QMRPACK: a Package of QMR Algorithms

ROLAND W. FREUND

Bell Laboratories, Lucent Technologies

and

NOËL M. NACHTIGAL

Sandia National Laboratories

The quasi-minimal residual (QMR) algorithm is a Krylov-subspace method for the iterative solution of large non-Hermitian linear systems. QMR is based on the look-ahead Lanczos algorithm that, by itself, can also be used to obtain approximate eigenvalues of large non-Hermitian matrices. QMRPACK is a software package with FORTRAN-77 implementations of the QMR algorithm and variants thereof, and of the three-term and coupled two-term look-ahead Lanczos algorithms. In this paper, we discuss some of the features of the algorithms in the package, with emphasis on the issues related to using the codes. We describe in some detail two routines from the package, one for the solution of linear systems, and the other for the computation of eigenvalue approximations. We present some numerical examples from applications where QMRPACK was used.

Categories and Subject Descriptors: F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*computations on matrices*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*eigenvalues*; *linear systems* (*direct and iterative methods*); G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis*; *efficiency*; *reliability and robustness*

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Linear systems, eigenvalues, iterative method, quasi-minimal residual property, non-Hermitian matrices, Krylov subspace, Lanczos method, look-ahead, three-term recurrences, coupled two-term recurrences, transfer functions

---

1. INTRODUCTION

The nonsymmetric Lanczos algorithm [Lanczos 1950] can be viewed as a procedure for building a basis for two Krylov subspaces. Given a matrix $A \in \mathbb{C}^{N\times N}$ and two vectors $v_1, w_1 \in \mathbb{C}^N$ with $\|v_1\|_2 = \|w_1\|_2 = 1$, the classical Lanczos algorithm builds two sets of vectors

$$\{v_j\}_{j=1}^n \quad \text{and} \quad \{w_j\}_{j=1}^n \tag{1}$$

Part of this research was performed while the authors were in residence at the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035, and it was supported in part by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration and the Universities Space Research Association. The research of the second author was also supported in part by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy, under contract DE–AC05–840R21400 with the Martin Marietta Energy Systems, Inc.

Authors' current addresses: R. W. Freund, Bell Laboratories, Lucent Technologies, Room 2C–420, 700 Mountain Avenue, Murray Hill, NJ 07974–0636, email: freund@research.bell-labs.com; N. M. Nachtigal, Mathematical Sciences Section, Sandia National Laboratories, P.O. Box 969, MS 9214, Livermore, CA 94551–0969, email: santa@z.ca.sandia.gov

such that, for $n = 1, 2, \ldots$,

$$\begin{aligned} \operatorname{span}\{v_1, v_2, \ldots, v_n\} &= K_n(v_1, A), \\ \operatorname{span}\{w_1, w_2, \ldots, w_n\} &= K_n(w_1, A^T). \end{aligned} \tag{2}$$

Here,

$$K_n(v_1, A) = \operatorname{span}\{v_1, Av_1, \ldots, A^{n-1}v_1\} \quad \text{and} \quad K_n(w_1, A^T) = \operatorname{span}\{w_1, A^T w_1, \ldots, (A^T)^{n-1} w_1\}$$

denote the nth *Krylov subspaces* of $\mathbb{C}^N$ generated by v_1 and A, and w_1 and A^T, respectively. In addition, the two basis sets are generated *biorthogonal*, so that, for the next pair of Lanczos vectors v_{n+1} and w_{n+1}, we have

$$w_{n+1}^T v_j = w_j^T v_{n+1} = 0 \quad \text{for all} \quad j = 1, 2, \ldots, n. \tag{3}$$

Furthermore, the Lanczos vectors (1) are usually scaled to have unit Euclidean length, i.e.,

$$\|v_{n+1}\|_2 = \|w_{n+1}\|_2 = 1 \quad \text{for all} \quad n = 0, 1, \ldots \ . \tag{4}$$

However, it turns out that it may not always be possible or desirable to construct vectors satisfying (3), as construction of the pair v_{n+1} and w_{n+1} involves division by $w_n^T v_n$, which may be close to zero in some sense, or even exactly zero. Such an event is called an *exact breakdown* ($w_n^T v_n = 0$) or a *near-breakdown* ($w_n^T v_n$ close to zero), and one remedy is to relax the biorthogonality relation (3), requiring instead that it only hold for all $j = 1, 2, \ldots, n_l - 1$, where $n_l \leq n + 1$. This leads to so-called *look-ahead* Lanczos algorithms.

We stress that the Lanczos process is not the only practical approach to building bases for Krylov subspaces. An alternative is to use the Arnoldi process [Arnoldi 1951], which produces a sequence of orthogonal basis vectors for the subspaces $K_n(v_1, A)$, $n = 1, 2, \ldots$. Unlike the Lanczos process, the Arnoldi process cannot break down. However, the Arnoldi process employs long recurrences that involve all previously generated basis vectors. Consequently, the algorithm can only be run for a limited number of steps, after which one restarts the procedure. In contrast, the Lanczos process, even with look-ahead, employs short recurrences that involve only a few of the most recently generated basis vectors. Therefore, the Lanczos process can be and usually is run without any restarts.

When viewed as a procedure to build Krylov subspaces, the Lanczos process lends itself to algorithms for the solution of linear systems or for the computation of eigenvalue estimates. However, the Lanczos algorithm has many other connections and applications, such as orthogonal polynomials, Hankel matrices, control theory, Padé approximation, and coding theory. In all these areas, the problem of breakdowns was evident from the beginning, and remedies for the case of exact breakdowns have been devised in each of these areas; we refer the reader to Freund [1994] and the references given there. However, for the case of near-breakdowns, it was not until 1982 that an algorithm was proposed. Taylor [1982] and Parlett et al. [1985] devised a look-ahead Lanczos algorithm and coined the term "look-ahead". However, they presented details only for the case of look-ahead steps of size at most 2, and their procedure does not easily generalize to larger blocks. Interest in look-ahead Lanczos algorithms was revived in the early 1990s, with a flurry of papers; see the references given in Freund [1994]. Freund et al. [1993] also proposed a look-ahead Lanczos algorithm that has several attractive features: it reduces to the classical Lanczos algorithm in the absence of look-ahead steps, it can handle look-ahead steps of arbitrary size, and all steps of the algorithm require as few inner products as the classical Lanczos algorithm. Freund and Nachtigal [1991] used this look-ahead Lanczos algorithm to extend the quasi-minimal residual (QMR) linear-systems solver proposed by Freund [1989; 1992a] for the special case of complex symmetric matrices to general non-Hermitian matrices. Later, Freund and Nachtigal [1994] devised a coupled two-term version of the look-ahead Lanczos algorithm, which seems to have better numerical properties when used in conjunction with the QMR method. Finally, Freund [1993] proposed TFQMR, a transpose-free QMR algorithm, which makes it possible to use the QMR approach in cases where application of the transpose of the matrix is not possible or desirable.

As a product of their papers, Freund and Nachtigal have produced QMRPACK, a package of FORTRAN routines that implement the look-ahead Lanczos algorithm proposed by Freund et al. [1993], as well as the coupled two-term recurrence algorithm devised by Freund and Nachtigal [1994]. QMRPACK also includes several QMR algorithms based on these two look-ahead Lanczos implementations, the TFQMR algorithm [Freund 1993], as well as an eigenvalue solver. The objective of this paper is to describe the content and the use of the software package QMRPACK.

We would like to emphasize two points about QMRPACK. The first concerns preconditioners for the linear systems solvers in the package. Typically, the efficiency of these algorithms is greatly improved when combined with a good preconditioner. However, the most efficient preconditioners are usually specific to the type of problem to be solved. The objective of QMRPACK is not to provide code for all these problem-specific preconditioners. Instead, QMRPACK only includes code for two standard preconditioners: SSOR preconditioning [Axelsson 1985] and a version of an incomplete LU factorization technique proposed by Saad [1992]. However, the linear systems solvers in QMRPACK are set up such that it is easy for the user to incorporate his own favorite preconditioner. The second point concerns the ultimate reliability of the algorithms in QMRPACK. In general, the convergence behavior and the efficiency of iterative methods, such as the ones in this software package, depend on the eigenvalue distribution and the possible non-normality of the matrix A, and also on other factors. In particular, for a given arbitrary matrix, iterative methods may or may not work, or they may converge too slowly to be practical. These problems can be avoided by employing a good preconditioner, but still the point is that iterative methods are less reliable than direct methods.

The remainder of the paper is organized as follows. In Section 2, we discuss some of the applications of the look-ahead Lanczos algorithms from QMRPACK. In Section 3, we give a brief description of the three-term and coupled two-term Lanczos algorithms. In Section 4, we recall how the Lanczos process simplifies in certain cases. In Section 5, we discuss general features of QMRPACK, and we describe in some detail two routines, one for solving linear systems and another for eigenvalue computations. In Section 6, we present some numerical examples from applications where QMRPACK was used.

2. APPLICATIONS

2.1 Background

The Lanczos vectors $\{v_j\}_{j=1}^n$ and $\{w_j\}_{j=1}^n$ are generated by means of recurrences that can be compactly summarized as follows:

$$\begin{aligned} AV_n &= V_n H_n + [\,0 \quad \cdots \quad 0 \quad v_{n+1}\,]\,\rho_{n+1}, \\ A^T W_n &= W_n \tilde{H}_n + [\,0 \quad \cdots \quad 0 \quad w_{n+1}\,]\,\eta_{n+1}. \end{aligned} \tag{5}$$

Here, ρ_{n+1}, $\eta_{n+1} > 0$ are scalars that are chosen to ensure the scaling (4),

$$V_n = [\,v_1 \quad v_2 \quad \cdots \quad v_n\,] \quad \text{and} \quad W_n = [\,w_1 \quad w_2 \quad \cdots \quad w_n\,] \tag{6}$$

are $N \times n$ matrices whose columns are just the Lanczos vectors (1),

$$H_n = \begin{bmatrix} \Theta_1 & \Delta_2 & 0 & \cdots & 0 \\ \Upsilon_2 & \Theta_2 & \ddots & \ddots & \vdots \\ 0 & \Upsilon_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \Delta_l \\ 0 & \cdots & 0 & \Upsilon_l & \Theta_l \end{bmatrix}, \quad \text{where} \quad l = l(n), \tag{7}$$

is a block-tridiagonal matrix of size $n \times n$ that is also upper Hessenberg, and

$$\tilde{H}_n = \Gamma_n^{-1} H_n \Gamma_n, \quad \text{where} \quad \Gamma_n = \operatorname{diag}(\gamma_1, \gamma_2, \ldots, \gamma_n) > 0, \tag{8}$$

is diagonally similar to (7). The integer $l = l(n)$ in (7) is a counter for the number of look-ahead steps performed so far. Furthermore, the diagonal blocks $\Theta_1, \Theta_2, \ldots, \Theta_l$ of the matrix H_n are all square, and their sizes are just the lengths of the look-ahead steps. In particular, if no true look-ahead steps of length bigger than one occur, then $l = n$ and H_n is just the scalar tridiagonal matrix generated by the classical Lanczos algorithm.

The Lanczos algorithm is mostly applied to very large $N \times N$ matrices A, and typically, $n \ll N$. In fact, a key property of the Lanczos process is that it produces a very useful small-dimensional approximation, namely the matrix H_n, to the large-dimensional matrix A. The matrix H_n is—in the sense of an oblique projection process—the best $n \times n$ approximation to A that can be obtained using only information from the two Krylov subspaces $K_n(v_1, A)$ and $K_n(w_1, A^T)$. Indeed, it can be shown that H_n is the result of the projection of A onto $K_n(v_1, A)$ and orthogonally to $K_n(w_1, A^T)$.

Next, we briefly review three applications of the Lanczos process in large-scale matrix computations.

2.2 Approximate Eigenvalues

The relations in (5) suggest to use the eigenvalues of the tridiagonal matrix H_n as approximate eigenvalues of A. Indeed, if $v_{n+1}\rho_{n+1} = 0$ or $w_{n+1}\eta_{n+1} = 0$ in (5), then each eigenvalue of H_n would be an exact eigenvalue of A. The resulting procedure thus consists of two steps. First, we run the look-ahead Lanczos algorithm for n iterations to obtain the block-tridiagonal Lanczos matrix (7), H_n. Second, we compute some or all of the eigenvalues of H_n. Note that, unless we are also interested in computing approximate right or left eigenvectors of H_n, it is not necessary to store the Lanczos vectors. For an actual implementation of this procedure, a number of important issues need to be addressed, such as the solution of the eigenvalue problem for H_n, convergence tests, and the detection of spurious eigenvalues caused by loss of biorthogonality. For a discussion of these practical aspects, we refer the reader to the paper by Cullum and Willoughby [1986], where the classical Lanczos algorithm without look-ahead is considered.

2.3 Iterative Solution of Linear Systems

Next, we turn to systems of linear equations,

$$Ax = b, \tag{9}$$

with $A \in \mathbb{C}^{N \times N}$ as coefficient matrix. Let $x_0 \in \mathbb{C}^N$ be an arbitrary initial guess for (9), and set $r_0 = b - Ax_0$. Our goal is to construct approximate solutions of (9) of the form

$$x_n \in x_0 + K_n(r_0, A), \quad n = 1, 2, \ldots . \tag{10}$$

We now choose $v_1 = r_0/\|r_0\|_2$ and any $w_1 \in \mathbb{C}^N$, $\|w_1\|_2 = 1$, as the starting vectors for the look-ahead Lanczos algorithm. Then, by (2), the Lanczos vectors $\{v_j\}_{j=1}^n$ span the Krylov subspace $K_n(r_0, A)$ in (10). Therefore, any iterate (10) can be represented in the form

$$x_n = x_0 + V_n z_n, \quad \text{where} \quad z_n \in \mathbb{C}^n, \tag{11}$$

and V_n is the matrix defined in (6). It remains to select the free parameter vector z_n in (11). Based on the Lanczos matrix (7), H_n, and its extended version, $H_n^{(e)}$, defined by

$$H_n^{(e)} = \begin{bmatrix} H_n \\ \varrho_n^T \end{bmatrix}, \quad \text{where} \quad \varrho_n = [0 \quad 0 \quad \cdots \quad 0 \quad \rho_{n+1}]^T \in \mathbb{C}^n, \tag{12}$$

there are two natural approaches.

The first one is to choose z_n as the solution of the linear system

$$H_n z_n = f_1^{(n)}, \quad \text{where} \quad f_1^{(n)} = [\,\|r_0\|_2 \quad 0 \quad \cdots \quad 0]^T \in \mathbb{R}^n. \tag{13}$$

To motivate (13), we remark that, by the first relation in (5), the residual vector $r_n = b - Ax_n$ corresponding to any iterate (11) satisfies

$$r_n = r_0 - AV_n z_n = V_n \left(f_1^{(n)} - H_n z_n\right) - v_{n+1}\rho_{n+1}(z_n)_n. \tag{14}$$

Here, $(z_n)_n$ denotes the last component of the vector z_n. Hence the choice (13) of z_n just eliminates all components along V_n in the representation (14) of r_n. It can be shown that the resulting iterative procedure is mathematically equivalent to the classical biconjugate gradient (BCG) algorithm due to Lanczos [1952], provided that no look-ahead steps are performed in the underlying Lanczos algorithm. The problem with this first approach is that the coefficient matrix H_n in (13) may be singular or ill-conditioned. In fact, this is one reason for the typically erratic convergence behavior of BCG with wild oscillations in the residual norm.

In order to obtain a method with better numerical and theoretical properties than BCG, Freund [1989; 1992a] and Freund and Nachtigal [1991] suggested to choose z_n as the solution of the least-squares problem

$$\left\|\Omega_{n+1}\left(f_1^{(n+1)} - H_n^{(e)} z_n\right)\right\|_2 = \min_{z \in \mathbb{C}^n} \left\|\Omega_{n+1}\left(f_1^{(n+1)} - H_n^{(e)} z\right)\right\|_2. \tag{15}$$

Here, $f_1^{(n+1)} \in \mathbb{R}^{n+1}$ is defined in (13) (with n replaced by $n+1$), and

$$\Omega_{n+1} = \operatorname{diag}(\omega_1, \omega_2, \ldots, \omega_{n+1}), \quad \omega_j > 0, \quad j = 1, 2, \ldots, n+1, \tag{16}$$

is a weight matrix. The default choice is $\omega_j = 1$ for all j, but there are also situations where other weights are useful; see Freund and Szeto [1992]. We remark that $H_n^{(e)}$ is an unreduced upper Hessenberg matrix, and hence, in contrast to H_n, the extended matrix $H_n^{(e)}$ is always guaranteed to have full column rank n. Therefore, the least-squares problem (15) always has a unique solution z_n. The motivation for this choice of z_n is as follows. Using (5) and (12), we can rewrite the formula (14) for the residual vector of any iterate (11) as follows:

$$r_n = V_{n+1}\Omega_{n+1}^{-1}\left(\Omega_{n+1}\left(f_1^{(n+1)} - H_n^{(e)} z_n\right)\right). \tag{17}$$

Hence, in view of (15), the iterate x_n is characterized by a minimization of the second factor in the representation (17) of its residual vector r_n. This is called the *quasi-minimal residual* (QMR) property, and the resulting iterative scheme for solving linear systems (9) is the QMR method. We refer the reader to Freund and Nachtigal [1991; 1993; 1994] for a detailed description of two different implementations of the QMR method. Theoretical properties of the QMR method, such as error bounds, are derived in Freund and Nachtigal [1991] and Freund [1992b; 1992c].

2.4 Padé Approximation of Transfer Functions

Let $A \in \mathbb{C}^{N\times N}$ and $b,\, c \in \mathbb{C}^N$ with $b,\, c \neq 0$ be given, and let R be the rational function

$$R(\sigma) \equiv c^T \left(I - \sigma A\right)^{-1} b \equiv \sum_{j=0}^{\infty} c^T A^j b\, \sigma^j. \tag{18}$$

Functions of the type (18) arise as transfer functions, for instance, in circuit simulation where R describes the Laplace-domain impulse response of electronic circuits; see, e.g., Feldmann and Freund [1994]. Often, N is so large that the evaluation of R itself becomes prohibitive, and instead, one approximates R by a Padé approximant R_n for some $n \ll N$. Here, R_n is a rational function of type $(n-1, n)$ (i.e., its numerator degree is at most $n-1$, and its denominator degree is at most n) such that its Taylor expansion about $\sigma = 0$ agrees with the Taylor expansion of R in at least the first $2n$ terms, *i.e.*, $R_n(\sigma) = R(\sigma) + \mathcal{O}(\sigma^{2n})$. In general, the series (18) converges only for small $|\sigma|$, and this may suggest that R_n is a useful approximation to R only for small σ. However, due to the fact that R itself is a rational function, this is not the case in general; note that $R_N = R$. Indeed, typically, R_n is a very good approximation to R over a large range of σ's, even for $n \ll N$. For example, Figure 6 in Section 6 below shows an almost perfect match of $|R_n(\sigma)|$ and $|R(\sigma)|$ for all $\sigma = 2\pi i\omega$ with $0 \leq \omega \leq 5 \times 10^9$.

It is well known [Gragg 1974; Gragg and Lindquist 1983] that the Padé approximant R_n can be obtained from the Lanczos process. Indeed, let H_n be the block-tridiagonal matrix (7) that is generated by running n steps of the look-ahead Lanczos Algorithm, with starting vectors $v_1 = b/\|b\|_2$ and $w_1 = c/\|c\|_2$, and set $D_n = W_n^T V_n$, with V_n and W_n the matrices defined in (6). Using (5), one readily verifies that

$$c^T A^j b = \frac{c^T b}{e_1^T D_n e_1}\, e_1^T D_n H_n^j e_1, \quad j = 0, 1, \ldots, 2n-1,$$

where e_1 is the first unit vector in $\mathbb{R}^n$, and together with (18), it follows that

$$R_n(\sigma) \equiv \frac{c^T b}{e_1^T D_n e_1}\, e_1^T D_n \left(I - \sigma H_n\right)^{-1} e_1 \equiv \sum_{j=0}^{2n-1} c^T A^j b\, \sigma^j + \mathcal{O}(\sigma^{2n})$$

is the desired Padé approximant. The resulting PVL algorithm for computing **P**adé approximants **V**ia the look-ahead **L**anczos process, and the application of PVL in circuit simulation is described in detail in Feldmann and Freund [1994].

3. LOOK-AHEAD LANCZOS ALGORITHMS

We now give brief descriptions of the three-term and coupled two-term look-ahead Lanczos algorithms. Full details can be found in the corresponding references, and in fact this paper assumes a degree of familiarity with the algorithms and related terms. Whenever necessary, we will use the notation from the papers [Freund et al. 1993; Freund and Nachtigal 1994] in describing the algorithms.

3.1 The Three-Term Look-Ahead Lanczos Algorithm

Like the classical algorithm, the look-ahead Lanczos algorithm [Freund et al. 1993] generates two sequences of vectors

$$\{v_j\}_{j=1}^n \quad \text{and} \quad \{w_j\}_{j=1}^n, \quad n = 1, 2, \ldots, \tag{19}$$

such that (2) is fulfilled. In addition, they satisfy the biorthogonality relation

$$W_n^T V_n = D_n, \tag{20}$$

where D_n is a block-diagonal matrix whose structure is discussed below, and V_n and W_n are the matrices defined in (6). This means that some of the Lanczos vectors do in fact satisfy the full biorthogonality (3). These vectors are called *regular*, and they form subsequences $\{v_{n_j}\}_{j=1}^{l}$ and $\{w_{n_j}\}_{j=1}^{l}$ of (19), where

$$1 =: n_1 < n_2 < \cdots < n_l \leq n < n_{l+1}, \quad l = l(n). \tag{21}$$

All vectors that are not regular are called *inner*. The regular vectors are used to partition $\{v_j\}_{j=1}^{n}$ and $\{w_j\}_{j=1}^{n}$ into blocks. By convention, one defines blocks $V^{(j)}$, of size $N \times h_j$, containing the regular vector v_{n_j} and all inner vectors—if any—between v_{n_j} and $v_{n_{j+1}}$:

$$V^{(j)} = [\, v_{n_j} \quad v_{n_j+1} \quad \cdots \quad].$$

A look-ahead step is then defined in terms of building such a block. Hence the integer l in (21) is just the number of look-ahead steps taken during the first n steps of the Lanczos algorithm. Moreover, h_j is called the length of the jth look-ahead step. The structure of the sequence $\{w_j\}_{j=1}^{n}$ parallels that of the sequence $\{v_j\}_{j=1}^{n}$, so that V_n and W_n can be written as

$$V_n = [\, V^{(1)} \quad V^{(2)} \quad \cdots \quad V^{(l)} \,] \quad \text{and} \quad W_n = [\, W^{(1)} \quad W^{(2)} \quad \cdots \quad W^{(l)} \,],$$

and D_n in (20) is given by

$$D_n = \operatorname{diag}(D^{(1)}, D^{(2)}, \ldots, D^{(l)}), \quad D^{(j)} = (W^{(j)})^T V^{(j)}, \quad j = 1, 2, \ldots, l.$$

The choice of whether to build a regular or an inner vector is determined at each step, based on the look-ahead strategy. We describe our look-ahead strategy in Section 3.2.

The main point of the three-term look-ahead Lanczos process is that vectors satisfying (2) and (20) can be constructed by means of short block three-term recurrences. In fact, if v_{n+1} and w_{n+1} are constructed as regular vectors, then these recurrences are as follows:

$$\begin{aligned} v_{n+1}\rho_{n+1} &= Av_n - V^{(l)} \left(D^{(l)}\right)^{-1} \left(W^{(l)}\right)^T Av_n - V^{(l-1)} \left(D^{(l-1)}\right)^{-1} \left(W^{(l-1)}\right)^T Av_n, \\ w_{n+1}\eta_{n+1} &= A^T w_n - W^{(l)} \left(D^{(l)}\right)^{-T} \left(V^{(l)}\right)^T A^T w_n - W^{(l-1)} \left(D^{(l-1)}\right)^{-T} \left(V^{(l-1)}\right)^T A^T w_n. \end{aligned} \tag{22}$$

If v_{n+1} and w_{n+1} are constructed as inner vectors, then they are computed as follows:

$$\begin{aligned} v_{n+1}\rho_{n+1} &= Av_n - v_n h_{nn} - V^{(l-1)} \left(D^{(l-1)}\right)^{-1} \left(W^{(l-1)}\right)^T Av_n, \\ w_{n+1}\eta_{n+1} &= A^T w_n - w_n h_{nn} - W^{(l-1)} \left(D^{(l-1)}\right)^{-T} \left(V^{(l-1)}\right)^T A^T w_n. \end{aligned} \tag{23}$$

Here, ρ_{n+1} and η_{n+1} in (22) and (23) are positive scalars chosen such that the Lanczos vectors are scaled as in (4). Furthermore, the inner recurrence coefficient h_{nn} in (23) is a free parameter; in our code, we use the choice in equation (25) below as the default. Each iteration of the algorithm can thus be described as follows.

Algorithm 1 (Overview of the three-term algorithm with look-ahead)

1) *Decide whether to construct* v_{n+1} *and* w_{n+1} *as regular or inner vectors.*
2) *Compute* v_{n+1} *and* w_{n+1} *as either regular or inner vectors, using* (22) *or* (23)*, respectively.*

Step 1) is the basis of the look-ahead strategy, and it consists of three checks, described in the next section. Since the recurrences used to build v_{n+1} and w_{n+1} are short, the look-ahead algorithm has low work and storage requirements, making it an attractive method for building bases for Krylov subspaces. However, it has been observed that, in finite-precision arithmetic, three-term vector recurrences are less robust than mathematically equivalent coupled two-term recurrences. This was our motivation in proposing in Freund and Nachtigal [1994] a different implementation of the look-ahead Lanczos algorithm, based on coupled two-term recurrences. Next, we briefly review this algorithm.

3.2 The Coupled Two-Term Look-Ahead Lanczos Process

The coupled two-term Lanczos process is an alternate way of generating the Lanczos basis vectors. The discussion that follows will not cover all the details and will not justify all the statements made. For full details, we refer the reader to Freund and Nachtigal [1994]. The algorithm generates, in addition to the Lanczos vectors $\{v_j\}_{j=1}^n$ and $\{w_j\}_{j=1}^n$, a second set of basis vectors, $\{p_j\}_{j=1}^n$ and $\{q_j\}_{j=1}^n$, such that, for $n = 1, 2, \ldots,$

$$\operatorname{span}\{p_1, p_2, \ldots, p_n\} = K_n(v_1, A),$$
$$\operatorname{span}\{q_1, q_2, \ldots, q_n\} = K_n(w_1, A^T).$$

For simplicity, we will sometimes refer to the Lanczos vectors $\{v_j\}_{j=1}^n$ and $\{w_j\}_{j=1}^n$ as the VW sequence, and to the auxiliary vectors $\{p_j\}_{j=1}^n$ and $\{q_j\}_{j=1}^n$ as the PQ sequence. The four sets of basis vectors are generated using coupled two-term recurrences of the form:

$$\begin{aligned} V_n &= P_n U_n, & A P_n &= V_{n+1} L_n, \\ W_n &= Q_n \Gamma_n^{-1} U_n \Gamma_n, & A^T Q_n &= W_{n+1} \Gamma_{n+1}^{-1} L_n \Gamma_n, \end{aligned}$$

Here,

$$P_n = [\, p_1 \quad p_2 \quad \cdots \quad p_n \,] \quad \text{and} \quad Q_n = [\, q_1 \quad q_2 \quad \cdots \quad q_n \,],$$

while U_n is an upper triangular matrix and L_n is an upper Hessenberg matrix, given by

$$U_n = \begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \cdots & 0 & 1 \end{bmatrix} \quad \text{and} \quad L_n = \begin{bmatrix} l_{11} & l_{12} & \cdots & l_{1n} \\ \rho_2 & l_{22} & & \vdots \\ 0 & \rho_3 & \ddots & \vdots \\ \vdots & \ddots & \ddots & l_{nn} \\ 0 & \cdots & 0 & \rho_{n+1} \end{bmatrix},$$

and Γ_n is the diagonal matrix from (8). As was shown in Freund and Nachtigal [1994], the matrices L_n and U_n define a factorization of the extended matrix $H_n^{(e)}$ generated by the three-term look-ahead Lanczos algorithm,

$$H_n^{(e)} = L_n U_n.$$

In addition, it is possible to reduce L_n and U_n to block-bidiagonal matrices, by constructing the basis vectors p_n and q_n so as to be block A-biorthogonal. Here, similar to the VW sequence, the vectors p_n and q_n are also constructed using look-ahead techniques. For example, we again have blocks $P^{(j)}$,

$$P^{(j)} = [\, p_{m_j} \quad p_{m_j+1} \quad \cdots \quad],$$

where p_{m_j} is called *regular*, the other vectors in the block are called *inner*, and the indices m_j satisfy

$$1 =: m_1 < m_2 < \cdots < m_k < n \leq m_{k+1}, \quad k = k(n).$$

The regular vectors p_{m_j} satisfy the A-biorthogonality condition

$$q_i^T A p_{m_j} = 0 \quad \text{for all} \quad i = 1, 2, \ldots, m_j - 1,$$

while the inner vectors satisfy only a relaxed version of this condition. Once again, the structure of Q_n parallels that of P_n, and the A-biorthogonality of the two sets of basis vectors can be written as:

$$Q_n^T A P_n = E_n = \mathrm{diag}(E^{(1)}, E^{(2)}, \ldots, E^{(k)}), \quad E^{(j)} = (Q^{(j)})^T A P^{(j)}.$$

Before we describe the look-ahead strategy, let us briefly discuss an outline of the algorithm. At each iteration, the process consists of the following four basic steps.

Algorithm 2 (Overview of the coupled two-term algorithm with look-ahead)

1) *Decide whether to construct p_n and q_n as regular or inner vectors.*
2) *Compute p_n and q_n as either regular or inner vectors.*
3) *Decide whether to construct v_{n+1} and w_{n+1} as regular or inner vectors.*
4) *Compute v_{n+1} and w_{n+1} as either regular or inner vectors.*

Steps 1) and 3) are the basis of the look-ahead strategy, and they each consist of three checks. Recall that the Lanczos vectors v_{n+1} and w_{n+1} can be obtained from the previous Lanczos vectors by a block three-term recurrence. Similarly, it is possible to show that the vectors p_n and q_n also have a block three-term recurrence, of the form

$$AP_{n-1} = P_n G_{n-1} \quad \text{and} \quad A^T Q_{n-1} = Q_n \Gamma_n^{-1} G_{n-1} \Gamma_{n-1},$$

where

$$G_{n-1} = U_n L_{n-1}.$$

The look-ahead strategy for the two pairs of sequences is then similar, and was first proposed in Freund et al. [1993]. In Step 1), the algorithm checks whether:

$$\sigma_{\min}(E^{(k)}) \geq eps,$$

$$n(A)\|p_n\|_2 \geq \sum_{i=m_{k-1}}^{n-1} \left|(U_n L_{n-1})_{i,n-1}\right| \|p_i\|_2,$$

$$n(A)\|q_n\|_2 \geq \sum_{i=m_{k-1}}^{n-1} \frac{\gamma_{n-1}}{\gamma_i} \left|(U_n L_{n-1})_{i,n-1}\right| \|q_i\|_2.$$

Here, $\sigma_{\min}(M)$ denotes the smallest singular value of matrix M, eps is machine epsilon, and $n(A)$ is an estimate for the norm of A. The vectors p_n and q_n are built as regular vectors only if all three of the above checks are satisfied. Likewise, in Step 3), the algorithm checks whether:

$$\sigma_{\min}(D^{(l)}) \geq eps,$$

$$n(A) \geq \sum_{i=n_{l-1}}^{n} |(L_n U_n)_{in}|,$$

$$n(A) \geq \sum_{i=n_{l-1}}^{n} \frac{\gamma_n}{\gamma_i} |(L_n U_n)_{in}|,$$

and again, the vectors v_{n+1} and w_{n+1} are built as regular vectors only if all three of the checks are passed. The motivation for these checks can be found in Freund et al. [1993] and Freund and Nachtigal [1994]. Here, we will only note that the look-ahead strategy proposed will build regular vectors in preference to inner vectors, and thus it will take as few look-ahead steps as possible.

4. SIMPLIFIED LANCZOS ALGORITHMS

It is well known [Lanczos 1956, p. 176] that the Lanczos process simplifies for Hermitian matrices $A = A^H$ and for complex symmetric matrices $A = A^T$. For $A = A^H$, one chooses $w_1 = \overline{v_1}$ as the complex conjugate of v_1, and then $w_n = \overline{v_n}$ for all n. Therefore, the updates for the left Lanczos vectors w_n can be deleted, and the resulting Hermitian Lanczos algorithm requires only half the work and storage of the general algorithm. Similarly, if $A = A^T$, one chooses $w_1 = v_1$, and then $w_n = v_n$ for all n.

Many important applications lead to matrices A that are neither Hermitian nor complex symmetric, but are P-symmetric or P-Hermitian in the sense that

$$A^T P = PA, \; P = P^T, \quad \text{or} \quad A^H P = PA, \; P = P^H, \tag{24}$$

for some nonsingular matrix P. For example, if A is a Toeplitz matrix, then the first relation in (24) is satisfied with P being the antidiagonal identity matrix.

Freund and Zha [1995] observed that the nonsymmetric Lanczos process also simplifies for P-symmetric and P-Hermitian matrices by choosing initial Lanczos vectors v_1 and w_1 that are coupled by $w_1 = Pv_1/\|Pv_1\|_2$ and $w_1 = \overline{Pv_1}/\|Pv_1\|_2$, respectively. In fact, using (24), one easily verifies that the Lanczos vectors satisfy $w_n = Pv_n/\|Pv_n\|_2$, respectively $w_n = \overline{Pv_n}/\|Pv_n\|_2$, for all n. Therefore, the computation of each left Lanczos vectors w_n can be replaced by essentially a matrix-vector multiplication with P. If the computation of such products is cheap (as is the case for Toeplitz matrices), the resulting simplified Lanczos algorithm requires only half the work and storage of the general algorithm.

The current version of QMRPACK contains the resulting simplified Lanczos algorithms and its QMR counterparts only for the two simplest special cases, $A = A^T$ and $A = A^H$. Codes for other important classes of matrices for which the simplified Lanczos algorithms are practical will be added in the future.

5. QMRPACK

5.1 General Description of the Package

QMRPACK was designed as a package of linear solvers and an eigenvalue solver built around the look-ahead Lanczos algorithms proposed by Freund, Nachtigal, and their collaborators. It is intended to supply several QMR algorithms and an eigenvalue solver as black-box routines that require a minimum of parameters from the user. The package is written in ANSI FORTRAN-77, for portability, and currently includes three main QMR algorithms for the solution of linear systems:

- The original QMR algorithm [Freund and Nachtigal 1991] based on the look-ahead three-term Lanczos process described in [Freund et al. 1993];

- The QMR algorithm based on the look-ahead coupled two-term recurrence Lanczos process [Freund and Nachtigal 1993; 1994];
- The TFQMR algorithm [Freund 1993].

Even though the main emphasis of the package is on the linear systems solvers, QMRPACK also includes a routine for obtaining approximate eigenvalues from the look-ahead three-term Lanczos algorithm. Finally, the package includes simplified no-look-ahead versions of both the three-term and the coupled two-term QMR algorithms, as well as the QMR-from-BCG algorithm proposed by Freund and Szeto [1992]. The codes are available for general matrices, Hermitian and non-Hermitian, in single and double precision, real and complex data types. Whenever available, the look-ahead versions of an algorithm are recommended over their no-look-ahead counterparts.

Together with the algorithms above, which are provided as library routines, the package also includes example driver codes and test matrices, to indicate how the library routines are used and to test the software. The example driver codes also include support for the compressed sparse format, as well as several preconditioners for linear systems: right, left, and two-sided variants of the ILUT preconditioner proposed by Saad [1992], and right-, left-, and two-sided variants of the SSOR preconditioner [Axelsson 1985]. We note in passing that the two-sided SSOR preconditioner is implemented using the Eisenstat trick [Eisenstat 1981], and thus the application of the preconditioned matrix is only marginally more expensive than the application of the unpreconditioned matrix.

5.2 Code-Usage Issues

There are several issues that arise when developing a software package based on iterative methods for the solution of linear systems or for the computation of eigenvalues. One needs to decide how the matrix-vector products are implemented, how the preconditioner is handled for linear systems, and also what convergence criterion will be used to terminate the iterations. We will briefly address each of these issues.

For the matrix-vector products, QMRPACK uses *reverse communication*[1]. This is a mechanism whereby the solver routines will return to the caller whenever a matrix-vector multiplication is needed, expecting the caller to perform the operation and call back the solver. With the exception of TFQMR, all the other nonsymmetric algorithms in QMRPACK require multiplications both with the matrix A and with its transpose A^T, so a flag is used to indicate to the caller which multiplication is desired. The advantage of using reverse communication is that all the details about the matrix, such as its data structure and the particulars of the implementation of the matrix-vector products, are external to the solvers and left entirely up to the user. This is considerably more flexible than hard-coding routine names and calling sequences into the solvers and expecting the user to fit their matrices in the parameter sequences provided. The following snippet of pseudo-code exemplifies the typical setup and usage of a solver routine in QMRPACK:

```
    Initial setup
 10 CALL SOLVER (...,X,B,FLAG)
    IF (FLAG.EQ.1) THEN
       Multiply B by A, store in X
       GO TO 10
    ELSE IF (FLAG.EQ.2) THEN
       Multiply B by the transpose of A, store in X
```

[1] Reverse communication is a mechanism for obtaining from the user specific information required within a subroutine by returning to the calling program. The idea of reverse communication goes back to Lawson and Krogh [1969]; see also Sections 3.8 and 4.4 in [Gill et al. 1979].

```
        GO TO 10
      END IF
   20 Check for convergence or errors
```

In addition, in the package, the vectors X and B are passed only by their column indices in a larger work array, and X must not be destroyed by the caller, but the general setup is the same. The example drivers demonstrate how to set up the calls to the various routines and how to handle the reverse-communication protocol.

In addition to the matrix-vector products, the linear systems solvers also raise issues about the implementation of preconditioning and of the convergence criteria. The solvers in QMRPACK are set up to solve an already preconditioned system: the application of the preconditioner rests with the user, inside the matrix-vector routines, and the codes solve the already preconditioned system (in fact, the codes further assume that the starting guess is always zero). Thus, instead of solving $Ax = b$, with starting guess x_0, the codes are set up to solve the preconditioned system $By = c$, with starting guess $y_0 = 0$, where $M = M_1 M_2$ is the preconditioner, $B = M_1^{-1} A M_2^{-1}$, $y = M_2(x - x_0)$, and $c = M_1^{-1}(b - Ax_0)$. To recover the iterates x_n for the original system, one sets

$$x_n = x_0 + M_2^{-1} y_n.$$

This means, however, that all the quantities generated by the codes belong to the preconditioned system $By = c$, and not the original system. In particular, the convergence criterion will use the preconditioned residual (unless right preconditioning is used) to determine convergence. Even though it would be possible to incorporate explicit calls to preconditioner routines in all the algorithms in the package, we felt that our choice allows more flexibility. In particular, cases where the preconditioner is applied in a way that eliminates the matrix A completely, such as the Eisenstat trick for the SSOR two-sided preconditioner, are handled quite naturally by the current setup. The other issue for linear systems solvers is the choice of the convergence criterion. The criterion used in QMRPACK is the relative residual norm $\|r_n\|_2 / \|r_0\|_2$. This choice is hard-coded in the algorithms, and there is no facility for the user to change it without additional coding. There are many other possible choices. Indeed, the variety is such that we could not find an elegant way to accommodate all the options encountered, without entering into details about data representation internal to the solvers.

The routines in QMRPACK have several control parameters, allowing the user complete control over such aspects of the algorithm as the generation of the auxiliary starting vectors, the choice between the computation of the true residual norm at every step or the use of the residual norm upper bounds that are available in the QMR algorithms, output history, and so forth. Also, all the vector operations in QMRPACK are implemented via calls to BLAS routines, which means that the package will benefit from optimized versions of these routines where available.

Finally, all the solver routines in QMRPACK have names of the form form `PMALG`, where:

P denotes the precision and data type:

C	single precision complex
D	double precision real
S	single precision real
Z	double precision complex

M denotes the matrix type:

S	symmetric
U	unsymmetric

ALG denotes the algorithm:

CPL	QMR based on coupled two-term look-ahead Lanczos
CPX	QMR based on coupled two-term no-look-ahead Lanczos
LAL	eigenvalue solver based on three-term look-ahead Lanczos
QBG	QMR-from-BCG
QMR	QMR based on three-term look-ahead Lanczos
QMX	QMR based on three-term no-look-ahead Lanczos
TFX	TFQMR

For example, ZUCPL denotes the double precision complex codes for the QMR method based on coupled two-term Lanczos with look-ahead, for unsymmetric matrices.

5.3 Routine DUCPL

After these general comments about the routines in QMRPACK, let us turn to a more detailed discussion of two routines from the package, DUCPL and DULAL. For each routine, we will discuss its parameters, support routines, and we will give an example usage based on the test problems distributed with QMRPACK. This section discusses the linear system solver DUCPL, while the next section will treat the eigenvalue solver DULAL.

DUCPL is the double precision real nonsymmetric QMR algorithm based on the coupled two-term look-ahead Lanczos variant proposed in [Freund and Nachtigal 1994]. It is both the most robust and the most complex linear systems solver in the package, hence its selection for discussion. Its calling sequence is:

```
CALL DUCPL (NDIM,NLEN,NLIM,MAXPQ,MAXVW,M,MVEC,NORMS,DWK,IDX,IWK,VECS,TOL,INFO)
```

5.3.1 Calling Parameters

The calling parameters and their dimensions are:

NDIM – INTEGER, leading dimension of the array VECS.

NLEN – INTEGER, row and column dimension of the matrix A.

NLIM – INTEGER, the maximum allowed number of look-ahead Lanczos steps.

MAXPQ – INTEGER, the maximum allowed block size for the PQ sequence.

MAXVW – INTEGER, the maximum allowed block size for the VW sequence.

M – INTEGER, leading dimension of the arrays IWK and DWK.

MVEC – INTEGER, the maximum number of consecutive vectors that can be built in any sequence, determines the size of the main vector array VECS.

NORMS(2) – DOUBLE PRECISION, estimates for the norm of A.

DWK(M,8*M+18) – DOUBLE PRECISION, work array, stores internal scalar quantities.

IDX(6,NLIM+2) – INTEGER, work array, stores internal indices.

IWK(M,13) – INTEGER, work array, stores internal indices.

VECS(NDIM,5*MVEC+3) – DOUBLE PRECISION, main vector array.

TOL – DOUBLE PRECISION, the convergence level of the scaled QMR residual norm.

INFO(4) – DOUBLE PRECISION, information passing array:

INFO(1) – general status flag.
INFO(2) – used for reverse-communication info.
INFO(3) – used for reverse-communication info.
INFO(4) – used for reverse-communication info.

5.3.2 Input Parameters

The following are input parameters and, where appropriate, their valid range of values:

NDIM: must be $\texttt{NDIM} \geq 1$.

NLEN: must be $1 \leq \texttt{NLEN} \leq \texttt{NDIM}$.

NLIM: must be $\texttt{NLIM} \geq 1$.

MAXPQ: must be $\texttt{MAXPQ} \geq 1$.

MAXVW: must be $\texttt{MAXVW} \geq 1$.

Look-ahead can be turned off in the VW or PQ sequence by setting $\texttt{MAXVW} = 1$ or $\texttt{MAXPQ} = 1$, respectively. While in general there do not seem to be any good reasons to set MAXPQ and MAXVW to different values, there is one case where this would make sense: if the matrix A is Hermitian indefinite, and if the algorithm is started with $v_1 = w_1$, then the VW sequence cannot have any breakdowns, while the PQ sequence can. In this case, MAXVW can be set to 1, while MAXPQ should be set to allow look-ahead steps in the PQ sequence.

M: must be $\texttt{M} \geq \texttt{MAXPQ} + \texttt{MAXVW} + 2$.

MVEC: must be $\texttt{MVEC} \geq \max(\texttt{MAXPQ}, \texttt{MAXVW}) + 1$.

The actual memory requirements for any given run of the algorithm will depend on MAXPQ, MAXVW, and the configuration of the blocks built in the two sequences. For a worst case configuration of the breakdowns, where the blocks in the two sequences are as large as MAXPQ and MAXVW permit, and overlap in just the "right" way, each sequence will require $\texttt{MAXPQ} + \texttt{MAXVW}$ vectors. Hence, setting $\texttt{MVEC} = \texttt{MAXPQ} + \texttt{MAXVW}$ will ensure that the array VECS is large enough to allow any possible block structure. However, for a typical run, the algorithm might not actually build blocks as large as MAXPQ and MAXVW might allow, or it might not encounter the worst case configuration of the breakdowns. Thus, it might be possible to set MVEC to a value less that $\texttt{MAXPQ} + \texttt{MAXVW}$ and the algorithm might still be able to run within the allocated space. If the algorithm aborts due to insufficient memory, an error code is returned in INFO(1) (see below). For many commonplace matrices, it seems that setting $\texttt{MAXPQ} = \texttt{MAXVW}$ between 3 and 5, and $\texttt{MVEC} = \max(\texttt{MAXPQ}, \texttt{MAXVW}) + 2$ seems to give good results.

NORMS: These estimates are used in the look-ahead strategy to determine whether to build regular or inner vectors. NORMS(1) is the estimate used for the PQ sequence, while NORMS(2) is the estimate used for the VW sequence. When previous estimates are not available, they should be set to 1. Setting one of the estimates to a very large value would effectively turn off look-ahead for the corresponding sequence.

The algorithm will dynamically update the norm estimates, by using two mechanisms: block closures, and norm estimating. Any time a block larger than `MAXPQ` or `MAXVW` is about to be built, the algorithm will reset the norm estimate to the smallest value that will ensure the block can close upon rebuilding. This value will also be multiplied by an internal factor `ADJUST`, which is set to 10.0 by default, to provide some "slack". Setting $\texttt{ADJUST} = 1.0$ will result in many blocks being rebuilt—which is wasted effort—before a suitable norm estimate is found, hence the default of 10.0. The second mechanism is to compute the norms of the vectors Ap_n and $A^T q_n$, and obtain estimates for the norm of A from these (the norms of p_n and q_n are known). This is done at the first few steps of the algorithm, and involves two extra norm computations per step. The counter that governs the number of steps is called `NUMCHK` in the code, and it is set to 25 by default. Typically, the estimates found are useful only if the initial estimate was low.

The updating mechanism will raise the level of the two estimates whenever either is updated. For example, suppose that $\texttt{NORMS(1)} = 10.0$, for the PQ sequence, and $\texttt{NORMS(2)} = 1000.0$, for the VW sequence. Suppose that a block of size `MAXPQ` is built in the PQ sequence, which means the algorithm will force a block closure, and suppose that the smallest value that would ensure that the block would close when rebuilt is 150.0. Then the algorithm will set $\texttt{NORMS(1)} = \texttt{ADJUST} * 150.0$ and $\texttt{NORMS(2)} = \max(\texttt{ADJUST} * 150.0, 1000.0)$. For the default value of `ADJUST`, this would give $\texttt{NORMS(1)} = \texttt{NORMS(2)} = 1500.0$.

`VECS`: This array stores all the long vectors. Its structure is:

Columns	Vectors
1	the current QMR iterate x_n
2	the right-hand side b
3 and 4	temporary vectors
5 through $\texttt{MVEC} + 4$	the right Lanczos vectors v_n
$\texttt{MVEC} + 5$ through $2 * \texttt{MVEC} + 4$	the left Lanczos vectors w_n
$2 * \texttt{MVEC} + 5$ through $3 * \texttt{MVEC} + 4$	the vectors p_n
$3 * \texttt{MVEC} + 5$ through $4 * \texttt{MVEC} + 4$	the vectors q_n
$4 * \texttt{MVEC} + 5$ through $5 * \texttt{MVEC} + 3$	the QMR search directions s_n

The vectors v_n, w_n, p_n, q_n, and s_n are stored in a circular fashion, meaning that the later vectors overwrite the earlier ones. For example, the vector v_i is stored in column $5 + ((i-1) \bmod \texttt{MVEC})$. The right-hand side vector b must be specified in column 2. In addition, the starting left Lanczos vector w_1 can be specified in column 3, in which case the flag should be set in `INFO(1)` below. If the second starting vector is not specified, then the routine will generate one using a built-in random number generator.

`TOL`: When the relative residual norm $\|r_n\|_2 / \|r_0\|_2$ drops below `TOL`, the algorithm is deemed to have converged. If a non-positive value is given for `TOL`, then the routine will use $\sqrt{\epsilon}$, where ϵ is machine epsilon.

`INFO`: Only `INFO(1)` is an input parameter. It specifies the output units, and it controls the generation of the convergence history. If it is represented as the decimal number `tuyyzz`, then:

- **`t`:** if not 0, then the caller has provided the second Lanczos starting vector w_1. Otherwise, the routine will generate its own starting vector, using a built-in random number generator.

`u`: if not 0, then the true residual norm is computed at every step by computing the iterate x_n, then its residual $r_n = b - Ax_n$, and finally the norm. Otherwise, the algorithm will use the residual norm upper bound, which is available at no extra cost, until it gets within an order of magnitude of the convergence tolerance `TOL`, and only then does it switch to computing the actual residual norm.

`yy`: if not 0, then it denotes the unit number for the output unit to which convergence history data is sent; the unit is assumed to already be open for output. The data is sent as three number on a line: the iteration number, the residual norm upper bound, and the norm of the last computed residual.

`zz`: if not 0, the it denotes the unit number for the output unit to which miscellaneous verbose trace messages are sent; the unit is assumed to already be open for output.

For example, `INFO(1)` $= 0$ means to generate w_1 internally, to use the residual norm upper bound, and not to generate any output, while `INFO(1)` $= 551314$ means to use the starting vector w_1 provided in column 3 of `VECS`, to always compute the true residual norm, to send trace messages to unit 14, and to send convergence data to unit 13.

5.3.3 Output Parameters

The following are output parameters and their meaning:

`NLIM`: returns the number of QMR steps taken.

`MAXPQ`: returns the size of the largest block built in the PQ sequence.

`MAXVW`: returns the size of the largest block built in the VW sequence.

`NORMS`: returns the last norm estimates used. The values returned also provide some feedback on the look-ahead blocks constructed. For example, if the algorithm was forced to close a block and build a regular vector at a point where an inner vector should have been built—for example, if the largest block allowed is not large enough—then a large jump in the norm estimates will appear at that step, and the value returned in `NORMS` will be very large.

`VECS`: returns the last computed QMR iterate in column 1.

`TOL`: returns the value used, which is different from the input value only if the input value was non-positive.

`INFO`: returns reverse-communication parameters during execution, and an error code upon termination.

`INFO(1)` returns an error code, with the following possible values:

`0` normal termination, algorithm converged

`1` invalid reverse-communication call

`2` invalid inputs, at least one of the input parameters has a value out of its valid range

`4` algorithm reached the limit of look-ahead Lanczos step without converging

`8` the last block in one of the sequences cannot be closed, nor is it possible to force closure

`16` an A-invariant subspace has been found

`32` an A^T-invariant subspace has been found

`48` both invariant subspaces have been found

`64` insufficient memory allocated in `VECS`

128 could not rebuild an regular vector

If INFO(1) $= 8$, then all inner vectors were built because the corresponding diagonal block of the biorthogonality matrix was singular, and a block of size larger than the limits imposed would need to be built to proceed. If an A-invariant subspace is found, then in theory the algorithm should return the exact solution, and thus terminate normally. Thus, INFO(1) $=$ 16 and INFO(1) $= 48$ indicate that probably the convergence tolerance is too low, or that the check for invariant subspaces is too loose. If INFO(1) $= 64$, then MVEC was less than MAXPQ + MAXVW, and increasing it should allow the algorithm to proceed. If INFO(1) $= 128$, then the algorithm was unable to rebuild a regular vector; this should not happen, as it would indicate an internal error.

INFO(2) – INFO(4) are used for reverse communication. The caller sets INFO(2) to 0 when the routine is called for the first time, to signal the initialization phase. Thereafter, INFO(2) must not be modified. The routine returns to caller as part of the reverse-communication process, with INFO(2) $= 1$ if a multiplication with A is needed, or with INFO(2) $= 2$ if a multiplication with A^T is needed. The multiplication takes as its argument column INFO(3) of VECS, and must return the result in column INFO(4) of VECS, while preserving its argument. The following pseudo-code illustrates this procedure:

```
   INFO(2) = 0
10 CALL DUCPL (NDIM,...,INFO)
   REVCOM = INFO(2)
   COLX   = INFO(3)
   COLB   = INFO(4)
   IF (REVCOM.EQ.1) THEN
      Multiply VECS(1,COLX) by A
      Store result in VECS(1,COLB)
      GO TO 10
   ELSE IF (REVCOM.EQ.2) THEN
      Multiply VECS(1,COLX) by the transpose of A
      Store result in VECS(1,COLB)
      GO TO 10
   END IF
20 Check INFO(1) for convergence or errors
```

5.3.4 Support Routines

The routine DUCPL has several support functions that can be modified by the user. These routines are: DUCPLL, DUCPLU, and DUCPLO. DUCPLL and DUCPLU are used to generate the inner recurrence coefficients for the VW and PQ sequences, respectively, while DUCPLO is used to generate the entries ω_j of the weight matrix (16) for the QMR least-squares problem (15). By default, DUCPLL returns

$$l_{ij} = \begin{cases} n(A) & i = j, \\ 0 & \text{otherwise,} \end{cases}$$

where $n(A)$ is the internally computed estimate for the norm of A, DUCPLU returns $u_{ij} = 0$ for all i and j, and DUCPLO returns $\omega_j = 1$ for all j. One problem with the default values is that DUCPLL assumes that the largest eigenvalue of A is positive, and as a result sets $l_{ii} = +n(A)$, when in fact, if the largest eigenvalue of A is negative, a better choice is $l_{ii} = -n(A)$.

5.4 Routine DULAL

DULAL is the double precision real nonsymmetric eigenvalue solver based on the three-term look-ahead Lanczos variant proposed in [Freund et al. 1993]. Its calling sequence is:

```
CALL DULAL (NDIM,NLEN,NLIM,MAXN,MAXVW,M,NORM,DWK,IDX,IWK,HWK,VECS,INFO)
```

5.4.1 Calling Parameters

The calling parameters and their dimensions are:

- NDIM – INTEGER, leading dimension of the array VECS.
- NLEN – INTEGER, row and column dimension of the matrix A.
- NLIM – INTEGER, the maximum allowed number of look-ahead Lanczos steps.
- MAXN – INTEGER, leading dimension of the array HWK.
- MAXVW – INTEGER, the maximum allowed block size for the VW sequence.
- M – INTEGER, leading dimension of the arrays IWK and DWK.
- NORM – DOUBLE PRECISION, estimate for the norm of A.
- DWK(M,5*M+7) – DOUBLE PRECISION, work array, stores internal scalar quantities.
- IDX(3,NLIM+2) – INTEGER, work array, stores internal indices.
- IWK(M,4) – INTEGER, work array, stores internal indices.
- HWK(MAXN,2*MAXN+8) – DOUBLE PRECISION, work array, stores the block-tridiagonal Lanczos matrix, as well as temporary matrices used during the computation of eigenvalue estimates.
- VECS(NDIM,2*MAXVW+6) – DOUBLE PRECISION, main vector array.
- INFO(4) – DOUBLE PRECISION, information passing array:
  - INFO(1) – general status flag.
  - INFO(2) – used for reverse-communication info.
  - INFO(3) – used for reverse-communication info.
  - INFO(4) – used for reverse-communication info.

5.4.2 Input Parameters

The following are input parameters and, where appropriate, their valid range of values:

- NDIM: must be $\texttt{NDIM} \geq 1$.
- NLEN: must be $1 \leq \texttt{NLEN} \leq \texttt{NDIM}$.
- NLIM: must be $\texttt{NLIM} \geq 1$.
- MAXN: must be $\texttt{MAXN} \geq \texttt{NLIM} + 2$.
- MAXVW: must be $\texttt{MAXVW} \geq 1$.

  Look-ahead can be turned off in the VW sequence by setting $\texttt{MAXVW} = 1$.
- M: must be $\texttt{M} \geq 2 * \texttt{MAXVW} + 2$.

NORM: This estimate is used in the look-ahead strategy to determine whether to build regular or inner vectors. When a previous estimate is not available, it should be set to 1. Setting the estimate to a very large value would effectively turn off look-ahead.

The algorithm will dynamically update the norm estimate. The mechanisms used are the same as for the coupled two-term recurrence look-ahead Lanczos algorithm, described in the previous section.

VECS: This array stores all the long vectors. Its structure is:

Columns	Vectors
1 and 2	temporary vectors
3 through MAXVW + 4	the right Lanczos vectors v_n
MAXVW + 5 through 2 * MAXVW + 4	the left Lanczos vectors w_n

The vectors v_n and w_n are stored in a circular fashion, similar to the storage scheme used in the routine DUCPL.

The starting vectors v_1 and w_1 can be specified in column 1 and column 2 of VECS, respectively, in which case the flag should be set in INFO(1) below. If any of the starting Lanczos vectors is not specified, then the routine will generate one using a built-in random number generator.

INFO: Only INFO(1) is an input parameter. It specifies the output units and it controls the generation of the convergence history. If it is represented as the decimal number tuyyzz, then:

t: if not 0, then the caller has provided the right Lanczos starting vector w_1. Otherwise, the routine will generate its own starting vector, using a built-in random number generator.

u: if not 0, then the caller has provided the left Lanczos starting vector v_1. Otherwise, the routine will generate its own starting vector, using a built-in random number generator.

yy: if not 0, then it denotes the unit number for the output unit to which eigenvalue output data is sent; the unit is assumed to already be open for output.

zz: if not 0, the it denotes the unit number for the output unit to which miscellaneous verbose trace messages are sent; the unit is assumed to already be open for output.

For example, INFO(1) = 0 means to generate both v_1 and w_1 internally, and not to generate any output, while INFO(1) = 551314 means to use the starting vectors v_1 and w_1 provided in columns 1 and 2 of VECS, to send trace messages to unit 14, and to send eigenvalue data to unit 13.

5.4.3 Output Parameters

The following are output parameters and their meaning:

NLIM: returns the number of look-ahead Lanczos steps completed.

MAXVW: returns the size of the largest block built in the VW sequence.

NORM: returns the last norm estimate used. The values returned also provide some feedback on the look-ahead blocks constructed. For example, if the algorithm was forced to close a block and build a regular vector at a point where an inner vector should have been built—for example, if the largest block allowed is not large enough—then a large jump in the norm estimate will appear at that step, and the value returned in NORM will be very large.

INFO: returns reverse-communication parameters during execution, and an error code upon termination.

INFO(1) returns an error code, with the following possible values:

- 0 normal termination, algorithm converged
- 1 invalid reverse-communication call
- 2 invalid inputs, at least one of the input parameters has a value out of its valid range
- 8 the last block in one of the sequences cannot be closed, nor is it possible to force closure
- 16 an A-invariant subspace has been found
- 32 an A^T-invariant subspace has been found
- 48 both invariant subspaces have been found

If INFO(1) $= 8$, then all inner vectors were built because the corresponding diagonal block of the biorthogonality matrix was singular, and a block of size larger than the limits imposed would need to be built to proceed.

INFO(2) – INFO(4) are used for reverse communication. The caller sets INFO(2) to 0 when the routine is called for the first time, to signal the initialization phase. Thereafter, INFO(2) must not be modified. The routine returns to caller as part of the reverse-communication process, with INFO(2) $= 1$ if a multiplication with A is needed, or with INFO(2) $= 2$ if a multiplication with A^T is needed. The multiplication takes as its argument column INFO(3) of VECS, and must return the result in column INFO(4) of VECS, while preserving its argument. The following pseudo-code illustrates this procedure:

```
   INFO(2) = 0
10 CALL DULAL (NDIM,...,INFO)
   REVCOM = INFO(2)
   COLX   = INFO(3)
   COLB   = INFO(4)
   IF (REVCOM.EQ.1) THEN
      Multiply VECS(1,COLX) by A
      Store result in VECS(1,COLB)
      GO TO 10
   ELSE IF (FLAG.EQ.2) THEN
      Multiply VECS(1,COLX) by the transpose of A
      Store result in VECS(1,COLB)
      GO TO 10
   END IF
20 Check INFO(1) for convergence or errors
```

5.4.4 Support Routines

Like the routine DUCPL, DULAL has a support function that can be modified by the user. This routine is DULALH, and it is used to generate the inner recurrence coefficient h_{nn} (see (23)) for the VW sequence. By default, DULALH returns

$$h_{ij} = \begin{cases} n(A) & i = j, \\ 0 & \text{otherwise}, \end{cases} \tag{25}$$

where $n(A)$ is an internally computed estimate for the norm of A. One problem with the default values is that DULALH assumes that the largest eigenvalue of A is positive, and as a result sets $h_{ii} = +n(A)$, when in fact, if the largest eigenvalue of A is negative, a better choice is $h_{ii} = -n(A)$.

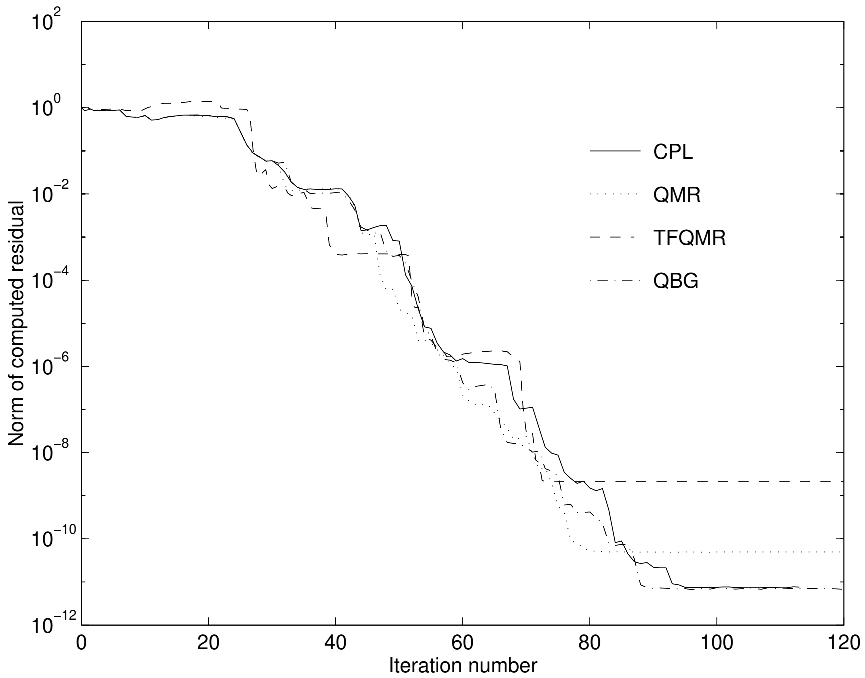

Figure 1: Computed residual norms for Example 1.

6. NUMERICAL EXAMPLES

In this section, we present a few examples of applications where the codes from QMRPACK were used. We restrict ourselves to the solution of linear systems (Examples 1–4) and the approximation of a transfer function (Example 5). For examples of computations with the eigenvalue solver from QMRPACK, we refer the reader to [Freund et al. 1993]. In particular, some of the examples given there clearly demonstrate the need for and benefit of look-ahead in eigenvalue computations with the Lanczos process.

Example 1. Our first example is a real nonsymmetric linear system that arose within a structure computation at the NASA Langley Research Center. The coefficient matrix, NASA1824, is of order $N = 1824$ and it has 39,208 nonzero elements. We preconditioned this linear system, using the ILUT preconditioner from QMRPACK with up to 8 fill-in elements per row and a dropping tolerance of 10^{-2}. We ran all linear systems solvers in QMRPACK, using double precision. This is an example where no look-ahead is necessary, and therefore, CPL and CPX, respectively QMR and QMX, gave identical results. In Figure 1, we show the true residual norms (computed from the actual iterates) for CPL, QMR, TFQMR, and QBG (QMR-from-BCG). All algorithms were run until they reached their accuracy limits, i.e., the true residual norms stagnated. The convergence behavior in Figure 1 is fairly typical in that the CPL residuals usually stagnate at a slightly lower level than the QMR residuals. Moreover, just like other transpose-free Lanczos-based methods, the TFQMR algorithm stagnates with considerably larger residuals than methods like CPL and QMR that involve the transpose.

Example 2. A second example of an application of QMRPACK is the investigation by Nachtigal and Semeraro [1995] into the use of QMR iterations as a replacement for the Beam-Warming factorization [Beam and Warming 1976] in the solution of the Euler equations. The physical test problem used was 2-D flow in a channel with a 10% circular arc on the lower surface, with 80 mesh points streamwise and 50 mesh points across, at Mach number 1.8. The flow field was initialized to the free stream conditions and the Euler

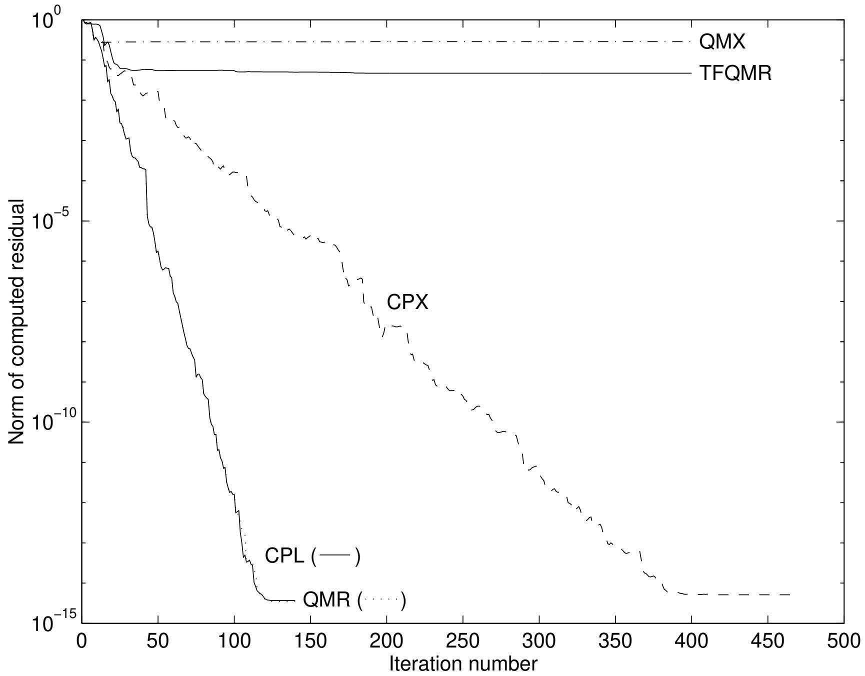

Figure 2: Computed residual norms for Example 2.

solution was iterated in time for 800 time steps, using a time step of 0.01 over the entire mesh to obtain time-accurate solutions. At each time step, one solves a linear system of order 14,976. These systems are fairly easy to solve, even without preconditioning. In fact, CPL (without preconditioning) typically takes only 25 iteration to solve any of these linear systems to the accuracy required for the outer Euler iteration. However, for some of the linear systems near-breakdowns do occur, and then the iteration counts go up when look-ahead is turned off. In Figure 2, we show the true residual norms for CPL, QMR, CPX, QMX, and TFQMR applied to one of the systems for which a near-breakdown was encountered. In this case, the near-breakdown occurred at iteration $n = 11$. Clearly, the look-ahead algorithms CPL and QMR are not affected by this near-breakdown. The no-look-ahead counterpart QMX of QMR, however, stagnates at an unacceptably high residual level. The no-look-ahead counterpart CPX of CPL recovers from the near-breakdown, but it takes a totally different path than CPL. In particular, due to the near-breakdown, CPX converges considerably slower than CPL. Finally, TFQMR is also affected by the near-breakdown, and it stagnates at a fairly high residual level. This example clearly demonstrates the benefit and need of look-ahead.

Despite the occasional increase of iteration counts for the TFQMR solves due to near-breakdowns, the use of TFQMR within the Euler solver is quite efficient, compared to the use of the Beam-Warming factorization. In Figure 3, we show the residual of the Euler equations as a function of CPU time for the two approaches. The curve "Beam-Warming" was obtained by solving the linear systems at each time step by means of the Beam-Warming factorization. The curve "TFQMR" was obtained by means of TFQMR as implemented in the QMRPACK routine `DUTFX`, without any preconditioning. The TFQMR algorithm is quite competitive, and in addition, it is more flexible than the Beam-Warming approach, which is restricted to meshes and discretizations that give rise to a block-pentadiagonal matrix (in 2-D).

Example 3. This example was provided to us by V. Venkatakrishnan. It comes from an unstructured

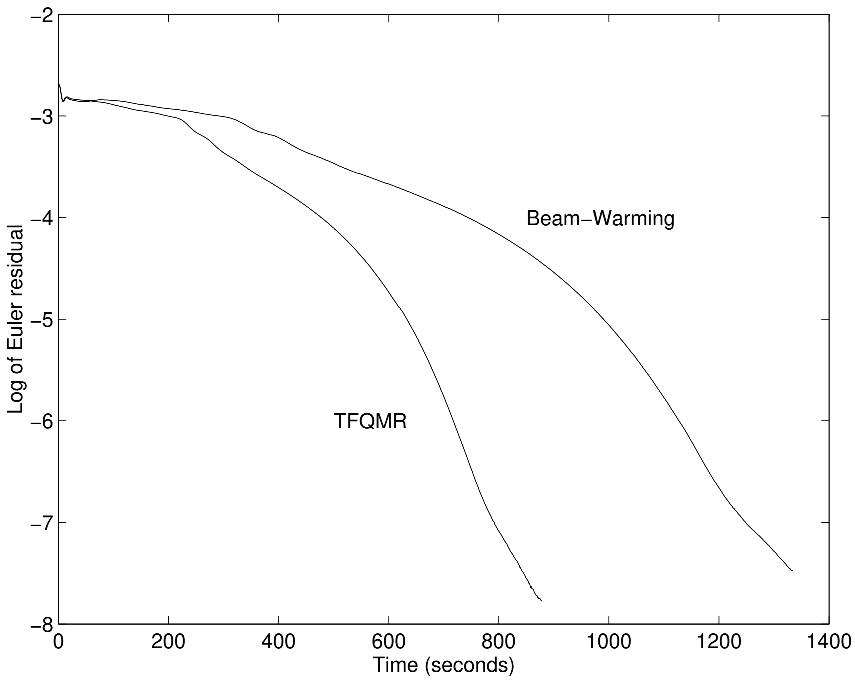

Figure 3: Euler residual norm vs. time for Example 2.

2-D Euler solver, and it corresponds to the system at the beginning of time-stepping; see Venkatakrishnan and Barth [1992]. The linear system is of order $N = 62,424$ with $1,717,792$ nonzero elements, and it was solved using the coupled two-term recurrence QMR routine `DUCPL`. The right-hand side b and the starting vector w_1 were both random vectors. The example was run on a Cray-2 at the NASA Ames Research Center, with a right SSOR preconditioner. The algorithm was stopped once it reached 1.0E−20, after 107 steps. For the VW sequence, it built 3 blocks of size 2 and 1 block of size 3. For the PQ sequence, it built no blocks, but forced a block closure once. Figure 4 shows both the computed residual norm (solid line) and the residual norm upper bound (dotted line) available at no extra cost in the algorithm. Note that the upper bound is an excellent indicator for the true residual norm.

Example 4. Nachtigal et al. [1994] have recently incorporated several QMR algorithms from QMRPACK into codes for the computation of the electronic structure of materials using an approach based on first principles. At the heart of these computations is the explicit inversion of a complex, non-Hermitian matrix. The operation is repeated at many points on a grid, and many times in a self-consistent loop, leading to thousands or possibly millions of matrix inversions; this is the most time-consuming part of the computation. Previous versions of the codes used a direct approach to compute the inverses (either Gauss-Jordan or an LU decomposition). The new version replaces the direct solver with the TFQMR solver `ZUTFX` from QMRPACK. Results indicate that the new approach is significantly faster than the old one and that it scales much better as well. Figure 5 shows average times of a computation for Molybdenum, at 0K, where TFQMR and an LU solver were used. Note that TFQMR and LU perform comparably if the cluster size is 9 atoms, but the TFQMR solver becomes significantly faster as the cluster size is increased and reaches a factor of 7.5 speedup in CPU time for a cluster size of 229 atoms. The difference is even more pronounced in user time, where the TFQMR solver is more than 40 times faster, as the direct-solver codes are swapped out more often. Thus, the new approach is shown to have significant potential both to speed up the computation and to extend it to larger cluster sizes than those previously attainable.

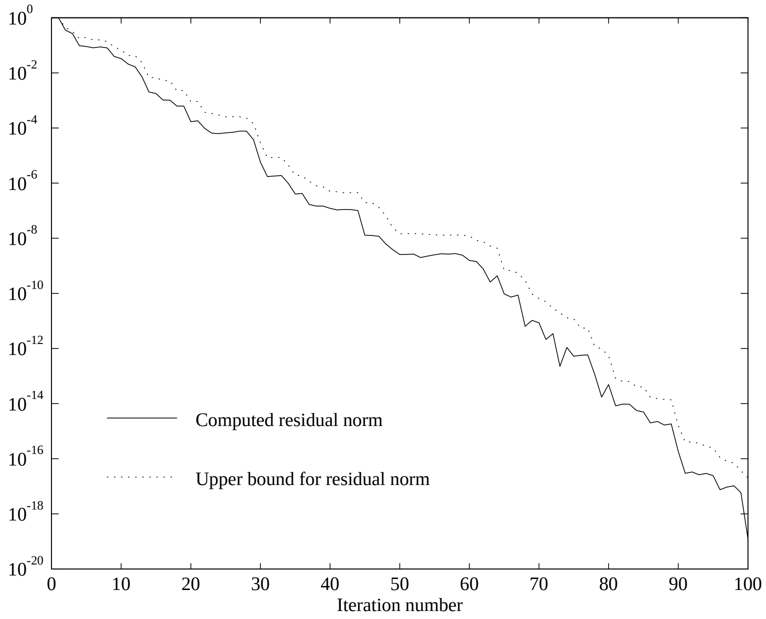

Figure 4: Upper bound and computed residual norm for Example 3.

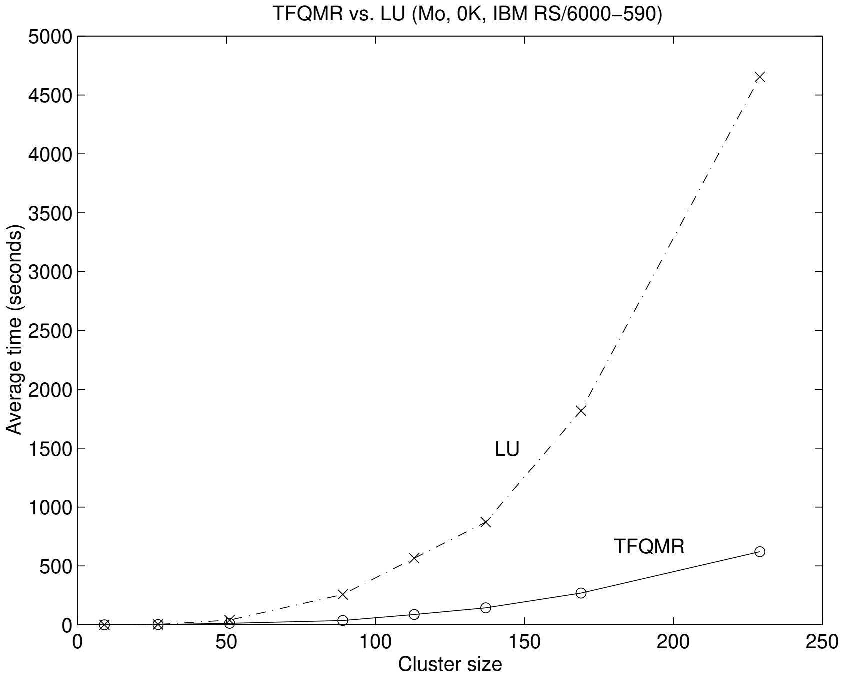

Figure 5: Average times for LU and TFQMR for Example 4.

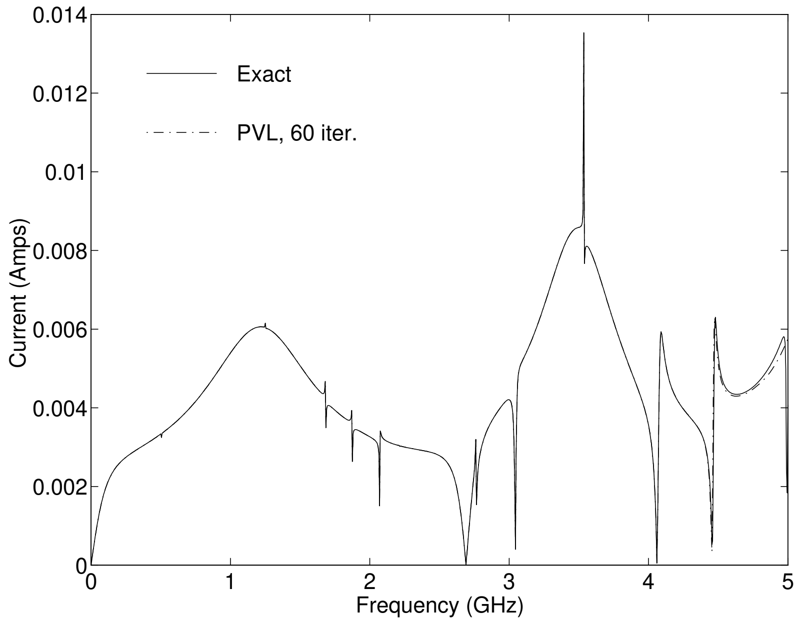

Figure 6: Simulation of the PEEC circuit from Example 5, 60 PVL steps.

Example 5. Finally, another example of an application of QMRPACK is the recent work of Feldmann and Freund [1994] who use the look-ahead Lanczos algorithm from QMRPACK to compute Padé approximations to the frequency-domain transfer function of large linear electronic circuits. The resulting procedure is the PVL algorithm sketched in Section 2.4. We illustrate the behavior of the PVL algorithm with one typical example taken from Feldmann and Freund [1994]. In Figure 6, we show the exact frequency response (solid line) and the approximation obtained after 60 Lanczos steps (dash-dotted line) for the PEEC circuit, which consists of 2100 capacitors, 172 inductors, and 6990 inductive couplings. Notice the excellent agreement of the two curves over a wide frequency range.

ACKNOWLEDGMENTS

We would like to thank Bart Semeraro for his help with Example 2. We are grateful to the anonymous referees for comments and suggestions that improved our paper.

APPENDIX — DISTRIBUTION OF QMRPACK

QMRPACK is available from netlib [Dongarra and Grosse 1987], either as a compressed tar file with the complete package, or as individual algorithm files.

The complete package is distributed in the form of the gzipped tar file `qmrpack.tgz` in the `linalg` section of netlib. The file can be obtained by using xnetlib or by anonymous ftp. In the latter case, ftp to `netlib.org` and then get the file `qmrpack.tgz` from the directory `netlib/linalg`.

It is also possible to retrieve single algorithms from QMRPACK. The single algorithms are contained in the `linalg/qmr` section of netlib. To obtain a list of the available algorithms, send an email message consisting of the single line

```
send index from linalg/qmr
```

to `netlib.org` on the Internet.

REFERENCES

ARNOLDI, W. E. 1951. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quart. Appl. Math. 9*, 1, 17–29.

AXELSSON, O. 1985. A survey of preconditioned iterative methods for linear systems of algebraic equations. *BIT 25*, 1, 166–187.

BEAM, R. M., AND WARMING, R. F. 1976. An implicit finite-difference algorithm for hyperbolic systems in conservation-law form. *J. Comput. Phys. 22*, 1, 87–110.

CULLUM, J., AND WILLOUGHBY, R. A. 1986. A practical procedure for computing eigenvalues of large sparse nonsymmetric matrices. In *Large Scale Eigenvalue Problems*, J. Cullum and R. A. Willoughby, Eds., North-Holland, Amsterdam, The Netherlands, pp. 193–240.

DONGARRA, J. J., AND GROSSE, E. H. 1987. Distribution of mathematical software via electronic mail. *Commun. ACM 30*, 5, 403–407.

EISENSTAT, S. C. 1981. Efficient implementation of a class of preconditioned conjugate gradient methods. *SIAM J. Sci. Statist. Comput. 2*, 1, 1–4.

FELDMANN, P., AND FREUND, R. W. 1994. Efficient linear circuit analysis by Padé approximation via the Lanczos process. Numerical Analysis Manuscript 94–01, AT&T Bell Laboratories, Murray Hill, NJ.

FREUND, R. W. 1989. Conjugate gradient type methods for linear systems with complex symmetric coefficient matrices. RIACS Tech. Report 89.54, NASA Ames Research Center, Moffett Field, CA.

FREUND, R. W. 1992a. Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices. *SIAM J. Sci. Statist. Comput. 13*, 1, 425–448.

FREUND, R. W. 1992b. Quasi-kernel polynomials and their use in non-Hermitian matrix iterations. *J. Comput. Appl. Math. 43*, 1–2, 135–158.

FREUND, R. W. 1992c. Quasi-kernel polynomials and convergence results for quasi-minimal residual iterations. In *Numerical Methods of Approximation Theory*, D. Braess and L. Schumaker, Eds., Birkhaüser, Basel, Switzerland, pp. 77–95.

FREUND, R. W. 1993. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput. 14*, 2, 470–482.

FREUND, R. W. 1994. The look-ahead Lanczos process for nonsymmetric matrices and its applications. In *Proceedings of the Cornelius Lanczos International Centenary Conference*, J. D. Brown, M. T. Chu, D. C. Ellison, and R. J. Plemmons, Eds., SIAM, Philadelphia, PA, pp. 33–47.

FREUND, R. W., GUTKNECHT, M. H., AND NACHTIGAL, N. M. 1993. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. *SIAM J. Sci. Comput. 14*, 1, 137–158.

FREUND, R. W., AND NACHTIGAL, N. M. 1991. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math. 60*, 3, 315–339.

FREUND, R. W., AND NACHTIGAL, N. M. 1993. Implementation details of the coupled QMR algorithm. In *Numerical Linear Algebra*, L. Reichel, A. Ruttan, and R. S. Varga, Eds., W. de Gruyter, Berlin, Germany, pp. 101–121.

FREUND, R. W., AND NACHTIGAL, N. M. 1994. An implementation of the QMR method based on coupled two-term recurrences. *SIAM J. Sci. Comput. 15*, 2, 313–337.

FREUND, R. W. AND SZETO, T. 1992. A transpose-free quasi-minimal residual squared algorithm for non-Hermitian linear systems. In *Advances in Computer Methods for Partial Differential Equations – VII*, R. Vichnevetsky, D. Knight, and G. Richter, Eds., IMACS, pp. 258–264.

FREUND, R. W., AND ZHA, H. 1995. Simplifications of the nonsymmetric Lanczos process and a new algorithm for Hermitian indefinite linear systems. AT&T Numerical Analysis Manuscript, Bell Laboratories, Murray Hill, NJ.

GILL, P. E., MURRAY, W., PICKEN, S. M., AND WRIGHT, M. H. 1979. The design and structure of a Fortran program library for optimization. *ACM Trans. Math. Softw. 5*, 3, 259–283.

GRAGG, W. B. 1974. Matrix interpretations and applications of the continued fraction algorithm. *Rocky Mountain J. Math. 4*, 2, 213–225.

GRAGG, W. B., AND LINDQUIST, A. 1983. On the partial realization problem. *Linear Algebra Appl. 50*, 277–319.

KROGH, F. T. 1969. VODQ/SVDQ/DVDQ—Variable order integrators for the numerical solution of ordinary differential equations. Subroutine Write-Up, Sec. 314, Jet Propulsion Lab., Pasadena, CA.

LANCZOS, C. 1950. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Standards 45*, 4, 255–282.

LANCZOS, C. 1952. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bur. Standards 49*, 1, 33–53.

LANCZOS, C. 1956. *Applied Analysis*. Prentice-Hall, Englewood Cliffs, NJ.

NACHTIGAL, N. M., AND SEMERARO, B. D. 1995. QMR methods in computational fluid dynamics. In *Solution Techniques for Large-Scale CFD Problems*, W. G. Habashi, Ed., John Wiley & Sons, New York.

NACHTIGAL, N. M., SHELTON, W. A., AND STOCKS, G. M. 1994. Combining the QMR method with first principles electronic structure codes. Oak Ridge National Laboratory, Technical Report ORNL/TM–12873, Oak Ridge, TN.

PARLETT, B. N., TAYLOR, D. R., AND LIU, Z. A. 1985. A look-ahead Lanczos algorithm for unsymmetric matrices. *Math. Comp. 44*, 105–124.

SAAD Y. 1992. ILUT: a dual threshold incomplete LU factorization. Research Report UMSI 92/38, University of Minnesota Supercomputer Institute, Minneapolis, MN.

Taylor, D. R. 1982. Analysis of the look ahead Lanczos algorithm. Ph.D. Thesis, Department of Mathematics, University of California, Berkeley, CA.

Venkatakrishnan, V., and Barth, T. J. 1992. Unstructured grid solvers on the iPSC/860. In *Parallel Computational Fluid Dynamics*, Rutgers, New Jersey, May 1992.