategy described in section 4.3 in this situation.

To demonstrate the performance of NITSOL in this context, we investigated how the amount of overlap between subdomains affected the performance of the nonlinear solver. The results for a 2×2 partition of a 256×256 grid are in Table 4.4. Each experiment was repeated five times, and the numbers that appear in this table were obtained by removing the largest and smallest times and averaging the remaining three.

In all but two cases, increasing the overlap between subdomains increased the execution time. This general trend is expected since the size of the local problems grows as the overlap increases. The total number of linear iterations also increased

with increasing overlap in all but two cases. However, in only one instance did both execution time and the total number of linear iterations decrease. Indeed, an interesting phenomenon occurs for TFQMR when the overlap is increased from two to three: while the number of linear iterations increases, the execution time decreases. This is due to the fact that our implementation of TFQMR uses an inexpensive estimate of the residual norm to check for convergence (cf. [12]), but it does not terminate until the directly computed residual satisfies the stopping criterion. When the overlap was set to two, the residual was evaluated directly 97 times, whereas this calculation was performed only 17 times when three overlapping grid points were used. This was easily determined by using NITSOL's output options to provide convergence histories of the linear iterative method.

Acknowledgment. The Center for High Performance Computing at the University of Utah provided the computing facilities that were used to obtain the computational results in this paper. The IBM SP has been partially funded by a Shared University Research (SUR) grant from IBM.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide*, SIAM, Philadelphia, PA, 1992.

[2] P. Bjørstad, *Fast numerical solution of the biharmonic Dirichlet problem on rectangles*, SIAM J. Numer. Anal., 20 (1983), pp. 59–71.

[3] P. N. Brown and Y. Saad, *Hybrid Krylov methods for nonlinear systems of equations*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 450–481.

[4] P. N. Brown and H. F. Walker, *GMRES on (nearly) singular systems*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 37–51.

[5] T. F. Chan and T. P. Mathew, *Domain decomposition algorithms*, Acta Numerica, 3 (1994), pp. 61–143.

[6] A. R. Conn, N. I. M. Gould, and P. L. Toint, *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*, Springer Ser. Comput. Math. 17, Springer-Verlag, Heidelberg, Berlin, New York, 1992.

[7] G. de Vahl Davis, *Natural convection of air in a square cavity: A benchmark numerical solution*, Internat. J. Numer. Methods Fluids, 3 (1983), pp. 249–264.

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

[9] S. C. Eisenstat and H. F. Walker, *Globally convergent inexact Newton methods*, SIAM J. Optim., 4 (1994), pp. 393–422.

[10] S. C. Eisenstat and H. F. Walker, *Choosing the forcing terms in an inexact Newton method*, SIAM J. Sci. Comput., 17 (1996), pp. 16–32.

[11] D. A. Frank-Kamenetskii, *Diffusion and Heat Exchange in Chemical Kinetics*, Plenum Press, New York, 1969.

[12] R. W. Freund, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems*, SIAM J. Sci. Comput., 14 (1993), pp. 470–482.

[13] R. W. Freund, G. H. Golub, and N. M. Nachtigal, *Iterative solution of linear systems*, Acta Numerica, 1 (1992), pp. 57–100.

[14] R. Glowinski and O. Pironneau, *Numerical methods for the first biharmonic equation and for the two-dimensional Stokes problem*, SIAM Rev., 21 (1979), pp. 167–212.

[15] W. D. Gropp, L. Curfman McInnes, and B. F. Smith, *Using the Scalable Nonlinear Equations Solvers Package*, Tech. report ANL/MCS-TM-193, Argonne National Laboratory, Argonne, IL, 1995.

[16] W. D. Gropp and B. F. Smith, *PETSc: Portable, Extensible Tools for Scientific Computing*, Tech. report ANL-93/00, Argonne National Laboratory, Argonne, IL, 1993.

[17] S. A. Hutchinson, J. N. Shadid, and R. S. Tuminaro, *Aztec User's Guide*, Tech. report SAND95 1559, Sandia National Laboratories, Albuquerque, NM, 1995.

[18] D. A. Knoll and P. R. McHugh, *Enhanced nonlinear iterative techniques applied to a nonequilibrium plasma flow*, SIAM J. Sci. Comput., 19 (1998), pp. 291–301.

[19] R. Lazarov, I. Mishev, and P. Vassilevski, *Finite volume methods for convection-diffusion problems*, SIAM J. Numer. Anal., 33 (1996), pp. 31–55.
[20] P. R. McHugh and D. A. Knoll, *Fully coupled finite volume solutions of the incompressible Navier–Stokes and energy equations using an inexact Newton method*, Internat. J. Numer. Methods Fluids, 19 (1994), pp. 439–455.
[21] S. G. Nash, *Newton-type minimization via the Lanczos method*, SIAM J. Numer. Anal., 21 (1984), pp. 770–788.
[22] S. G. Nash, *User's Guide for TN/TNBC: Fortran Routines for Nonlinear Optimization*, Tech. report 397, Mathematical Sciences Department, The Johns Hopkins University, Baltimore, MD, 1984.
[23] Y. Saad, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comp., 37 (1981), pp. 105–126.
[24] Y. Saad, *ILUT: a dual threshold incomplete LU factorization*, J. Numer. Linear Algebra Appl., 1 (1994), pp. 387–402.
[25] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual method for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.
[26] J. N. Shadid, S. A. Hutchinson, H. K. Moffat, G. L. Hennigan, B. A. Hendrickson, and R. W. Leland, *A 65+ Gflop/s unstructured finite element simulation of chemically reacting flows on the Intel Paragon*, in Proc. Supercomputing '94, Washington, DC, November 14–18, 1994, pp. 673–679.
[27] J. N. Shadid, H. K. Moffat, S. A. Hutchinson, G. L. Hennigan, K. D. Devine, and A. G. Salinger, *MPSalsa: A Finite Element Computer Program for Reacting Flow Problems. Part* 1: *Theoretical Development*, Tech. report SAND96-2752, Sandia National Laboratories, Albuquerque, NM, 1996.
[28] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, MIT Press, Cambridge, MA, 1996.
[29] P. N. Swartztrauber and R. A. Sweet, *Algorithm* 541: *Efficient Fortran subprograms for the solution of separable elliptic partial differential equations*, ACM Trans. Math. Software, 5 (1979), pp. 352–364.
[30] K. Turner and H. F. Walker, *Efficient high accuracy solutions with GMRES(m)*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 815–825.
[31] H. A. van der Vorst, *Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.
[32] C. J. van Duijn and J. M. de Graaf, *Large time behaviour of solutions of the porous medium equation with convection*, J. Differential Equations, 84 (1990), pp. 183–203.
[33] H. F. Walker and L. Zhou, *A simpler GMRES*, Numer. Linear Algebra Appl., 1 (1994), pp. 571–581.