IRBL: AN IMPLICITLY RESTARTED BLOCK LANCZOS METHOD FOR LARGE-SCALE HERMITIAN EIGENPROBLEMS

J. BAGLAMA*, D. CALVETTI†, AND L. REICHEL‡

Abstract. The `irbleigs` code is an implementation of an implicitly restarted block-Lanczos method for computing a few selected nearby eigenvalues and associated eigenvectors of a large, possibly sparse, Hermitian matrix A. The code only requires the evaluation of matrix-vector products with A; in particular, factorization of A is not demanded, nor is the solution of linear systems of equations with the matrix A. This, together with a fairly small storage requirement, makes the `irbleigs` code well suited for large-scale problems. Applications of the `irbleigs` code to certain generalized eigenvalue problems and to the computation of a few singular values and associated singular vectors are also discussed. Numerous computed examples illustrate the performance of the method and provide comparisons with other available codes.

Key words. block Lanczos method, eigenvalue computation, singular value computation, polynomial acceleration.

AMS subject classifications. 65F15, 65F10, 65F20

1. Introduction. This paper discusses the performance and some implementation issues of a new MATLAB code for the computation of a few eigenvalues and associated eigenvectors of a large sparse Hermitian $n \times n$ matrix A with real- or complex-valued elements. The code can be applied to compute a few of the largest eigenvalues, a few of the smallest eigenvalues or a few eigenvalues in the vicinity of a specified point on the real axis. In addition, the code can be used to compute a few eigenvalue-eigenvector pairs of certain large generalized eigenvalue problems, or to determine a few singular values and associated singular vectors of a general matrix. The order n of the matrix A is assumed to be so large, that its factorization is not feasible. A user only has to provide computer code for the evaluation of matrix-vector products with the matrix A; in particular, the matrix does not have to be stored. Functions or subroutines for the evaluation of matrix-vector products can be written in MATLAB, FORTRAN or C. The eigenvalue code typically only requires the storage of a few n-vectors, in addition to storage of the computed eigenvectors. The fairly small storage requirement makes it possible to compute eigenvalue-eigenvector pairs of large matrices on personal computers.

The MATLAB code, denoted by `irbleigs`, implements an Implicitly Restarted Block Lanczos (IRBL) method. This method generalizes the implicitly restarted Lanczos method, which was first described in [9, 41]. The `irbleigs` code is available from the authors' web sites. Advantages of this code, compared with implementations of the (standard) Lanczos or block-Lanczos algorithms, include smaller storage requirement and the possibility of computing eigenvalues in the interior of the spectrum without factoring the matrix A.

The `irbleigs` code has been developed in several steps. An implicitly restarted

*Department of Mathematics, University of Rhode Island, Kingston, RI 02881. E-mail: `jbaglama@math.uri.edu`. Home page: http://hypatia.math.uri.edu/∼jbaglama.

†Department of Mathematics, Case Western Reserve University, Cleveland, OH 44106. E-mail: `dxc57@po.cwru.edu`. Home page: http://lanczos.math.cwru.edu/∼dxc57. Research supported in part by NSF grants DMS-9806702 and DMS-0107841.

‡Department of Mathematical Sciences, Kent State University, Kent, OH 44242. E-mail: `reichel@math.kent.edu`. Home page: http://reichel.mcs.kent.edu/∼reichel. Research supported in part by NSF grants DMS-9806413 and DMS-0107858.

Lanczos method that can be used to compute extreme eigenvalues or a few eigenvalues in the vicinity of a user-specified point on the real axis was presented in [2]. However, we found that when there are multiple or very close eigenvalues a block-version of the code performs better. Therefore, an implicitly restarted block-Lanczos method was developed and described in [4], where also an application to liquid crystal modeling was discussed. This application gives rise to large-scale path-following problems. Eigenvalues and associated eigenvectors of large Jacobian matrices are determined in order to detect turning points and bifurcation points. The null space of a Jacobian matrix at a bifurcation point yields information relevant for deciding how to follow the paths across the bifurcation point. Our wish to carry out path-following interactively made it desirable to perform the computations on a workstation. The limited amount of fast computer memory available on a workstation and the large sizes of the Jacobian matrices that arise in this application made it necessary to develop a code that does not demand the factorization of the Jacobian matrices, and only requires the storage of very few n-vectors, in addition to the computed eigenvectors. Further discussion on numerical methods for large-scale bifurcation problems based on the implicitly restarted block-Lanczos method can be found in [4, 7, 8].

The implicitly restarted block-Lanczos algorithm discussed in [4] is designed for the computation of a few extreme eigenvalues and associated eigenvectors, but can not be applied to determine eigenvalues in the vicinity of an arbitrary point on the real axis. The code discussed in the present paper removes this restriction by applying a judiciously chosen acceleration polynomial.

The implicitly restarted Lanczos method is analogous to the implicitly restarted Arnoldi method, which was proposed by Sorensen [41] and has been further developed by Lehoucq, Sorensen and Yang [21, 23, 25, 42]; ARPACK, a set of FORTRAN subroutines that implements the implicitly restarted Arnoldi method, is described in [25]. MATLAB, version 6.0, makes this code available through the function `eigs`. An implicitly restarted block-Arnoldi method has recently been described in [22].

ARPACK is designed for the computation of a few eigenvalues and eigenvectors of a large nonsymmetric matrix, but can be applied to symmetric matrices as well. By focusing on symmetric eigenvalue problems, we have been able to develop a code that is more reliable than ARPACK and typically requires less computer storage. This is illustrated by computed examples reported in the present paper. Related examples can be found in [2, 9].

We remark that when the block-size is chosen to be one in the `irbleigs` code, the method simplifies to an implicitly restarted Lanczos method. We have found that choosing the block-size larger than one gives faster convergence if the desired eigenvalues are of multiplicity larger than one or are very close. This is illustrated in Section 5.

When a suitable preconditioner for A is known, the Davidson method and extensions thereof can be competitive for the computation of a few eigenvalues and associated eigenvectors; see Murray et al. [31] and Sleijpen and van der Vorst [40] for descriptions of such methods. Experiments comparing our MATLAB code `irbleigs` with the Jacobi-Davidson method by Sleijpen and van der Vorst [40] are presented in Section 5.

This paper is organized as follows. Section 2 reviews the block-Lanczos method, develops the recursion formulas for the IRBL method and discusses our strategy for handling singular blocks. Section 3 is concerned with the choice and computation of the acceleration polynomial. Section 4 outlines variants of the IRBL method for

the solution of certain generalized eigenvalue problem and for the computation of a few singular values and associated singular vectors of a large general matrix, and Section 5 presents numerous computed examples that illustrate the performance of the `irbleigs` code and compare the code to several other available methods. Concluding remarks can be found in Section 6.

2. The IRBL method. Let $\{v_j\}_{j=1}^r$ be a given set of orthonormal n-vectors, and introduce the matrix $V_r = [v_1, v_2, \ldots, v_r]$. Define the Krylov subspace

$$\mathcal{K}_{mr}(A, V_r) := \text{span}\{V_r, AV_r, A^2V_r, \ldots, A^{m-1}V_r\}. \tag{2.1}$$

Application of m steps of the block-Lanczos method with initial matrix $V_r \in \mathbb{C}^{n\times r}$ yields the block-Lanczos decomposition

$$AV_{mr} = V_{mr}T_{mr} + F_rE_r^*, \tag{2.2}$$

where $V_{mr} \in \mathbb{C}^{n\times mr}$, $V_{mr}I_{mr\times r} = V_r$, $V_{mr}^*V_{mr} = I_{mr}$ and $F_r \in \mathbb{C}^{n\times r}$ satisfies $V_{mr}^*F_r = 0$. Here $I_{mr} \in \mathbb{R}^{mr\times mr}$ denotes the identity matrix, the matrix $I_{mr\times r} \in \mathbb{R}^{mr\times r}$ consists of the first r columns of I_{mr} and the matrix $E_r \in \mathbb{R}^{mr\times r}$ consists of the last r columns of I_{mr}. The superscript $*$ denotes transposition and, when applicable, complex conjugation. Finally,

$$T_{mr} = V_{mr}^*AV_{mr} \tag{2.3}$$

is an $mr \times mr$ Hermitian block-tridiagonal matrix of the form

$$T_{mr} = \begin{bmatrix} D_1 & B_1^* & & & & 0 \\ B_1 & D_2 & B_2^* & & & \\ & B_2 & D_3 & B_3^* & & \\ & & & \ddots & & \\ & & & \ddots & \ddots & B_{m-1}^* \\ 0 & & & & B_{m-1} & D_m \end{bmatrix}, \tag{2.4}$$

with Hermitian diagonal blocks $D_j \in \mathbb{C}^{r\times r}$ and nonsingular upper triangular subdiagonal blocks $B_j \in \mathbb{C}^{r\times r}$. It follows from (2.2) that the range of V_{mr} is the Krylov subspace (2.1). We refer to the columns of the matrix V_{mr} as Lanczos vectors.

We have tacitly assumed that the initial matrix V_r and the matrix A allow the block-Lanczos decomposition (2.2) with the stated properties to be computed. At the end of this section, we will discuss how to handle the situation when this is not the case. Until then, we assume that the block-Lanczos decomposition (2.2) exists.

Let $\{\theta, y\}$ be an eigenvalue-eigenvector pair of the matrix T_{mr} and define the vector $x := V_{mr}y$. Then θ and x are commonly referred to as a Ritz value and a Ritz vector of A, respectively. It follows from (2.2) that the residual error $Ax - x\theta$ associated with the Ritz pair $\{\theta, x\}$ satisfies

$$\|Ax - x\theta\| = \|(AV_{mr} - V_{mr}T_{mr})y\| = \|F_rE_r^*y\|. \tag{2.5}$$

Throughout this paper $\|\cdot\|$ denotes the Euclidean vector norm as well as the associated induced matrix norm. Thus, the norm of the residual error can be computed without explicitly computing the Ritz vector x by evaluating the right-hand side of (2.5).

When the norm (2.5) is small, the Ritz value θ is an accurate approximation of an eigenvalue of A.

In block-Lanczos methods that do not employ restarts, see, e.g., Chatelin [10, Section 6.4], Parlett [34, Chapter 13] or Ruhe [36] as well as [5] for discussions of such methods, the number of Lanczos steps m is increased until the right-hand side of (2.5) is sufficiently small. Then the Ritz pair $\{\theta, x\}$ of A is computed and used as an approximate eigenpair of A. However, this approach may require the use of secondary computer storage when the matrix A is very large, because of the storage requirement of the matrix V_{mr}. The use of secondary computer storage typically increases the computational time significantly. To avoid using secondary storage, the block-Lanczos algorithm can be restarted periodically. The IRBL method is an implementation of a restarted block-Lanczos method, which allows the application of a judiciously chosen acceleration polynomial.

Another approach to reducing the computer storage required, and thereby avoiding the use of secondary computer storage, is to discard all but the most recently computed Lanczos vectors. The discarded Lanczos vectors have to be recomputed when determining the eigenvectors. Since Lanczos vectors are discarded, it is difficult to maintain orthogonality of all the Lanczos vectors computed in the presence of round-off errors. Loss of orthogonality of the Lanczos vectors may lead to the computation of spurious eigenvalues. Lanczos method of this kind are discussed by Cullum and Willoughby [11].

Assume that the block-Lanczos decomposition (2.2) has been computed by m steps of the block-Lanczos algorithm, and let m be the largest number of block-Lanczos steps that we wish to carry out between restarts. Let the residual error (2.5) be larger than a specified tolerance for the Ritz values of interest. We then apply recursion formulas derived in [4] to compute the matrix

$$U_r := p_m(A)V_r, \tag{2.6}$$

where p_m is a polynomial of degree m, to be specified below. We refer to p_m as an acceleration polynomial. Given the block-Lanczos decomposition (2.2), the matrix U_r can be computed without the evaluation of matrix-vector products with A; see [4] for details. Orthogonalization of the columns of U_r yields the matrix V_r^+; thus,

$$U_r = V_r^+ R_r^+, \qquad V_r^+ \in \mathbb{C}^{n\times r}, \qquad R_r^+ \in \mathbb{C}^{r\times r}, \tag{2.7}$$

where $(V_r^+)^* V_r^+ = I_r$ and R_r^+ is upper triangular.

The computations that determined the matrix V_r^+ from V_r are now repeated with the matrix V_r^+ replacing V_r. Thus, application of m steps of the block-Lanczos method to A with initial block V_r^+ yields the block-Lanczos decomposition

$$AV_{mr}^+ = V_{mr}^+ T_{mr}^+ + F_r^+ E_r^*. \tag{2.8}$$

If the desired Ritz values have not been determined with sufficient accuracy by this decomposition, then a new acceleration polynomial p_m^+ of degree m is chosen and the matrix

$$U_r^+ := p_m^+(A)V_r^+ \tag{2.9}$$

is evaluated by using recursion formulas described in [4], without evaluation of matrix-vector products with the matrix A. Combining (2.6) and (2.9) yields

$$U_r^+ = p_m^+(A)p_m(A)V_r(R_r^+)^{-1}. \tag{2.10}$$

Orthogonalization of the columns of U_r^+ now gives the matrix V_r^{++}. New block-Lanczos decompositions are evaluated in this manner until approximations of all desired eigenvalues, as well as associated eigenvectors, have been computed with sufficient accuracy.

The performance of the IRBL method crucially depends on the choice of the sequence of acceleration polynomials $p_m, p_m^+, \ldots$. These polynomials are determined by specifying their zeros. Let the polynomial ψ_{mk} be the product of the k first acceleration polynomials, each one of degree m, and let $z_1, z_2, \ldots, z_{mk}$ denote the zeros of ψ_{mk}. After k evaluations of block-Lanczos decompositions of the forms (2.2) and (2.8), we have, analogously to (2.10),

$$V_r' = \psi_{mk}(A) V_r \hat{R}_r^{-1}, \tag{2.11}$$

where $V_r' \in \mathbb{C}^{n\times r}$ has orthogonal columns, $\hat{R}_r \in \mathbb{C}^{r\times r}$ is upper triangular and

$$\psi_{mk}(z) = \prod_{j=1}^{mk} (z - z_j). \tag{2.12}$$

We also refer to the polynomial ψ_{mk} as an acceleration polynomial. Sorensen [41] refers to the zeros z_j as shifts, because they are shifts in a truncated QR-algorithm used to evaluate the matrices U_r and U_r^+ in (2.6) and (2.9).

The choice of acceleration polynomial ψ_{mk}, or equivalently the choice of zeros z_j, in the `irbleigs` code depends on whether we would like to compute a few of the smallest or largest eigenvalues of A, or a few eigenvalues in a neighborhood of a specified point on the real axis. The zeros should be chosen so that the acceleration polynomial ψ_{mk} is of large magnitude in the vicinity of the eigenvalues that we wish to compute and of small magnitude at the other eigenvalues of A. We discuss the choice of zeros in Section 3.

Recently, Gupta [19] proposed a related approach for computing eigenpairs of a symmetric matrix. Gupta [19] first applies an acceleration polynomial of fairly high degree to one or several initial vectors by using the recursion formulas of nonstationary Richardson iteration, and then uses the Lanczos or block-Lanczos method to determine approximations of desired eigenpairs. The degree of the acceleration polynomial and the number of steps of the Lanczos or block-Lanczos methods are chosen so as to minimize the computational work required under certain assumptions on the distribution of the eigenvalues. The method typically requires more computer storage than the `irbleigs` code because, generally, more consecutive Lanczos or block-Lanczos steps are carried out.

Assume for the moment that the subdiagonal blocks of the block-tridiagonal matrix T_{mr} in (2.2) with $r \times r$ blocks are nonsingular. Then the eigenvalues of the matrix T_{mr} are of multiplicity at most r; see, e.g., the proof of Proposition 3.1 below. If some of the desired eigenvalues of A are of multiplicity $\ell > r$, then special care has to be taken so that the `irbleigs` code will detect all eigenvalues of multiplicity ℓ. The implicitly restarted block-Lanczos algorithm described in [4] introduced random vectors, orthogonalized against converged eigenvectors and the other vectors of V_r, as columns of the initial block V_r. The algorithm was restarted with this initial block V_r. Convergence of a Ritz value towards an already determined eigenvalue then showed that the proper invariant subspace of that eigenvalue had not yet been determined. The algorithm was restarted with new random columns in the initial block until no Ritz value converged to one of the desired eigenvalues. We found this approach to reliably approximate eigenvalues of multiplicity, say $j > 1$, by sets of j close

or identical eigenvalues in many numerical experiments. However, this approach is quite expensive; often many matrix-vector product evaluations are required to make the random vector converge to an eigenvector. Therefore, the `irbleigs` code also gives the user the possibility to continue the computations with a smaller tolerance in the convergence criterion, after the original convergence criterion is satisfied. This approach has often determined the correct multiplicity of multiple eigenvalues for a large number of test problems. It can be motivated heuristically as follows. Assume that we wish to compute the smallest $r+1$ eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_{r+1}$ of A, where $\lambda_1 = \lambda_2 = \ldots = \lambda_{r+1}$. When the tolerance in the stopping criterion is small, i.e., when the value of the parameter ϵ in (2.13) below is small, then generally only a few Ritz vectors associated with these eigenvalues are determined at a time. This allows the initial blocks V_r in subsequently generated Krylov subspaces to become rich in basis vectors of the invariant subspace associated with the desired eigenvalues that have not been determined yet.

The criteria for accepting Ritz values and Ritz vectors differ in general. The Ritz value θ is accepted as an approximate eigenvalue of A when the residual error (2.5) is smaller than a prescribed tolerance, i.e., when

$$\|F_r E_r^* y\| \leq \epsilon \|A\|, \qquad x = V_{mr} y, \tag{2.13}$$

for a user-specified value of ϵ. The value of $\|A\|$ in the bound (2.13) is approximated by the eigenvalue of largest magnitude of all symmetric tridiagonal matrices T_{mr} computed so far. The acceptance criterion for Ritz vectors generally is more stringent in order to avoid that subsequently generated Krylov spaces are orthogonalized against poor eigenvector approximations. In order to accept a Ritz vector as an approximate eigenvector against which subsequent Krylov subspaces will be orthogonalized, the bound (2.13) has to be satisfied by the Ritz pair with ϵ equal to the minimum of the square-root of machine epsilon and the user supplied value of ϵ. When such a Ritz pair has been found, and an approximation of the eigenvalue already is available, we keep the most accurate of the available eigenvalue approximations.

All computations in our implementation of the block-Lanczos method are performed block-wise in order to take full advantage of Level 3 BLAS for matrix-matrix multiplication; see [13] for a discussion of these subroutines. To secure that the columns of the matrix V_{mr} in (2.2) are numerically orthonormal, as well as numerically orthogonal against already computed eigenvectors, they are reorthogonalized. The orthonormality of the columns of V_{mr} and their orthogonality against already computed eigenvectors prevents convergence of different Ritz vectors towards the same vector and convergence of Ritz vectors towards already computed eigenvectors. Since the number of columns of V_{mr} typically is fairly small, the computational cost of reorthogonalization is not large. However, it may be advantageous to implement partial or selective reorthogonalization when the number of vectors to reorthogonalize is larger; see Parlett [33, 34] for discussions of these techniques. Recently, Larsen posted a MATLAB code for computing a Lanczos decomposition (2.2) with block-size $r = 1$ using partial reorthogonalization [20]. This code does not employ restarts and is therefore not suited for very large problems. For instance, when we applied Larsen's code to the eigenvalue problem in Example 1 of Section 5, an out-of-memory error caused the computations to be terminated before any eigenvalues had been found. We therefore do not include this code in the comparison of restarted Lanczos methods reported in Section 5.

So far, we have assumed that the triangular subdiagonal blocks of the matrix T_{mr}

are nonsingular. However, the block-Lanczos method may generate a singular subdiagonal block in, say, step $\ell \leq m$. This indicates that the Krylov subspace $\mathcal{K}_{\ell r}(A, V_r)$ is of dimension strictly smaller than ℓr. When the block-size r is one, a singular subdiagonal block signals that an invariant subspace has been found. However, this may not be the case when the block-size r is strictly larger than one. In the latter case, we replace each linearly dependent vector in the Krylov subspace by a random vector. Specifically, assume that a diagonal entry in the ℓth subdiagonal $r \times r$ block, with $r > 1$, is smaller than a prescribed tolerance. Then this entry is set to zero and the corresponding column of the matrix $V_{\ell r}$ is chosen to be a random unit vector that is orthogonal against all other columns of $V_{\ell r}$ and all computed eigenvectors. Now $m - \ell$ steps of the block-Lanczos method are carried out until a decomposition of the form (2.2) has been determined. If this decomposition yields an acceptable approximate eigenpair, then this pair is stored and an acceleration polynomial of degree m is applied in the same fashion as described above. If on the other hand the block-Lanczos decomposition does not determine an approximate eigenpair with sufficient accuracy, then straightforward application of an acceleration polynomial and application of m block-Lanczos steps yields a new singular subdiagonal block, at least in exact arithmetic. In the presence of round-off errors, we may obtain a singular or nearly singular subdiagonal block. In order to avoid computations with such blocks, we identify the vector in the initial block V_r that gives rise to the singular or nearly singular subdiagonal block, and replace this vector by a unit random vector, which is orthogonal to the other columns of V_r as well as to already converged eigenvectors; see [1] for further details.

Assume that we already have computed ℓ of k desired eigenpairs, and that we are to apply the block-Lanczos method to the matrix $V_r \in \mathbb{C}^{n \times r}$ with orthonormal columns to determine the remaining $k - \ell$ wanted eigenvalues. The eigenvectors already found have to be stored, and in order not to increase the demand of computer storage significantly, we apply only $m - j$ steps of the block-Lanczos algorithm, where j is the unique positive integer, such that $(j-1)r < \ell \leq jr$, unless this bound yields $m - j = 1$ in which case we set $j = m - 2$. Then the computed matrix $V_{(m-j)r}$ and the eigenvectors already found, together, require about the same storage as the matrix V_{mr} would have required. Having determined the matrix $V_{(m-j)r}$, an acceleration polynomial of degree $m - j$ is applied. This reduction in the number of block-Lanczos steps is appropriate when the number of consecutive block-Lanczos steps m is limited by the size of the available fast computer storage. The `irbleigs` code implements this reduction in the number of consecutive block-Lanczos steps as eigenpairs are determined when extreme eigenvalues of A are sought. The selection of acceleration polynomial when a few nonextreme eigenvalues are desired is more complicated when the number of block-Lanczos steps is varied, and for the latter kind of problems, the number of block-Lanczos steps taken after each restart is kept fixed.

3. Computation of the acceleration polynomial. The acceleration polynomial ψ_{mk}, defined by (2.12), determines which eigenpairs of A will be computed as well as the rate of convergence. The polynomial is defined by specifying its zeros z_j and it is applied by using the recurrence relations of the IRBL method, as described in Section 2. Ideally, we would like ψ_{mk} to be of magnitude one in the vicinity of the desired eigenvalues and to vanish at the other eigenvalues of A. When this is the case, the columns of the matrix V_r' defined by (2.11) are linear combinations of the desired eigenvectors, and an application of the block-Lanczos method with initial block V_r' yields the desired eigenpairs. In actual computations, we seek to determine an accel-

eration polynomial ψ_{mk} that is of large magnitude at the desired eigenvalues of A and of small magnitude at the other eigenvalues. This section discusses the construction of such a polynomial.

The zeros z_j of the acceleration polynomial ψ_{mk} are allocated on a set $\mathbb{K}$ that contains some of the undesired eigenvalues of A and none of the desired ones. For instance, if we wish to compute a few of the largest eigenvalues of A, then $\mathbb{K}$ is an interval on the real axis to the left of the desired eigenvalues. If, instead, a few of the smallest eigenvalues of A are desired, then $\mathbb{K}$ is a real interval to the right of the desired eigenvalues. When we wish to determine a few nonextreme eigenvalues, the set $\mathbb{K}$ generally consists of two real intervals, one on each side of the set of desired eigenvalues. First we discuss how to allocate the zeros z_j on a given set $\mathbb{K}$ and then we consider the choice of sets $\mathbb{K}$.

Assume that the zeros $z_1, z_2, \ldots, z_\ell$ already have been allocated. We then let the zeros $z_{\ell+1}, z_{\ell+2}, \ldots, z_{\ell+m}$ be approximate solutions of a sequence of m maximization problems. Specifically, for $j = 1, 2, \ldots, m$, we let the zero $z_{\ell+j}$ be an approximate solution of

$$w(z_{\ell+j}) \prod_{i=1}^{\ell+j-1} |z_j - z_i| = \max_{z\in\mathbb{K}} w(z) \prod_{i=1}^{\ell+j-1} |z - z_i|, \qquad z_{\ell+j} \in \mathbb{K}, \tag{3.1}$$

where w is a nonnegative weight function on the real axis to be defined below. The points $z_{\ell+1}, z_{\ell+2}, \ldots, z_{\ell+m}$ determined by (3.1) might not be unique. We call any sequence of points $z_{\ell+1}, z_{\ell+2}, \ldots, z_{\ell+m}$ determined in this manner Leja points for $\mathbb{K}$, because of their close relation to points investigated by Leja [26]. When $\mathbb{K}$ consists of one interval on which the points $z_1, z_2, \ldots, z_\ell$ already have been allocated, the new points $z_{\ell+1}, z_{\ell+2}, \ldots, z_{\ell+m}$ are distributed so that all the points $z_1, z_2, \ldots, z_{\ell+m}$ are distributed roughly like zeros of Chebyshev polynomials for the interval $\mathbb{K}$. The asymptotic distribution can be expressed in terms of the normal derivate of a certain Green's function for the complement in the complex plane of $\mathbb{K}$, and this characterization carries over to sets $\mathbb{K}$ that consist of two intervals; see [2, 26] for details.

The exact solution of the sequence of maximization problems (3.1) can be cumbersome when ℓ or m are large. Easily computable approximations of the Leja points are furnished by the fast Leja points introduced in [3]. The computation of s fast Leja points requires only $\mathcal{O}(s^2)$ arithmetic operations.

We turn to the choice of sets $\mathbb{K}$ in the `irbleigs` code, and consider the case when the k smallest eigenvalues and associated eigenvectors of the matrix A are desired, where $k \ll n$. The sets $\mathbb{K}$ are chosen analogously when we wish to determine the k largest eigenvalues and associated eigenvectors of A.

Enumerate the eigenvalues λ_j or A and θ_j of T_{mr} in increasing order,

$$\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n \tag{3.2}$$

and

$$\theta_1 \leq \theta_2 \leq \ldots \leq \theta_{mr}. \tag{3.3}$$

Then the following relation between the λ_j and θ_j holds.

PROPOSITION 3.1. *Assume that the subdiagonal blocks of the block-tridiagonal matrix T_{mr} with block-size r, defined by (2.2), are nonsingular, and let the eigenvalues θ_j of T_{mr} and λ_j of A be ordered according to (3.3) and (3.2), respectively. Then*

$$\lambda_j \leq \theta_j, \qquad 1 \leq j \leq mr, \tag{3.4}$$

$$\lambda_n \geq \theta_{mr}. \tag{3.5}$$

Moreover,

$$\lambda_j < \theta_{j+r}, \qquad 1 \le j \le (m-1)r. \tag{3.6}$$

Proof. The inequalities (3.4) and (3.5) follow from the minimax properties of the eigenvalues of A and T_{mr} and from (2.3). The fact that the subdiagonal blocks are nonsingular implies that $\mathrm{rank}(T_{mr} - \theta I_{mr}) \ge rm - r$ for all eigenvalues θ of T_{mr}, and therefore each eigenvalue of T_{mr} has multiplicity at most r. This observation and (3.4) show (3.6). □

Throughout this section we assume that the conditions of Proposition 3.1 hold. The parameter *sizint* of the `irbleigs` code determines the size of the interval $\mathbb{K}$. Let the number k of desired smallest eigenvalues of A and the integer *sizint* satisfy

$$k < (m-1)r, \qquad 1 \le sizint \le (m-1)r - k. \tag{3.7}$$

Then, by (3.6), the interval $\mathbb{K} = [\theta_{mr-sizint}, \theta_{mr}]$ does not contain any of the k smallest eigenvalues of the matrix A. We may therefore allocate zeros of the acceleration polynomial on $\mathbb{K}$. The smallest interval is obtained for $sizint$=1, which is the default value. Experience from numerous computed examples suggests that this value often yields the desired eigenpairs with least computational effort. However, for matrices A with a large condition number a value of *sizint* larger than unity sometimes gave faster convergence.

The set $\mathbb{K}$ is updated whenever a new block-tridiagonal matrix T_{mr} with nonsingular subdiagonal blocks is available. When the first such matrix has been computed, we define the endpoints of $\mathbb{K} = [a, b]$ by

$$a := \theta_{mr-sizint}, \qquad b := \theta_{mr}. \tag{3.8}$$

We let the m first zeros $z_1, z_2, \ldots, z_m$ be fast Leja points for $\mathbb{K}$ and determine a new matrix V_r^+ according to (2.7). Application of m block-Lanczos steps yields the block-Lanczos decomposition (2.8) with block-tridiagonal matrix T_{mr}^+. Denote the eigenvalues of the latter matrix also by θ_j and order them according to (3.3). The endpoints of the set $\mathbb{K} = [a, b]$ are then updated according to

$$a := \min\{a, \theta_{mr-sizint}\}, \qquad b := \max\{b, \theta_{mr}\} \tag{3.9}$$

or

$$a := \theta_{mr-sizint}, \qquad b := \max\{b, \theta_{mr}\}. \tag{3.10}$$

The `irbleigs` code allows a user to choose which pair of updating formulas, (3.9) or (3.10), to be applied. The formulas (3.9) yield a nested sequence of increasing intervals $\mathbb{K}$ and is used if the parameter *endpt* of the `irbleigs` code is set to MON. The updating formulas (3.10) allows the endpoint a closest to the desired eigenvalues to"float," i.e., to vary in a nonmonotonic fashion. These formulas are used when the parameter *endpt* is set to FLT.

The computations of approximations of Leja points in the `irbleigs` code is carried out as described above if the parameter *zertyp* is set to WL (for Weighted Leja points). The weight function w in (3.1) is

$$w(z) := |z - a|. \tag{3.11}$$

We have found in numerous computed examples that the IRBL method yields faster convergence with this weight function than with $w(z) := 1$ when approximate Leja points are determined as outlined above.

The code also provides another, simpler, way of generating zeros of the acceleration polynomial. This alternate approach is used when the parameter *zertyp* in the `irbleigs` code is set to ML, which stands for Mapped Leja points. Fast Leja points are generated for the interval $[-2, 2]$ with weight function $w(z) := 1$, and then mapped to intervals $\mathbb{K}$ by a linear transformation. When the sets $\mathbb{K}$ form nested intervals, these intervals typically converge to an interval, which we denote by $\hat{\mathbb{K}}$. As the number of mapped Leja points increases, their distribution will approximate that of zeros of Chebyshev polynomials for the set $\hat{\mathbb{K}}$. We have found that letting the zeros be mapped Leja points often gives faster convergence than if we let the zeros be zeros of the mth degree Chebyshev polynomial of the first kind for each set $\mathbb{K}$ generated. A numerical example where zeros of the acceleration polynomial ψ_{mk} are zeros of Chebyshev polynomials of degree m for each set $\mathbb{K}$ is reported in [4]. We have observed that mapped Leja points generated with parameter *endpt*=MON often give better performance of the `irbleigs` code than mapped Leja points with parameter *endpt*=FLT. We choose the interval $[-2, 2]$ when computing mapped Leja points, because this interval has capacity one (in the sense of potential theory) and therefore allows the generation of a large number of Leja points without overflow or underflow; see [3] for details.

We turn to the case when the sets $\mathbb{K}$ consist of two real intervals, $[a, b]$ and $[\tilde{b}, \tilde{a}]$, one on each side of the k desired eigenvalues. For definiteness, we assume that the k desired eigenvalues are in a vicinity of the origin and that $\tilde{b} < \tilde{a} < 0 < a < b$. The method of generating fast Leja points generalizes in a straightforward manner from sets that consist of one interval to sets that are made up of two intervals. Only the option *zertyp*=WL applies and the weight function in (3.1) is given by

$$w(z) := \begin{cases} |z - a|, & z \geq a, \\ |z - \tilde{a}|, & z \leq \tilde{a}. \end{cases}$$

The endpoint b is updated according to (3.9) and the endpoint $\tilde{b}$ is updated analogously,

$$\tilde{b} := \min\{\tilde{b}, \theta_1\}, \tag{3.12}$$

where θ_1 is the smallest Ritz value of the computed block-tridiagonal matrix T_{mr}; cf. (3.3).

We now consider the updating formulas for the endpoints $\tilde{a}$ and a of $\mathbb{K}$ closest to the wanted eigenvalues. These endpoints have to be chosen so that the interval $[\tilde{a}, a]$ does not contain any one of the k desired eigenvalues. An approach for achieving this when the block-size is one, based on the use of harmonic Ritz values of A, is described in [2]. Here we generalize this approach to block-size r larger than one.

In the Rayleigh-Ritz method for computing approximations of eigenvalues of A^{-1}, a matrix $P \in \mathbb{R}^{n \times \ell}$ is chosen and the eigenvalues of the generalized eigenvalue problem

$$P^* A^{-1} P y = \hat{\theta} P^* P y, \qquad y \in \mathbb{C}^\ell \backslash \{0\},$$

are considered approximations of eigenvalues of A^{-1}. The particular choice $P := AV_{mr}$ yields

$$V_{mr}^* A V_{mr} y = \hat{\theta} V_{mr}^* A^2 V_{mr} y, \qquad y \in \mathbb{C}^{mr} \backslash \{0\}, \tag{3.13}$$

and obviates the need to apply the matrix A^{-1}, or equivalently, to solve linear systems of equations with the matrix A. Using (2.2) and (2.3), we obtain

$$V_{mr}^* A^2 V_{mr} = (T_{mr}^* V_{mr}^* + E_r F_r^*)(V_{mr} T_{mr} + F_r E_r^*) = T_{mr}^2 + E_r F_r^* F_r E_r^* = T_{mr}^2 + E_r B_m^* B_m E_r^*, \tag{3.14}$$

where B_m is the upper triangular matrix in the QR-factorization of F_r. Substituting (2.3) and (3.14) into (3.13) yields the generalized eigenvalue problem

$$T_{mr} y = \hat{\theta}(T_{mr}^2 + E_r B_m^* B_m E_r^*) y. \tag{3.15}$$

Throughout this section we assume that the matrix T_{mr} is nonsingular. How this condition can be enforced is discussed below.

PROPOSITION 3.2. *Assume that the matrix T_{mr} in the block-Lanczos decomposition (2.2) is nonsingular. Then the eigenvalues $\hat{\theta}$ of the generalized eigenvalue problem (3.15) are real and satisfy $0 < |\hat{\theta}| < \infty$.*

Proof. Since T_{mr} is nonsingular and Hermitian, the matrix $T_{mr}^2 + E_r B_m^* B_m E_r^*$ is Hermitian positive definite. Let L denote its lower triangular Cholesky factor. The eigenvalues $\hat{\theta}$ of the generalized eigenvalue problem (3.15) are eigenvalues of the Hermitian nonsingular matrix $L^{-1} T_{mr} L^{-*}$ and therefore are real, nonvanishing and bounded.

An alternative proof, that does not use the Cholesky factor L, can be based on Theorem 8.7.1 and Corollary 8.7.2 in [17]. □

Substitute $\tilde{\theta} := 1/\hat{\theta}$ into (3.15) to obtain the generalized eigenvalue problem

$$(T_{mr}^2 + E_r B_m^* B_m E_r^*) y = \tilde{\theta} T_{mr} y. \tag{3.16}$$

The eigenvalues $\tilde{\theta}$ of (3.16) are referred to as harmonic Ritz values, because their reciprocal values are weighted averages of the reciprocal values of eigenvalues of A. Equation (3.16) is discussed by Morgan [30], Paige et al. [32] and in [2] when the block-size r is one.

Our interest in the harmonic Ritz values stems from that their distribution around the origin reveals how the eigenvalues of A are distributed in a vicinity of the origin. A nice recent survey of properties of harmonic Ritz values and their relation to Lehmann intervals is provided by Beattie [6].

THEOREM 3.3. *Assume that the matrix T_{mr} is nonsingular, and enumerate the harmonic Ritz values according to*

$$\tilde{\theta}_1 \leq \tilde{\theta}_2 \leq \ldots \leq \tilde{\theta}_\ell < 0 < \tilde{\theta}_{\ell+1} \leq \tilde{\theta}_{\ell+2} \leq \ldots \leq \tilde{\theta}_{mr}, \tag{3.17}$$

where ℓ is an integer, such that $0 \leq \ell \leq mr$. If $\ell > 0$, then the matrix A has at least j eigenvalues in the interval $[\tilde{\theta}_{\ell-j+1}, 0)$ for $j = 1, 2, \ldots, \ell$. Conversely, if $\ell < mr$, then A has at least j eigenvalues in the interval $(0, \tilde{\theta}_{\ell+j}]$ for $j = 1, 2, \ldots, mr - \ell$.

Proof. It follows from Proposition 3.2 that the harmonic Ritz values $\tilde{\theta}_j$ are nonvanishing. Therefore an index ℓ, such that (3.17) holds can be found. A proof of the relations between harmonic Ritz values and the eigenvalues of A, based on results by Lehmann, has recently been presented by Beattie [6, Section 3]. □

Before discussing the application of harmonic Ritz values to the determination of the endpoints $\tilde{a}$ and a of $\mathbb{K}$, we consider their computation. It is not necessary to solve the generalized eigenvalue problem (3.16) to compute the harmonic Ritz values. The following, simpler, approach to the computation of the harmonic Ritz values has

previously been discussed by Paige et al. [32] for the case when the block-size r is one.

Introduce the $(mr+r)\times(mr+r)$ Hermitian block-tridiagonal matrix

$$\tilde{T}_{mr+r} = \begin{bmatrix} D_1 & B_1^* & & & & & 0 \\ B_1 & D_2 & B_2^* & & & & \\ & B_2 & D_3 & B_3^* & & & \\ & & & \ddots & & & \\ & & \ddots & \ddots & B_{m-1}^* & & \\ & & & B_{m-1} & D_m & B_m^* & \\ 0 & & & & B_m & \tilde{D}_{m+1} & \end{bmatrix}, \tag{3.18}$$

whose leading principal $mr\times mr$ submatrix is given by (2.4), where B_m is the upper triangular matrix in the QR-factorization of F_r, and

$$\tilde{D}_{m+1} := B_m E_r^* (T_{mr})^{-1} E_r B_m^*. \tag{3.19}$$

Assume for the moment that T_{mr} is nonsingular. Then the matrix $\tilde{T}_{mr+r}$ easily can be determined from the block-Lanczos decomposition (2.2).

THEOREM 3.4. *Let the matrix T_{mr} be nonsingular. Then the nonvanishing eigenvalues of $\tilde{T}_{mr+r}$ are harmonic Ritz values of A.*

Proof. The proof generalizes a proof by Paige et al. [32] for block-size one to block-size r larger than one. Writing the generalized eigenvalue problem (3.16) in the form

$$(T_{mr}(T_{mr}-\tilde{\theta}I_{mr}) + E_r B_m^* B_m E_r^*)y = 0$$

shows that the zeros of the polynomial

$$p(\tilde{\theta}) = \det(T_{mr}(T_{mr}-\tilde{\theta}I_{mr}) + E_r B_m^* B_m E_r^*) \tag{3.20}$$

are the harmonic Ritz values. We now demonstrate that the characteristic polynomial of $\tilde{T}_{mr+r}$,

$$q(\tilde{\theta}) = \det(\tilde{T}_{mr+r} - \tilde{\theta}I_{mr+r}), \tag{3.21}$$

is divisible by $p(\tilde{\theta})$.

Consider the partitioning of a square matrix M into submatrices

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \tag{3.22}$$

with square diagonal blocks. Then

$$\det(M) = \begin{cases} \det(M_{11})\det(M_{22} - M_{21}M_{11}^{-1}M_{12}), & \text{if } \det(M_{11}) \neq 0, \\ \det(M_{22})\det(M_{11} - M_{12}M_{22}^{-1}M_{21}), & \text{if } \det(M_{22}) \neq 0. \end{cases} \tag{3.23}$$

Partitioning the matrix $\tilde{T}_{mr+r}$ according to (3.22) with $M_{11} = T_{mr}$, we obtain from (3.21) and (3.23) that, when $\tilde{\theta}$ is not an eigenvalue of T_{mr},

$$q(\tilde{\theta}) = \det(T_{mr}-\tilde{\theta}I_{mr})\det(\tilde{D}_{m+1} - \tilde{\theta}I_r - B_m E_r^*(T_{mr}-\tilde{\theta}I_{mr})^{-1}E_r B_m^*). \tag{3.24}$$

Substituting (3.19) into (3.24) and using the identity

$$T_{mr}^{-1} - (T_{mr} - \tilde{\theta} I_{mr})^{-1} = -\tilde{\theta}(T_{mr} - \tilde{\theta} I_{mr})^{-1} T_{mr}^{-1}$$

gives

$$(3.25) \quad q(\tilde{\theta}) = (-\tilde{\theta})^r \det(T_{mr} - \tilde{\theta} I_{mr}) \det(B_m E_r^* (T_{mr} - \tilde{\theta} I_{mr})^{-1} T_{mr}^{-1} E_r B_m^* + I_r).$$

Applying the determinant formulas (3.23) to the matrix

$$\begin{bmatrix} I_{mr} & -T_{mr}^{-1} E_r B_m^* \\ B_m E_r^* (T_{mr} - \tilde{\theta} I_{mr})^{-1} & I_r \end{bmatrix}$$

shows that

$$\begin{aligned} &\det(I_r + B_m E_r^* (T_{mr} - \tilde{\theta} I_{mr})^{-1} T_{mr}^{-1} E_r B_m^*) \\ &= \det(I_{mr} + T_{mr}^{-1} E_r B_m^* B_m E_r^* (T_{mr} - \tilde{\theta} I_{mr})^{-1}), \end{aligned}$$

and substituting this identity into (3.25) yields

$$\begin{aligned} q(\tilde{\theta}) &= (-\tilde{\theta})^r \det(T_{mr} - \tilde{\theta} I_{mr}) \det(T_{mr}^{-1}) \det(T_{mr} + E_r B_m^* B_m E_r^* (T_{mr} - \tilde{\theta} I_{mr})^{-1}) \\ &= (-\tilde{\theta})^r \det(T_{mr}^{-1}) \det(T_{mr}(T_{mr} - \tilde{\theta} I_{mr}) + E_r B_m^* B_m E_r^*). \end{aligned}$$

It now follows from (3.20) that

$$q(\tilde{\theta}) = (-\tilde{\theta})^r \det(T_{mr}^{-1}) p(\tilde{\theta}).$$

This identity is valid for all values of $\tilde{\theta}$ and completes the proof of the theorem. □

We are in a position to discuss the choice of sets $\mathbb{K}$. Thus, assume that we would like to determine k eigenvalues of A in the vicinity of the origin, as well as associated eigenvectors. For notational simplicity, we assume in the remainder of this section that k is even. The formulas presented have to be modified slightly when k is odd. Let the integer ℓ, determined by (3.17), satisfy $k/2 < \ell < mr - k/2$. Then it follows from Theorem 3.3 that the interval $[\tilde{\theta}_{\ell-k/2+1}, \tilde{\theta}_{\ell+k/2}]$ contains at least k eigenvalues of A. Hence, the set $\mathbb{K} = [\tilde{b}, \tilde{a}] \cup [a, b]$ with

$$(3.26) \qquad \tilde{a} := \tilde{\theta}_{\ell-k/2}, \qquad a := \tilde{\theta}_{\ell+k/2+1}, \qquad \tilde{b} \leq \tilde{a}, \qquad a \leq b,$$

does not contain any of the desired eigenvalues and the zeros of the acceleration polynomial ψ_{mk} could be allocated in $\mathbb{K}$. However, extensive numerical experience with the implicitly restarted block-Lanczos method indicates that faster convergence often can be achieved by choosing the endpoints $\tilde{a}$ and a as far away from the origin as possible. The description of the choices of the endpoints $\tilde{a}$ and a in the `irbleigs` code is somewhat technical and we only provide an outline. These choices depend on the values of the parameters *sizint* and *endpt*. The former parameter specifies how large the intervals that make up the sets $\mathbb{K}$ should be, the latter whether successive sets $\mathbb{K}$ should be nested.

We first consider the case when *endpt*=MON, which gives monotonically varying endpoints $\tilde{a}$ and a, and therefore a nested sequence of sets $\mathbb{K}$. Let the integer ℓ be defined by (3.17) and assume that $k/2 \leq \ell \leq mr - k/2$. This condition secures that we may determine $k/2$ positive and $k/2$ negative eigenvalues. A similar requirement on ℓ is imposed when k is odd. If $sizint = 1$, $\theta_1 \leq \tilde{\theta}_2$ and $\tilde{\theta}_{mr-1} \leq \theta_{mr}$, then we let

$$(3.27) \qquad \tilde{a} := \max\{\tilde{a}, \tilde{\theta}_2\}, \qquad a := \min\{a, \tilde{\theta}_{mr-1}\}$$

and the endpoints b and $\tilde{b}$ are updated according to (3.9) and (3.12). If, instead, $sizint = 2$ and $\theta_1 \leq \tilde{\theta}_3$ and $\tilde{\theta}_{mr-2} \leq \theta_{mr}$, then the formulas (3.27) are replaced by

$$\tilde{a} := \max\{\tilde{a}, \tilde{\theta}_3\}, \qquad a := \min\{a, \tilde{\theta}_{mr-2}\}. \tag{3.28}$$

The updating formulas (3.27) and (3.28) are modified if the relations between Ritz and harmonic Ritz values are violated. For instance, if $sizint = 1$ and $\tilde{\theta}_2 < \theta_1 \leq \tilde{\theta}_3$, then $\tilde{a}$ is updated according to (3.28) instead of by (3.27).

If *endpt*=FLT, the endpoints $\tilde{a}$ and a are allowed to float, i.e., to vary in a nonmonotonic manner, and the sets $\mathbb{K}$ determined are not guaranteed to be nested. The assignments (3.27) are replaced by

$$\tilde{a} := \tilde{\theta}_2, \qquad a := \tilde{\theta}_{mr-1}. \tag{3.29}$$

Other assignments, such as (3.28), are modified analogously.

The computation of k eigenvalues of the matrix A in the vicinity of an arbitrary point μ on the real axis can be carried out by replacing the matrix A by $A - \mu I_n$ and proceed as described above. Since the Lanczos decomposition (2.2) yields

$$(A - \mu I_n)V_{mr} = V_{mr}(T_{mr} - \mu I_{mr}) + F_r E_r^*, \tag{3.30}$$

it follows that the matrix A does not have to be modified. Instead, we compute the Lanczos decomposition (2.2) and then subtract μ from the diagonal entries of the block-tridiagonal matrix T_{mr}. The new block-tridiagonal matrix, which we also denote by T_{mr} is used in the formulas for computing Ritz and harmonic Ritz values of $A - \mu I_n$.

In our discussion on the computation of k eigenvalues of A around the origin, we assumed that the matrix T_{mr} is nonsingular. The nonsingularity can be enforced as follows. If we detect that T_{mr} is numerically singular when computing the matrix $\tilde{D}_{m+1}$, cf. (3.19), then we replace T_{mr} by $T_{mr} - \mu I_{mr}$ for some $\mu \in \mathbb{R}$ of small magnitude, such that $T_{mr} - \mu I_{mr}$ is nonsingular. It follows from our discussion above, that this has the effect that the algorithm seeks to determine k eigenvalues in a vicinity of μ.

4. Extensions. The `irbleigs` code can be used to solve certain generalized eigenvalue problem as well as to compute a few singular values and associated singular vectors of a general $n \times \ell$ matrix. For a recent discussion on the application of the Lanczos and Arnoldi methods to the generalized eigenvalue problem, we refer to Ruhe [37].

Consider the generalized eigenvalue problem

$$Hx = \lambda Mx, \tag{4.1}$$

where the matrices $H, M \in \mathbb{C}^{n\times n}$ are Hermitian and M is positive definite. Assume that M has a structure, such as small bandwidth, that makes it feasible to compute its upper triangular Cholesky factor R; thus, $M = R^*R$. The generalized eigenvalue problem (4.1) can be a transformed into a standard eigenvalue problem for the Hermitian matrix

$$A := R^{-*}HR^{-1}, \tag{4.2}$$

and it follows that the eigenvalues λ are real and the eigenvectors of (4.1) can be chosen to be pairwise M-orthonormal. The `irbleigs` code computes block-Lanczos

decompositions of the matrix (4.2) without explicitly forming the matrix; only matrix-vector product evaluations with A are required. Each such evaluation requires the computation of one matrix-vector product with the matrix H and the solution of two linear systems of equations with the triangular matrices R and R^*. A user can provide either the Cholesky factor R or the matrix M. In the latter case, R is computed using the MATLAB command `chol`. We remark that codes based on a shift-and-invert approach, which requires factorization of a linear combination of H and M, have been written by Grimes et al. [18], Marques [27] and Meerbergen and Scott [29]. The `irbleigs` code is designed for the computation of a few eigenpairs of generalized eigenproblems that are so large that factorization of H is not feasible.

We turn to the computation of a few singular values and associated right and left singular vectors of a large matrix $C \in \mathbb{C}^{n\times\ell}$. The `irbleigs` code can be applied in two ways. We may use the code to compute the eigenpairs associated with the corresponding eigenvalues of the Hermitian matrix

$$A := \begin{bmatrix} 0 & C \\ C^* & 0 \end{bmatrix} \in \mathbb{C}^{(n+\ell)\times(n+\ell)}. \tag{4.3}$$

Let

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\min\{n,\ell\}} \tag{4.4}$$

denote the singular values of C. Then the matrix A has the eigenvalues

$$\pm\sigma_1, \pm\sigma_2, \ldots, \pm\sigma_{\min\{n,\ell\}}$$

as well as $|m-n|$ zero eigenvalues. The eigenvector of A associated with the eigenvalue σ_j yields both the right and left singular vectors of C associated with this singular value. This approach is often appropriate when a few of the smallest singular values and associated singular vectors are desired. If instead we would like to determine a few of the largest singular values of C, then it may be attractive to apply the `irbleigs` code to one of the matrices C^*C or CC^*. The largest singular values of these matrices are better separated than the largest singular values of the matrix (4.3), and this generally speeds up the convergence of Lanczos-type methods. The eigenvectors of C^*C and CC^* are the singular vectors of C.

5. Numerical examples. This section presents computed examples that illustrate the performance of the `irbleigs` code and compare it with other available codes for the computation of a few selected nearby eigenvalues and associated eigenvectors of large Hermitian matrices. Specifically, we compare the `irbleigs` code with Sleijpen's MATLAB implementation `jdqr` of the Jacobi-Davidson QR method by Fokkema et al. [16] and with two implementations of the implicitly restarted Arnoldi/Lanczos method.

The `jdqr` code is available at Sleijpen's home page[1]. It computes partial Schur decompositions of A and can determine extreme and nonextreme eigenvalues. The Jacobi-Davidson method is a powerful scheme when a good preconditioner for the linear system of equations that has to be solved is available. In our computed examples, we assume that no good preconditioner is known, and apply the Jacobi-Davidson method either with no preconditioner or with a diagonal preconditioner made up by the diagonal entries of A. The linear system of equations is solved by an iterative

[1] http://www.math.uu.nl/people/sleijpen/JD_software/JDQR.html

TABLE 5.1
Parameters for `jdqr`.

Name	Description	Default Value
tol	Tolerance	10^{-8}
jmin	Minimum dimension of search subspace	$k+5$
jmax	Maximum dimension of search subspace	$jmin+5$
k	Number of desired eigenvalues	5
MaxIt	Maximum number of iterations	100
$v0$	Starting vector	ones+0.1·rand
Schur	Schur decomposition	no
sigma	Location of the desired eigenvalues	LM
TestSpace	For using harmonic Ritz values	Standard
Disp	Display eigenvalues	0
LSolver	Linear solver	GMRES
LS_ℓ	ℓ for BiCGstab(ℓ)	5
LS_MaxIt	Max. # of iterations for the linear system solver	4
LS_Tol	Residual reduction by linear solver	$1, 0.7, 0.7^2, \ldots$
Precond	Preconditioner	M=[ ]

method. The BICGSTAB, CG, GMRES, MINRES and SYMMLQ iterative methods are available, with GMRES being the default method. We used the iterative method that required the fewest number of matrix-vector product evaluations for each example. Several parameters can be specified by a user of the code; see Table 5.1. For all examples, we used the default values for the parameters *jmin*, *Schur*, *TestSpace*, *Disp*, *LS_Tol*, and set *LS_MaxIt* to the value that gave the best results. For further details on the code, we refer to Sleijpen's home page.

The two implementations of the implicitly restarted Arnoldi/Lanczos method used in our examples are furnished by the functions `eigs` in MATLAB versions 5.3 and 6.0, and are denoted by `eigs5.3` and `eigs6.0`, respectively. We used patches from The MathWorks to remedy the memory leakage in MATLAB version 6.0 and to correct the call routines in `eigs6.0`.

TABLE 5.2
Parameters for `eigs5.3` *LM stands for largest magnitude.*

Name	Description	Default Value
cheb	Polynomial acceleration indicator	0
disp	# of eigenvalues displayed in each iteration	20
issym	Positive if the matrix is symmetric	0
k	Number of desired eigenvalues	6
maxit	Maximum number of iterations	300
p	Number of Arnoldi vectors	$2k$
sigma	Location of the desired eigenvalues	LM
stagtol	Stagnation tolerance	10^{-6}
tol	Tolerance	10^{-10}
$v0$	Starting vector	$\mathrm{rand}(n,1)-0.5$

The function `eigs5.3` is discussed in Radke's Master's thesis [35] and implements the Implicitly Restarted Arnoldi and Lanczos methods. It uses a shift-and-invert

approach to accelerate convergence to the desired eigenvalues when the matrix A is stored explicitly. To avoid factorization of matrices of the form $A - sI_n$, $s \in \mathbb{R}$, we supplied a MATLAB function for the evaluation of matrix-vector products with A. The `eigs5.3` function then does not factor matrices of the form $A - sI_n$, and instead applies an acceleration polynomial to determine a few desired eigenvalues and associated eigenvectors of the matrix A; see [35, 42] for details. A user may choose the values of several parameters that affect the performance of the `eigs5.3` code. Table 5.2 lists these parameters and their default values. In all examples, we set *stagtol* to machine precision, $disp = 0$, and $issym = 1$; see MATLAB version 5.3 for further details on the parameter values.

TABLE 5.3
Parameters for `eigs6.0`. *eps stands for machine epsilon and is about* $2.2 \cdot 10^{-16}$. *LM stands for largest magnitude.*

Name	Description	Default Value
$cholM$	Cholesky factorization of the matrix M	0
$disp$	Display eigenvalues	1
$isreal$	Positive if the matrix is real	1
$issym$	Positive if the matrix is symmetric	0
k	Number of desired eigenvalues	6
$maxit$	Maximum number of iterations	300
p	Number of Arnoldi vectors	$2k$
$permM$	Permutation of the Cholesky factorization	$[1:n]$
$sigma$	Location of the desired eigenvalues	LM
tol	Tolerance	eps
$v0$	Starting vector	random

The function `eigs6.0` uses a C-mex file called ARPACKC that processes the input and calls compiled FORTRAN subroutines of ARPACK; see [25] for a detailed description of the ARPACK code. MATLAB version 6.0 contains the following compiled FORTRAN subroutines of ARPACK: dsaupd, dseupd, dnaupd, dneupd, znaupd and zneupd. An important difference between the functions `eigs5.3` and `eigs6.0` is that only the former has the polynomial acceleration option *cheb*, which is used to determine nonextreme eigenvalues; see Sorensen and Yang [42]. When using `eigs6.0` to determine nonextreme eigenvalues and associated eigenvectors, a user must supply a linear system solver. This approach is attractive when it is feasible to factor matrices of the form $A - sI_n$ for $s \in \mathbb{R}^n$. Since we assume that these matrices cannot be factored, we do not compare the `irbleigs` code with `eigs6.0` for computing nonextreme eigenvalues and associated eigenvectors. Table 5.3 describes the parameters of the `eigs6.0` code and their default values. In all computed examples with the `eigs6.0` code, we used the default values for $isreal$, $cholM$ and $permM$, and we set $disp = 0$ and $issym = 1$; see MATLAB version 6.0 for further details on these parameters.

Table 5.4 describes parameters, whose values can be chosen by a user of the `irbleigs` code. The parameter $blsz$ defines the block-size of the block-tridiagonal matrix T_{mr} in (2.2) and corresponds to the parameter r in the previous sections. The parameter $nbls$ in the `irbleigs` code determines the maximum number of consecutive block-Lanczos steps and corresponds to the parameter m in the previous sections. When the parameter $cholM$ is positive, the upper triangular Cholesky factor R of the matrix M in the generalized eigenvalue problem (4.1) is provided instead of the matrix

TABLE 5.4
Parameters for `irbleigs`. *Default values marked by superscript* $*$ *are for the cases when extreme eigenvalues of A are desired, i.e., when the value of sigma is LE (largest eigenvalue) or SE (smallest eigenvalue). When sigma has a numerical value, and in particular when nonextreme eigenvalues of A are desired, the default values endpt =FLT, maxdpol = n and zertyp = WL are used.*

Name	Description	Default Value
blsz	Block size	3
cholM	Cholesky factorization of the matrix M	0
dispr	Display Ritz values and residuals	0
eigvec	Matrix of converged eigenvectors	[ ]
endpt	Endpoints of damping intervals	MON*
funpar	Parameters for matrix-vector product function	[ ]
k	Number of desired eigenvalues	3
nbls	Number of blocks	3
maxit	Maximum number of iterations	100
maxdpol	Maximum degree of the dampening polynomial	200*
permM	Permutation of the Cholesky factorization	$[1:n]$
zertyp	Type of zeros	ML*
sigma	Location of the desired eigenvalues	LE
sizint	Size of the dampening interval	1
tol	Tolerance used for convergence	10^{-6}
*v*0	A matrix of orthonormal starting vectors	randn

M. We note that a very sparse symmetric positive definite matrix M might not have a very sparse Cholesky factor R. A suitable permutation of the rows and columns of M may make the Cholesky factor sparser. Such a permutation can be supplied with the vector $permM$, i.e., we compute the Cholesky factor of $M(permM, permM)$. For instance, the MATLAB function `symamd` can be used to determine such a permutation. If the matrix *eigvec* is nonempty, then the `irbleigs` code determines a sequence of Krylov subspaces that are orthogonal to the columns of the matrix *eigvec*. When the columns of the matrix *eigvec* are made up of available orthonormal eigenvectors, the `irbleigs` code is forced to determine eigenvectors that are orthogonal to the available ones. The parameter *maxdpol* is the maximum number of Leja points computed before the computations of Leja points is restarted by setting ℓ in (3.1) to zero. For difficult problems, when the largest eigenvalue is much larger than the smallest eigenvalue, a large value of *maxdpol* may enhance convergence; see Example 5. The parameter *zertyp* determines how the zeros of the acceleration polynomial are defined. The value WL gives weighted fast Leja points and the value ML gives mapped Leja points described in Section 3. The value of the parameter *sizint* determines the length of the interval or intervals that make up the sets $\mathbb{K}$, see Section 3, and greatly affects the rate of convergence. The value 1 gives the smallest intervals. A larger value of *sizint* gives larger intervals. The value of the parameter *endpt* is either MON or FLT. The former value gives monotonically increasing or decreasing endpoints of the sets $\mathbb{K}$ and produces a nested sequence of sets. The value FLT allows the endpoint(s) of $\mathbb{K}$ closest to the desired eigenvalues to vary in a nonmonotonic fashion. The parameter *sigma* determines which eigenvalues will be computed. The value LE of *sigma* yields approximations of the k largest eigenvalues of A and the value SE gives approximations of the k smallest eigenvalues. A numerical value of *sigma* yields approximations of k eigenvalues in a vicinity of the value of *sigma*. The parameter *tol* corresponds

to ϵ in (2.13) and determines how accurately the computed Ritz pairs approximate eigenpairs of the matrix A. The value of $\|A\|$ in (2.13) is approximated by the absolute value of the Ritz value of largest magnitude determined during the computations. The parameter $funpar$ allows the user to pass additional parameters to the function for the evaluation of matrix-vector products with A. Finally, the parameter $dispr$ determines the display of Ritz values during the computations. When $dispr > 0$, the sequence of computed approximations of the desired eigenvalues is displayed. This allows a user to follow the progress of the computations. The default value $dispr = 0$ only gives the output of the accepted eigenvalue approximations. In all examples we used default values for the parameters $cholM$, $dispr$, $eigvec$, $funpar$, $permM$, and $endpt$.

There are numerous choices and combinations of parameter values for each one of the methods. Some choices and combinations yield faster convergence than others. The performances reported in this section are typical for the methods.

In all computed examples we determined the initial Lanczos block V_r for the `irbleigs` code by generating an $n \times blsz$ matrix with entries sampled from the standard normal distribution, and then orthonormalizing the columns. The initial vector $v0$ for the `jdqr`, `eigs5.3` and `eigs6.0` codes was chosen to be the first column of V_r.

In numerous computed examples we found that if the value of the parameter *tol* in `jdqr`, `eigs5.3` and `eigs6.0` is not chosen sufficiently small, these codes may be unable to detect some eigenvalues of multiplicity larger than one. Since the `irbleigs` code implements a block-Lanczos method, it generally determines the correct multiplicity even for non-tiny values of *tol* when the block-size is larger than or equal to the multiplicity. In the computed examples with multiple eigenvalues, we let, for each one of the codes `jdqr`, `eigs5.3`, and `eigs6.0`, the parameter *tol* be equal to the smallest power of 1/10 for which the code computes the desired eigenpairs to about the same accuracy as `irbleigs` with proper multiplicity.

In all examples the matrix A was accessed only by calls to a function with input $x \in \mathbb{R}^n$ and output Ax. This approach is "matrix-free" in the sense that the matrix A does not have to be stored. For several of the examples the function for matrix-vector product evaluation was written in C and interfaced with MATLAB using MEX files; see [28]. The matrix-free approach allowed us to work with matrices of very large size and prevented the codes `eigs5.3` and `eigs6.0` from factoring A. The CPU times (in seconds) recorded were determined using the tic-toc commands in MATLAB.

All numerical experiments for the present paper, except for Example 4, were carried out using MATLAB version 6.0 on a Gateway E-5200 workstation with two 450 MHz (512k cache) Pentium III processors and 128 MB (100 MHz) of memory. In particular, we moved the code `eigs5.3` to MATLAB version 6.0 to make a fair comparison of the performance of all codes possible. Machine epsilon was $2.2 \cdot 10^{-16}$.

A comparison of a preliminary version of the IRBL method with the 1996 FORTRAN code for ARPACK by Lehoucq at al. [24] and the FORTRAN code LASO2 by Scott [39] is reported in [4]. This comparison showed the IRBL method to perform significantly better than the other codes when Krylov subspaces with only few vectors can be stored. This is consistent with our experience with the `irbleigs` code.

Example 1 (Smallest eigenvalues). Let A be the 40000×40000 matrix obtained by discretizing the 2-dimensional negative Laplace operator on the unit square by the standard 5-point stencil with Dirichlet boundary conditions. We wish to determine the eigenpairs associated with the 3 smallest eigenvalues of A. The eigenvalues of the matrix A are known and the second and third smallest eigenvalues of A coincide, i.e.,

$$\lambda_1 < \lambda_2 = \lambda_3 < \dots ;$$

TABLE 5.5
Example 1: Parameter values. Default values are marked by superscript $*$. Preconditioning option D denotes the diagonal preconditioner diag(A) that consists of the (main) diagonal of the matrix A. SR stands for smallest real part and SA for smallest algebraic.

irbleigs	jdqr	eigs5.3	eigs6.0
$blsz = 2, 3^*$	$jmax = 20$	$cheb = 0^*$,1	$k = 3$
$k = 3^*$	$k = 3$	$k = 3$	$maxit = 1000$
$nbls = 5, 3^*$	$MaxIt = 9000$	$maxit = 1000$	$p = 20$
$maxit = 1000$	$Precond = [\,]^*$, D	$p = 20$	$sigma$ =SA
$maxdpol = 200^*$	$sigma$ =SR	$sigma$ =SR	$tol = 10^{-8}*$,10^{-10}
$zertyp$ =ML*,WL	$tol = 10^{-12}, 10^{-14}$	$tol = 10^{-10}*$, 10^{-12}	
$sigma$ =SE	$LSolver$ =CG		
$sizint = 1^*$	$LS\_MaxIt = 20$		
$tol = 10^{-6}*$			

TABLE 5.6
Example 1. 40000 × 40000 discretized negative Laplace operator. Superscript $*$ indicates that multiple eigenvalues were missed.

irbleigs

zertyp	*blsz*/*nbls*	# matrix-vector products	CPU time	magnitude of largest error
ML	3/3	1791	301s	$2.99 \cdot 10^{-8}$
WL	3/3	1422	244s	$1.43 \cdot 10^{-8}$
ML	2/5	2860	515s	$1.56 \cdot 10^{-8}$
WL	2/5	3890	713s	$1.05 \cdot 10^{-9}$

jdqr

tol	*Precond*	# matrix-vector products	CPU time	magnitude of largest error
10^{-10}	None	1425*	1875s	$2.78 \cdot 10^{-16}$
10^{-10}	diag(A)	1425*	1926s	$2.56 \cdot 10^{-16}$
10^{-12}	None	2307	3302s	$2.59 \cdot 10^{-16}$
10^{-12}	diag(A)	2349	3501s	$2.80 \cdot 10^{-16}$

eigs5.3

tol	*cheb*	# matrix-vector products	CPU time	magnitude of largest error
10^{-10}	0	1926*	2093s	$8.63 \cdot 10^{-17}$
10^{-10}	1	2647*	3767s	$2.66 \cdot 10^{-16}$
10^{-12}	0	10378	9975s	$1.08 \cdot 10^{-15}$
10^{-12}	1	3057*	4338s	$2.59 \cdot 10^{-16}$

eigs6.0

tol	# matrix-vector products	CPU time	magnitude of largest error
10^{-8}	3135*	384s	$7.84 \cdot 10^{-17}$
10^{-10}	5191	650s	$1.63 \cdot 10^{-16}$

see, e.g., [43, Section 8.4]. We would like the computed Ritz values to satisfy (2.13) with $\epsilon = 10^{-6}$.

The parameter values used for the different methods are given by Table 5.5. It is clear from Table 5.6 that the `irbleigs` code requires the smallest number of matrix-vector product evaluations with the matrix A and the smallest amount of computer memory. Throughout this paper, the number of matrix-vector product evaluations shows the number of evaluations of matrix-vector products Aw of the $n \times n$ matrix A with a single n-vector w.

Table 5.6 shows that the `eigs5.3`, `eigs6.0`, and `jdqr` codes did not always detect both multiple eigenvalues. Furthermore, the use of the polynomial acceleration option $cheb = 1$ in the `eigs5.3` code caused the multiple eigenvalue to be missed regardless of how small the value of the parameter tol was selected. The column labeled "largest magnitude of error" in Table 5.6 displays the magnitude of the largest error in the computed approximations of the three desired eigenvalues. We remark that the small values of the parameter tol required by the `jdqr`, `eigs5.3` and `eigs6.0` codes in order to determine the proper number of eigenvalues in the vicinity of the double eigenvalue yields computed eigenvalues with high accuracy. The table shows that using the diagonal preconditioner diag(A) did not reduce the number of matrix-vector product evaluations required by the `jdqr` code.

This example illustrates that the `irbleigs` code is able to determine accurate approximations of the desired eigenvalues and requires fewer matrix-vector product evaluations than the other methods. The `irbleigs` code requires the storage of at most 10 Lanczos vectors, while the other codes were allowed storage of 20 basis vectors. Decreasing the number of basis vectors for the other codes to 10 increased the number of matrix-vector product evaluations required significantly.

Note that `eigs6.0` requires less computational time than `eigs5.3` even when the number of matrix-vector product evaluations is larger. This depends on that the `eigs6.0` code is more efficient. We expect that a FORTRAN implementation of the IRBL method with a MEX user interface for MATLAB would require significantly less execution time than the `irbleigs` code available. □

TABLE 5.7
Example 2: Parameter values. Default values are marked by superscript $*$. Preconditioning option D denotes the diagonal preconditioner diag(A).

irbleigs	jdqr	eigs5.3
$blsz = 3^*$	$jmax = 15$	$cheb = 1$
$k = 5$	$k = 5$	$k = 5$
$nbls = 5$	$MaxIt = 15000$	$maxit = 2000$
$maxit = 2000$	$Precond = [\,]^*$, D	$p = 15$
$maxdpol = n^*$	$sigma = 0$	$sigma = 0$
$zertyp =$WL*	$tol = 10^{-6}$	$tol = 10^{-6}$
$sigma = 0$	$LSolver =$MINRES	
$sizint = 1^*$	$LS\_MaxIt = 100$	
$tol = 10^{-6*}$		

Example 2 (Interior eigenvalues). We consider a matrix that arises from the Anderson model of localization in quantum physics for investigation of quantum mechanical effects of disorder; see [15] for more details. The matrix, denoted by A, is real symmetric and indefinite. The diagonal entries represent disorder and are uniformly

TABLE 5.8
Example 2. Anderson model of localization.

	irbleigs		eigs5.3		jdqr	
	# matrix-vector products	CPU time	# matrix-vector products	CPU time	# matrix-vector products	CPU time
1	17145	176s	43112	268s	35062	289s
2	18615	194s	64608	405s	41021	352s
3	13770	135s	41668	261s	39506	327s
4	14520	147s	50761	316s	35668	301s
5	14820	147s	46741	289s	22134	183s

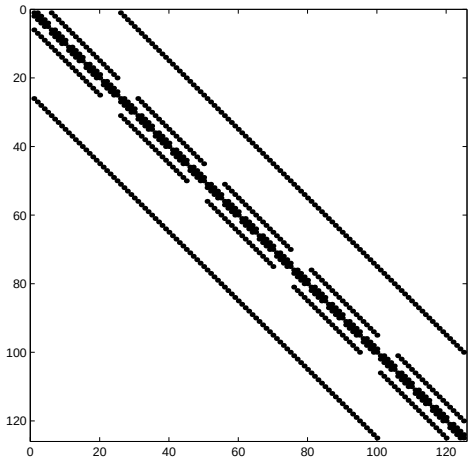

FIG. 5.1. *The sparsity pattern of a* 125×125 *matrix for the Anderson model of localization. The matrix has* 725 *nonvanishing entries.*

distributed random numbers in the interval $[\frac{-\omega}{2}, \frac{\omega}{2}]$. The nonvanishing off-diagonal entries are determined by the probability that electrons move from one site to a neighboring site; they are normalized to be unity. Figure 5.1 shows the sparsity pattern of such a matrix of size 125×125.

The eigenvalues of A represent quantum mechanical energy levels. Of particular interest are the eigenvalues closest to the origin. In the present example, we let $\omega := 16.5$; this models the critical disorder case. The order of the matrix is $n = 1728$. We would like to compute the five eigenvalues closest to the origin.

We computed approximations of the desired eigenvalues for five matrices of this kind with the `irbleigs`, `eigs5.3` and `jdqr` codes. The values of the parameters used for the methods are displayed in Table 5.7. The number of matrix-vector product evaluations and the CPU times required for each one of these matrices is reported in Table 5.8. The smallest eigenvalue of each matrix generated is about -10, and the largest eigenvalue about 10. The eigenvalue closest to the origin is of magnitude from about 10^{-3} to about 10^{-2}. For instance, the first one of the five matrices generated had the smallest eigenvalue -10.24, the largest eigenvalue 10.21, and the five eigenvalues closest to the origin were -0.020, -0.013, 0.0011, 0.0058 and 0.019.

The choice of parameters allows storage of at most 15 basis vectors for each method. Table 5.8 shows the `eigs5.3` and `jdqr` codes to require substantially more

matrix-vector product evaluations than the `irbleigs` code. Use of the diagonal preconditioner diag(A) did not reduce the number of matrix-vector product evaluations required by the `jdqr` code. □

TABLE 5.9
Example 3: Parameter values. Default values are marked by superscript $*$. Preconditioning option D denotes the diagonal preconditioner diag(A).

`irbleigs`	`jdqr`	`eigs5.3`
$blsz = 2, 3^*$	$jmax = 12$	$cheb = 1$
$k = 6$	$k = 6$	$k = 6$
$nbls = 6, 4$	$MaxIt = 100^*$	$maxit = 300^*$
$maxit = 100^*$	$Precond = [\,]^*$, D	$p = 12$
$maxdpol = n^*$	$sigma = 0.01205$	$sigma = 0.01205$
$zertyp =$WL*	$tol = 10^{-6}, 10^{-8*}, 10^{-9}$	$tol = 10^{-6}, 10^{-8}$
$sigma = 0.01205$	$LSolver =$GMRES	
$sizint = 1^*$	$LS\_MaxIt = 12$	
$tol = 10^{-6*}$		

TABLE 5.10
Example 3. 362×362 PLAT362 Harwell-Boeing matrix. Superscript $*$ indicates that not all multiple eigenvalues were found.

`irbleigs`

$blsz/nbls$	# matrix-vector products	CPU time	magnitude of largest error
2/6	432	2.01s	$2.74 \cdot 10^{-12}$
3/4	528	2.08s	$1.12 \cdot 10^{-12}$

`jdqr`

tol	# matrix-vector products	CPU time	magnitude of largest error
10^{-6}	542^*	3.38s	$1.22 \cdot 10^{-11}$
10^{-8}	922^*	5.53s	$8.36 \cdot 10^{-16}$
10^{-9}	1197	6.89s	$3.66 \cdot 10^{-16}$

`eigs5.3`

tol	# matrix-vector products	CPU time	magnitude of largest error
10^{-6}	651^*	3.13s	$4.01 \cdot 10^{-13}$
10^{-8}	1081	4.67s	$3.64 \cdot 10^{-16}$

Example 3 (Interior eigenvalues). We consider the 362×362 matrix PLAT362 from the Harwell-Boeing Sparse Matrix Collection [14]. This matrix arises in a finite difference model associated with the North Atlantic Ocean. Its eigenvalues are known to be of multiplicity two. The eigenvectors associated with eigenvalues in the interval $\mathbb{I} := [0.001, 0.024]$ correspond to natural modes that contribute to global tides and therefore are of interest. We seek to determine six eigenpairs associated with eigenvalues close to the midpoint 0.01205 of the interval $\mathbb{I}$. Table 5.9 displays the parameter values used for the different codes.

Table 5.10 shows that the `irbleigs` code did not miss any multiple eigenvalues, and required the fewest matrix-vector product evaluations and the least CPU time. Since we know that each eigenvalue is of multiplicity two, we chose the block-size to be two. The `eigs5.3` and `jdqr` codes failed to find any multiple eigenvalues when $tol = 10^{-6}$. Decreasing the tolerance to $tol = 10^{-8}$, the `eigs5.3` code was able to detect all multiple eigenvalues, but this resulted in a significant increase in the number of matrix-vector product evaluations and in CPU time. The `jdqr` code was only able to capture one multiple eigenvalue when $tol = 10^{-8}$. Decreasing the tolerance further to $tol = 10^{-9}$, the `jdqr` code successfully determined all multiple eigenvalues. When the diagonal preconditioner diag(A) was used for the `jdqr` code, no eigenvalues were found within the default maximum number of iterations. □

TABLE 5.11

Example 4: Parameter values. Default values are marked by superscript $*$. LR stands for largest real part and LA for largest algebraic.

`irbleigs`	`jdqr`	`eigs5.3`	`eigs6.0`
$blsz = 1, 2$	$jmax = 5, 10$	$cheb = 0^*$	$k = 2$
$k = 2$	$k = 2$	$k = 2$	$maxit = 300^*$
$nbls = 5$	$MaxIt = 1000$	$maxit = 300^*$	$p = 5, 10$
$maxit = 100^*$	$Precond = [\ ]^*$	$p = 5, 10$	$sigma =$LA
$maxdpol = 200^*$	$sigma =$LR	$sigma =$LR	$tol = 10^{-5}$
$zertyp =$ ML*	$tol = 10^{-5}$	$tol = 10^{-5}$	
$sigma =$LE*	$LSolver =$MINRES,CG		
$sizint = 1^*$	$LS\_MaxIt = 10$		
$tol = 10^{-5}$			

Example 4 (Largest eigenvalue). Let A be the matrix S3DKT3M2 from the Harwell-Boeing Sparse Matrix Collection [14]. This is a 90449×90449 real symmetric positive definite matrix with 3753461 nonzero entries. It is one of the largest symmetric matrices in this matrix collection. The matrix stems from a finite element discretization of a cylindrical shell. Its largest and smallest eigenvalues are $8.7984 \cdot 10^3$ and $2.4269 \cdot 10^{-8}$, respectively; see the web site http://math.nist.gov/MatrixMarket. The second and third largest eigenvalues, obtained from numerical calculations, are $8.7967 \cdot 10^3$ and $8.7939 \cdot 10^3$, respectively. We seek to compute the two largest eigenvalues of A. These eigenvalues are close, but simple. We used the parameter values displayed in Table 5.11 for the different codes.

The computations for this example were carried out in MATLAB version 6.5 on a Dell Precision workstation 530 with two 2.4 GHz (512k cache) Xeon processors and 2.0 GB (400 MHz) of memory.

This example shows that the `irbleigs` code can compute eigenvalues of a very large matrix quickly and efficiently with a Krylov subspace of only 5 Lanczos vectors. Table 5.12 shows the other methods to require more than 4 times as many matrix-vector product evaluations when storage of only 5 Lanczos vectors is allowed. The superior performance of the `irbleigs` code, compared with the codes `eigs5.3` and `eigs6.0`, depends on the different choices of shifts used by the codes.

When Krylov subspaces of 10 vectors were allowed, all methods successfully determined the two largest eigenvalues. Table 5.12 shows the `irbleigs` code to be competitive in this situation also.

Several different values were used for the parameter LS_MaxIt of the `jdqr` code; the best results were achieved for $LS\_MaxIt = 10$. The results displayed for the

TABLE 5.12
Example 4: 90449×90449 *S3DKT3M2 Harwell-Boeing matrix.* * *MINRES was used to solve the linear system in* `jdqr`. ** *CG was used to solve the linear system in* `jdqr`. *The magnitude of largest error only pertains to the largest eigenvalue.* `eigs5.3` *was unable to compute two eigenvalues with only* 5 *Lanczos vectors.*

`irbleigs`

$blsz/nbls$	# matrix-vector products	CPU time	magnitude of largest error
1/5	350	98s	$1.79 \cdot 10^{-4}$
2/5	680	129s	$6.39 \cdot 10^{-4}$

`jdqr`

# Lanczos vectors	# matrix-vector products	CPU time	magnitude of largest error
5	1543*	384s	$9.82 \cdot 10^{-11}$
10	832**	203s	$1.32 \cdot 10^{-10}$

`eigs5.3`

# Lanczos vectors	# matrix-vector products	CPU time	magnitude of largest error
5	–	–	–
10	1093	465s	$8.32 \cdot 10^{-5}$

`eigs6.0`

# Lanczos vectors	# matrix-vector products	CPU time	magnitude of largest error
5	2632	608s	$8.08 \cdot 10^{-4}$
10	753	174s	$2.42 \cdot 10^{-4}$

`jdqr` code are for the iterative method that required the smallest number of matrix-vector product evaluations. The required tolerance for the `jdqr` code was the same as for the other codes, but when the computations were terminated by the code, very accurate eigenvalue approximation had been determined. The use of the diagonal preconditioner diag(A) in the `jdqr` code did not reduce the number of matrix-vector product evaluations required. □

Example 5 (Generalized eigenvalue problem). We consider a generalized eigenvalue problem (4.1), where the matrices H and M are chosen to be the matrices BCSSTK08 and BCSSTM08 from the Harwell-Boeing Sparse Matrix Collection [14]. These matrices are of size 1074×1074 and arise from dynamic analysis in structural engineering for TV studios. The matrix BCSSTK08 is a Hermitian positive definite stiffness matrix and BCSSTM08 is a Hermitian positive definite mass matrix.

We seek to determine the four smallest eigenvalues. They are about 6.9, 18.14202, 18.142366 and 18.142366. The largest eigenvalue is about $1.7 \cdot 10^7$ and equals the spectral radius of the matrix A given by (4.2). This example poses many difficulties for eigensolvers because there are several clusters of eigenvalues, the second and third smallest eigenvalues are very close, the third smallest eigenvalue is numerically of multiplicity two and the spectral radius of the matrix A defined by (4.2) is much larger than any one of the desired eigenvalues.

We use the Cholesky factorization of the mass matrix M. With the parameter

$maxit = 5000$ and the other parameters for the `irbleigs` code assigned their default values, the code required 14283 matrix-vector product evaluations with the matrix H and 147 seconds of CPU time. The magnitude of the largest error in the computed approximations of the desired eigenvalues was $2.15 \cdot 10^{-1}$. The large error depends on the large spectral radius.

To increase the rate of convergence and reduce the error in the computed eigenvalue approximations, we set $maxdpol = 1000$, $tol = 10^{-8}$, $maxit = 5000$, and $sizint = 2$. Using these new values of the parameters, the `irbleigs` code required 12315 matrix-vector product evaluations with the matrix H and 126 seconds of CPU time. The magnitude of the largest error in the computed approximations of the desired eigenvalues was reduced to $6.6 \cdot 10^{-5}$. Both runs with the `irbleigs` code required storage of only 9 Lanczos vectors.

For comparison, we also tried to compute the desired eigenvalues with the `eigs6.0` code using the parameter values $tol = 10^{-8}$, $p = 9$, and $maxit = 5000$. This code failed to determine the four desired eigenvalues within 50000 matrix-vector product evaluations. Increasing the tolerance to $tol = 10^{-6}$ and increasing the number of Lanczos vectors p to 20 did not help; `eigs6.0` still failed to determine the desired eigenvalues within 50000 matrix-vector product evaluations. The `eigs5.3` code was not able to determine these eigenvalues either. The `jdqr` code is not designed for generalized eigenvalue problems and would require a different function for matrix-vector product evaluation than the one used for `irbleigs` and `eigs6.0`. We therefore do not report the performance of the `jdqr` code. □

Example 6 (Singular values). Consider the 1033×320 matrix WELL1033 and the 1850×712 matrix WELL1850 from the set LSQ in the Harwell-Boeing Sparse Matrix Collection [14]. These matrices arise from surveying problems. The condition number of a matrix $C \in \mathbb{R}^{m \times n}$, $m \geq n$, of full rank is given by $\kappa(C) := \sigma_1/\sigma_n$, where σ_1 and σ_n denote the largest and smallest singular values of C, respectively; cf. (4.4). We can compute the condition number of the matrix C by determining the largest and smallest positive eigenvalues of the matrix $A \in \mathbb{R}^{(m+n)\times(m+n)}$ defined by (4.3). Note that the matrix A is not explicitly formed; only a function for the evaluation of matrix-vector products with the matrices C and C^* is required. The computation of the smallest positive eigenvalue of A requires the determination of eigenvalues close to the origin. We applied the `irbleigs` code with parameter values $k = 1 + blsz$, $blsz = 3$, $nbls = 5$, $maxit = 1000$ and $sigma = 0$.

The number of matrix-vector product evaluations refers to the matrix A defined by (4.3) with C being one of the matrices WELL1033 or WELL1850. The computation of the smallest singular value and associated singular vectors of the matrix WELL1033 with the `irbleigs` code required the evaluation of 4635 matrix-vector products and 43 seconds of CPU time. The corresponding computations for the matrix WELL1850 required 3960 matrix-vector product evaluations and a CPU time of 64 seconds.

To compute the largest singular value and associated singular vectors of the matrices WELL1033 and WELL1850, we set $k = 1$ and $sigma =$LE. For the former matrix, the `irbleigs` code required only 105 matrix-vector product evaluations and 0.84 seconds of CPU time, and for the latter matrix 150 matrix-vector product evaluations and 2.17 seconds of CPU time. We obtained the approximations $1.8065/0.0109 \approx 1.66 \cdot 10^2$ and $1.7943/0.0161 = 1.11 \cdot 10^2$ of the condition numbers of the matrices WELL1033 and WELL1850, respectively.

We also used the code `eigs5.3` with parameters $k = 4$, $sigma = 0$, $p = 15$, $tol = 10^{-6}$, $maxit = 1000$ and $cheb = 1$. With this choice of parameters `eigs5.3`

failed to locate the smallest positive eigenvalue of the matrix A for both matrices WELL1033 and WELL1850. Increasing the number of Lanczos vectors to $p = 50$ did not help; `eigs5.3` still failed to determine the smallest positive eigenvalue.

The code `jdqr` with parameters $k = 4$, $sigma = 0$, $jmax = 15$, $MaxIt = 9000$ and $tol = 10^{-6}$, and without preconditioner, also had difficulties to compute the smallest positive eigenvalue of A for both matrices WELL1033 and WELL1850; over 20000 matrix-vector product evaluations with the matrix A were required.

We remark that both MATLAB versions 5.3 and 6.0 have functions `svds` for computing a few singular values and associated singular vectors of a large sparse matrix. The `svds` function of MATLAB version 5.3 calls `eigs5.3` and the `svds` function of MATLAB version 6.0 calls `eigs6.0` to determine appropriate eigenvalues and eigenvectors of the Hermitian matrix A defined by (4.3). The `svds` functions of both MATLAB versions apply a shift-and-invert approach to locate nonextreme eigenvalues of A. Therefore, these `svds` functions are not well suited for very large matrices. □

6. Conclusion. This paper presents a restarted block-Lanczos method for the computation of a few nearby extreme or nonextreme eigenvalues of a large Hermitian matrix A. The method does not require factorization of A, and can therefore be applied to very large problems. Numerical examples show the method to be competitive with other available codes with regard to the number of matrix-vector product evaluations required and with regard to storage demand. Applications to generalized eigenvalue problems and the computation of a few singular values and vectors are also discussed.

Acknowledgement. We would like to thank Richard Lehoucq and Axel Ruhe for valuable suggestions, as well as the referees for comments.

REFERENCES

[1] J. Baglama, *Dealing with linear dependence during the iterations of the restarted block Lanczos methods*, Numer. Algorithms, 25 (2000), pp. 23–36.

[2] J. Baglama, D. Calvetti and L. Reichel, *Iterative methods for the computation of a few eigenvalues of a large symmetric matrix*, BIT, 36 (1996), pp. 400–421.

[3] J. Baglama, D. Calvetti and L. Reichel, *Fast Leja Points*, Elec. Trans. Numer. Anal., 7 (1998), pp. 124–140.

[4] J. Baglama, D. Calvetti, L. Reichel and A. Ruttan, *Computation of a few small eigenvalues of a large matrix with applications to liquid crystal modeling*, J. Comput. Phys., 146 (1998), pp. 203–226.

[5] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst, eds., Templets for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, SIAM, Philadelphia, 2000.

[6] C. Beattie, *Harmonic Ritz values and Lehmann bounds*, Elec. Trans. Numer. Anal., 7 (1998), pp. 18–39.

[7] D. Calvetti and L. Reichel, *A block Lanczos method for large continuation problems*, Numer. Algorithms, 21 (1999), pp. 109–118.

[8] D. Calvetti and L. Reichel, *Iterative methods for large continuation problems*, J. Comput. Appl. Math., 123 (2000), pp. 217–240.

[9] D. Calvetti, L. Reichel and D. C. Sorensen, *An implicitly restarted Lanczos method for large symmetric eigenvalue problems*, Elec. Trans. Numer. Anal., 2 (1994), pp. 1–21.

[10] F. Chatelin, Eigenvalues of Matrices, Wiley, Chichester, 1993.

[11] J. K. Cullum and R. A. Willoughby, Lanczos Algorithms for Large Symmetric Eigenvalue Computations, vol. 1, Birkhäuser, Boston, 1985.

[12] E. R. Davidson, *The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices*, J. Comput. Phys., 17 (1975), pp. 87–94.

[13] J. J. Dongarra, J. DuCroz, I. S. Duff and S. Hammarling, *A set of level 3 basic linear algebra subprograms*, ACM Trans. Math. Software, 16 (1990), pp. 1–17.

[14] I. S. Duff, R. G. Grimes and J. G. Lewis, *User's Guide for the Harwell-Boeing Sparse Matrix Collection (Release I)*, Technical Report TR/PA/92/86, CERFACS, Toulouse, France, 1992. Matrices available at http://math.nist.bov/MatrixMarket/

[15] U. Elsner, V. Mehrmann, F. Milde, R. Römer and M. Schreiber, *The Anderson model of localization: A challenge for modern eigenvalue methods*, SIAM J. Sci. Comput., 20 (1999), pp. 2089-2102.

[16] D. R. Fokkema, G. L. G. Sleijpen and H. A. van der Vorst, *Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils*, SIAM J. Sci. Comput., 20 (1998), pp. 94–125.

[17] G. H. Golub and C. F. Van Loan, Matrix Computations, 3rd ed., Johns Hopkins University Press, Baltimore, 1996.

[18] R. G. Grimes, J. L. Lewis and H. D. Simon, *A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems*, SIAM J. Matrix Anal., 15 (1994), pp. 228–272.

[19] S. Gupta, *A Richardson Leja Lanczos algorithm to compute interior eigenvalues of very large matrices*, Lic. thesis, Department of Mathematics, Chalmers University of Technology, Gothenburg, Sweden, 2001.

[20] R. M. Larsen, *PROPACK: A software package for the symmetric eigenvalue problem and singular value problems based on Lanczos and Lanczos bidiagonalization with partial reorthogonalization*. Code available at the web site http://solar2.stanford.edu/~rmunk/PROPACK. Web site last updated 2001.

[21] R. B. Lehoucq, *Analysis and implementation of an implicitly restarted Arnoldi iteration*, Ph.D. Thesis, Rice University, Houston, 1995.

[22] R. Lehoucq and K. Maschhoff, *Block Arnoldi method*, Section 7.7 in [5], pp. 185–189.

[23] R. B. Lehoucq and D. C. Sorensen, *Deflation techniques for an implicitly restarted Arnoldi iteration*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 789–821.

[24] R. B. Lehoucq, D. C. Sorensen, P. A. Vu and C. Wang, ARPACK: An implementation of an implicitly restarted Arnoldi method for computing some of the eigenvalues and eigenvectors of a large sparse matrix, 1996.

[25] R. B. Lehoucq, D. C. Sorensen and C. Yang, ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods, SIAM, Philadelphia, 1998. Code available at the web site http://www.caam.rice.edu/software/ARPACK

[26] F. Leja, *Sur certaines suits liées aux ensemble plan et leur application à la representation conforme*, Ann. Polon. Math., 4 (1957), pp. 8–13.

[27] O. Marques, *BLZPACK: Description and user's guide*, TR/PA/95/30, CERFACS, Toulouse, France, 1995.

[28] The MathWorks, MATLAB Application Program Interface Guide, Version 5, The MathWorks, Inc., Natick, 1998.

[29] K. Meerbergen and J. Scott, *Design and development of a block rational Lanczos method with partial reorthogonalization and implicit restarting*, Report, Rutherford Appleton Laboratory, Oxon, England, 2000. Available at the web site http://www.numerical.rl.ac.uk/reports/reports.html

[30] R. B. Morgan, *Computing interior eigenvalues of large matrices*, Linear Algebra Appl., 154-156 (1991), pp. 289–309.

[31] C. R. Murray, S. C. Racine and E. R. Davidson, *Improved algorithms for the lowest few eigenvalues and associated eigenvectors of large matrices*, J. Comput. Phys., 103 (1992), pp. 382–389.

[32] C. C. Paige, B. N. Parlett and H. A. van der Vorst, Approximate solutions and eigenvalue bounds from Krylov subspaces, Numer. Lin. Alg. Appl., 2 (1995), pp. 115–134.

[33] B. N. Parlett, *The rewards for maintaining semi-orthogonality among Lanczos vectors*, Numer. Linear Alg. Appl., 1 (1992), pp. 243–267.

[34] B. N. Parlett, The Symmetric Eigenvalue Problem, SIAM, Philadelphia, 1998.

[35] R. Radke, A MATLAB Implementation of the Implicitly Restarted Arnoldi Method for Solving Large-Scale Eigenvalue Problems, Master's Thesis, Rice University, Houston, 1996.

[36] A. Ruhe, *Implementation aspect of band Lanczos algorithms for computation of eigenvalues of large sparse symmetric matrices*, Math. Comp., 33 (1979), pp. 680–687.

[37] A. Ruhe, *Lanczos methods*, Section 5.5 in [5], pp. 116–123.

[38] Y. Saad, Numerical Methods for Large Eigenvalue Problems, Halstead Press, New York, 1992.

[39] D. S. Scott, LASO2 - FORTRAN implementation of the Lanczos process with selective orthogonalization. Code and documentation available from Netlib.

[40] G. L. G. Sleijpen and H. A. van der Vorst, *A Jacobi-Davidson iteration method for linear eigenvalue problems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 401–425.

[41] D. C. Sorensen, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J.

Matrix Anal. Appl., 13 (1992), pp. 357–385.

[42] D. C. Sorensen and C. Yang, *Accelerating the Lanczos Algorithm via polynomial spectral transformations*, Technical Report TR 97-29, Dept. of Comp. and Appl. Math., Rice University, Houston, 1997.

[43] J. Stoer and R. Bulirsch, Introduction to Numerical Analysis, 2nd ed., Springer, New York, 1993.