Mathematics and Computers in Simulation 35 (1993) 211–232
North-Holland

MATCOM 938

Iterative methods for nonsymmetric systems in DAEs and stiff ODEs codes *

A.T. Chronopoulos and C.T. Pedro
Department of Computer Science, University of Minnesota, Minneapolis, MN, United States

Abstract

Chronopoulos, A.T. and C.T. Pedro, Iterative methods for nonsymmetric systems in DAEs and stiff ODEs codes, Mathematics and Computers in Simulation 35 (1993) 211–232.

Past research demonstrated that the Newton method coupled with Krylov subspace iterative methods is more efficient than the Newton method coupled with Gaussian elimination when used to solve the corrector equation in stiff systems of Ordinary Differential Equations (ODEs) with large and sparse Jacobian. Some Krylov subspace methods have been incorporated in the mathematical software codes (such as LSODE), which solve numerical stiff systems of ODEs. In this article we review the past results on this topic and three iterative methods for solving nonsymmetric linear systems of equations. We then incorporate these iterative methods in DASSL, which is a program for numerical integration of systems of stiff ODEs and Differential-Algebraic Equations (DAEs). We present numerical tests (with systems of stiff ODEs) and timing comparisons on the CRAY-2 supercomputer.

1. Introduction

A very important problem in scientific computing is the efficient solution of systems of Initial Value Problems (IVPs) or systems of stiff ODEs and DAEs. Systems of ODEs or DAEs arise frequently in the modeling of problems from engineering and physics. Systems of DAEs often arise as systems of ODEs coupled with algebraic equations representing constraints in the model. Some problems are directly modeled as systems of ODEs or DAEs (e.g., chemical kinetics, electrical networks, mechanical systems). Problems modeled as time-dependent partial differential equations (PDEs) when semi-discretized by the method of lines also give rise to systems of ODEs or DAEs (e.g., fluid dynamics).

Past research in the area of systems of stiff ODEs has resulted in deriving efficient numerical methods with suitable stability and accuracy properties (see [3,18,22,25,28,36,37]). A system of DAEs is characterized by its index, which is defined to be the number of time differentiations that must be applied to all or a few of the equations of the system in order to obtain an explicit system of ODEs. Stable numerical methods have been proposed to solve systems of DAEs of

Correspondence to: Dr. A.T. Chronopoulos, Department of Computer Science, University of Minnesota, 4-192 EE/CSci Building, 200 Union Street S.E., Minneapolis, MN 55455, United States. e-mail: chronos@cs.umn.edu.
* The Minnesota Supercomputing Institute provided time on the CRAY-2.

index zero or one. Some results also exist for important classes of systems of DAEs of index higher than one. For more details, see [3,17,19,22,29] and the references therein.

This paper will discuss the application of iterative methods to the solution of systems of DAEs of the form

$$F(y, y', t) = 0, \tag{1.1}$$

where the initial values of at least y are given and $\partial F/\partial y'$ is nonsingular. Systems of DAEs of index one or systems of implicit or explicit ODEs are such problems. Special attention will be paid to systems of explicit stiff ODes of the form

$$F(y, y', t) = y' - g(t, y) = 0, \quad y(t_0) = y_0. \tag{1.2}$$

The Backward Differentiation Formulae (BDFs) can be used to numerically solve (1.1) (see [29]). In applying a BDF method at time t_n, we replace $y'(t_n)$ with $\rho y_n/h_n$, where ρ is the difference operator defined by

$$\rho y_n = \sum_{j=1}^{k} \tilde{\beta}_j y_{n-j}, \tag{1.3}$$

k is the order of the method, $h_n = t_n - t_{n-1}$ is the time step, y_{n-j}, $j = 1, \ldots, k$, are approximations to the solution computed at previous time steps, and β_j are the BDF coefficients, to obtain the system of nonlinear equations

$$F\left(y_n, \frac{\rho y_n}{h_n}, t_n\right) = 0. \tag{1.4}$$

The Jacobian of this system equals

$$\frac{\tilde{\beta}_0}{h_n}\frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y}. \tag{1.5}$$

For explicit systems of ODEs (see (1.2)) the Jacobian reduces to

$$\frac{\tilde{\beta}_0}{h_n} I - \frac{\partial g}{\partial y}.$$

For more details see [29].

The numerical solution of systems of stiff ODEs or DAEs yields in general nonlinear algebraic systems which must be solved at each integration step in all commonly used methods. A substantial portion of the total computational work and storage is devoted to solving these nonlinear algebraic systems, particularly if these systems are large. These nonlinear systems cannot be solved via functional iteration (as for nonstiff problems) because stiffness requires that a Newton-like method be used to solve (the corrector) equation (1.4) in order to avoid a severe restriction on the stepsize.

Over the past decade, several efficient iterative methods have been developed to solve large sparse (nonsymmetric) systems of linear algebraic equations (see [1,7,9,11,12,14,15,21,26,30,34, 38–42] and the references therein). The use of the Conjugate Gradient method or Krylov subspace methods in codes for stiff IVPs has been studied by several authors (see [4–6,8,20,24,31,33,35]). These authors incorporated inexact Newton methods coupled with iterative methods for nonsymmetric linear systems (e.g., Orthomin, IOM, GMRES) in codes implementing the BDF methods for the numerical solution of systems of stiff ODEs.

Theoretical arguments and numerical tests showed that inexact Newton coupled with a linear iterative solver is superior to inexact Newton coupled with a direct linear solver in terms of both computational work and storage, when the Jacobian of the IVP is large and sparse. The selection of the Krylov subspace iterative methods was based on the following (see [20]). (i) The methods require matrix-vector multiplications for which no explicit Jacobian need to be computed (matrix-free methods). (ii) The error is reduced faster in the eigendirections of the Jacobian corresponding to the stiff components of the system of ODEs. These components are the most inaccurate (by the predictor) and must be corrected in order to maintain stability. (iii) The eigenvalues of the Jacobian lie in the right half-plane. Krylov subspace based iterative methods (e.g., Orthomin, GMRES) seem to work efficiently for this type of linear systems. The adaptive Chebyshev iteration has also been shown to work well for this type of systems and it has been proved to satisfy properties (i)–(iii) (see [30]).

Recent research results have been published in the area of look-ahead biorthogonal Lanczos methods with aim to derive robust biorthogonal Lanczos type methods (see [2,16,21,27,32]). These methods require two matrix-vector products per iteration (one using the matrix of the linear system and one using its transpose). Squared (biorthogonal) Lanczos methods have also been proposed in [7,11,15,26,38,40]. This type of methods do not require multiplication by the transpose of the matrix of the linear system and they are (in general) faster than the standard biorthogonal Lanczos methods. We derived a squared Lanczos method (SqLanczos) by directly squaring the matrix polynomial of the biorthogonal Lanczos method (see [11]). This method requires three matrix-vector products. SqLanczos has been proved to be more robust than the other squared Lanczos methods (see [11]). For this reason we chose to incorporate this method in DASSL.

In this paper we review the inexact Newton method and three state-of-the-art iterative methods (Orthomin [14], GMRES [34] and the Squared Lanczos method [11]) for nonsymmetric linear systems. We incorporate these iterative methods in an inexact Newton method and use them in solving the corrector equation in a code (DASSL, see [3]) solving ODEs or DAEs by use of the BDF methods. All three methods can be applied in matrix-free fashion. Our work overlaps with previously published articles in this area in using the inexact Newton method coupled with Orthomin or GMRES in codes for systems of stiff ODEs. However, it differs from them because we also use a newer method (SqLanczos) from the class of biorthogonal Lanczos methods and we incorporate these methods in a code for systems of stiff ODEs and DAEs (up to index one).

The structure of the paper is as follows. In Section 2 we review the inexact Newton methods. In Section 3 we review the incomplete LU preconditioning and the three linear iterative methods we implemented. In Section 4 we describe the DASSL code and our modifications for linking the iterative methods to it. In Section 5 we present the test problems we used. In Section 6 we discuss the results of the numerical tests. In Section 7 we draw conclusions.

2. Inexact Newton

A large part of the work needed to solve systems of stiff ODEs or DAEs lies in solving the corrector equation at each integration step. This is done using the Newton method for nonlinear systems of equations.

Algorithm 2.1 (Newton).
Start with initial guess $y_n^{(0)}$, $m = 0$
Compute $F(y_n^{(0)})$
While ($\|F(y_n^{(m)})\| > \epsilon$ and $m < M_{\max}$) **Do**
(1) Compute the Jacobian

$$A = \frac{\tilde{\beta}_0}{h_n}\frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y} \quad \text{at } y_n^{(m)}$$

(2) Solve the linear system $A\,\Delta y_n^{(m)} = -F(y_n^{(m)})$
(3) Update the solution $y_n^{(m+1)} = y_n^{(m)} + \Delta y_n^{(m)}$
(4) Compute $F(y_n^{(m+1)})$
EndWhile

Another stopping criterion is $\| y_n^{(m)} - y_n^{(m-1)} \| < \epsilon$. A Newton step entails evaluating the Jacobian and the nonlinear function, and applying a linear system solver such as Gaussian elimination repeatedly until the desired stopping criteria are met.

Despite the fact that the Newton method exhibits quadratic convergence, it can be very expensive to compute the Jacobian (step (1)) and solve for $\Delta y_n^{(m)}$ exactly (to machine accuracy) (step (2)). Inexact Newton methods are usually implemented to use the same Jacobian for a few steps. The Jacobian is recomputed if Newton fails to converge in the maximum number of steps allowed. When the Newton method is coupled with a linear iterative solver, then the inexact Newton consists of an outer (nonlinear or Newton) iteration and an inner (linear) iteration. Nonlinear iterative methods which are applied directly to the nonlinear system have also been studied in [10]. Such an inexact Newton has the same form as Algorithm 2.1, except step (2) becomes

(2′) Solve the linear system $A\,\Delta y_n^{(m)} = -F(y_n^{(m)}) + r_n^{(m)}$,

where $r_n^{(m)}$ is an error vector. This means that the linear system (2) is only solved approximately with linear residual error equal to $r_n^{(m)}$. There are two stopping criteria for an iterative method used to solve (2′) that have been considered. It was proved [8] that an inexact Newton method coupled with Orthomin applied to solve systems of stiff ODES works well if $\| r_n^{(m)} \| \leqslant \epsilon$ is used as a stopping criterion for the inner iteration. Also, if the relative residual errors are bounded:

$$\frac{\| r_n^{(m)} \|}{\| F(t, y_n^{(m)}) \|} \leqslant \eta_m \quad \text{and} \quad \eta_m \leqslant \eta < 1,$$

then the inexact Newton converges at least at a linear rate (see [13]).

The number of outer iterations in an inexact Newton method coupled with an iterative method is (in general) expected to be higher than in an inexact Newton method coupled with

the LU decomposition. However, the overall computational cost for convergence is expected to be smaller.

Computing and storing the full Jacobian can be avoided because Krylov subspace methods require the Jacobian only to perform Jacobian times vector products. The Jacobian times vector product can be computed (in a matrix-free fashion) using a first-order Taylor's approximation:

$$Av \approx \frac{F(t,\ y_n + \epsilon v) - F(t,\ y_n)}{\epsilon}, \tag{2.1}$$

where for example ϵ can be taken equal to the error tolerance in the Newton method.

3. Iterative methods

In the Newton method (see Algorithm 2.1), step (2) requires the solution of the linear system

$$A\ \Delta y_n^{(m)} = b, \tag{3.1}$$

where $b = -F(y_n^{(m)})$.

One of the standard ways to solve this linear system is to use direct methods such as Gaussian elimination. We replace the direct method by an inner iteration using a linear iterative solver. For this purpose we consider three Krylov subspace iterative methods as inner iteration methods: Orthomin, Generalized Minimal Residual and Squared Lanczos. In these methods convergence is declared when the norm of the residual $\| b - A\ \Delta y_n^{(m)} \|$ is less than the given tolerance ϵ. If the inner iteration method does not converge within the maximum number of steps allowed, then the Newton iteration is also considered to have failed.

The convergence of iterative methods can be accelerated by applying preconditioning to the linear system. In preconditioning, we look for matrices P, such that

$$P_r A \approx I$$

or $P_r A$ has clustered eigenvalues, and multiplication by P_r is easy to perform. One type of preconditioning is the incomplete LU (ILU(0)) factorization of A (see [39]). The matrix A is written as $A = LU + E$, where $L_{ij} = U_{ij} = 0$ if $A_{ij} = 0$, and $E_{ij} = 0$ if $A_{ij} \neq 0$. The matrices L, U are lower and upper triangular respectively and E is an error matrix. L and U are obtained by applying Gaussian elimination only on the nonzero entries of A. The L and U factors are computed by the following algorithm.

Algorithm 3.0 (ILU(0) preconditioning).
(1) **For** $i = 1$ to N **Do**
(2) **For** $j = 1$ to N **Do**
(3) **If** $A_{ij} \neq 0$ **then**
(4) $s_{i,j} = A_{i,j} - \sum_{t=1}^{\min(i,j)-1} L_{i,t} U_{t,j}$
(5) If $i \geqslant j$ then $L_{i,j} = s_{i,j}$
(6) If $i < j$ then $U_{i,j} = s_{i,j}/L_{i,i}$
EndIf
EndFor
Endfor

The preconditioning matrix is $P_r = (LU)^{-1}$. The matrix-vector multiplication by the preconditioning matrix $P_r r$ consists of solving the two triangular systems $L\bar{v} = r$ and $Uv = \bar{v}$, where the triangular matrices L and U are the incomplete LU-factors computed in Algorithm 3.0. Solving this type of triangular linear systems on a supercomputer may be slow because the operations involved are difficult to vectorize. We resolve this difficulty by using the vectorizable ILU(0) preconditioning approach as proposed in [39].

We next describe the iterative methods that we implemented without the preconditioning step for simplicity of notation.

3.1. Orthomin

The generalized conjugate residual method GCR (see [14]) for nonsymmetric linear systems is generalization of the conjugate residual method CR for symmetric and positive definite linear systems. The main difference between GCR and CR is in computing the direction vectors. In GCR, the new direction vector is generated from the residual vector forcing A^TA-orthogonality against all the preceding direction vectors. In CR, A^TA-orthogonality against the immediately preceding vector suffices. The GCR method converges for A nonsymmetric with its symmetric part positive definite (see [14]).

The Orthomin(k) method is a truncated version of GCR and requires the storage of k direction vectors (see [14]). Let the residuals and direction vectors be denoted by r_i and p_i, respectively. Let $\Delta y_{n,0}^{(m)}$ be an initial guess (in solving (3.1)) and compute the initial residual $r_0 = b - A\ \Delta y_{n,0}^{(m)}$.

The Orthomin(k) algorithm is given next.

Algorithm 3.1 (Orthomin(k)).
(1) $p_0 = r_0$
(2) Compute Ar_0
For $i = 0$ **Step** 1 **Until** convergence **Do**
(3) $c_i = (r_i, Ar_i)/(Ap_i, Ap_i)$
(4) $\Delta y_{n,i+1}^{(m)} = \Delta y_{n,i}^{(m)} + c_i p_i$
(5) $r_{i+1} = r_i - c_i Ap_i$
(6) Compute Ar_{i+1}
(7) $b_j^i = -(Ar_{i+1}, Ap_j)/(Ap_j, Ap_j)$, $j_i \leqslant j \leqslant i$, where $j_i = \max(0, i - k + 1)$
(8) $p_{i+1} = r_{i+1} + \sum_{j=j_i}^{i} b_j^i p_j$
(9) $Ap_{i+1} = Ar_{i+1} + \sum_{j=j_i}^{i} b_j^i Ap_j$
EndFor

3.2. Generalized minimal residual (GMRES)

The GMRES algorithm for nonsymmetric linear systems was first introduced by Saad and Schultz [34]. The Arnoldi method is used to form an orthonormal basis for the Krylov subspace K_k of dimension k generated by the basis $\{r_0, Ar_0, \ldots, A^{k-1}r_0\}$. GMRES then minimizes the norm of the residual error over the subspace K_k. GMRES(k) is a restarted form of the algorithm using k direction vectors to approximate the solution. Let $\Delta y_{n,0}^{(k)}$ be an initial guess

(in solving (3.1)) and compute the initial residual $r_0 = b - A\ \Delta y_{n,0}^{(m)}$. We next give the algorithm performing one *cycle* (consisting of k iterations) of GMRES(k).

Algorithm 3.2 (GMRES(k)).
(1) $q_1 = r_0 / \| r_0 \|$
For $i = 1, 2, \ldots, k$ **Do**
(2) Compute Aq_i
(3) $\hat{q}_{i+1} = Aq_i - \sum_{j=1}^{i} \tilde{h}_{ji} q_j$ with $\tilde{h}_{ji} = (Aq_i, q_j)$, $j = 1, 2, \ldots, i$,
(4) $\tilde{h}_{i+1,1} = \| \hat{q}_{i+1} \|$
(5) $q_{i+1} = \hat{q}_{i+1} / \tilde{h}_{i+1,i}$
EndFor
(6) Form the approximate solution $\Delta y_{n,k}^{(m)} = Q_k z_k + \Delta y_{n,0}^{(m)}$, where $Q_k = [q_1, \ldots, q_k]$
(7) Compute $r_k = b - A\ \Delta y_{n,k}^{(m)}$.

The vector z_k (of size k) minimizes $J(z) = \| \| r_0 \| e_1 - \bar{H}_k z \|$, where $z \in \mathbb{R}^k$ and $e_1 = [1, 0, \ldots, 0]^T$. If $\| r_k \| > \epsilon$, the algorithm restarts with $r_0 = b - A\ \Delta y_{n,k}^{(m)}$.

The matrix $\bar{H}_k$ (of dimension $(k+1) \times k$) consists of the upper Hessenberg matrix $\tilde{H}_k$ (generated by GMRES(k)) plus the additional $(k+1)$st row whose nonzero element is $\tilde{h}_{k+1,k}$ in the $(k+1, k)$ position. Minimizing the error functional $J(z)$ is equivalent to solving

$$\min_{z \in K_k} \left\| b - A\left[\Delta y_{n,0}^{(m)} + Q_k z \right] \right\|.$$

This linear least-squares problem is solved by use of the QR decomposition of the matrix $\bar{H}_k$. More details can be found in [34].

3.3. Squared Lanczos

The restarted squared Lanczos method (SqLanczos) is based on the squared recurrences of the biorthogonal Lanczos method for solving nonsymmetric linear systems [11]. Unlike the biorthogonal Lanczos type methods (see [11]), SqLanczos does not require multiplication of the transpose of Jacobian times a vector. In [11] it was proved that this method is more robust (as it has fewer non-breakdown conditions) than the Conjugate Gradient Squared method (CGS) [38]. Let SqLanczos(k) denote the restarted form of the method. It consists of two parts. The first part consists of squaring the matrix polynomials, which represent the direction vector recurrences in the biorthogonal Lanczos method. The second part consists of squaring the matrix polynomials, which represent the residual and solution vector recurrences (see [11] for details). Let $\Delta y_{n,0}^{(m)}$ be an initial guess (in solving (3.1)) and compute the initial residual $r_0 = b - A\ \Delta y_{n,0}^{(m)}$. Let the "squared" biorthogonal Lanczos direction vectors and residuals be denoted by q_i and r_i, respectively. We next give the algorithm performing one *cycle* (consisting of k iterations) of the restarted squared Lanczos method.

Algorithm 3.3 (SqLanczos(k)).
Part I
(1) $\gamma_1 = 0$, $q_0 = 0$
(2) $p_1 = q_1 = r_0 / \| r_0 \|$
For $i = 1, \ldots, k$ **Do**

(3) Compute Aq_i, A^2q_i
(4) $\alpha_i = (q_1, Aq_i)$
(5) $\bar{q}_i = Aq_i - \alpha_i q_i$
(6) $A\bar{q}_i = A^2q_i - \alpha_i Aq_i$
(7) $\hat{q}_{i+1} = A\bar{q}_i - \alpha_i\bar{q}_i + 2\alpha_i p_i - 2Ap_i + \gamma_i q_{i-1}$
(8) $\gamma_{i+1} = (q_1, \hat{q}_{i+1})$
(9) $q_{i+1} = \hat{q}_{i+1}/\gamma_{i+1}$
(10) $p_{i+1} = \bar{q}_i - p_i$
(11) $Ap_{i+1} = A\bar{q}_i - Ap_i$
EndFor

Let T_k equal the tridiagonal matrix $[\delta_{i+1}, \alpha_i, \beta_{i+1}]$, with α_i from step (5), $\delta_{i+1} = |\gamma_{i+1}|^{1/2}$ from step (8) and $\beta_{i+1} = \delta_{i+1}\text{sign}(\gamma_{i+1})$. Then we use QR decomposition to solve $T_k z_k = \| r_0 \| e_1$, where $e_1 = [1, 0, \ldots, 0]^T$. If T_k is singular, then we reduce the size of the cycle to $k-1$. It is guaranteed that at least one of T_k or T_{k-1} is nonsingular. Let i_s be the number of sign changes in the sequence β_j, for $j = 1, \ldots, i$ (see [11]).

Part II
(1) $w_1 = r_0/\| r_0 \|$, $\beta_1 = 0$
For $i = 1, \ldots, k$ **Do**
(2) If $i \leqslant 1$ goto (7)
(3) If $2 < i$ goto (5)
(4) $\bar{\bar{w}}_{i-1} = w_{i-2} - z_m^{i-2}(-1)^{(i-2)_s} Aq_{i-2}$
(5) $\bar{w}_{i-1} = [Aw_{i-1} - \alpha_{i-1}w_{i-1} - \beta_{i-1}\bar{\bar{w}}_{i-1}]/\delta_i$
(6) $w_i = \bar{w}_{i-1} - z_m^{i-1}(-1)^{i_s} Ap_i/\beta_i$
(7) Compute Aw_i
(8) $r_i = r_{i-1} - 2z_m^i Aw_i + (-1)^{i_s}(z_m^i)^2 A^2 q_i$
(9) $\Delta y_{n,i}^{(m)} = \Delta y_{n,i-1}^{(m)} + 2z_m^i w_i - (-1)^{i_s}(z_m^i)^2 Aq_i$
EndFor

Parts I and II are one cycle of the method. If $\| r_k \| > \epsilon$, the algorithm restarts a new cycle with $r_0 = r_k$.

4. DASSL

The Differential-Algebraic System Solver (DASSL) (see [3]) is a software package designed to numerically solve systems of ODEs or systems of DAEs of index not exceeding one, written in the form

$$F(t, y, y') = 0, \qquad y(t_0) = y_0, \quad y'(t_0) = y_0', \tag{4.1}$$

where F, y and y' are N-dimensional vectors. In the underlying theory (see [3,17,22,29]), the derivative in (4.1) is replaced with a backward difference and then a nonlinear system of equations is solved at the current time t_n using the Newton method.

We next discuss the main aspects of DASSL: the time step advancement, stopping criteria and our changes to incorporate linear iterative methods.

4.1. Time step procedure

The heart of DASSL is a procedure called DASTP which moves the solution ahead one time step from t_n to t_{n+1}. This is achieved by applying a predictor-corrector BDF method of variable order up to five to the system at t_n in order to obtain a predicted value of y_n and y_n' at t_{n+1}.

From the predicted value of y_n, DASSL evaluates the Jacobian matrix (see (1.5)) using divided differences. The corrector step uses an inexact Newton method with the standard LINPACK Gaussian elimination routines to solve the system. DASTP allows no more than four iterations of the Newton method before declaring the failure of the corrector step. The initial vector $y_n^{(0)}$ in Newton is chosen to be the predicted solution vector $y_n^{(p)}$.

DASSL uses the fixed-leading coefficients implementation of the BDF methods (see [23]). DASSL chooses an order for the BDF and a step size h_n based on the error in the solution. An error estimate is obtained under the assumption that the last few time steps were taken at constant stepsize, at the current order k_n and orders at $k_n - 2$, $k_n - 1$ and $k_n + 1$. If these error estimates increase as k_n increases, then the order is reduced; if they decrease, it is raised. The new stepsize h_{n+1} is selected so that the error estimate computed under the assumption of constant stepsizes h_{n+1} at order k_{n+1} satisfies the error test. If the Newton iteration (which solves the corrector equation) has not converged within four steps, the stepsize h_n is reduced and/or a new iteration matrix is formed based on the current approximations to y, and y', and the step is attempted again.

4.2. Stopping criteria for the corrector iteration

DASSL uses two stopping criteria for the corrector step [3]:

$$\|\Delta y_n^{(m)}\| < 100\bar{\epsilon}\,\|y_n^{(p)}\|, \qquad \frac{\rho}{1-\rho}\|\Delta y_n^{(m)}\| < 0.3, \quad \rho \leqslant 0.9, \tag{4.2}$$

where $y_n^{(m)}$ is the Newton approximation to the corrected solution, $\Delta y_n^{(m)}$ is the amount of improvement of the solution in one Newton step, $y_n^{(p)}$ is the predicted solution, $\bar{\epsilon}$ is the machine epsilon and

$$\rho = \text{rate of convergence} = \left(\frac{\|\Delta y_n^{(m)}\|}{\|\Delta y_n^{(0)}\|}\right)^{1/m}.$$

Satisfying either criterion declares convergence of the corrector step. The first criterion is a strict convergence test on $\Delta y_n^{(m)}$, requiring that the norm of $\Delta y_n^{(m)}$ relative to the norm of the predicted value of y_n be less than 100 times the machine epsilon. The second is a looser restriction based on the rate of convergence, requiring that the solution converges at a linear rate.

4.3. Modifications to DASSL

In order to take advantage of the iterative methods, DASSL has been modified. In particular, parts of the corrector step have been replaced with other subroutines or eliminated.

In place of the DASSL subroutine DASLV, which calls the LINPACK subroutines DGESL and DGBSL to perform the Gaussian elimination for general and banded systems, iterative methods as described in Section 3 are substituted. The stopping criteria (4.2) were changed to a simple test of the norm of the (nonlinear) residual:

$$\left\| F\left(t_n, y_n, \frac{y_n - y_{n-1}}{h_n}\right) \right\| < \epsilon, \tag{4.3}$$

where ϵ is the tolerance for the solution of system of ODEs or DAEs.

The substitution of Gaussian elimination by iterative methods creates an additional convergence test for the Newton method inner iteration. This stopping criterion was also based on the (linear) residual norm:

$$\| r_i \| < \epsilon.$$

The combination of these two stopping criteria has been used in [8] for a system of explicit stiff ODEs and seems to work well for our test problems. The choice of right preconditioning allows the computation of the linear residual r_i instead of $P_r r_i$ (which is computed by the left preconditioning). However, scaled norms may have to be used as in the DASSL based on Newton with LU decomposition. If the iterative method was unable to converge within the maximum number of steps, taken to be $2\sqrt{N}$, where N is the size of the Jacobian, the corrector step is also considered to have failed. The number of allowable Newton iterations per time step was also modified, increased from 4 to 6.

Since the iterative methods themselves have been enhanced to reduce memory storage (using only matrix-free matrix-vector operations), it is no longer necessary to directly compute the Jacobian matrix J (1.5). Thus we have eliminated from the DASSL procedure DAJAC, which forms the Jacobian and computes an LU decomposition using the LINPACK subroutines DGEFA and DGBFA.

5. Numerical tests

To test the modifications of DASSL and the iterative methods, the following three test problems were used. In the cases presented here, we used $2\sqrt{N}$ as the maximum number of iterations allowed by an iterative method per call.

5.1. Heat equation

The first test problem is the linear two-dimensional heat equation as described in [20]:

$$\Delta u = \frac{\partial u}{\partial t}, \tag{5.1}$$

on the unit square $\Omega = \{(x_1, x_2)\colon 0 \leqslant x_1 \leqslant 1, 0 \leqslant x_2 \leqslant 1\}$, where u is the temperature distribution.

The initial temperature distribution was taken to be

$$u_0 = \begin{cases} 1, & (x_1, x_2) \in \Omega, \\ 0, & (x_1, x_2) \in \partial\Omega. \end{cases} \tag{5.2}$$

A uniform mesh and a five-point discretization were used to approximate the equation. We experimented with $\nu = 16, 32, 64$ and 128 mesh points in each direction. This yields a system of ODEs of dimension $N = \nu^2$. An absolute tolerance of 10^{-6} was used both by DASSL and the iterative methods as a convergence criterion. This system of ODEs is linear. The Jacobian (see (1.5)) for this problem is symmetric and positive definite. We used the ILU(0) preconditioning described in Section 3. However, due to the preconditioning, the Jacobian times vector products were not performed in a matrix-free fashion.

5.2. Predator–prey equation

The second test problem is based on a reaction-diffusion system arising from a Lotka–Volterra predator–prey model in two dimensions as described in [4]. There are two species variables $c^1(x_1, x_2, t)$ and $c^2(x_1, x_2, t)$, representing the prey and predator species densities over the spatial habitat $\Omega = \{(x_1, x_2)\colon 0 \leqslant x_1 \leqslant 1, 0 \leqslant x_2 \leqslant 1\}$, and time t in seconds, $0 \leqslant t \leqslant 3$.

The equations used to represent this model (see [4]) are

$$\begin{aligned} &\frac{\partial c^i}{\partial t} = d_i\left(\frac{\partial^2 c^i}{\partial x_1^2} + \frac{\partial^2 c^i}{\partial x_2^2}\right) + f^i(c^1, c^2), \quad i = 1, 2, \\ &f^1(c^1, c^2) = c^1\left(b_1 - a_{12}c^2\right), \qquad f^2(c^1, c^2) = c^2\left(b_2 - a_{21}c^1\right), \\ &d_1 = 0.05, \qquad d_2 = 1.0, \qquad b_1 = 1, \qquad b_2 = -1000, \qquad a_{12} = 0.1, \qquad a_{21} = -100. \end{aligned} \tag{5.3}$$

The boundary conditions used are homogeneous Neumann:

$$\frac{\partial c^i}{\partial x_1} = 0, \quad \text{on } x_1 = 0 \text{ and } x_1 = 1, \qquad \frac{\partial c^i}{\partial x_2} = 0, \quad \text{on } x_2 = 0 \text{ and } x_2 = 1. \tag{5.4}$$

The initial conditions used are chosen to be consistent with the boundary conditions

$$\begin{aligned} c^1(x_1, x_2, 0) &= 10 - 5\cos(\pi x_1)\cos(10\pi x_2), \\ c^2(x_1, x_2, 0) &= 17 + 5\cos(10\pi x_1)\cos(\pi x_2). \end{aligned} \tag{5.5}$$

The boundary conditions were approximated by first- and second-order one-sided approximations. For example $\partial c^i/\partial x_1 = 0$ on $x_1 = 0$ is approximated by either

$$\frac{\partial c^i}{\partial x_1}(0, x_2, t) = \frac{c^i(\Delta x_1, x_2, t) - c^i(0, x_2, t)}{\Delta x_1} \tag{5.6}$$

or

$$\frac{\partial c^i}{\partial x_1}(0, x_2, t) = \frac{-3c^i(0, x_2, t) + 4c^i(\Delta x_1, x_2, t) - c^i(2\,\Delta x_1, x_2, t)}{2\,\Delta x_1},$$

where Δx_i, for $i = 1, 2,$ is the mesh size in the x_i-direction.

A uniform mesh and a five-point discretization were used to approximate the partial differential equation. We experimented with $\nu = 16$, 32 and 64 mesh points in each direction. This yields a (nonlinear) system of ODEs of dimension $N = 2\nu^2$. An absolute tolerance of 10^{-6} and relative tolerance of 10^{-4} was used by DASSL, and a tolerance of 10^{-6} was used in the iterative methods convergence criterion.

5.3. *Krogh equation*

The third problem is an ODE model problem (see [20]) given by the following equations. Let

$$\frac{\mathrm{d}\tilde{z}}{\mathrm{d}t} = \tilde{\Delta}\tilde{z} + \tilde{\gamma}\tilde{g}(\tilde{z}), \tag{5.7}$$

where

$$\tilde{z} = [\tilde{z}(1), \tilde{z}(2), \ldots, \tilde{z}(N)]^{\mathrm{T}}, \tag{5.8}$$

$\tilde{\Delta} = \mathrm{Diag}(\tilde{\delta}(i))$ (where $\mathrm{Diag}(\cdot)$ denotes a diagonal matrix), $(\tilde{g}(\tilde{z}))(i) = \tilde{z}(i)^2$, for $i = 1, \ldots, N$, and

$$\begin{aligned} &\tilde{\delta}(1) = -1000, \quad \tilde{\delta}(2) = -800, \quad \tilde{\delta}(3) = -500, \quad \tilde{\delta}(4) = -300, \\ &\tilde{\delta}(i) = \frac{-100(N-i+1)}{(N-5)}, \quad \text{for } i \geqslant 5. \end{aligned} \tag{5.9}$$

We chose the initial values

$$\tilde{z}^{(0)} = [-1, -1, \ldots, -1]^{\mathrm{T}}. \tag{5.10}$$

The exact solution is given by

$$\tilde{z}(i)(t) = \frac{-\tilde{\delta}(i)}{\tilde{\gamma} + c(i)\,\mathrm{e}^{-\tilde{\delta}(i)t}}, \tag{5.11}$$

where the constants $c(i)$ satisfy the initial conditions.

The Jacobian (see (1.5)) for this system of ODEs is diagonal. We can transform this problem to a system of ODEs with nonsymmetric Jacobian by use of the linear transformation $y = B\tilde{z}$. We consider the matrix

$$B = I - \frac{2}{v^{\mathrm{T}}u}uv^{\mathrm{T}}, \tag{5.12}$$

where

$$u = [0,\ 1,\ 1, \ldots, 1]^{\mathrm{T}}, \qquad v = [1,\ 1,\ 1, \ldots, 1]^{\mathrm{T}}. \tag{5.13}$$

Then the system of ODEs (5.7) is transformed to

$$\frac{\mathrm{d}y}{\mathrm{d}t} = F(y) = Ay + g(y), \tag{5.14}$$

where

$$g(y)(i) = \sum_{j=1}^{N} B_{(i,j)} \left(\sum_{k=1}^{N} B_{(j,k)} y(k) \right)^2,$$

$A = B\ \mathrm{Diag}(\tilde{\delta}(i))B$ and $B^{-1} = B$. Then the Jacobian $\partial F/\partial y$ of the transformed system of ODEs (5.14) equals $B[\mathrm{Diag}(\tilde{\delta}(i) + 2\tilde{z}(i))]B$ and it is nonsymmetric since B is nonsymmetric.

In our tests we experimented with the following dimensions: $N = 16^2, 32^2, 64^2$ and 128^2. An absolute tolerance of 10^{-6} was used in DASSL and in the iterative methods convergence criterion.

6. Test results

All tests were run on a CRAY-2 supercomputer at the Minnesota Supercomputer Institute. The CRAY-2 is a four-processor shared-memory machine. Each processor can execute independent tasks concurrently. The CRAY-2 at the Minnesota Supercomputer Institute has 512 megawords of central memory. Each processor has eight 64-word vector registers and has data access through a single path between its vector registers and main memory. Each processor has 16K words of local memory. There are also six parallel pipelines: common memory to vector register, load/store vector register to local memory, load/store floating addition/subtraction, floating multiplication/division, integer addition/subtraction, and logical pipelines. CRAY-2 uses 64-bit arithmetic. We compile our codes using the automatic vectorization option. We measured the execution times using the timing facility of CRAY-2. Although our runs were not performed in single user mode, we observed that more than 75% of a single processor CPU was devoted to our execution. Thus the timings are expected to be about as accurate as if they were performed in single user mode.

We tabulated the following measurements, which can be used to compare the three iterative methods in terms of efficiency in computing the solution of the test problems. The following abbreviations appear in the results tables.

Steps: the total number of successful step taken by the integrator.

F-S: the total number of function evaluations of $F(t,\ y,\ y')$.

Calls: the total number of calls to the iterative method.

Iter steps: the number of iterations by the iterative method.

Iter F-S: the number of function evaluations of $F(t,\ y,\ y')$ done by the iterative method.

Time: CPU run time in seconds.

The following remarks can be made about the results tabulated. For each test problem, we ran Orthomin(k) (with $k = 1, 2, 4$) and the restarted methods GMRES(k) and SqLanczos(k)

Table 1
Heat equation without ILU(0) precond; $N = 256$ (16×16 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	179	997	179	818	818	0.331
GMRES(4)	177	1443	178	1012	1265	0.294
GMRES(8)	177	1798	178	1440	1620	0.396
SqLanczos(2)	177	2746	178	642	2568	0.397
SqLanczos(4)	177	3244	178	876	3066	0.479
SqLanczos(8)	177	4806	178	1424	4628	0.608

Table 2
Heat equation with ILU(0) precond; $N = 256$ (16×16 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	177	601	178	423	423	0.348
GMRES(4)	177	1083	178	724	905	0.522
GMRES(8)	177	1780	178	1424	1602	0.838
SqLanczos(2)	177	1754	178	394	1576	0.628
SqLanczos(4)	177	2684	178	716	2506	0.948
SqLanczos(8)	177	4806	178	1424	4628	1.654

(with cycle sizes $k = 2, 4, 8$). Other values of $k = 1, 2, \ldots, 10$ were tried but they were not tabulated because the results did not differ significantly from the ones presented here. The largest cost in the iterative methods is due to matrix-vector products (Matvec) expressed as

Table 3
Heat equation without ILU(0) precond; $N = 1024$ (32×32 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	206	1847	209	1038	1038	1.130
GMRES(4)	206	3184	209	2380	2975	1.320
GMRES(8)	206	2837	209	2336	2628	1.584
SqLanczos(2)	206	5177	209	1242	4968	2.070
SqLanczos(4)	206	5585	209	1536	5376	1.776
SqLanczos(8)	206	7385	209	2208	7176	2.568

Table 4
Heat equation with ILU(0) precond; $N = 1024$ (32×32 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	206	911	209	702	702	1.236
GMRES(4)	206	1454	209	996	1245	1.596
GMRES(8)	206	2099	209	1680	1890	2.166
SqLanczos(2)	206	2626	209	604	2416	2.099
SqLanczos(4)	206	3373	209	904	3164	2.698
SqLanczos(8)	206	5721	209	1696	5512	4.167

Table 5
Heat equation without ILU(0) precond; $N = 4096$ (64×64 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	226	3226	226	3000	3000	6.017
GMRES(4)	226	6426	226	4960	6200	7.267
GMRES(8)	226	4762	226	4032	4536	6.986
SqLanczos(2)	226	9498	226	2318	9272	10.644
SqLanczos(4)	226	9046	226	2520	8820	11.527
SqLanczos(8)	226	10704	226	3224	10478	11.525

Table 6
Heat equation with ILU(0) precond; $N = 4096$ (64×64 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	226	1386	226	1160	1160	6.064
GMRES(4)	226	2131	226	1524	1905	7.732
GMRES(8)	226	2512	226	2032	2286	8.076
SqLanczos(2)	226	3954	226	932	3728	9.101
SqLanczos(4)	226	4594	226	1248	4368	12.007
SqLanczos(8)	226	7324	226	2184	7098	17.668

function evaluations (see (2.1)). Orthomin and GMRES require one Matvec/iteration whereas SqLanczos requires three Matvec/iteration. Both Orthomin and GMRES reduce smoothly the norm of the linear residual whereas SqLanczos does not (see [11]). For all the test cases the

Table 7
Heat equation without ILU(0) precond; $N = 16384$ (128×128 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	254	6693	256	6437	6437	37.191
GMRES(4)	260	34066	266	27040	33800	161.964
GMRES(8)	254	13720	256	11968	13464	77.239
SqLanczos(2)	254	20800	256	5136	20544	88.026
SqLanczos(4)	254	18862	256	5316	18606	80.509
SqLanczos(8)	254	19600	256	5952	19344	85.877

Table 8
Heat equation with ILU(0) precond; $N = 16384$ (128×128 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	254	2571	256	2315	2315	31.698
GMRES(4)	254	4391	256	3308	4135	42.590
GMRES(8)	254	4063	256	3384	3807	41.484
SqLanczos(2)	254	7384	256	1782	7128	55.923
SqLanczos(4)	254	7746	256	2140	7490	60.214
SqLanczos(8)	254	10110	256	3032	9854	78.487

Table 9
Predator–prey equation with first-order boundary conditions; $N = 512$ (16×16 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	1240	7639	1278	6366	6361	4.295
Orthomin(2)	1249	6633	1280	5353	5353	4.025
Orthomin(4)	1224	6582	1258	5324	5324	4.018
GMRES(4)	1254	9087	1287	6240	7800	4.181
GMRES(8)	1235	12700	1270	10160	11430	5.903
SqLanczos(2)	1230	21678	1262	5104	20416	7.813
SqLanczos(4)	1242	34841	1269	9592	33572	12.185
SqLanczos(8)	1234	63355	1267	19104	62088	21.476

Table 10
Predator–prey equation with second-order boundary conditions; $N = 512$ (16×16 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	1142	8509	1151	7358	7358	4.561
Orthomin(2)	1143	8226	1148	7075	7075	4.696
Orthomin(4)	1143	8002	1149	6853	6853	4.814
GMRES(4)	1143	11922	1147	8620	10775	4.953
GMRES(8)	1142	12044	1145	9688	10899	5.503
SqLanczos(2)	1142	23037	1149	5472	21888	8.045
SqLanczos(4)	1142	31063	1145	8548	20918	10.738
SqLanczos(8)	1142	55199	1145	16632	54054	18.207

execution time for Orthomin and GMRES are comparable and much less than that of SqLanczos. However, the SqLanczos gives the smallest number of iterations. We next make some observations on the results obtained from each test problem.

(1) *The heat equation* (Tables 1–8). The system of ODEs is linear and stiff and the Jacobian (see (1.5)) is symmetric and positive definite. The preconditioning in this case becomes the incomplete Cholesky instead of ILU. Due to the ILU(0) computations, the Jacobian times vector products were not performed in a matrix-free fashion. For symmetric (positive definite) linear systems, Orthomin(k) with $1 \leqslant k$ reduces to CR. In this case the preconditioned CR is

Table 11
Predator–prey equation with first-order boundary conditions; $N = 2048$ (32×32 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	1271	21939	1388	20603	20551	28.080
Orthomin(2)	1116	13199	1168	12038	12031	19.069
Orthomin(4)	1103	10754	1163	9591	9591	17.237
GMRES(4)	1104	16576	1146	12344	15430	18.869
GMRES(8)	1096	13427	1133	10928	12294	17.151
SqLanczos(2)	1104	35067	1147	8480	33920	33.151
SqLanczos(4)	1103	40217	1143	11164	39074	37.890
SqLanczos(8)	1103	59147	1141	17848	58006	55.119

Table 12
Predator–prey equation with second-order boundary conditions; $N = 2048$ (32×32 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	811	14163	831	13333	13332	17.343
Orthomin(2)	811	13087	831	12257	12256	17.726
Orthomin(4)	811	12119	825	11295	11294	18.703
GMRES(4)	809	19594	814	15024	18780	21.867
GMRES(8)	809	16345	811	13808	15534	21.386
SqLanczos(2)	809	39548	828	9680	38720	40.552
SqLanczos(4)	809	39819	815	11144	39004	40.473
SqLanczos(8)	809	54897	817	16640	54080	50.457

Table 13
Predator–prey equation with first-order boundary conditions; $N = 8192$ (64×64 mesh)

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	1310	64312	1480	62914	62832	222.088
Orthomin(2)	1282	51165	1452	49784	49713	199.093
Orthomin(4)	1082	24535	1162	23378	23373	117.297
GMRES(4)	1102	69914	1219	54956	68695	200.381
GMRES(8)	1081	30643	1123	26240	29052	108.511
SqLanczos(2)	1079	77043	1267	18944	75776	222.854
SqLanczos(4)	1075	75936	1134	21372	74802	219.623
SqLanczos(8)	1075	101309	1131	30824	100178	289.464

the optimal Krylov subspace method for solving such linear systems. Also, GMRES(k) reduces to the restarted CR, which (for k small) is less efficient than standard CR. The use of preconditioning reduces by about 50% the number of iterations and total function evaluations. However, the reduction in total execution time varies in each case. Modest reduction is observed in Orthomin(1), while higher reduction is observed in SqLanczos and GMRES (e.g., for $N = 128^2$). This happens because (in the unpreconditioned case) for large Jacobian dimension the cycles of size $k = 2, 4, 8$ are too small and several cycles are spent in order to obtain a small linear residual error.

Table 14
Predator–prey equation with second-order boundary conditions; $N = 8192$ (64×64 mesh)

Method	Steps	F-S	Calls	Iter-steps	Iter F-S	Time
Orthomin(1)	892	32284	951	31340	31333	128.781
Orthomin(2)	875	29923	935	28991	28988	126.329
Orthomin(4)	870	27124	920	26206	26204	137.231
GMRES(4)	868	57756	911	45476	56845	170.557
GMRES(8)	868	37055	875	32160	36180	129.413
SqLanczos(2)	868	83280	1176	20526	82015	239.965
SqLanczos(4)	868	80387	979	22688	79408	235.914
SqLanczos(8)	868	96405	907	29384	95498	281.423

Table 15
Krogh equation with $\tilde{\gamma} = 0$; $N = 256$

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	176	844	177	667	667	0.320
Orthomin(2)	176	844	177	667	667	0.316
Orthomin(4)	176	843	177	666	666	0.320
GMRES(4)	176	1322	177	916	1145	0.343
GMRES(8)	176	1770	177	1416	1593	0.481
SqLanczos(2)	176	2425	177	562	2248	0.468
SqLanczos(4)	176	3075	177	828	2898	0.594
SqLanczos(8)	176	4805	177	1424	4628	0.880

Table 16
Krogh equation with $\tilde{\gamma} = 0$; $N = 1024$

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	168	855	169	686	686	0.809
Orthomin(2)	168	853	169	684	684	0.872
Orthomin(4)	168	854	169	685	685	0.853
GMRES(4)	168	1309	169	912	1140	0.945
GMRES(8)	168	1690	169	1352	1521	1.368
SqLanczos(2)	168	2433	169	566	2264	1.450
SqLanczos(4)	168	2997	169	808	2828	1.766
SqLanczos(8)	168	4563	169	1352	4394	2.113

(2) *The predator–prey equation* (Tables 9–14). The system of ODEs is nonlinear and stiff (see [4]) and the Jacobian (see (1.5)) is nonsymmetric. Since this problem is nonlinear and the Jacobian times vector operations are performed in a matrix-free fashion, we did not implement the ILU preconditioning for this problem. The second-order approximation to the boundary conditions seems to give the best performance results. This is expected as the partial derivatives are also discretized using second-order differences.

(3) *Krogh's ODE model equation* (Tables 15–22). This system of ODEs is (non)linear and stiff. Since the Jacobian is mildly nonsymmetric, the Orthomin method is the most efficient for

Table 17
Krogh equation with $\tilde{\gamma} = 0$; $N = 4096$

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	162	895	164	731	731	2.786
Orthomin(2)	162	893	164	729	729	2.696
Orthomin(4)	162	894	164	730	730	2.867
GMRES(4)	162	1409	164	996	1245	2.052
GMRES(8)	162	1649	164	1320	1485	3.801
SqLanczos(2)	162	2508	164	586	2344	4.385
SqLanczos(4)	162	3076	164	832	2912	5.342
SqLanczos(8)	162	4480	164	1328	4316	7.369

Table 18
Krogh equation with $\tilde{\gamma} = 0$; $N = 16384$

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	158	890	159	731	731	9.727
Orthomin(2)	158	891	159	732	732	10.687
Orthomin(4)	158	891	159	732	732	10.716
GMRES(4)	158	1475	160	1052	1315	12.196
GMRES(8)	158	1627	160	1304	1467	13.685
SqLanczos(2)	158	2583	159	606	2424	17.832
SqLanczos(4)	158	3072	160	832	2912	20.051
SqLanczos(8)	158	4372	160	1296	4212	26.121

Table 19
Krogh equation with $\tilde{\gamma} = 1$; $N = 256$

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	176	905	206	699	699	0.353
Orthomin(2)	176	905	206	699	699	0.363
Orthomin(4)	176	903	206	697	697	0.367
GMRES(4)	176	1474	204	1016	1270	0.421
GMRES(8)	176	2030	203	1624	1827	0.562
SqLanczos(2)	176	2684	204	620	2480	0.601
SqLanczos(4)	176	3521	203	948	3318	0.697
SqLanczos(8)	176	5507	203	1632	5304	1.140

Table 20
Krogh equation with $\tilde{\gamma} = 1$; $N = 1024$

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	169	926	203	723	723	0.848
Orthomin(2)	169	932	203	729	729	0.972
Orthomin(4)	169	928	203	725	725	0.948
GMRES(4)	169	1544	204	1072	1340	1.221
GMRES(8)	169	2040	204	1632	1836	1.606
SqLanczos(2)	169	2807	207	650	2600	1.596
SqLanczos(4)	169	3622	206	976	3416	1.886
SqLanczos(8)	169	5535	205	1640	5330	2.898

Table 21
Krogh equation with $\tilde{\gamma} = 1$; $N = 4096$

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	163	1005	219	786	786	3.169
Orthomin(2)	163	1004	219	785	785	3.505
Orthomin(4)	163	1008	219	789	789	3.206
GMRES(4)	163	1767	217	1240	1550	4.135
GMRES(8)	163	2159	215	1728	1944	5.385
SqLanczos(2)	163	2992	216	694	2776	5.806
SqLanczos(4)	163	3898	216	1052	3682	7.263
SqLanczos(8)	163	6014	216	1784	5798	10.805

Table 22
Krogh equation with $\tilde{y} = 1$; $N = 16384$

Method	Steps	F-S	Calls	Iter steps	Iter F-S	Time
Orthomin(1)	158	1024	229	795	795	12.061
Orthomin(2)	158	1024	229	795	795	11.829
Orthomin(4)	158	1024	229	795	795	13.970
GMRES(4)	158	1840	230	1288	1610	16.962
GMRES(8)	158	2299	229	1840	2070	22.204
SqLanczos(2)	158	3152	232	730	2920	24.446
SqLanczos(4)	158	4053	231	1092	3822	29.604
SqLanczos(8)	158	6366	230	1888	6136	45.506

both the linear and the nonlinear case. The number of function evaluations for the nonlinear case is higher than the linear case for all three methods (as expected).

7. Conclusions

We reviewed past research results in the area of iterative methods applied to large and sparse linear systems arising in the numerical integration of systems of stiff ODEs. We discuss the problem of using inexact Newton coupled with iterative methods in the numerical solution of systems of DAEs. We incorporated in a ODEs or DAEs code (DASSL) an inexact Newton method coupled with three iterative methods (Orthomin(k), GMRES(k) and SqLanczos(k)). We also included ILU(0) preconditioning for linear systems of ODEs. We tested this modified code on three problems which are systems of stiff ODEs.

In almost all tests, Orthomin needed the fewest number of total function evaluations and required the least amount of CPU-time. It was only in the nonlinear predator–prey test problem with first-order boundary conditions and 32×32 or 64×64 mesh points that GMRES became comparable to Orthomin. Other than this case, Orthomin was more efficient than GMRES and SqLanczos in total evaluations and CPU-time. Similarly, GMRES was more efficient than SqLanczos. SqLanczos gave the lowest total number of iterations. However, in SqLanczos three Matvec/iteration are performed versus one Matvec/iteration for either Orthomin or GMRES. This accounts for the low efficiency demonstrated by SqLanczos. One of the reasons for the superior efficiency of Orthomin is due to the fact that the Jacobian in these test problems is very close to symmetric. For symmetric (positive definite) linear systems Orthomin reduces to CR. Also, the restarted GMRES(k) reduces to restarted CR, which (for small k) is not as efficient as CR.

The addition of the incomplete Cholesky preconditioning in the heat equation test problem resulted in a decrease in both the total number of iterations, and the number of function evaluations performed by the iterative method of 30–70%. The CPU run time increased in all cases except for the largest dimension (128×128 mesh points). The CPU run time is expected to further decrease for larger dimensions.

Acknowledgements

The authors thank the anonymous referees whose comments helped enhance significantly the quality of presentation of this article.

References

[1] O. Axelsson, A generalized conjugate gradient, least square method, *Numer. Math.* 51 (1987) 209–227.

[2] D. Boley, S. Elshay, G.H. Golub and M.H. Gutknecht, Nonsymmetric Lanczos and finding orthogonal polynomial associated with indefinite kernels, *Numer. Algorithms* 1 (1991) 21–43.

[3] K.E. Brennan, S.L. Campbell and L.R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations* (Elsevier, New York, 1989).

[4] P.N. Brown and A.C. Hindmarsh, Matrix-free methods in the solution of stiff systems of ODEs, *SIAM J. Numer. Anal.* 23 (3) (1986) 610–638.

[5] P.N. Brown and A.C. Hindmarsh, Reduced storage matrix methods in stiff ODE systems, *Appl. Math. Comput.* 31 (1989) 40–91.

[6] G.D. Byrne, Pragmatic experiments with Krylov methods in the stiff ODE setting, in: J. Cash and I. Gladwell, eds., *Numerical Ordinary Differential Equations* (Clarendon Press, Oxford, to appear); also: Invited talk at the IMA Conf. on Computational ODEs, Imperial College, London, 1989.

[7] T.F. Chan, L. De Pillis and H. van der Vorst, A transpose-free squared Lanczos algorithm and application to solving nonsymmetric linear systems, Tech. Report CAM 91-17, Univ. California, Los Angeles, 1991.

[8] T. Chan and K. Jackson, The use of iterative linear-equation solvers in codes for large systems of stiff IVPs for ODEs, *SIAM J. Sci. Statist. Comput.* 7 (2) (1986) 378–417.

[9] A.T. Chronopoulos, *s*-step iterative methods for (non)symmetric (in)definite linear systems, *SIAM J. Numer. Anal.* 28 (6) (1991) 1776–1789.

[10] A.T. Chronopoulos, Nonlinear CG-like iterative methods, *J. Comput. Appl. Math.* 40 (1) (1992) 73–89.

[11] A.T. Chronopoulos, On the squared unsymmetric Lanczos method, *J. Comput. Appl. Math.* (1994), to appear; also: Tech. Report UMSI 91/310, Supercomputing Inst., Univ. Minnesota, 1991.

[12] P. Concus, G.H. Golub and D.P. O'Leary, A generalized conjugate gradient method for the numerical solution of partial differential equations, in: J.R. Bunch and D.J. Rose, eds., *Sparse Matrix Computations* (Academic Press, New York, 1976) 309–322.

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

[14] 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) 345–357.

[15] R.W. Freund, A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems, Tech. Report 91.18, NASA Ames Research Center, Moffett Field, 1991.

[16] R.W. Freund and N.M. Nachtigal, QMR: A quasi-minimal residual method for non-Hermitian linear systems, *Numer. Math.* 60 (1991) 315–339.

[17] C.W. Gear, Simultaneous numerical solution of differential algebraic equations, *IEEE Trans. Circuit Theory* CT-18 (1) (1971) 89–95.

[18] C.W. Gear, *Numerical Initial Value Problems in ODEs* (Prentice-Hall, Englewood Cliffs, NJ, 1971).

[19] C.W. Gear and L. Petzold, ODE methods for the solution of differential algebraic systems, *SIAM J. Numer. Anal.* 21 (1984) 716–728.

[20] W. Gear and Y. Saad, Iterative solution of linear equations in ODE codes, *SIAM J. Sci. Statist. Comput.* 4 (1983) 583–601.

[21] M.H. Gutknecht, The unsymmetric Lanczos algorithms and their relations to Padé approximation, continued fractions and the QD algorithms, in: T.A. Manteuffel, ed., *Proc. Copper Mountain Conf. on Iterative Methods*, Univ. Colorado, Denver, CO (1990).

[22] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*, Comput. Math. 14 (Springer, New York, 1991).

[23] K.R. Jackson and R. Sacks-Davis, An alternative implementation of variable step-size multistep formulas for stiff ODEs, *ACM Trans. Math. Software* 6 (3) (1980) 295–318.

[24] K.R. Jackson and W.L. Seward, Adaptive linear equation solvers in codes for large stiff ODEs, Tech. Report CS-91-33, Dept. Comput. Sci., Univ. Waterloo, Ont., 1991.

[25] C. Johnson, Error estimates and adaptive time-step control for a class of one-step methods for stiff ordinary differential equations, *SIAM J. Numer. Anal.* 25 (1988) 908–926.

[26] W.D. Joubert, Generalized gradient and Lanczos methods for the solution of nonsymmetric systems of linear equations, Report CNA-238, Univ. Texas, Austin, 1990.

[27] S.K. Kim and A.T. Chronopoulos, An efficient nonsymmetric Lanczos method on parallel vector computers, *J. Comput. Appl. Math.* 42 (3) (1992) 357–374.
[28] F.T. Krogh and K. Stewart, Asymptotic absolute stability ($h \to \infty$) for BDFs applied to stiff differential equations, *ACM Trans. Math. Software* 10 (1984) 45–56.
[29] P. Lotstedt and L.R. Petzold, Numerical solution of nonlinear differential equations with algebraic constraints I: Convergence results for backward differentiation formulas, *Math. Comp.* 46 (1986) 491–516.
[30] T.A. Manteuffel, The Tchebyshev iteration for nonsymmetric linear systems, *Numer. Math.* 28 (1977) 307–327.
[31] W.L. Miranker and I.L. Chern, Dichotomy and conjugate gradients in the stiff initial value problems, Tech. Report RC 8032, IBM, T.J. Watson Research Center, 1980.
[32] B.N. Parlett, D.R. Taylor and Z.A. Liu, A look-ahead Lanczos algorithm for unsymmetric matrices, *Math. Comp.* 44 (1985) 105–124.
[33] G. Rockswold and A. Chronopoulos, Implementation of some matrix-free iterative methods in stiff ODE codes, Tech. Report UMSI 96/16, Supercomputing Inst., Univ. Minnesota, 1991.
[34] Y. Saad and M. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* 7 (1986) 856–869.
[35] P.E. Saylor and R.D. Skeel, Linear iterative solvers for implicit ODE methods, Tech. Report 90-51, ICASE, NASA Langley Research Center, 1990.
[36] L.F. Shampine, Implementation of implicit formulas for the solution of ODEs, *SIAM J. Sci. Statist. Comput.* 1 (1) (1980) 103–118.
[37] R.D. Skeel, Computational error estimates for stiff ODEs, in: *Computational Mathematics I* (Boole, Dublin, 1985) 1–20.
[38] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric systems, *SIAM Sci. Statist. Comput.* 10 (1989) 36–52.
[39] H. van der Vorst, A vectorizable variant of some ICCG methods, *SIAM Sci. Statist. Comput.* 3 (3) (1982) 350–356.
[40] H. 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 (2) (1992) 631–644.
[41] O. Widlund, A Lanczos method for a class of nonsymmetric systems of linear equations, *SIAM J. Numer. Anal.* 15 (1978) 801–812.
[42] D.M. Young and K.C. Jea, Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods, *Linear Algebra Appl.* 34 (1980) 159–194.