The Lanczos Algorithm
With Selective Orthogonalization*

By B. N. Parlett and D. S. Scott

Abstract. The simple Lanczos process is very effective for finding a few extreme eigenvalues of a large symmetric matrix along with the associated eigenvectors. Unfortunately, the process computes redundant copies of the outermost eigenvectors and has to be used with some skill. In this paper it is shown how a modification called selective orthogonalization stifles the formation of duplicate eigenvectors without increasing the cost of a Lanczos step significantly. The degree of linear independence among the Lanczos vectors is controlled without the costly process of reorthogonalization.

1. Introduction. The Lanczos method is well suited to the task of computing a few \((p)\) eigenvalues and eigenvectors of a large \((n \times n)\) symmetric matrix \(A\). The wanted eigenvalues may be at either, or both, ends of the spectrum. Typical values are \(p = 4\) and \(n = 1000\); in a typical application the smallest eigenvalues of \(A\) will correspond to the natural frequencies which can be excited in some structure after it is perturbed away from equilibrium.

It seems appropriate to give a brief review of the history of the method. Simple processes, like the Power Method, require, in principle, an infinite number of matrix-vector products to converge to an eigenvector. On the other hand, the method of Minimized Iterations, which Lanczos announced in 1950, expands each eigenvector in a convergent series with at most \(n\) terms.** However, Lanczos' method was promptly switched to a different channel. It was used as a process for computing a tridiagonal matrix \(T\) orthogonally congruent to \(A\); \(T = Q^*AQ, Q = (q_1, q_2, \ldots, q_n), Q^* = Q^{-1}\).

Despite its theoretical attractions the Lanczos process was soon displaced by the Givens [1954] and Householder [1958] methods which employ explicit similarity transformations on \(A\). To compete in accuracy the Lanczos process has to be supplemented with the explicit orthogonalization of the Lanczos vectors \(\{q_i\}\) which, in exact arithmetic, would be orthogonal automatically.

In 1971 C. Paige showed that the simple Lanczos procedure, without orthogonalization, was very effective for finding a few of the extreme eigenvalues and their matching eigenvectors.

Received February 14, 1978.


Key words and phrases. Eigenvalues, eigenvectors, symmetric matrices, Lanczos method.

* Research supported by Office of Naval Research Contract N00014-76-C-0013.

** Convergence is usually very quick for eigenvectors belonging to extreme eigenvalues.

© 1979 American Mathematical Society
0025-5718/79/0000-0014/$06.50

217
In part this is because the only way $A$ enters the Lanczos algorithm is through a subprogram which computes $Ax$ for any given vector $x$. The user is free to exploit sparseness and compact storage of $A$ in the coding of this subprogram. Equally important is the fact that the algorithm need not go the whole way. It builds up $Q_j = (q_1, \ldots, q_j)$ and $T_j = Q_j^*AQ_j$ by step $j$ and can often be stopped at values of $j$ as small as $2\sqrt{n}$. Paige [1971] showed that loss of orthogonality among the Lanczos vectors $\{q_1, q_2, \ldots\}$ was a necessary and sufficient condition, in finite precision arithmetic, for convergence of at least one of $T_j$'s eigenvalues to one of $A$'s eigenvalues.

This left the Lanczos algorithm as a very powerful tool in the hands of an experienced user. However, it did not provide a black box program which could be used "off the shelf" in the same way as eigenvalue programs for small matrices. There are several rather technical reasons for this. For one thing suitable criteria for accepting good approximations, rejecting spurious approximations, or stopping were all rather elusive. Left to itself a simple Lanczos program will run forever, doggedly finding more and more copies of the outer eigenvalues for each new inner eigenvalue it discovers.

This uncertainty about the amount of storage which is needed prompted the suggestion, by Golub and others, that the Lanczos method be used iteratively. That is, after $k$ steps the best approximation to an eigenvector is computed and it, or some modification of it, is used as a new starting vector. With this approach the old difficulties take on new forms: how to choose $k$ and how to select the new starting vector.

Another variation which has been used with success is the block form of the Lanczos method. Each step becomes more costly but fewer are needed, and this seemed to be the only way to find small clusters of close or multiple eigenvalues. However, the user has to make the difficult choice of the block size.

The remainder of this article describes an inexpensive modification of the simple algorithm (we call it Lanczos with selective orthogonalization) which permits the simple Lanczos process to be used as a black box. Moreover,

1. No redundant copies of eigenvectors are computed.

2. A posteriori error bounds and estimates cost almost nothing and are used in order to stop the program as soon as possible.

3. Multiple eigenvalues, and their eigenvectors, are found naturally, thanks to roundoff error.

Not surprisingly, the idea of purifying Lanczos vectors did not come out of the blue. Cullum and Donath [1974] found it necessary to deflate converged Ritz vectors from their blocks, Lewis [1977] found that some deflation helped in a difficult calculation of interior eigenvalues, and Underwood [1975] removed such vectors from his blocks when restarting the iterative version of Lanczos. However, we do not regard deflation as an aid in adversity but as a tool for producing orthogonal Ritz vectors; and thus, our orthogonalization is independent of convergence and will occur beforehand, especially when the user wants high accuracy.
LANCZOS ALGORITHM WITH SELECTIVE ORTHOGONALIZATION

2. Notational Conventions.
Integers—\(i, j, k, l, m, n, p\).
Scalars—small Greek letters \(\alpha, \beta, \ldots\).
(Column) Vectors—small roman letters \(x, y, \ldots\) (except for the integers).
Matrices—capital roman letters.
Identity matrix—\(I = (e_1, e_2, \ldots, e_n)\).
Diagonal matrices—capital Greek letters.
SYMmetric (nondiagonal) MATRICES—SYMmetric LETTERS \(A, H, M, U, V, W, X\).
Tridiagonal matrices—

\[
T_j = \begin{bmatrix}
\alpha_1 & \beta_1 & & \\
& \beta_1 & \alpha_2 & \\
& & \ddots & \\
& & & \alpha_{j-1} & \beta_{j-1} \\
& & & & \beta_{j-1} & \alpha_j
\end{bmatrix}
\]

All vectors are \(n\)-dimensional unless the contrary is stated. All square matrices
are \(n \times n\) unless the contrary is stated.
\(A - \xi I\) is written for \(A - \xi I\).
\(\text{Span}(b_1, \ldots, b_j)\) denotes the subspace generated by \(b_1, \ldots, b_j\).
\(x^*\) is the transpose of \(x\).
\(\|x\| = \sqrt{x^*x}\), the Euclidean norm.
\(\lambda_i[M]\) —the \(i\)th eigenvalue of \(M\) (from the left).

Eigenvalue Ordering:

\[
\begin{cases}
\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n, \\
\lambda_{-n} \leq \cdots \leq \lambda_{-2} \leq \lambda_{-1}.
\end{cases}
\]

\(\|M\| = \max_i |\lambda_i[M]| = \max \|Mv\|/\|v\|, v \neq 0\).
(j)—a formula in the current section.
(k)—a formula in section \(k\).

3. Lanczos in Exact Arithmetic. \(A\) can be reduced to tridiagonal form \(T_n\) in
many different ways. Let

\[
Q_n^*AQ_n = T_n
\]

be one such reduction, where \(Q_n = (q_1, \ldots, q_n)\) is orthogonal. If the off-diagonal
elements \(\beta_i, i = 1, \ldots, n-1,\) of \(T_n\) are positive then, in fact, \(T_n\) and \(Q_n\) are
completely determined by \(q_1\) or by \(q_n\). Let us write (1) in the form \(AQ_n = Q_nT_n\)
and see what it says about the \(n \times j\) submatrix \(Q_j = (q_1, \ldots, q_j), j < n\).
The last column on the right is \( r_j = q_{j+1} \beta_j \). Now (2) can be written compactly as

\[
AQ_j = Q_j T_j + r_j e_j^* \quad j = 1, \ldots, n,
\]

where \( e_j^* = (0, \ldots, 0, 1) \) has \( j \) elements and \( \beta_n = 0 \). From the orthogonality of \( Q_n \) follows

\[
Q_i^* Q_j = \delta_{ij}, \quad \text{whereas} \quad Q_i^* Q_j = \text{an orthogonal projector onto span } Q_j.
\]

Note that if \( \beta_j \) (i.e. \( \| r_j \| \)) = 0, then span \( Q_j \) is an invariant subspace and \( T_j \) is the restriction of \( A \) to it. In genuine applications \( \beta_j = 0 \) never happens, even for \( j > n! \)

The Lanczos algorithm builds up \( Q_j \) and \( T_j \) one column per step. Some important relations follow from (3) and (4) and are independent of the specific implementation of the algorithm.

Orthogonality. Since \( r_j \) is a multiple of \( q_{j+1} \) it must be orthogonal to all previous \( q_i \), \( i = 1, \ldots, j \). In fact, this property can be deduced from (3) and (4) without invoking (1).

**Lemma 1.** Let \( Q_j \) be any matrix satisfying (3) and (4). Then \( Q_{j-1}^* r_j = 0 \) and, if \( \alpha_j = q_j^* A q_j \), then \( Q_{j-1}^* r_j = 0 \) too.

A proof is given in Kahan and Parlett [1974].

The Lanczos algorithm proceeds from \( Q_j \) to \( Q_{j+1} \) by forcing \( q_j^* r_j = 0 \) via the choice of \( \alpha_j \) and then normalizing \( r_j \) to get \( \beta_j \) and \( q_{j+1} \). What could be simpler? Note that \( Q_{j-1}^* r_j \) is not forced to vanish because, in exact arithmetic, the lemma guarantees it. From (3)

\[
r_j = r_j e_j^* e_j = (AQ_j - Q_j T_j) e_j = A q_j - q_{j-1} \beta_{j-1} - q_j \alpha_j.
\]

Observe that \( q_1, \ldots, q_{j-2} \) are not needed for the computation of \( \alpha_j, r_j, \beta_j, q_{j+1} \) (i.e. the \( j \)th step) and so may be put out to a secondary storage medium. This is a very attractive feature of the method.
Suppose that the Lanczos algorithm pauses at the $j$th step and makes a subsidiary computation of some, or all, of the eigenvalues and eigenvectors of $T_j$. Let

$$(6) \quad T_j = S_j \Theta_j S_j^*,$$

where $S_j = (s_1, \ldots, s_j)$ is $j \times j$ and orthogonal and $\Theta_j = \text{diag}(\theta_1, \ldots, \theta_j)$ has the eigenvalues of $T_j$. Since $T_j = Q_j^* A Q_j$, the values $\theta_i$ and the vectors $y_i = Q_j s_i$ are the (optimal) Rayleigh-Ritz approximations to $A$ from the information on hand, i.e. from span($Q_j$). Note that

$$(7) \quad \|A Q_j - Q_j T_j\| = \|r_j e_j^*\| = \beta_j.$$

How good are these optimal approximations?

**Theorem 1.** There are $j$ eigenvalues of $A$, call them $\lambda_1, \ldots, \lambda_j$, such that

$$|\lambda_i - \theta_i| \leq \beta_j, \quad i = 1, \ldots, j.$$

A proof is given in Kahan [1967]. See also Kahan and Parlett [1976]. The bound covers all the $\theta_i$ and does not discriminate between them. We can do better. Let $s_{ji}$ denote the bottom ($j$th) element of $T_j$'s eigenvector $s_j$.

**Theorem 2.** To each $i$ there is a corresponding eigenvalue of $A$, call it $\lambda_i'$, such that

$$|\lambda_i' - \theta_i| \leq \beta_j |s_{ji}| = \beta_i, \quad i = 1, \ldots, j.$$

A proof is given in Kahan and Parlett [1976]. The quantity $\beta_i = \|(A - \theta_i) y_i\|_i$, the $i$th residual.

**Theorem 3.** Let $A z_i = z_i' \lambda_i'$, let $\psi_i$ be the angle between $z_i'$ and its Ritz vector $y_i = Q_j s_i$, and let the gap $\gamma_i = \min_{k \neq i} |\lambda_k - \theta_i|$. Then, for $i = 1, \ldots, j$,

$$|\lambda_i' - \theta_i| \leq \beta_i^2 / \gamma_i, \quad \tan \psi_i \leq \beta_i / \gamma_i.$$

Proofs are given in Davis and Kahan [1970].

In principle $\gamma_i$ is unknown and these bounds are not computable. However, $\delta_i = \min_{k \neq i} |\theta_k \pm \beta_{jk} - \theta_i|$ can be used in place of $\gamma_i$ to give an estimate.

The following result, proved in Kahan [1967], shows that the previous bounds fail gracefully when $Q_j$ is not orthonormal. Specifically, the bounds must be multiplied by $\sqrt{2}/\sigma_1$, where $\sigma_1^2$ is the smallest eigenvalue of $Q_j^* Q_j$.

**Theorem 4.** Given $A, n \times n, Q, n \times m, H, m \times m$, with $\lambda_i[H] = \theta_i$, $i = 1, \ldots, m$, there are $m$ of $A$'s eigenvalues, call them $\lambda_i'$, so that for $i = 1, \ldots, m$,

$$|\lambda_i' - \theta_i| \leq \sqrt{2} \|A Q - Q H\|_1 / \sigma_1(Q),$$

where $\sigma_1^2 = \lambda_1[Q^* Q]$.

The Kaniel-Paige theory (Kaniel [1966]) shows that it is the extreme (leftmost and rightmost) eigenvalues which are most likely to be approximated by some of the $\theta_i$. Moreover, the rapidity of convergence, as $j$ increases, depends on the (unknown)
gaps between \( A \)'s eigenvalues. Cases have occurred in which an unusual distribution of eigenvalues coupled with special \( q_1 \)'s have caused interior eigenvalues to come out first. See Cline, Golub and Platzman [1976].

In principle, then, the Lanczos algorithm should be continued, with periodic pauses, until, and only until, adequate approximations to the wanted eigenvalues and eigenvectors are in hand. This sometimes happens for values of \( j \) as small as \( 2\sqrt{n} \), another attraction of the method.

In practice, things are not this simple. With finite precision computation convergence goes hand in hand with loss of linear independence among the \( q_i \), and so the error bounds cease to be valid by the time the first of the \( \theta_i \) converges.

Before leaving the context of exact arithmetic we want to emphasize the value of the bounds \( \beta_{ij} \). They show why the absence of small \( \beta_j \) does not impede convergence of some of the \( \theta_i \) to eigenvalues and the computable numbers \( s_{ij} \) show which of the \( \theta_i \) are converging. There are extensions of Theorem 3, which allow a bunch of close \( \theta \)'s and their \( y \)'s to be treated simultaneously; the gap then becomes the distance of the cluster from eigenvalues not associated with the cluster.

4. Orthogonality Versus Convergence. The use of finite precision arithmetic provokes significant departures from the exact version of the Lanczos algorithm described above. In order to examine these effects we turn our backs on the quantities which would be produced by use of exact arithmetic and make a standard change of notation. The symbols \( Q_p, T_j, \alpha_j, \beta_j \) from now on denote the computed quantities stored in the computer under these names. We shall not try to compare them with their Platonic counterparts but instead we will seek the (more complicated) relations which do hold between the objects on hand.

The fundamental equation (3.3) becomes

\[
AQ_j = Q_j T_j + r_j e_j^+ - \bar{F}_j, \tag{1}
\]

where \( \bar{F}_j \) accounts for local round-off effects. Paige has shown that if the algorithm is implemented correctly, \( \bar{F}_j \) is harmless, satisfying an inequality of the form \( \| \bar{F}_j \| \leq \phi(n)e\|A\| \) for some almost linear function \( \phi \) (Paige [1972], [1976]). The orthogonality relation (3.4) fails and in its place we write

\[
\| 1 - Q_j^*Q_j \| \leq \kappa_j. \tag{2}
\]

In the last section we given an expression for \( \kappa_j \); but here we focus on the more special and more important issue of orthogonality loss among the vectors \( y_i = Q_p \beta_p, i = 1, \ldots, j \), which we continue to call Ritz vectors despite the fact that the optimality with which they approximate eigenvectors of \( A \) departs hand in hand with \( Q_j \)'s orthogonality.
In Paige [1971] can be found the following remarkable results in which the bottom elements $s_{jk} (= e_s s_i)$ of $T_j$'s eigenvectors $s_i$ appear again.

**Theorem 5.** Consider the $j$th step of the simple Lanczos algorithm, and drop the index $j$ on which all the quantities depend. The computed approximate eigenpairs $(\theta_i, y_i)$, $i = 1, \ldots, j$, satisfy

$$y_i^T y_k = (G_i(s_{jk} / s_{ji}) - g_k(s_{jk} / s_{jk}) + f_{ik}^T)/(\theta_i - \theta_k), \quad i \neq k,$$

where $G$ and $F$ are round-off matrices; $\|G\| \div \|F\| \div \sqrt{\varepsilon n} \|T\|$, where $\varepsilon$ is the relative precision of the arithmetic. Moreover,

$$y_{i+1}^T q_{j+1} = g_{ij}^T g_{ji} = g_{ii}^T (\beta_j | s_{ji} |), \quad i = 1, \ldots, j.$$

The bottom elements of the $s_i$ appear in a special way. With any good program, $S$ will be orthonormal (to working accuracy) so that $\sum_{i=1}^j s_{ji}^2 = 1$. If

$$(3) \quad |s_{jk}| \div |s_{ji}| \div j^{-1/2}, \quad |\theta_i - \theta_k| > \|T\|/100,$$

then the error bounds (Theorems 3 and 4) on $\theta_i$ and $\theta_k$ indicate that they are poor eigenvalue approximations while Theorem 5 shows that $y_i$ and $y_k$ are orthogonal to working accuracy. Conversely, if $|s_{ji}| < 10^{-3}$, say, then $\theta_i$ (if isolated) is a good eigenvalue approximation, $y_i$ is good too, and $y_i$ will not be orthogonal to any unconverted $y_k$ (indicated by $s_{jk} \div j^{-1/2}$). Since $S$ is orthonormal to working accuracy, it is $Q_j$ which must have lost orthonormality. The better the approximations $\theta_i$ and $y_i$ the greater the departure of $Q_j$ from orthogonality.

A further analysis (Paige [1971]) shows that $\frac{1}{2} \leq \|y_i\| \leq 2$ provided that the $\theta$'s are not too close. What this means in practice is that Ritz vectors $y_i$ cannot shrink alarmingly unless there are two or more $\theta$'s approximating a single eigenvalue $\lambda$. Our orthogonalization forestalls this calamity.

As the Lanczos algorithm proceeds with increasing $j$, the loss of orthogonality among the Lanczos vectors $\{q_j\}$ is widespread but has no apparent structure. It is the Ritz vectors $\{y_j\}$ which display the pattern of the loss of orthogonality. Unconverted Ritz vectors will be mutually orthogonal while both the unconverted Ritz vectors and $q_{j+1}$ will have strong components in the direction of Ritz vectors which have nearly converged.

**Example of Loss of Orthogonality.**

$n = 6$.

$A = \text{diag}(0., .00025, .0005, .00075, .001, 10.)$.

$q_1 = 6^{-1/2}(1., 1., 1., 1., 1., 1.)^T$.

Unit round off $\div 10^{-14}$.

Simple Lanczos was run for six steps. $Y_6 = Q_6 S_6$. 

License or copyright restrictions may apply to redistribution; see http://www.ams.org/journal-terms-of-use
Note that the general loss of orthogonality seen in $Q_6^*Q_6$ is represented in $Y_6^*Y_6$ as the second copy of the eigenvector associated with the eigenvalue 10.

5. **Selective Orthogonalization.** One way to restore orthogonality to $Q_f$ is to use the modified Gram-Schmidt process in order to force $q_{f+1}$ to be orthogonal to all previous $q$'s. Besides the ever increasing expense in arithmetic operations, this reorthogonalization process requires the presence of all the $q_i$ at each step. Paige’s result suggests that linear independence of the $q$’s can be maintained by merely orthogonalizing the $q$’s against a few selected vectors, namely the Ritz vectors which have nearly converged. Hence, the name of the algorithm.

The modification of the simple Lanczos process is as follows. At each pause $T_j$ is diagonalized and the bounds on the not-yet-computed Ritz vectors are inspected. Those Ritz vectors with error bounds less than $\sqrt{\varepsilon\|A\|}$ are declared good, are computed, orthonormalized, and then stored in the fast memory. From that point until the next pause all future $q$’s are kept nearly orthogonal to these directions.

It might appear to be necessary to orthogonalize only $q_{f+1}$ and $q_{f+2}$ against these good $y$’s. It follows from (3.5) that all subsequent $q$’s would remain orthogonal to them. In finite precision, however, the error vector in each computation of $Aq_k$ will bring back small multiples of all $A$’s eigenvectors. Fortunately, it is not necessary
to orthogonalize $r_j$ against the $y$'s at every step as Section 7 reveals.

The purpose of selective orthogonalization is to prevent the computation of many unwanted copies of all the well-separated outer eigenvectors. This reduces the number of Lanczos steps required to compute the wanted eigenvalues and eigenvectors and so keeps the number of calls on the large matrix $A$ as low as possible. The algorithm must compute and store good Ritz vectors even if some of them are not wanted by the user. For example, if the three eigenvectors at the left end of the spectrum are wanted, the algorithm may well have computed three or more eigenvectors at the right end as well, should they happen to be better separated from the rest of the spectrum than are the ones we want.

**Example of Selective Orthogonalization.**

$n = 6$.

$$A = \text{diag}(0, .00025, .0005, .00075, .001, 10).$$

$$q_1 = 6^{-\frac{1}{2}}(1, 1, 1, 1, 1, 1)^T.$$  

Unit round off $\approx 10^{-14}$.  

The Lanczos algorithm with selective orthogonalization was run for six steps. It paused after four steps and computed a good Ritz vector for the eigenvalue 10. It then took two more steps orthogonalizing against this vector.

$$Q_{6}^{*}Q_{6} \text{ for Selective Orthogonalization}$$

\[
\begin{bmatrix}
.10E+01 & .75E-14 & -.30E-10 & .25E-06 & -.11E-09 & .92E-10 \\
.75E-14 & .10E+01 & .33E-10 & .55E-06 & .51E-10 & -.36E-10 \\
-.30E-10 & .33E-10 & .10E+01 & -.97E-10 & -.44E-10 & -.37E-07 \\
.25E-06 & .55E-06 & -.97E-10 & .10E+01 & .24E-07 & -.64E-08 \\
-.11E-09 & .51E-10 & -.44E-10 & .24E-07 & .10E+01 & .10E-13 \\
.92E-10 & -.36E-10 & -.37E-07 & -.64E-08 & .10E-13 & .10E+01 \\
\end{bmatrix}
\]

Note that the leading $4 \times 4$ principal minor is the same as in the earlier example. Robust linear independence has been maintained by selective orthogonalization.

6. When to Pause. There are five possible strategies for deciding when to pause. The simplest (and cheapest) is to pause every $m$ steps, where $m$ is some constant, possibly depending on $n = \text{dim}(A)$, but independent of all other characteristics of $A$. Such a plan is completely insensitive to the loss of orthogonality in $Q_j$ and is unsatisfactory in practice.

Paige and others have suggested keeping $q_1$ in fast store and computing $q_1^* q_j$ as a measure of the loss of orthogonality. This is not cheap since it requires the storage of an $n$-vector as well as the computation of a vector inner product at each step. This scheme usually works quite well. However, this estimate is a lower bound rather than an upper bound on $\|I - Q_j^* Q_j\|$. Therefore, on occasion, the pause may come too late and disastrous failures of this kind are possible in practice. Furthermore, it is
not clear how to apply this scheme, after the first pause, for deciding when to pause again.

Kahan and Parlett [1974], [1976] have described two different schemes for bounding $\|I - Q^*Q\|$. The scoreboard majorizes the matrix $I - Q^*Q$, which requires $j^2$ storage locations. Since the orthogonalizations will permit the Lanczos process to continue well beyond $j = \sqrt{n}$ steps, this storage cost becomes excessive. The other scheme is a scalar bound $\kappa$ on $\|I - Q^*Q\|$. Only a few arithmetic operations are needed to update $\kappa$ at each step, independent of both $j$ and $n$.

Rather than monitoring the loss of orthogonality, it is also possible to monitor convergence of the Ritz vectors instead. If the $\beta_{ji}$ are calculated at each step (or even a few of them from each end of the spectrum), the moment to pause can be determined exactly. One way of doing this would be to calculate all the Ritz values at each step and use a formula from Paige [1971] that states that

$$s_{ji}^2 = \chi_{j-1}(\theta_i)/\chi_j'(\theta_i),$$

where $\chi_j(\mu)$ is the characteristic polynomial of $T_j$. Another way would be to calculate a few eigenvalues of $T_j$ at each end of the spectrum, and then use inverse iteration to find the bottom elements of the corresponding eigenvectors.

The program described in the rest of this paper uses the kappa bound exclusively to determine when to pause. The details of the implementation are given in Section 14.

On the other hand the possibility of directly monitoring convergence is quite appealing. This approach, used on its own or in conjunction with the kappa bound, is being actively investigated.

7. Monitoring the Return of Banished Ritz Vectors. Let $y$ be a good normalized Ritz vector, and let $\tau_j$ be a bound on $|y^*q_j|$, the unwanted component of $y$ in $q_j$.

There is a simple three term recurrence governing the $\tau$'s. We have

$$Ay = \theta y + r \quad (r \text{ is not to be confused with } r_j).$$

The quantities computed in the $j$th step of the Lanczos algorithm satisfy

$$q_{j+1}\beta_j = Aq_j - q_j\alpha_j - q_{j-1}\beta_{j-1} + f_j,$$

where $f_j$ accounts for the round off and $\|f_j\| \leq \nu\|A\|$ for some constant $\nu$ which depends on $A$ but not on $j$. Hence,

$$y^*q_{j+1}\beta_j = y^*Aq_j - y^*q_j\alpha_j - y^*q_{j-1}\beta_{j-1} + y^*f_j.$$  

Because $|y^*q_j| \leq \tau_j$, (1) and (2) yield
(4) 
\[ |y^*q_{j+1}| \leq [|\vartheta - \alpha_j|\tau_j + \beta_{j-1}\tau_{j-1} + |r^*q_j| + \nu \|A\|] / \beta_j = \tau_{j+1} \cdot \]

Moreover, \( r = q_{k+1}\beta_k \) for some \( k < j \), so \( |r^*q_j| \leq \beta_{ki}q_k^*q_j \leq \beta_{ki} \sqrt{\epsilon} = O(\epsilon\|A\|) \). Since \( \nu \) and \( \|A\| \) are not readily available, the program simply drops the last two terms in (4). Each time that a pair of \( q \)'s are explicitly orthogonalized against \( y \) the corresponding \( r \)'s are set to \( \epsilon \). Then the recurrence is updated by (4) at each step and tested. As soon as \( \tau_j \) again exceeds the tolerance, \( y \) is explicitly deflated out of \( q_j \) and \( q_{j+1} \). The tolerance is not critical (\( \sqrt{\epsilon} \) seems to be an appropriate value).

Along with each computed Ritz vector is stored the associated eigenvalue \( \theta_i \), the residual norm estimate \( \beta_{ij} \), and cells for the current and previous \( \tau \)-values. The cost of updating this information is negligible. Thus, \( \tau \) may be thought of as a two-rowed array of length equal to the number of good Ritz vectors.

8. Flowchart I. Lanczos with selective orthogonalization, ample storage and no multiple eigenvalues.

\begin{itemize}
  \item Initialize.
  \item Permits input of \( q_1 \).
  \item Take a Lanczos step.
  \item Update \( \tau \).
  \item Purge if necessary.
  \item Update \( \kappa \).
  \item Is \( \kappa \geq \sqrt{\epsilon} \)?
  \item Compute partial Eigensystem of \( T \) and test the \( \beta_{ij} \).
  \item Enough acceptable vectors?
  \item \( \text{Compute and store good Ritz vectors. Initialize } \tau \text{-vector. Store Ritz values. Reset } \kappa \).
  \item Compute and store accepted vectors. Store Ritz values.
\end{itemize}

Notes.
1. \( \kappa \) monitors loss of orthogonality and is described in Section 14.
2. \( \tau \) monitors the components of the current Lanczos vector in the directions of the good Ritz vectors. See Section 7.
3. A good Ritz vector need not be one which is wanted. It may not be quite accurate enough to be accepted yet or it may belong to the wrong end of the spectrum.
4. Good Ritz vectors will be recomputed at each pause.
9. Running Out of Storage. Information is always lost when the Lanczos process is restarted. Since our algorithm maintains semiorthogonality among the $q$-vectors at a modest cost, the usual reason for restarting is not present. However, since the available storage may be quite limited on some computer systems, the program must be capable of restarting when necessary and as much information as possible should be retained.

Every time that storage has been exhausted, the program calculates and permanently stores all acceptable wanted $R$-vectors and all good vectors not among those desired. It also stores the corresponding Ritz values and sorts all the permanently stored Ritz vectors by increasing Ritz value.

There remains the question of what the starting vector should be. The new $q_1$ is currently taken to be a linear combination of some of the Ritz vectors which are not acceptable. The one with smallest residual $\beta_{\mu}$ is always used as well as any others which have converged to half the acceptable accuracy. The weights for the linear combination are the reciprocals of the $\beta_{\mu}$. Other choices for the restart vector could be made. Before restarting $q_1$ is orthogonalized against all the permanent vectors, and the $\tau$-vector (see Section 7) is initialized.

10. Multiple Eigenvalues. Since the Lanczos algorithm only examines the subspace spanned by the vectors $(q_1, Aq_1, A^2q_1, \ldots, A^lq_1)$, it is unable to detect any eigenvector which is orthogonal to $q_1$. In particular, it is incapable of finding multiple eigenvalues. If $V$ is the eigenspace of a multiple eigenvalue $\lambda$, then the Lanczos algorithm will find only the single eigenvector in the direction of the projection of $q_1$ onto $V$.

Despite this, the program finds multiple eigenvalues quite naturally. Rounding errors introduce components in all directions. After one eigendirection of a multiple eigenvalue has been found the components in orthogonal directions will persist after purification. These components will grow as the algorithm continues until a second eigenvector, orthogonal to the first, has been found.

Since multiple eigenvalues are found sequentially instead of simultaneously, a more sophisticated termination criterion is needed. For example, if $A$ has a double eigenvalue at zero, a simple eigenvalue at 1, and the rest of the spectrum larger than 2, then the program will find an eigenvector of 0 and the eigenvector of 1 at about the same time. Therefore, if the program finds enough acceptable vectors it must decide whether to start over again to test for undisclosed multiplicities. Currently, the program makes a test run if, at the last pause, more than one acceptable eigenvalue is found, or if the only one found is in the convex hull of the rest of the acceptable eigenvalues found so far. This strategy is rather conservative and will often make test runs which are unnecessary. However, with this criterion multiple eigenvalues will always be correctly unearthed.

11. Can Low Accuracy Be Achieved Safely? Yes. The user desired accuracy is used only in determining which of the desired vectors should be saved permanently.
when the process is started. A simple perturbation argument shows that any eigenvalue found after a restart is perturbed by no more than the maximum of the norms of the residuals of the permanent vectors. Consequently, eigenvalues found on later passes will be of the same order of accuracy as those found earlier.

12. Flowchart II. Modifications of Flowchart I to cope with limited storage and multiple eigenvalues.


diagram

13. Some Numerical Comparison for LANSO. We present some comparisons of LANSO (Lanczos Algorithm with Selective Orthogonalization) with published examples of block Lanczos programs written by R. Underwood and by J. Cullum and W. E. Donath. We have numbered the examples as they appear in the references. Unfortunately, the number of vector inner products needed by the Cullum and Donath program is not available. After the comparison we also trace the history of LANSO as it solved the problem. In all but the last example, significant effort was spent on the final multiple
eigenvalue check. A modification is planned which would reduce the length of the check run significantly.

**Underwood Examples.** (Example 2 omitted.)

**Example 1.** This example has a cluster of three eigenvalues well separated from the rest. Three eigenvalues were requested (nval = 3).

\[ n = 453, \text{nval} = 3, \text{ifig} = 8 = \text{no. of correct decimal digits desired} \]

\[ \lambda_1 = -10., \lambda_2 = -9.99, \lambda_3 = -9.98, \lambda_4 = -9. + .02 \times (i - 4), i = 4, 5, \ldots, 454 \]

<table>
<thead>
<tr>
<th></th>
<th>matrix-vector products</th>
<th>vector inner products</th>
<th>max error in eigenvalues</th>
<th>max residual norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Lanczos</td>
<td>165</td>
<td>1265</td>
<td>$10^{-13}$</td>
<td>$3 \times 10^{-6}$</td>
</tr>
<tr>
<td>LANSO</td>
<td>70</td>
<td>191</td>
<td>$10^{-11}$</td>
<td>$3 \times 10^{-6}$</td>
</tr>
</tbody>
</table>

History of LANSO (mxstep = 50. = maximum no. of Lanczos steps permitted in a run)

<table>
<thead>
<tr>
<th>pause at j =</th>
<th>new ( \kappa )</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>2.4</td>
<td>( 1.6 \times 10^{-12} )</td>
</tr>
<tr>
<td>44</td>
<td>3.4</td>
<td></td>
</tr>
<tr>
<td>restart</td>
<td>26</td>
<td>1.1</td>
</tr>
</tbody>
</table>

**Example 3.** This example is a purely linear distribution. It is the most difficult of the Underwood examples since the eigenvalues desired are not well separated from the rest.

\[ n = 101, \text{nval} = 6, \text{ifig} = 5 \]

\[ \lambda_i = -(101 - i)/100 \]

<table>
<thead>
<tr>
<th></th>
<th>matrix-vector products</th>
<th>vector inner products</th>
<th>max error in eigenvalues</th>
<th>max residual norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Lanczos</td>
<td>350</td>
<td>1974</td>
<td>$10^{-9}$</td>
<td>$2 \times 10^{-5}$</td>
</tr>
<tr>
<td>LANSO</td>
<td>112</td>
<td>383</td>
<td>$10^{-7}$</td>
<td>$1 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

History of LANSO (mxstep = 50.)

<table>
<thead>
<tr>
<th>pause at j =</th>
<th>new ( \kappa )</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>2.8</td>
<td>( 1 \times 10^{-12} )</td>
</tr>
<tr>
<td>46</td>
<td>2.4</td>
<td>( 6 \times 10^{-10} )</td>
</tr>
<tr>
<td>50</td>
<td>( 3 \times 10^{-7} )</td>
<td></td>
</tr>
<tr>
<td>restart</td>
<td>20</td>
<td>3.1</td>
</tr>
<tr>
<td>36</td>
<td>.92</td>
<td></td>
</tr>
<tr>
<td>restart</td>
<td>26</td>
<td>2.2</td>
</tr>
</tbody>
</table>

License or copyright restrictions may apply to redistribution; see http://www.ams.org/journal-terms-of-use
**Example 4.** This example has two double eigenvalues separated from the rest of the spectrum. Note that LANSO finds the eigenpairs to higher precision than desired. This is very common when low accuracy is desired for well-separated multiplicities.

\[ n = 180, \text{ nval } = 4, \text{ ifig } = 4 \]

\[ \lambda_1 = \lambda_2 = 0, \lambda_3 = \lambda_4 = .1, \lambda_i = .25 + .01 \times (i - 5), \text{ for } i = 5, 6, \ldots, 180 \]

<table>
<thead>
<tr>
<th>block size</th>
<th>matrix-vector products</th>
<th>vector inner products</th>
<th>max error in eigenvalues</th>
<th>max residual norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>158</td>
<td>997</td>
<td>( 2 \times 10^{-9} )</td>
<td>( 1.5 \times 10^{-4} )</td>
</tr>
<tr>
<td>2</td>
<td>125</td>
<td>725</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>140</td>
<td>699</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>317</td>
<td>1330</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LANSO</td>
<td>120</td>
<td>361</td>
<td>( 3 \times 10^{-13} )</td>
<td>( 5 \times 10^{-7} )</td>
</tr>
</tbody>
</table>

History of LANSO (mxstep = 50.)

<table>
<thead>
<tr>
<th>pause at j</th>
<th>( \kappa )</th>
<th>new ( \kappa )</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>1.8</td>
<td>( 3 \times 10^{-9} )</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>3.5</td>
<td>( 6 \times 10^{-5} )</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>1.0</td>
<td>( 1 \times 10^{-5} )</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>restart</td>
<td>19</td>
<td>2.7</td>
<td>( 1 \times 10^{-10} )</td>
</tr>
<tr>
<td>34</td>
<td>2.5</td>
<td>( 2 \times 10^{-7} )</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>2.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>restart</td>
<td>j = 26</td>
<td>1.6</td>
<td>terminate</td>
</tr>
</tbody>
</table>

**Example 5.** This example has a triple eigenvalue between a single and the rest. Note that the rest of the spectrum is not linear which improves convergence. Note that LANSO finds five vectors, all with better accuracy than desired. The new multiplicity check will make the most difference on this example.

\[ n = 300, \text{ nval } = 3, \text{ ifig } = 3 \]

\[ \lambda_1 = 0, \lambda_2 = \lambda_3 = \lambda_4 = .1, \lambda_i = 1 - 3/(i - 1), \text{ for } i = 5, 6, \ldots, 300 \]

block sizes of 1, 2, and 3 were tried but only the best, block size = 3, was reported

<table>
<thead>
<tr>
<th></th>
<th>matrix-vector products</th>
<th>vector inner products</th>
<th>max error in eigenvalue</th>
<th>max residual norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Lanczos</td>
<td>36</td>
<td>288</td>
<td>( 2 \times 10^{-9} )</td>
<td>( 3.3 \times 10^{-4} )</td>
</tr>
<tr>
<td>LANSO</td>
<td>67</td>
<td>249</td>
<td>( 2 \times 10^{-13} )</td>
<td>( 5 \times 10^{-9} )</td>
</tr>
</tbody>
</table>
History of LANSO (mxstep = 50)

<table>
<thead>
<tr>
<th>pause at j =</th>
<th>k</th>
<th>new k</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>restart</td>
<td>17</td>
<td>2.9</td>
<td>0, .1, and .25 found</td>
</tr>
<tr>
<td>(check)</td>
<td>16</td>
<td>9.1</td>
<td>another .1 found</td>
</tr>
<tr>
<td>restart</td>
<td>16</td>
<td>1.6</td>
<td>another .1 found</td>
</tr>
<tr>
<td>(check)</td>
<td>18</td>
<td>5.1</td>
<td>terminate with 5 vectors</td>
</tr>
</tbody>
</table>

**Example 6.** This is just the previous example with the triple eigenvalue slightly perturbed.

\[ n = 300, \text{ nval } = 4, \text{ ifig } = 3 \]
\[ \lambda_1 = 0, \lambda_2 = .0999999, \lambda_3 = .1, \lambda_4 = .1000001, \lambda_i = 1 - 3/(i - 1), i = 5, 6, \ldots, 300 \]

<table>
<thead>
<tr>
<th></th>
<th>matrix-vector products</th>
<th>vector inner products</th>
<th>max error in eigenvalues</th>
<th>max residual norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Lanczos</td>
<td>54</td>
<td>408</td>
<td>(2 \times 10^{-8})</td>
<td>(9 \times 10^{-4})</td>
</tr>
<tr>
<td>LANSO</td>
<td>58</td>
<td>204</td>
<td>(2 \times 10^{-7})</td>
<td>(6 \times 10^{-8})</td>
</tr>
</tbody>
</table>

History of LANSO:

<table>
<thead>
<tr>
<th>pause at j =</th>
<th>k</th>
<th>new k</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>restart</td>
<td>18</td>
<td>5.5</td>
<td>(5 \times 10^{-7})</td>
</tr>
<tr>
<td>(check)</td>
<td>24</td>
<td>2.4</td>
<td>4 vectors found: 0, .1, .1, and .25</td>
</tr>
<tr>
<td>restart</td>
<td>16</td>
<td>1.5</td>
<td>another .1 found</td>
</tr>
<tr>
<td>(check)</td>
<td>18</td>
<td>8.0</td>
<td>terminate</td>
</tr>
</tbody>
</table>

**Cullum and Donath Examples.**

**Example 7.1B.** This example has two eigenvalues with a good separation from the rest.

\[ n = 316, \text{ nval } = 2, \text{ ifig } = 9 \]
\[ \lambda_n = 0, \lambda_{n-1} = -1, \lambda_{n-i} = -0.03 \times (i - 2), i = 2, 3, \ldots, n - 1 \]

<table>
<thead>
<tr>
<th></th>
<th>matrix-vector products</th>
<th>vector inner products</th>
<th>max error in eigenvalue</th>
<th>max residual norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAN</td>
<td>94</td>
<td>*</td>
<td>(6 \times 10^{-10})</td>
<td>(2 \times 10^{-5})</td>
</tr>
<tr>
<td>LANSO</td>
<td>69</td>
<td>179</td>
<td>(6 \times 10^{-12})</td>
<td>(3 \times 10^{-6})</td>
</tr>
</tbody>
</table>
History of LANSO:

<table>
<thead>
<tr>
<th>pause at j =</th>
<th>κ</th>
<th>new κ</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>1.2</td>
<td>$8 \times 10^{-12}$</td>
<td>2 vectors found</td>
</tr>
<tr>
<td>43</td>
<td>1.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>restart</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(check)</td>
<td>26</td>
<td>2.7</td>
<td>terminate</td>
</tr>
</tbody>
</table>

Example 7.4A, A. This example has two eigenvalues with less separation than the previous one. A higher value for mxstep (say 100 instead of 50) would improve the convergence rate for LANSO. Note again that LANSO produces more accuracy.

$n = 201$, $nval = 2$, ifig = 11

$\lambda_n = 0$, $\lambda_{n-1} = -.01$, $\lambda_{n-i} = - .1 - .05 \times (i - 2)$, for $i = 2, 3, \ldots, n - 1$

<table>
<thead>
<tr>
<th>matrix-vector products</th>
<th>vector inner products</th>
<th>max error in eigenvalues</th>
<th>max residual norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAN</td>
<td>184</td>
<td>$3 \times 10^{-9}$</td>
<td>$5 \times 10^{-6}$</td>
</tr>
<tr>
<td>LANSO</td>
<td>142</td>
<td>$5 \times 10^{-14}$</td>
<td>$2 \times 10^{-7}$</td>
</tr>
</tbody>
</table>

History of LANSO (mxstep = 50)

<table>
<thead>
<tr>
<th>pause at j =</th>
<th>κ</th>
<th>new κ</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>2.5</td>
<td>$9 \times 10^{-13}$</td>
<td>mxstep</td>
</tr>
<tr>
<td>46</td>
<td>1.2</td>
<td>$6 \times 10^{-12}$</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>$3 \times 10^{-9}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>restart</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(check)</td>
<td>21</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>2.0</td>
<td>$1 \times 10^{-10}$</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>.57</td>
<td>$2 \times 10^{-9}$</td>
<td></td>
</tr>
<tr>
<td>restart</td>
<td></td>
<td></td>
<td>mxstep, 1 vector found</td>
</tr>
<tr>
<td>(check)</td>
<td>15</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>3.1</td>
<td>terminate</td>
</tr>
</tbody>
</table>

Example 7.4A, B. The two eigenvalues are much closer together this time. This slows the convergence rate for LANSO somewhat.

Same as 7.4A, A except $\lambda_{n-1} = -.0001$

<table>
<thead>
<tr>
<th>matrix-vector products</th>
<th>vector inner products</th>
<th>max error in eigenvalues</th>
<th>max residual norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAN</td>
<td>184</td>
<td>$7 \times 10^{-12}$</td>
<td>$2 \times 10^{-6}$</td>
</tr>
<tr>
<td>LANSO</td>
<td>156</td>
<td>$3 \times 10^{-14}$</td>
<td>$2 \times 10^{-7}$</td>
</tr>
</tbody>
</table>
History of LANSO (mxstep = 50)

<table>
<thead>
<tr>
<th>pause at j =</th>
<th></th>
<th>new κ</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2.7</td>
<td>1 × 10^{-12}</td>
</tr>
<tr>
<td></td>
<td>46</td>
<td>1.6</td>
<td>7 × 10^{-11}</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>3 × 10^{-8}</td>
<td>mxstep</td>
</tr>
<tr>
<td>restart</td>
<td>20</td>
<td>1.0</td>
<td>1 × 10^{-11}</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>.98</td>
<td>1 × 10^{-10}</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>3.2 × 10^{-2}</td>
<td>mxstep</td>
</tr>
<tr>
<td>restart</td>
<td>20</td>
<td>2.3</td>
<td>4 × 10^{-7}</td>
</tr>
<tr>
<td>(check)</td>
<td>30</td>
<td>1.0</td>
<td>2 vectors found</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>2.5</td>
<td>terminate</td>
</tr>
</tbody>
</table>

Example 7.4A, C. With the two eigenvalues equal the convergence is about the same as for BLAN. Setting mxstep = 100 would improve LANSO.

Same as 7.4A, A except λ_{n-1} = 0.

<table>
<thead>
<tr>
<th></th>
<th>matrix-vector products</th>
<th>vector inner products</th>
<th>max error in eigenvalues</th>
<th>max residual norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAN</td>
<td>184</td>
<td>*</td>
<td>2 × 10^{-12}</td>
<td>2 × 10^{-6}</td>
</tr>
<tr>
<td>LANSO</td>
<td>186</td>
<td>490</td>
<td>1 × 10^{-14}</td>
<td>2 × 10^{-7}</td>
</tr>
</tbody>
</table>

History of LANSO (mxstep = 50)

<table>
<thead>
<tr>
<th>pause at j =</th>
<th></th>
<th>new κ</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2.7</td>
<td>1 × 10^{-12}</td>
</tr>
<tr>
<td></td>
<td>46</td>
<td>1.7</td>
<td>7 × 10^{-11}</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>3 × 10^{-8}</td>
<td>mxstep</td>
</tr>
<tr>
<td>restart</td>
<td>20</td>
<td>1.5</td>
<td>2 × 10^{-8}</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>3.9</td>
<td>3 × 10^{-7}</td>
</tr>
<tr>
<td></td>
<td>43</td>
<td>1.1</td>
<td>7 × 10^{-6}</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>.26</td>
<td>mxstep, 1 vector found</td>
</tr>
<tr>
<td>restart</td>
<td>15</td>
<td>2.5</td>
<td>.1 found</td>
</tr>
<tr>
<td>restart</td>
<td>27</td>
<td>3.0</td>
<td>1 × 10^{-12}</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>1.3</td>
<td>1 × 10^{-10}</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>3 × 10^{-7}</td>
<td>mxstep</td>
</tr>
<tr>
<td>restart</td>
<td>20</td>
<td>1.9</td>
<td>other 0 found, terminate</td>
</tr>
</tbody>
</table>
14. The Kappa Bound.*** Let \( \kappa_1 \) be an upper bound on the error committed in normalizing an \( n \)-vector, so

\[
|1 - \| q_j \|_2^2 | \leq \kappa_1, \quad \text{for all } j.
\]

\( \kappa_1 \) will depend on the arithmetic unit, the square root routine and other details of the program. Suppose that numbers \( \xi_j \) are known such that

\[
\| Q_j^* q_{j+1} \| \leq \xi_j, \quad \text{for all } j.
\]

Then define \( \kappa_{j+1} \) by

\[
\kappa_{j+1} = \sqrt{\frac{\| \xi_j \|^2 + 4 \xi_j^2}{\kappa_j + \kappa_1 + \sqrt{(\kappa_j - \kappa_1)^2 + 4 \xi_j^2}}}/2.
\]

**Lemma 1.** If \( \| 1 - Q_j^* Q_j \| \leq \kappa_j \) then \( \| 1 - Q_{j+1}^* Q_{j+1} \| \leq \kappa_{j+1} \).

**Proof.**

\[
\| 1 - Q_{j+1}^* Q_{j+1} \| \leq \left\| \begin{array}{cc} \| 1 - Q_j^* Q_j \| & \| -Q_j^* q_{j+1} \| \\ \| -q_{j+1}^* Q_j \| & \| 1 - q_{j+1}^* q_{j+1} \| \end{array} \right\|.
\]

In order to compute \( \kappa_j \), we must first compute \( \xi_j \). The vector \( q_{j+1} \) is obtained by dividing \( r_j \) of (4.1) by \( \beta_j \). Hence,

\[
\| Q_j^* q_{j+1} \| \leq \| Q_j^* g_j \| /\beta_j + \| Q_j^* g_j \|,
\]

where \( g_j \) accounts for the error introduced by the division by \( \beta_j \). \( g_j \) is always small and satisfies \( \| g_j \| < \epsilon \), where \( \epsilon \) is the relative machine precision. To bound \( \| Q_j^* r_j \| \) we first prove the following result.

**Lemma 2.**

\[
Q_j^* r_j = [(1 - Q_j^* Q_j) T_j - (1 - e_j e_j^*) T_j (1 - Q_j^* Q_j)] e_j - \bar{F}^* q_{j+1}
\]

\[
+ (q_j^* A g_j - \alpha_j) e_j + Q_j^* \bar{f}_j.
\]

**Proof.** Recall Eq. (4.1), namely

\[
A Q_j = Q_j T_j + r_j e_j^* - \bar{F}_j.
\]

*** The three lemmas are taken from the unpublished report (Kahan and Parlett [1974]).
Then we have
\[ Q^*r_i = Q^*(AQ_i - Q_jT_j + \bar{F}_j)e_j, \]
using (5),
\[ = (AQ^*_iQ_j - Q^*_iQ_jT_j)e_j + Q^*_i\bar{F}_j, \]
where \( \bar{F}_j = F_j e_j \),
\[ = [(AQ^*_i)Q_j - Q^*_iQ_jT_j]e_j + Q^*_i\bar{F}_j, \]
using \( A^* = A \),
\[ = [T_jQ^*_jQ_j - \bar{F}_j Q_j + e_j Q_j - Q^*_j Q_jT_j]e_j + Q^*_j\bar{F}_j, \]
using (5) again,
\[ = [ - T_j(1 - Q^*_j Q_j) + (1 - Q^*_j Q_j)T_j - F^*_j Q_j]e_j + e_j Q_j + Q^*_j\bar{F}_j, \]
adding and subtracting \( T_j \).

Finally, using the second line above,
\[ e_j Q^*_j r_j = e_j(Q^*_j A Q_j - Q^*_j Q_j T_j)e_j \]
\[ = q^*_j A q_j - e_j Q^*_j Q_j T_j e_j \]
\[ = q^*_j A q_j - \alpha_j + e_j(1 - Q^*_j Q_j)T_j e_j. \]

After transposing, substituting, and rearranging terms the lemma's assertion is obtained.

The expansion for \( Q^*_j r_j \) falls into two parts \( Q^*_j r_j = c_j + d_j \), where
\[ c_j = [(1 - Q^*_j Q_j)T_j - (1 - e_j e^*_j)T_j(1 - Q^*_j Q_j)]e_j, \]
\[ d_j = -F^*_j q_{j+1} + q^*_j A q_j - \alpha_j)e_j + Q^*_j\bar{F}_j. \]

Using the stable implementation of the algorithm analyzed by Paige [1976], \( d_j \) is always tiny. In practice, it has been found that the contribution of \( d_j \) to \( Q^*_j r_j \) can be ignored completely in computing the bound \( \xi_j \).

**Lemma 3.** Let \( \|Q^*_i q_{i+1}\| \leq \xi_i \) for \( i < j \). Then
\[ \|q_j\| \leq \|T_{j-1} - 1\| \xi_{j-1} + \beta_{j-1}(\xi_{j-1} + \xi_{j-2} + 2\kappa_1) + \alpha_j k_1. \]

**Proof.** Partition \( T_j \) and \( 1 - Q^*_j Q_j \) to find
\[ (1 - Q^*_j Q_j)T_j e_j = \left[ \begin{array}{c} -Q^*_{j-1}q_1 \\ 1 - \|q_j\|^2 \end{array} \right] \alpha_j + \left[ \begin{array}{c} -Q^*_{j-1}q_{j-1} \\ 1 - \|q_{j-1}\|^2 \end{array} \right] \beta_{j-1}, \]
\[ T_j(1 - Q^*_j Q_j)e_j = \left[ \begin{array}{c} -T_{j-1} Q^*_{j-1} q_j + e_{j-1} \beta_{j-1}(1 - \|q_j\|^2) \\ \alpha_j(1 - \|q_j\|^2) - \beta_{j-1} q_{j-1} q_j \end{array} \right]. \]

The factor \( (1 - e_j e^*_j) \) simply annihilates the bottom element. Moreover, by (1),
\[ 1 - \|q_j\|^2 \leq \kappa_1, \quad \text{and} \quad |q^*_j q_{j-1}| = |q^*_j q_j| \leq \xi_{j-1}. \]

The bound is obtained by collecting terms. \( \square \)

Finally, to compute numbers we need a value for \( \kappa_1 \). It can be shown that \( \kappa_1 = (n + 6)e \) will do. We also use \( \|T_{j-1} - \alpha_j\|_{\infty} \) as an upper bound on \( \|T_{j-1} - \alpha_j\| \).
After the program has made a pause, it is necessary to reset the kappa recurrence in order to use it to determine the next pause. To do this we make use of Theorem 5 (Section 4) which states

\[ y_j q_{j+1} = g_j / \hat{\beta}_j. \]

(7)

The quantities \( \hat{\beta}_j \) are available at a pause. A current estimate of \( \|A\| (= \max |\theta|) \) is also available. An estimate of \( \xi_j \) is obtained from the formula

\[ \xi_j \approx \epsilon \|A\| / \hat{\beta}_j. \]

(8)

where \( \hat{\beta}_j \) is the minimum of the \( \beta_j \) above the tolerance. To restart the recurrence \( \xi_{j-1} \) and \( \kappa_j \) are set to this value as well.

To avoid a search over all of \( T_{j-1} \) to compute \( \|T_{j-1} - \alpha_j\|_\infty \), the program maintains values for AMAX, AMIN, and BMAX which are, respectively, the largest \( \alpha \), the smallest \( \alpha \), and the largest beta from \( T_{j-1} \). These values permit the program to compute the right-hand side of

\[ \|T_{j-1} - \alpha_j\| \leq \max \{ \text{AMAX} - \alpha_j, \alpha_j - \text{AMIN} \} + 2 \times \text{BMAX}. \]

(9)

Further, if the program pauses at step \( j \), the eigenvalues of \( T_j \) must be computed. If TLARGE and TSMALL are the largest and smallest eigenvalues of \( T_j \), respectively, then

\[ \|T_{k-1} - \alpha_k\| \leq \max \{ \text{max(TLARGE} - \alpha_k, \alpha_k - \text{TSMALL}) \|T_{k-1} - \alpha_k\| \} + \beta_j, \]

(10)

where

\[ T'_{k-1} = \begin{bmatrix} \alpha_{j+1} & \beta_{j+1} \\ \beta_{j+1} & \ddots \\ & \ddots & \beta_{k-2} \\ & & \beta_{k-2} & \alpha_{k-1} \end{bmatrix}, \]

and \( \|T'_{k-1} - \alpha_k\| \) is estimated as in (9).


