College of
William & Mary
Department of Computer Science

WM-CS-2005-02

Iterative Validation of Eigensolvers: A Scheme for Improving the Reliability of Hermitian Eigenvalue Solvers

Andreas Stathopoulos

July 2005

ITERATIVE VALIDATION OF EIGENSOLVERS: A SCHEME FOR IMPROVING THE RELIABILITY OF HERMITIAN EIGENVALUE SOLVERS *

JAMES R. MCCOMBS † AND ANDREAS STATHOPOULOS †

Abstract. Iterative eigenvalue solvers for large, sparse matrices may miss some of the required eigenvalues that are of high algebraic multiplicity or tightly clustered. Block methods, locking, *a-posteriori* validation, or simply increasing the required accuracy are often used to avoid missing or to detect a missed eigenvalue, but each has its own shortcomings in robustness or performance. To resolve these shortcomings, we have developed a post-processing algorithm, *iterative validation of eigensolvers* (IVE), that combines the advantages of each technique. IVE detects numerically multiple eigenvalues among the approximate eigenvalues returned by a given solver, adjusts the block size accordingly, then calls the given solver using locking to compute a new approximation in the subspace orthogonal to the current approximate eigenvectors. This process is repeated until no additional missed eigenvalues can be identified. IVE is general and can be applied as a wrapper to any Rayleigh-Ritz-based, hermitian eigensolver. Our experiments show that IVE is very effective in computing missed eigenvalues even with eigensolvers that lack locking or block capabilities, although such capabilities may further enhance robustness. By focusing on robustness in a post-processing stage, IVE allows the user to decouple the notion of robustness from that of performance when choosing the block size or the convergence tolerance.

1. Introduction.

Sparse, iterative eigenvalue solvers are effective at computing the extremal eigenvalues of large, hermitian matrices or matrices represented as functions [17, 7, 28, 27]. However, iterative methods may miss some of the desired eigenvalues if these are of high algebraic multiplicity or are tightly clustered. Moreover, certain initial guesses, eigenvalue distributions, and preconditioning can cause eigenvalues to converge out of the expected order, and thus to be missed. Some techniques have been developed to detect missed eigenvalues or attempt to avoid missing them all together, but a robust and automated process that combines the advantages of each technique has yet to be investigated.

A few *a-posteriori* techniques have been used to deal with missed eigenpairs. Sylvester's matrix inertia [11] is one technique that can be used to determine how many eigenvalues of a Hermitian matrix exist within a particular interval. Although this technique has enabled the development of robust eigenvalue software [13], it requires the LDL^T factorization of the matrix, which can be prohibitively expensive for large, sparse matrices, or even impossible if the matrix is represented as a function. In such cases, practitioners may call the solver again with a random initial guess to obtain an additional eigenvector in the orthogonal complement of the current set of converged approximations. Assuming the smallest eigenvalues are sought, the new approximate eigenvalue converges towards a missed eigenvalue if it is smaller than the largest previously obtained eigenvalue.

Locking and block iterations are two techniques that attempt to reduce the number of eigenvalues missed *during* the iterative process. Locking [4, 31, 18] is performed by freezing converged eigenvector approximations (locked vectors), and keeping current and future vectors in the search space orthogonal to them. Because locked vectors do not participate in the Rayleigh-Ritz minimization they remain unchanged, and the search space can approximate other eigenpairs without duplicating the locked ones. Alternatively, solvers may be implemented in a non-locking way, [19, 33], where converged eigenvectors keep participating in the search space, thereby improving their accuracy. Regardless of implementation, locking a set of a-priori known eigenvectors, X, can be performed outside a solver by providing the matrix $(I-XX^T)A(I-XX^T)$.

*Work supported by National Science Foundation grants: (ITR/DMR 0325218) and (ITR/AP-0112727), and performed using the computational facility at the College of William and Mary, which was enabled by grants from the National Science Foundation (EIA-9977030) and Sun Microsystems (SAR EDU00-03-793).

†Department of Computer Science, College of William and Mary, Williamsburg, Virginia 23187-8795, (mccombjr, andreas@cs.wm.edu).

Locking can be effective in obtaining more than one eigenvector belonging to a multiple eigenvalue, especially with Krylov methods which otherwise can obtain only one such eigenvector in exact arithmetic. It is also used with many preconditioned or non-Krylov solvers [4, 29], as it allows for a smaller active search space which reduces Rayleigh Ritz and restarting costs. Whether locking delivers the purported robustness depends significantly on the implementation details, and typically the most robust implementation may not be the most efficient.

Block iterative methods [27, 3, 26, 37, 25, 36, 13] that extend the subspace by k vectors per iteration, where k is the block size, are a more robust alternative to locking. Assume the single-vector Lanczos process converges to an eigenvalue with algebraic multiplicity k. Then, with appropriate initial guesses, the block Lanczos method with block size k will find all k eigenvectors of that eigenvalue [24]. Although not guaranteed in general, this property is observed in other methods as well. Block methods can also result in better cache utilization and improved performance if a fine-tuned implementation is available. However, block methods are not without drawbacks. Excessively large block sizes can increase the number of matrix-vector multiplications and result in more frequent restarts, thereby slowing convergence.

Locking and blocking can be combined to improve the robustness of iterative eigensolvers [5, 10, 2, 20]. Still, the resulting robustness depends on the problem solved and the parameters chosen by the user. What makes this choice difficult is the conflicting requirements between robustness and performance. A lower than required convergence threshold is less likely to miss eigenvalues but it takes more time. Moreover, the choice of a block size that optimizes simultaneously cache performance, convergence, and robustness is not possible to know a-priori. Our approach in this paper is to let the user choose the most efficient parameters as default, and to provide a post-processing technique that corrects any robustness shortcomings, and only when these are needed. This way, we decouple the issue of robustness from that of performance (cache or convergence).

Our goal is not to provide another hermitian eigensolver, but to develop a robust detection algorithm that uses any given solver in a methodical way to identify whether the requirements posed by the user have been met by the solver. Borrowing a term from Software Engineering, we call our algorithm *iterative validation of eigensolvers* (IVE), as it does not verify whether the approximations are correct, but rather adjusts the solver to find the correct approximations. IVE works in conjunction with any Rayleigh-Ritz-based hermitian eigenvalue solver. The IVE algorithm accepts as input and locks the converged eigenvector approximations. Locking can be performed implicitly in IVE, if not provided by the solver. If a block solver is available, IVE adjusts the block size based on the largest numerical multiplicity found, and calls the solver again so that any missed eigenvalues in the orthogonal complement of the locked vectors can be computed. Any missed eigenvalues detected by IVE are locked out and the process is repeated until no additional missed eigenvalues are computed. Although the structure of the IVE algorithm is general, we only apply it to hermitian eigenproblems where the monotonic convergence of eigenvalues facilitates a robust detection of missed eigenvalues.

We have used our IVE to wrap several well known eigensolvers. The solvers may differ substantially in their ability to use locking or blocking efficiently and robustly, but they invariably miss eigenvalues. Our results show that IVE restores robustness in several pathologically hard cases. Moreover, running the solver through default parameters followed by a certain IVE configuration improves performance, thus demonstrating the desired decoupling of performance and robustness.

In Section 2, we discuss why eigensolvers miss eigenvalues, and the available techniques for increasing robustness. In Section 3, we describe in detail the IVE algorithm. In Section 4, we present results using IVE on a variety of solvers. Throughout the paper we assume that we seek the lowest *nev* eigenvalues $\tilde{\lambda}_i$ and eigenvectors $\tilde{x}_i,\ i = 1,\ldots,nev$ of a sparse, hermitian matrix A.

2. Reasons behind missing eigenvalues and current approaches. A large arsenal of sparse hermitian eigenvalue solvers is available, some based on Krylov methods, others on nonlinear optimization, preconditioning, and a host of other underlying principles [32]. Most of these methods employ the Rayleigh-Ritz (RR) minimization to extract their approximations from a search space. The eigenvalue and eigenvector approximations are called Ritz values and Ritz vectors, respectively. For hermitian eigenproblems, the Ritz values minimize the Rayleigh-quotient over the search space, thus leading to a monotonic convergence toward the required eigenvalues.

Despite their individual strengths, solvers can miss some of the required extreme eigenvalues. The reasons, which are often interrelated, are: (a) the presence of multiple or highly clustered eigenvalues, (b) the out-of-order convergence of some eigenvalues, (c) a starting vector that is deficient in certain required eigenvector components.

It is well known that in exact arithmetic Krylov methods cannot compute more than one eigenvector belonging to a multiple eigenvalue. Krylov methods produce the next vector in the search space as $r = p(A)v_0$, where v_0 is the starting vector and $p(A)$ is the characteristic polynomial of the Lanczos tridiagonal matrix [27]. If a multiple eigenvalue λ_i and one of its corresponding eigenvectors have already been found, then $p(\lambda_i) = 0$ and r contains no components in the kernel: $\text{Null}(A-\lambda_i I)$. In practice, floating-point arithmetic introduces noise toward these directions allowing their eventual computation. To allow the solver enough time to amplify the missing eigencomponents, a low convergence tolerance must be required. Otherwise, a more interior eigenpair may converge first as one of the *nev* smallest ones. Interestingly, non-Krylov methods often encounter similar problems with multiple eigenvalues.

Clustered eigenvalues present similar difficulties as multiple ones. This is not surprising because, until the solver has resolved an eigenvalue to an error that is smaller than its distance from a nearby eigenvalue, it views both eigenvalues as multiple and could miss one if larger tolerances are specified. The slow convergence of solvers toward clustered eigenvalues exacerbates the problem.

More generally, eigenvalues are missed when an interior eigenvalue converges before the solver has identified the existence of an outer one. In Krylov methods this is not the expected order of convergence, causing robustness problems to solvers that are based on the assumption that the *nev* smallest eigenvalues will be found first. The assumption, although usually true, does not hold in general. As shown recently [16], eigenvalue distributions exist for which Krylov methods converge to the eigenvalues in the middle of the spectrum first, and to the extreme ones last. In practice, smaller order reversals may be observed, which can cause eigenvalues to be missed.

The out of order convergence is more frequently observed with non-Krylov methods that use preconditioning or solve approximately a correction equation to speed up convergence [7, 29, 23]. For example, consider a matrix A, whose absolute smallest eigenvalues are required, and the Generalized Davidson method using a preconditioner $M \approx A^{-1}$. Because M and A do not share the same eigenvectors, the convergence benefits from M are not necessarily greater for eigenvalues closest to zero, thus yielding an arbitrary convergence order. Davidson-type methods also use various targeting schemes [33], i.e., which Ritz vector is improved at every step, directly affecting the order in which eigenvalues converge. In floating point arithmetic, targeting effects are observed even without preconditioning [22, 35]. Finally, whether the implementation of the solver or the eigenvalue distribution of the matrix could lead to misconvergence depends also on the initial guess.

2.1. Current approaches. To increase confidence in the computed eigenpairs, practitioners typically ask for more accuracy than needed. Although beneficial in many cases, in general we may not know the accuracy that is sufficient for resolving multiple eigenvalues or avoiding out of order convergence. Also, this approach may be unnecessarily expensive, especially when a very low tolerance is used to find a large number of eigenvalues of which only a few would have been missed with a larger tolerance. Finally, the problem with higher multiplicities in Krylov methods persists.

Another common strategy is to compute more eigenpairs than required. This works if the reversals in the order of convergence are localized, e.g., the tenth smallest eigenvalue converges before the ninth, but not before the second. In general, we do not know how many more eigenpairs to compute, so the strategy can be unnecessarily expensive. Most importantly, it still may not find multiple or clustered eigenpairs.

Deflation of converged eigenpairs is commonly used in many eigensolvers so that more eigenpairs can be obtained without repeating the already computed ones. Locking is the preferred form of deflation because of its numerical stability [27, 4]. When an eigenvector converges, it is extracted from the search space and all current and future vectors of the search space are kept orthogonal to that (locked) vector. Locked vectors are inactive otherwise, they do not participate in the RR (as future Ritz vectors should not have any components in them), and therefore they are not modified further. Computationally, the smaller active search space reduces the costs of the RR and restarting phases of eigensolvers; more so when a large number of eigenpairs are sought. This is one of the reasons that locking is preferred to non-locking implementations for subspace iteration and Jacobi-Davidson type methods [5, 29].

Locking provides also a more reliable means of computing the invariant spaces of multiple eigenvalues, especially with Krylov methods. Assume that a subset X of the invariant subspace associated with λ has been computed and locked. This is equivalent to calling the solver with the deflated operator $(I-XX^T)A(I-XX^T)$, which has all the eigenvalues associated with X equal to zero. Unlike the polynomial deflation $p(A)v_0$ in Krylov methods which removes all the invariant subspace of λ from the vector iterates, locking removes only the computed part of $\mathrm{Null}(A-\lambda I)$. More eigenvectors of λ can then be obtained. This deflation behavior is not restricted only to Krylov methods. Finally, locking also speeds up the convergence to any missed, single eigenvalues which are amidst other converged ones. In the deflated operator such eigenvalues are better isolated and therefore are easier to find. However, even with locking, misconvergence can still occur if some tightly clustered or multiple eigenpairs converge out of order. Moreover, not all solvers implement locking, and when they do, their implementation details and thus robustness may vary. For example, locking can be implemented transparently to the solver if we use the deflated operator in the matrix-vector multiplication, but the solver must return the converged eigenpairs before they are locked. Alternatively, some solvers implement locking as part of the orthogonalization phase. We discuss these issues further in the next section.

Block methods are usually more effective in computing multiple eigenvalues. As long as the k initial guesses in the block contain sufficient components in the direction of the eigenvectors associated with the multiple eigenvalue, then block Krylov methods are guaranteed to find at least k of the multiple values [24]. Moreover, more uniform convergence of the block vectors is observed, reducing the likelihood of out-of-order misconvergence. Blocking is known to improve robustness in all iterative methods, but the effects are more emphasized in Krylov methods [12]. Block methods also improve cache performance with larger k [8], seemingly offering a panacea to all problems.

Despite the better cache utilization, robustness comes at a cost in execution time. Although the number of iterations decreases, the overall number of matrix-vector operations (and thus flops) performed is usually higher than their single-vector counterparts [14]. Especially with spectra that are easily computed, block methods tend to require close to k times more matrix-vector operations than single-vector methods, completely canceling any caching benefits. Additionally, the number of iterations may also increase if k is too large relative to the maximum basis size. This would cause very frequent restarts impairing the convergence of most methods; this is apparent in some experiments later in this paper. When k equals the maximum basis size, the solver reverts to subspace iteration, losing the subspace acceleration [32]. In general, an optimal choice for k must ensure appropriate cache utilization, robustness, and the smallest possible increase in flops. Except for caching which depends on the architecture, none of the other variables are known ahead of time. In particular, the k needed to detect all the required eigenvectors is not known, and even if it were, it could conflict with the optimal choice for cache performance and convergence. Finally, unlike locking, blocking can be used only if implemented by the available solver.

We conclude our discussion, with the importance of the initial guess. In all the aforementioned techniques, a critical assumption is that the initial vector has sufficient components in the desired directions. In lack of better information, a random vector is typically the best initial guess. Yet, as the iterative process progresses, certain required directions may be diminished or removed as in the case of Krylov methods and multiple eigenvalues. A possibility is to insert a random vector in the search space or in the block, replacing an eigenpair that converges. This can be performed trivially in (Jacobi)-Davidson or subspace iteration methods, but techniques have also been proposed for the implicitly restarted Lanczos method [31, 30]. Still, the newly introduced directions may not be amplified fast enough to prevent out of order convergence of an interior eigenpair which was almost converged before the insertion of the random vector. A drastic solution would be to dispense with the whole search space after an eigenpair converges and restart the solver anew with a random guess. A related approach that does not completely rid of the search space is suggested in a yet unpublished report [21]. Based on our discussion on locking, a newly rebuilt search space would offer the best possible robustness. However, it would also be very slow, especially if no eigenvalues were going to be missed.

A recurring theme is the dichotomy between robustness and performance. To achieve robustness, solvers would have to make extreme choices on parameters such as block size and tolerance that would impair performance. In the following section we present our post-processing approach to validating the parameters of a given solver that are required to achieve robustness.

3. Iterative validation of eigensolvers. We propose a new algorithm which we call *iterative validation of eigensolvers* (IVE). IVE is implemented as a wrapper around any existing sparse, iterative, hermitian eigensolver that utilizes the Rayleigh-Ritz (RR) minimization procedure. IVE automates the process of computing eigenvalues missed by the solver, by combining as many of the techniques mentioned in Section 2.1 as the solver implements. In other words, IVE is not a solver, nor does it compute eigenvalues. IVE merely identifies difficulties in the computed spectrum, and sets locked vectors, initial guesses, and block size appropriately so that the chosen eigensolver may compute any missed eigenvalues on its own. In this sense, it is similar to the validation process of a software engineering cycle, where a software is modified appropriately (different solver parameters) to meet the customer requirements (which eigenpairs to find), rather than verifying whether the software meets the specifications (produces converged eigenpairs). Therefore, the measures of performance and robustness achieved by IVE can only be in reference to the underlying solver. Also, by being a post-processing algorithm, IVE may be switched off if no validation is needed.

The enabling property for IVE is that the Ritz values of RR-based hermitian eigensolvers converge monotonically, if their Ritz vectors are retained in the search space during restarts or truncations. Once a Ritz value becomes smaller than a converged eigenvalue, we are sure that a previously undetected eigenvalue exists. Although Newton eigensolver schemes can be derived without RR, most current algorithms and software incorporate it, so our assumption is not restrictive in practice.

Assume the solver has returned an initial set of *nev* converged Ritz pairs (λ_i, x_i), where $\lambda_i \leq \lambda_{i+1}$, $i = 1, \ldots, nev-1$. The IVE algorithm analyzes the converged λ_i and the norms of their residuals, $r_i = Ax_i - \lambda_i x_i$, to determine an appropriate block size. IVE then calls the solver with new initial guesses, using the converged $X = \{x_i\}$ as locked vectors, in an attempt to find any missed eigenvalues. An eigenvalue is a missed eigenvalue if it is smaller than the largest locked eigenvalue λ_{nev}. If it is, then its eigenvector replaces the locked eigenvector x_{nev} in the set of locked vectors. The IVE algorithm then repeats an analysis of the block size and proceeds as before. IVE works also with solvers that do not provide locking or blocking. First, we discuss some issues on locking and how to choose a block size. Then we present the IVE algorithm, describing how it chooses initial guesses and how many additional eigenpairs it finds.

3.1. Locking variants. One way to compute eigenvectors orthogonal to the set X is for the solver to provide a mechanism to lock (orthogonalize against) a set of externally provided vectors [20, 1]. This feature is a relatively simple extension to the orthogonalization procedure that most solvers employ. It is also numerically stable because the orthogonalization procedure should guarantee that no components of X appear in the search space. In this case, IVE simply passes X to the solver. Note that a solver that locks against an externally provided X, does not have to implement locking internally for the eigenpairs it computes.

When a solver does not provide the above external locking feature, an alternative is to provide all the converged X together with a random vector as initial guesses to the solver and ask for a few more eigenpairs. This works with many solvers such as (Jacobi-)Davidson, subspace iteration, or methods where X can be provided as part of the initial block, but it does not work with Lanczos algorithms. In addition, the solver must accommodate a large enough search space, and all X eigenvectors have to be re-checked and possibly modified.

A third alternative is to use explicit deflation, by calling the solver not with the matrix A, but with a function that implements the action of the operator:

$$(I - XX^T)(A - \sigma I). \tag{3.1}$$

The preconditioner may be deflated in a similar way as shown in [28]. The shift σ plays an important role. If $\sigma = 0$ and the smallest eigenvalues of A are positive, the eigenvalues of (3.1) associated with X are zero and thus they are still the smallest. In exact arithmetic this is equivalent to the external locking above, and no X components should appear in the search space. In floating point, however, when new vectors are orthogonalized against the search space inside any solver, they may lose their orthogonality against X. As X is not available within the solver, orthogonality cannot be recovered through reorthogonalization. Thus, components of X will start to emerge in the RR, and their "zero" eigenvalues will be re-computed.

The role of σ is to shift the spectrum so that zero is far from the smallest eigenvalues; ideally in the middle or at the other end of the spectrum. Then, the X components that emerge in the search space will be in the unwanted part of the spectrum and they will not be chosen by RR (purged internally [18]). When explicit locking is necessary, we have used $\sigma = \|A\|_F/\sqrt{n}$ as a lower bound to the largest eigenvalue of positive spectra, where $\|A\|_F$ is the Frobenious norm and n the dimension of A. Other shifts can also be used, if estimates of

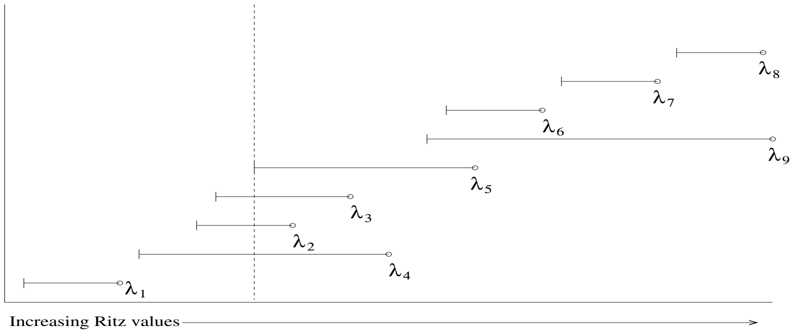

FIG. 3.1. *Sample distribution of converged Ritz values and their associated error intervals. The maximum algebraic numerical multiplicity in this example is four because a point of maximum overlap, indicated by the dashed vertical line, crosses four error intervals. The rest of the eigenvalues have multiplicities* $\lambda_1(1), \lambda_2(4), \lambda_3(4), \lambda_4(4), \lambda_5(4), \lambda_6(3), \lambda_7(2), \lambda_8(2), \lambda_9(3)$.

λ_n are known [27]. This technique yields significant autonomy to IVE, not depending on the locking features of the solver.

3.2. Determining block sizes. Assume that a block eigensolver is available. Our theoretical premise is that a block size equal to the largest multiplicity should be sufficient to resolve any problems. The IVE algorithm selects such a block size based on the largest "numerical multiplicity" observed in the converged Ritz pairs.

Let δ_i be an error bound and $[\lambda_i - \delta_i, \lambda_i]$ the error interval corresponding to the Ritz value λ_i. Because the matrix is hermitian, $\tilde{\lambda}_i \leq \lambda_i$. Depending on the eigenvalue distribution and on the size of the user-specified tolerance, the error intervals associated with the Ritz values may overlap. Two Ritz values whose error intervals overlap are *numerically multiple*. It is possible for three or more Ritz values to be numerically multiple, but this requires a more general definition. The Ritz values $\{\lambda_i, i \in \{i_1, i_2, \ldots, i_k\}\}$, are *numerically multiple* if and only if $[\lambda_i - \delta_i, \lambda_i]$ overlaps with $[\lambda_j - \delta_j, \lambda_j]$ for all $i, j \in \{i_1, i_2, \ldots, i_k\}$.

Sharp bounds on the δ_i are well-known, when the approximations are computed through the RR procedure [24]. Let $\tilde{\lambda}$ be the single eigenvalue closest to a Ritz value λ, and let r be the residual corresponding to λ. The following inequality holds:

$$|\tilde{\lambda} - \lambda| \leq \delta = \min\left(\|r\|_2, \frac{\|r\|_2^2}{\min\left(|\tilde{\lambda}_i - \lambda|,\ \tilde{\lambda}_i \neq \tilde{\lambda}\right)} \right). \tag{3.2}$$

Near convergence (which is the case in IVE), the upper bound δ in (3.2) can be accurately estimated by substituting the Ritz value closest to λ for $\tilde{\lambda}_i$.

Let the *algebraic numerical multiplicity*, m_i, of a Ritz value λ_i be defined as the size of the largest set of Ritz values that include λ_i and are numerically multiple. A robust choice for block size can be the maximum m_i taken over all Ritz values. The rationale is that other required eigenvalues may have been missed that have the same multiplicity. Although this works well for small m_i, for large m_i it is highly uncommon that the eigenvalue with the maximum m_i is still missing half of its actual multiplicity, making this a pessimistic choice. In addition, the block size should not become too large relative to the maximum size of the search space, depending on the solver. For example, for LOBPCG [15] the block size must be a third of the basis size; Lanczos and Davidson solvers work better when the block size is much smaller than the basis size. Because of these external to IVE factors, or because some information about the problem may be known (multiplicity or cluster size), the user may put

ALGORITHM: MaxAlgNumMult(λ, r, nev)
Compute δ_i the error estimate from (3.2) corresponding to λ_i.
MaxAlgNumMult = 0
lowerBound(i) = $\lambda_i - \delta_i$, $i = 1, \ldots, nev$
Let p_i be the number of overlapping intervals at endpoint $\lambda_i - \delta_i$
for $i = 1, nev$
    $p_i = 0$
    for $j = 1, nev$
        if (lowerBound(i) $\geq$ lowerBound(j) **and** lowerBound(i) $\leq \lambda_j$)
            $p_i = p_i + 1$
    end
    MaxAlgNumMult = **max**(MaxAlgNumMult, p_i)
end

FIG. 3.2. *Algorithm for computing the maximum algebraic numerical multiplicity among a set of Ritz values corresponding to the smallest eigenvalues of a Hermitian matrix.*

a maximum cap on the block size. We emphasize that a cap based on the underlying solver and its maximum basis size does not reduce robustness, because for larger block sizes that solver's own robustness and convergence would deteriorate. Also, blocking is just one of the robustness techniques that IVE employs.

Computing the maximum algebraic numerical multiplicity is equivalent to finding a "point of maximum overlap", i.e., a point that has the largest number of error intervals overlapping it. It can be shown that there will always be a point of maximum overlap at one of the endpoints of the intervals [6]. Figure 3.1 shows an example distribution of Ritz values and their associated error intervals. Figure 3.2 gives an algorithm, *MaxAlgNumMult*, for computing the maximum m_i in $O(nev^2)$. Interestingly, the p_i in the algorithm do not correspond to the m_i for all i, yet the maximum m_i is computed correctly. A more efficient, $O(nev \log_2 nev)$ time algorithm exists, but its implementation is involved. Moreover, as nev is usually $O(1000) \ll n$, the execution time of our algorithm is negligible even compared to a matrix-vector product, obviating the use of the lower complexity algorithm. To our knowledge, this is the first algorithm to find the maximum multiplicity in a set of approximate eigenpairs.

3.3. The IVE algorithm. The IVE algorithm given in Figure 3.3 first computes the residual norms of the converged Ritz vectors (step 4). This step is not required if the eigensolver returns this information. Then, IVE determines the maximum algebraic numerical multiplicity using *MaxAlgNumMult* (step 5). In step 6, if the solver implements a block method, the block size is chosen so that it does not increase beyond the maximum value *maxBlockSize*. Below that value, it is set at the maximum of the current maximum m_i, the block size chosen by the user to compute the initial set of converged Ritz values, and two. A block size of at least two is used to force a block method when no numerically multiple eigenvalues have been detected. If the initial run of the solver used a block size of k, there is no incentive to use anything else for validation. Still, the user can overwrite it by passing $initialBlockSize = 1$. In step 7, the initial guesses, X_0, are chosen. In the first IVE iteration, X_0 is a set of all random initial guesses. In subsequent IVE iterations, X_0 includes also any unconverged Ritz vectors that are known to be missed during the previous IVE iteration (see below). As we discussed in Section 2.1 it is always important to include some random guesses.

In step 8, the eigensolver is called with the initial guesses X_0 and locked vectors X to

ALGORITHM: $[X, \lambda]$ = IVE(A, X, λ, resNorms, initialBlockSize, maxBasisSize)
Let X and λ be the converged Ritz vectors and Ritz values.
Let initialBlockSize be the block size used to compute X.
Let maxBlockSize be the maximum block size allowed
1. numNew = 1
2. $X_{missed} = [\,]$, $X_{unconverged} = [\,]$
3. **repeat**
4. resNorms = ComputeResNorms(A, X, λ)
5. newMult = MaxAlgNumMult(λ, resNorms)
6. **if** (solver implements block)
   blockSize = **min**(maxBlockSize, **max**(newMult, initialBlockSize, 2))
   else
   blockSize = 1
7. X_0 = $[X_{missed}$, **rand**(blockSize - **size**$(X_{unconverged}))]$
8. Compute $[X_{new}, \lambda_{new}, X_{unconverged}, \lambda_{unconverged}]$ by calling the solver:
   if (solver implements external locking)
   Solver(A, numNew, X_0, X)
   else
   Solver($(I - XX^T)(A - \sigma I)$, numNew, X_0)
9. $[X, \lambda]$ = InsertionSort(X_{new}, λ_{new}, numNew, X, λ, resNorms)
10. numMissedUnconv = NumMissedValues(λ, $\lambda_{unconverged}$, blockSize-numNew)
11. $X_{missed} = X_{unconverged}$(:,1:numMissedUnconv)
12. numNew = **max**(1, numMissedUnconv)
13. **until** (X did not changed **and** numMissedUnconv $= 0$)

FIG. 3.3. *IVE algorithm for computing missed eigenpairs. IVE is a wrapper around an existing Hermitian, Rayleigh-Ritz based eigensolver.*

find *numNew* more eigenpairs. Depending on the solver, external or explicit locking can be performed. Note that the IVE algorithm assumes that the chosen solver returns not only the *numNew* converged Ritz values in λ_{new}, but also the remaining *blockSize−numNew* unconverged Ritz values in $\lambda_{unconverged}$. Solvers that do not return the unconverged Ritz pairs by default can be easily modified to do so. Otherwise, we can set *numMissedUnconv* $= 0$.

In step 9, an insertion sort (see Figure 3.4 for details) is called to insert any converged, missed eigenpairs in λ_{new} and X_{new} into λ and X, respectively. For every missed Ritz value inserted into the λ array, the largest λ_{nev} in the array and the corresponding vector in X are removed. If enough storage is available, those removed converged eigenvectors can still remain in the locked array to avoid possible recomputation. We choose not to take advantage of this feature in our experiments. If none of the recently converged Ritz values were missed, then the IVE algorithm will terminate at step 13.

Step 10 calls the algorithm *NumMissedValues* in Figure 3.5 to determine how many of the unconverged Ritz values in $\lambda_{unconverged}$ are missed values. A Ritz value in $\lambda_{unconverged}$ is considered missed if it is smaller than λ_{nev}. Because of monotonic convergence, these Ritz values will only continue to decrease and approach missed eigenvalues. Therefore, it is wise to use the corresponding unconverged Ritz vectors in $X_{unconverged}$ as initial guesses (step 11). These can significantly reduce validation time when many eigenvalues are missed, because they enable the solver to avoid repeating convergence through random guesses.

Finally, step 12 sets the new number of Ritz values the solver must compute. Clearly, we should continue and find the *numMissedUnconv* missed ones that step 10 may have identified.

ALGORITHM: $[X, \lambda]$ = InsertionSort(X_{new}, λ_{new}, numNew, X, λ, resNorms, nev)

for $i = 1$:numNew
  $j = nev$
  while $(j > 0$ **and** $\lambda(j) > \lambda_{new}(i))$, $j = j - 1$; **end**
  if $j == nev$, **break**;
  $X = [X(:, 1:j), X_{new}(:, i), X(:, j+1:nev-1)]$
  $\lambda = [\lambda(1:j), \lambda(i), \lambda(j+1:nev-1)]$
  $r_{new} \equiv$ the residual with respect to $(X_{new}(i), \lambda(i))$
  $resNorms = [resNorms(1:j), r_{new}, resNorms(j+1:nev-1)]$
end

FIG. 3.4. *Insertion sort algorithm for identifying and inserting missed eigenvalues into λ. Handling of boundary cases is not shown. In actual implementations, the eigenvalues are inserted first, and the eigenvectors are inserted only at the end to avoid unnecessary memory copying.*

ALGORITHM: NumMissedValues(λ, nev, $\lambda_{unconverged}$, numUconv)

NumMissedValues = 0
for $i = 1$, numUconv
  $j = nev$
  while $\left(j > 0 \text{ and } \lambda(j) < \lambda_{unconverged}(i)\right)$, $j = j - 1$; **end**
  if $j == nev$, **break** ;
  $\lambda = [\lambda(1:j), \lambda_{unconverged}(i), \lambda(j+1:nev-1)]$
  NumMissedValues = NumMissedValues + 1
end

FIG. 3.5. *Algorithm for computing the number of unconverged Ritz values that have been skipped. The corresponding Ritz vectors will be used as initial guesses at the next IVE iteration.*

If $numMissedUnconv = 0$, we have no indication that more eigenvalues may be missing, so it would be wasteful to find more than one additional eigenvalue. The IVE algorithm terminates in step 13 if no missed eigenvalues (converged or unconverged) were discovered, or continues with the next IVE iteration.

We present an example in Figure 3.6 to illustrate the steps taken by the iterative validation algorithm. Suppose we seek the $nev = 8$ smallest eigenvalues of a symmetric matrix. In steps 4-6 the residual norms are computed and the maximum algebraic numerical multiplicity is determined to be 3. All random initial guesses are selected in step 7 because it is still the first iteration. In step 8 the solver is called and returns $numNew = 1$ converged Ritz value in λ_{new}, and two other Ritz values in $\lambda_{unconverged}$, because the block size is three. Next, *InsertionSort* is called in step 9 to determine if λ_{new} is a missed eigenvalue. It is, so it is inserted into its proper position and the largest element in λ is discarded. In steps 10-12, *NumMissedValues* determines that one of the $\lambda_{unconverged}$ was converging towards a missed eigenvalue. The corresponding missed vector is placed in X_{missed} to be used as an initial guess in the next IVE iteration, and numNew is set to 1 to indicate that an additional eigenvalue has been skipped. At the next IVE iteration, steps 4-7 determine the algebraic numerical multiplicity to be 4, set the block size accordingly, and assign X_{missed} and three random vectors as initial guesses. The IVE iterations continue until no more missed eigenvalues are detected.

4. Experimental evaluation. Our primary goal is to show that IVE restores robustness in any hermitian eigensolver, even for pathological cases with large multiplicities, or tightly clustered eigenvalues. The importance of increased confidence in the computed results, sometimes at any cost, cannot be overstated. Of particular interest are cases of eigensolvers that fail

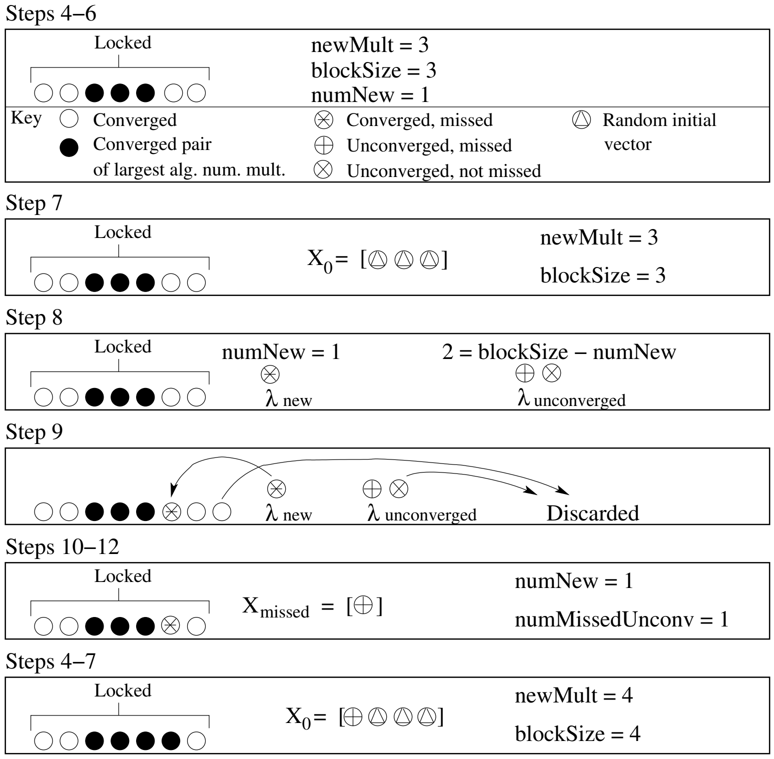

FIG. 3.6. *Example validation problem.*

to find all the required eigenvalues, regardless of the initial choice of parameters, yet guided by IVE, they succeed. A secondary goal is to demonstrate how IVE can decouple robustness from performance, so that solvers are run with default settings for best performance and any missed eigenpairs are obtained later in a shorter IVE cycle.

4.1. Eigensolvers used in the evaluation. We have used five different eigensolvers from three eigensolver packages. The first is IRBL, the Implicitly Restarted Block Lanczos method [2, 1], which uses implicit restarting combined with Leja shifts to improve convergence of the standard Lanczos method. IRBL is a block method and implements both external and internal locking. For our tests, we have added three lines of code in IRBL to allow it to return all Ritz pairs in the block, not only the converged ones. IRBL is implemented in MATLAB, and therefore it is used mainly to demonstrate the IVE benefits on robustness and convergence; not on timings.

The second symmetric solver is the function dsaupd from the popular software package ARPACK [19]. Henceforth, we will refer to the particular eigensolver as ARPACK. ARPACK is based also on implicit restarting, but unlike IRBL, it uses the Ritz values as the restarting shifts. ARPACK is a single vector method and does not implement external or internal locking. Thus, in a sense, it represents a worst case scenario, as IVE cannot employ all of its techniques. In particular, IVE is restricted to using *blockSize* $= 1$, *numMissedUnconv* $= 0$, and thus *numNew* $= 1$, and locking must be implemented explicitly in IVE. No modifications were made to the ARPACK code which is implemented in Fortran 77.

The remaining three solvers come from PRIMME, a software package that we have developed in C, and which is freely available [20]. PRIMME, or PReconditioned Iterative Mul-

```
N = 30000; clusters = 30; multiplicity = 8; DeltaCluster = 1e-8;
D = zeros(N,1); Ones = ones(multiplicity,1);
D(1:multiplicity) = eps*Ones; value = 1.e-6;
for i = 1:clusters-1
   D(i*multiplicity+1:(i+1)*multiplicity) = value*Ones;
   value = value+DeltaCluster;
end
% The rest well separated and equidistant
row = clusters*multiplicity+1; separation = (1-1e-3)/(N-row);
D(row:N) = 1e-3+ (row-1:N-1)*separation;
A = spdiags(D,0,N,N);
```

FIG. 4.1. *The Matlab code that generates the diagonal matrix DIAG.*

tiMethod Eigensolver, is based on a Davidson-type main iteration, but it implements various techniques such as blocking, external and internal locking, CG-type restarting, and adaptive inner-outer iterations, that allow it to transform to any current eigenvalue method. We choose three methods for their popularity and efficiency.

The first is the Generalized Davidson method with CG-restarting, or GD+k [34]. We use it with block and locking enabled. Its main characteristic is the near optimal convergence in terms of matrix-vector products for small *nev*. But this also means that there is little room for convergence improvement through larger block sizes.

The second method is the JDQMR variant of the Jacobi-Davidson method [32, 29]. It is used with block and locking options enabled, although the correction equation corresponding to each block vector is solved independently. JDQMR adaptively stops the inner iteration for each correction equation, yielding convergence near that of GD+k, but the per-iteration cost of JDQMR is much less expensive.

The third method is LOBPCG-W, which is a variant of the increasingly popular LOBPCG method [15], and it differs from it in the following two ways: First, LOBPCG-W maintains an orthogonal search space to guarantee numerical stability of the process. Second, LOBPCG-W uses a window approach, i.e., the blockSize is allowed to be smaller than *nev*, and as eigenvalues are found in the block they are locked out, and the window progresses until all *nev* pairs have been found.

4.2. Experimental setup and environment. We perform experiments on three matrices, each highlighting a different aspect of the IVE capabilities. The first matrix, BCSSTK16 from the Harwel-Boeing collection [9], has an eigenvalue with unusually high multiplicity. The second matrix, DIAG, is diagonal and is constructed as shown in Figure 4.1 to have a large number of clusters of multiple eigenvalues. The third matrix, LAPLACE, comes from the usual 7-point discretization of the 3-D Laplacian operator on the uniform grid $30\times30\times30$ with Dirichlet boundary conditions, and it has several eigenvalues with small multiplicity.

Although, we make an effort to use similar parameters for all methods (tolerance, basis and block sizes, etc.), the reader should not focus on comparisons across methods. Different implementations make such comparison difficult, but most importantly, IVE improvements can only be understood in reference to the same method.

We perform experiments with two or three different tolerances, which is the only variable from Section 2.1 that IVE does not vary. For IRBL, we request that the 2-norm of the residual is less than $\tau\lambda_{max}$. For ARPACK, we provide the tolerance τ to the method. For all PRIMME solvers we request that the 2-norm of the residual is less than $\tau\|A\|_F$. If $\varepsilon = 2.2204^{-16}$ is the machine precision, we consider three cases; $\tau = \varepsilon$, $\tau = \sqrt{\varepsilon}$, and a large τ that is just small enough to differentiate between wanted and unwanted eigenvalues. For these tolerances and

Table 4.1A: IRBL/BCSSTK16 Initial runs

	τ		
k	ε	$\varepsilon^{1/2}$	2.023e-5
1	57, 162390	28, 15728	21, 5514
2	68, 211526	32, 17616	23, 7706
3	74, 265125	32, 17844	23, 7665
4	73, 298884	32, 13748	24, 7168
5	74, 283880	32, 489830	24, 20700
6	74, 272868	38, 11340	20, 5562
7	74, 287693	29, 9541	21, 9723
8	74, 318568	32, 9560	19, 4896
9	74, 363123	36, 11007	23, 5121
10	74, 398170	32, 12890	20, 4470
15	74, 541230	39, 12600	22, 4875
20	74, 679760	56, 9140	37, 4560
74	74, 118696	74, 114848	74, 8362

Table 4.1B: IRBL/BCSSTK16 IVE runs

	τ		
mxk	ε	$\varepsilon^{1/2}$	2.023e-5
1	72992	6921	5828
2	97292	7821	6028
3	76223	9229	4286
4	59392	7721	3628
5	53892	6621	4528
6	41756	6745	3680
7	64478	6981	4244
8	40988	6937	3968
9	52166	7546	4484
10	58092	6121	4528
Best total MV for Initial + IVE run			
	203378	21849	9142
k,mxk	1, 8	1, 10	1, 4

TABLE 4.1

IRBL on matrix BCSSTK16. Table A shows results for computing the 74 smallest eigenvalues. For each block size (k) and tolerance ($\tau\lambda_{max}$) we report (nevFound, MV), where nevFound is how many of the 74 computed eigenvalues are the required ones, and MV the number of matrix-vector multiplications. We underline the case that provided the initial set of vectors for the later IVE run. Table B shows the IVE results. For all maximum block sizes (mxk) and tolerances, IVE recovers all the 74 required eigenvalues, so we only report the number of MV performed during validation alone. The bottom of Table B shows the total MV required (initial run plus IVE) considering the fastest IVE run. The initial block k and the best mxk are also provided.

for a variety of block sizes, we first perform an initial run of a solver, and record the number of desired eigenvalues it was able to find and its performance. Then, using the approximations produced by a certain initial run, we let IVE guide the solver under various values for maxBlockSize, and record the total number of desired eigenvalues found and the corresponding performance. For each method and matrix, results appear in one table.

IRBL experiments are run under MATLAB 6 on a Sun Microsystems Ultra 60 workstation. All other experiments are run on a PowerMac G5, with 1 GB of memory, 512 KB of L2 cache, 1 GHz memory bus, and two 2 GHz PowerPC processors, of which only one is used. ARPACK is compiled with g77 and -O optimization, while PRIMME solvers are compiled with gcc version 4.0.0, and -O optimization. All codes are linked with the Mac optimized VecLib library that includes LAPACK and BLAS.

4.3. Results with matrix BCSSTK16. The dimension of the matrix is 4884, and we seek the 74 smallest eigenvalues which are all 1.0. The next largest eigenvalue is $O(1e6)$, and the largest one is 4.5e-9. This matrix is useful for showing the behavior of solvers and IVE in the presence of an exact, high multiplicity.

First, we test IRBL in Table 4.1. In Table 4.1A we perform several initial runs with block sizes k, each using a maximum basis size of $m = k\lfloor 100/k \rfloor$, a maximum degree of the dampening polynomial equal to the dimension of the matrix, and a size of the dampening interval of $m/2$. For the smallest tolerance, IRBL finds all needed eigenvalues with $k > 4$. The best performance is with $k = 74$, but we assume we do not know the multiplicity a priori. For the larger tolerances, IRBL cannot produce the invariant space of the multiple eigenvalue without $k = 74$.

Table 4.1B shows the performance of various IVE runs, each for a different maximum block size (*mxk*) and starting after the underlined initial run in Table 4.1A was performed. IVE helps IRBL find all 74 required eigenvalues for all tolerances τ and *mxk*. Because, IRBL is a method that benefits in general from larger block sizes, it is advisable to allow larger values of *mxk*. Indeed, the number of matrix-vector operations taken by the initial run followed by the IVE($mxk = 10$) is comparable to or significantly smaller than the most robust initial run for $\tau = \varepsilon, \tau =$2e-5. For certain moderate values of *mxk*, performance of IRBL-IVE improves

Table 4.2A: GD+k/BCSSTK16 Initial runs

k	τ: ε	$\varepsilon^{1/2}$	1e-6
1	37, 15097, 93	20, 5469, 37	13, 3232, 25
2	39, 16994, 97	24, 6883, 42	16, 4059, 27
3	41, 19591, 113	25, 7593, 46	17, 4510, 30
4	45, 20247, 121	27, 7700, 48	19, 4652, 30
5	49, 22214, 135	28, 7974, 50	20, 4968, 33
6	44, 28814, 168	27, 10640, 67	19, 6046, 40
7	54, 23754, 149	30, 8834, 56	22, 5451, 37
8	51, 25012, 155	29, 9761, 63	21, 5884, 39
9	48, 35213, 225	28, 12922, 85	21, 7476, 53
10	74, 40940, 305	37, 9557, 72	27, 6167, 47
15	74, 37138, 370	36, 8134, 83	32, 5181, 49
20	74, 31794, 362	42, 9027, 104	26, 5018, 60
30	73, 24647, 364	42, 7249, 102	30, 4609, 71
40	74, 26451, 497	42, 7111, 135	42, 4473, 87
50	74, 20936, 458	55, 7089, 165	52, 4789, 111
74	74, 19058, 600	73, 6946, 220	71, 4575, 153

Table 4.2B: GD+k/BCSSTK16 IVE runs

mxk	τ: ε	$\varepsilon^{1/2}$	1e-6
1	9572, 82	5943, 45	4274, 33
2	9464, 77	6070, 42	4564, 32
3	8679, 69	6028, 41	4803, 33
4	9300, 77	6284, 45	4784, 34
5	9585, 78	6256, 47	5056, 38
6	9555, 78	5980, 45	5060, 36
7	11044, 95	6547, 49	4973, 38
8	15584, 152	6944, 52	4802, 35
Best total MV/times for Initial + IVE run			
	28270, 182	11497, 78	7796, 57
k,mxk	3, 3	1, 3	1, 2

TABLE 4.2

GD+k on matrix BCSSTK16. Table A shows results for computing the 74 smallest eigenvalues. For each block size (k) and tolerance ($\tau\|A\|_F$) we report (nevFound, MV, time), where nevFound is how many of the 74 computed eigenvalues are the required ones, MV the number of matrix-vector multiplications, and time is in seconds. We underline the case that provided the initial set of vectors for the later IVE run. Table B shows the IVE results. For all maximum block sizes (mxk) and tolerances, IVE recovers all the 74 required eigenvalues, so we only report the number of MV and time taken during validation alone. The bottom of Table B shows the total MV and time required (initial run plus IVE), considering the fastest IVE run. The initial block k and the best mxk are also provided. IVE is required for robustness ($\sqrt{\varepsilon}$ and 1e-6), and can improve time (ε).

further. Therefore, users are free to tune their code for performance without worrying about its robustness.

Table 4.2 shows similar results for the GD+k method. The maximum basis size used for both initial and IVE runs was 30 for $k < 10$ and $6k$ for $k \geq 10$. The GD+k method requires a larger block size to find the desired invariant space, however, at a great cost in performance. It is thus more effective to select a block size for its caching performance in the initial run (e.g., $k = 3$ in $\tau = \varepsilon$ case), and then let IVE resolve the multiplicity with a similar block size. For the $\tau = \varepsilon$ case, this strategy improves performance often by a factor of two or three over just using GD+k with a sufficiently large block. For the larger tolerance cases, IVE is the only way to compute the required eigenvalues.

Table 4.3 shows the results with the JDQMR method. The maximum basis sizes used were as in the GD+k method. Despite its efficiency, JDQMR has trouble identifying high multiplicities even with large block sizes. For inner-outer methods, such as JDQMR, this is not surprising as they focus mostly on improving particular eigenvectors, and thus are prone to misconvergence. Also, large block sizes are usually not beneficial. For all three tolerances, IVE restores robustness to JDQMR, and for a relatively low cost. Note, the best times observed would not increase substantially if a larger *mxk* were used.

Table 4.4 shows results with LOBPCG-W. This method requires that the maximum basis size is $3k$. LOBPCG is inherently a block method that should benefit from larger k. Although, the initial run of the method finds more required eigenvalues with small k than other methods do, a few ones are always missed, even with $k = 74$. A surprising exception is the $k = 1$ in the $\tau = \varepsilon$ case, which finds all of them, but such behavior is not expected in general. IVE guided LOBPCG-W finds all eigenvalues, and at very competitive times.

Table 4.5 shows results with ARPACK, for three different tolerances. ARPACK, as classical Krylov method without locking, can only identify a few of the eigenvectors in the multiplicity. As with the experiments in LOBPCG-W and JDQMR, decreasing the convergence tolerance does not solve the problem. Yet, with the guidance of the simplified IVE, ARPACK

Table 4.3A: JDQMR/BCSSTK16 Initial runs

	τ		
k	ε	$\varepsilon^{1/2}$	1e-6
1	37, 18116, 38	21, 7345, 16	12, 4233, 10
2	38, 20625, 44	22, 8089, 18	14, 4957, 12
3	38, 21991, 47	23, 8684, 20	15, 5347, 13
4	39, 24272, 52	24, 9431, 22	17, 6009, 15
5	40, 26171, 57	26, 10437, 25	18, 6744, 17
6	41, 25733, 56	26, 10282, 24	18, 6370, 16
7	42, 29610, 64	27, 11450, 28	20, 7760, 21
8	41, 27342, 59	28, 11423, 27	19, 7347, 19
9	43, 30933, 68	27, 11109, 27	19, 6898, 18
10	47, 37911, 85	30, 12470, 32	21, 8082, 23
15	47, 37919, 93	30, 11769, 36	23, 7434, 24
20	48, 30928, 81	32, 11106, 36	24, 8393, 32
30	49, 31587, 95	35, 11626, 47	28, 7076, 32
40	53, 29998, 104	40, 10522, 52	37, 8164, 43
50	70, 43311, 163	48, 10829, 61	43, 7212, 41
74	71, 35835, 181	62, 11150, 84	61, 9407, 83

Table 4.3B: JDQMR/BCSSTK16 IVE runs

	τ		
mxk	ε	$\varepsilon^{1/2}$	1e-6
1	11522, 29	6886, 20	5029, 19
2	12592, 33	8034, 23	6136, 22
3	13362, 33	8656, 23	6179, 20
4	13927, 34	8196, 24	6381, 20
5	15783, 37	8921, 25	6765, 20
6	16415, 40	8684, 25	7047, 21
7	14493, 34	8546, 22	7135, 21
8	16882, 39	8449, 24	7908, 22
Best total MV/times for Initial + IVE run			
	29638, 67	14231, 38	9269, 29
k,mxk	1, 1	1, 1	1, 1

TABLE 4.3

JDQMR on matrix BCSSTK16. Similarly to Table 4.2, Table A shows results for initial runs and Table B for various IVE runs, including the best performance obtained. IVE is required for robustness. Block sizes larger than one do not increase execution time significantly.

Table 4.4A: LOBPCG-W/BCSSTK16 Initial runs

	τ		
k	ε	$\varepsilon^{1/2}$	1e-6
1	74,450005,1848	60, 76687, 323	44,34279,156
2	62, 85413, 370	45, 29439, 148	32, 14384,75
3	62, 68730, 325	43, 23491, 120	30, 10998,61
4	65, 65387, 336	42, 19570, 105	28, 9467, 54
5	66, 58429, 319	39, 16659, 97	27, 8763, 51
6	67, 59416, 354	39, 16738, 104	27, 7996, 49
7	68, 53514, 330	41, 15692, 105	27, 8057, 52
8	67, 51463, 337	39, 13773, 92	27, 7440, 50
9	67, 46950, 310	39, 13534, 98	27, 7396, 54
10	69, 49260, 336	37, 11749, 80	27, 6590, 45
15	68, 39809, 322	38, 9898, 82	28, 5999, 50
20	68, 36599, 344	39, 9277, 91	30, 5678, 54
30	68, 34169, 401	43, 8277, 95	31, 4916, 58
40	72, 31679, 454	44, 7875, 118	41, 5074, 78
50	72, 38744, 655	52, 7845, 132	48, 5139, 87
74	73, 31654, 704	69, 7908, 169	64, 5240,116

Table 4.4B: LOBPCG-W/BCSSTK16 IVE runs

	τ		
mxk	ε	$\varepsilon^{1/2}$	1e-6
1	5799, 30	6225, 32	4554, 24
2	7938, 46	5774, 31	4656, 25
3	12846, 86	5580, 31	4412, 24
4	9800, 68	6052, 37	4692, 28
5	10998, 81	6025, 36	4641, 27
6	11442, 90	6272, 41	4540, 29
7	7496, 60	6020, 41	4704, 31
8	7586, 62	6704, 48	5256, 36
Best total MV/times for Initial + IVE run			
	52749, 340	25150, 136	11002, 69
k,mxk	9, 1	4, 3	10, 3

TABLE 4.4

LOBPCG-W on matrix BCSSTK16. Similarly to Table 4.2, Table A shows results for initial runs and Table B for various IVE runs, including the best performance obtained. IVE is required for robustness, and beyond the optimal block size, time does not increase significantly.

can solve the problem with no difficulties.

For BCSSTK16 and for all methods tested, a maximum block size (*mxk*) of one was sufficient for robustness in IVE, showing the synergetic effects of the rest of its features such as locking, computing more eigenpairs, and systematically introducing random guesses. As we show next, larger *mxk* may be needed in other cases.

4.4. Results with matrix DIAG. We seek the lowest eight 8 eigenvalues which are equal to machine precision. Despite the lower multiplicity, most eigensolvers misconverge to some of the eigenvalues in the second nearby cluster.

Results from GD+k tests appear in Table 4.6. For $\tau = \varepsilon$, the method misses a few eigenvalues if the block size is less than three, which is a typical block size for GD+k. With the larger τ, GD+k finds a dimension of the required invariant subspace equal to the block size. This is a typical case, where the block size must be increased for robustness in a code whose performance deteriorates with larger block sizes. IVE avoids this trade-off, thus decoupling

Table 4.5 ARPACK/BCSSTK16						
	Initial runs			IVE run		
τ	evals	MV	Time	evals	Total MV	Total time
1e-16	29	4435	86	74	20567	259
1e-8	6	1240	20	74	18233	226
1e-7	5	1150	18	74	16974	211

TABLE 4.5

ARPACK on matrix BCSSTK16. This is not a block method, so we only vary the tolerance τ. We report (nevFound, MV, time) for the initial run of ARPACK, and the total (nevFound, MV, time) taken by both initial and the subsequent IVE runs. The robustness gains are significant.

Table 4.6A: GD+k/DIAG Initial runs		
	τ	
k	ε	$5\varepsilon^{1/2}$
1	5, 6777, 178.1	1, 567, 14.9
2	6, 13118, 305.4	2, 564, 11.6
3	8, 203927, 5127.2	3, 1267, 28.7
4	8, 24124, 521.0	4, 1548, 31.1
5	8, 48110, 1147.3	5, 2230, 50.1
6	8, 43310, 1056.9	6, 2401, 55.1
7	8, 86573, 2142.4	7, 4376, 105.0
8	8, 88696, 2264.9	8, 7008, 168.1

Table 4.6B: GD+k/DIAG IVE runs		
	τ	
mxk	ε	$5\varepsilon^{1/2}$
1	8, 3858, 89.1	4, 598, 13.2
2	8, 7184, 162.0	6, 976, 19.5
3	8, 8337, 159.6	8, 1480, 29.1
4	8, 34628, 770.1	8, 2216, 44.6
5	8, 47253, 1014.4	8, 2300, 39.9
6	8, 508381, 11340.8	8, 3159, 59.2
7	8, >600K 13384.4	8, 3827, 76.9
8	8, >600K 16084.9	8, 3532, 78.5
Best total MV/times for Initial + IVE run		
	10635, 267.2	2044, 40.7
k,mxk	1, 1	2, 3

TABLE 4.6

GD+k on matrix DIAG. Table A shows results for computing the 8 smallest eigenvalues, and Table B shows the subsequent IVE results. We follow the same format as in previous tables, reporting (nevFound, MV, time) for the initial as well as the IVE runs. A higher block size is needed in case $\tau = 5\sqrt{\varepsilon}$ to find all the desired eigenvalues. Although GD+k does not benefit from large block size in general, performance improves through an efficient initial run followed by a robust IVE run.

performance and robustness. By choosing an *mxk* such as 2 or 3, which is reasonable for low tolerances in GD+k (and not depending on the particular problem), IVE helps GD+k find all required eigenvalues and much faster than any initial run by GD+k. As expected, a slightly larger block may be needed for larger tolerances, but its exact value does not significantly affect performance. Note that for $\tau = 5\varepsilon^{1/2}$, IVE must be allowed to set a block size larger than one to recover robustness.

In Table 4.7 we show results from JDQMR on the DIAG matrix. As with GD+k, IVE can recover robustness for JDQMR if it is allowed to use a larger block size. Moreover, *mxk* does not have to match the multiplicity sought (as is necessary in the initial runs), because the synergy of the many IVE features compensates for smaller block sizes. For this solver, IVE execution times are relatively insensitive to *mxk*, matching the best performance of the initial runs and with the added robustness.

In Table 4.8, LOBPCG-W finds all the required eigenvalues for $\tau = \varepsilon$ and for all k, except for $k = 2$ for which IVE recovers the missing eigenvalue for a relatively small additional cost. Surprisingly, for $\tau = 5\varepsilon^{1/2}$ LOBPCG-W cannot identify the full invariant subspace in the initial run, regardless of block size. When IVE is allowed to increase the block size above three, all eigenvalues are found.

Finally, in Table 4.9, we show the results from ARPACK on the DIAG matrix and for three different tolerances. Through a combination of locking, restarting and finding more eigenvalues IVE is able to recover all needed eigenvalues, except when $\tau = 1e-7$ which is close to the first intercluster gap. As with the rest of the methods, a slightly larger block size would have resolved the problems.

Up to now, we have seen test cases where keeping $mxk = 1$ achieved both the required

Table 4.7A: JDQMR/DIAG Initial runs

	τ	
k	ϵ	$5\epsilon^{1/2}$
1	5, 7412, 21.5	1, 776, 4.5
2	6, 9382, 29.1	2, 765, 4.2
3	7, 12230, 38.5	3, 1453, 6.9
4	8, 12998, 38.8	4, 1650, 7.6
5	8, 14537, 47.7	5, 1401, 6.3
6	8, 16583, 52.7	6, 1648, 7.4
7	8, 17990, 61.4	7, 2822, 12.7
8	8, 11402, 36.7	8, 3908, 17.4

Table 4.7B: JDQMR/DIAG IVE runs

	τ	
mxk	ϵ	$5\epsilon^{1/2}$
1	8, 4471, 13.6	1, 406, 2.3
2	8, 7967, 25.6	1, 1462, 8.4
3	8, 11723, 33.8	6, 1855, 8.4
4	8, 18817, 57.2	8, 3229, 15.2
5	8, 25618, 77.9	6, 2597, 11.0
6	8, 33243, 99.8	8, 3950, 18.4
7	8, 35727, 108.7	8, 4431, 19.6
8	8, 39984, 118.1	8, 3592, 17.8
Best total MV/times for Initial + IVE run		
	11883, 35.1	4005, 19.7
k,mxk	1, 1	1, 4

TABLE 4.7

JDQMR on matrix DIAG. Initial and IVE runs are shown similarly to Table 4.6. Typically, JDQMR is used with block size of one, although a larger one does not affect it significantly Except for the larger block sizes in $\tau = \epsilon$, a combination of initial and IVE runs matches the performance of the most robust JDQMR, but without a priori knowledge of optimal block.

Table 4.8A: LOBPCG-W/DIAG Initial runs

	τ	
k	ϵ	$5\epsilon^{1/2}$
1	8, 166099, 1416.6	1, 836, 7.3
2	7, 192388, 1948.3	2, 1370, 13.1
3	8, 432877, 4949.2	3, 1419, 16.6
4	8, 196039, 2542.5	3, 1367, 16.7
5	8, 189875, 2548.1	3, 1360, 16.5
6	8, 187032, 2807.7	4, 2686, 40.0
7	8, 230628, 3955.3	5, 2961, 48.7
8	8, 120816, 2053.3	4, 2072, 31.8

Table 4.8B: LOBPCG-W/DIAG IVE runs

	τ	
mxk	ϵ	$5\epsilon^{1/2}$
1	8, 8714, 69.5	6, 829, 8.1
2	8, 23982, 268.3	5, 654, 7.8
3	8, 62862, 841.4	7, 3897, 59.1
4	8, 89604, 1296.8	8, 4040, 53.7
5	8, 144105, 2092.0	8, 7688, 125.3
6	8, 94350, 11915.2	8, 8089, 116.5
7	8, >600K, 13285.9	8, 8230, 119.8
8	8, >600K, 14419.4	8, 8302, 121.1
Best total MV/times for Initial + IVE run		
	201102, 2017.8	5407, 70.4
k,mxk	2, 1	4, 4

TABLE 4.8

LOBPCG-W on matrix DIAG. Initial and IVE runs are shown similarly to Table 4.6. For $\tau = \epsilon$, LOBPCG-W is robust for most block sizes, but the IVE does not add substantial overhead. For $\tau = 5\sqrt{\epsilon}$, IVE is the only way to obtain the required eigenvalues. Requiring the smaller $\tau = \epsilon$ instead would be unnecessarily more expensive.

robustness and the minimum time; cases where a larger *mxk* was required for performance; and cases where a larger *mxk* was required for robustness. But the optimum choice of *mxk* does not depend on the problem but mainly on the solver and the accuracy needed. For example, given a Krylov solver with basis size m, it makes no sense in terms of convergence to set $mxk > m/3$, with values less than $m/5$ being more reasonable. In the above IVE tests and for all solvers, the maximum multiplicity was larger than $m/5$, hence the observed decrease in performance with larger *mxk*. Other solvers with larger basis sizes could behave differently. Nevertheless, IVE restores robustness even with very modest block sizes, and therefore users can decouple the choices of parameters that are needed for convergence and cache performance from the parameters needed for robustness on a particular problem.

4.5. Results with a Laplacian matrix. The LAPLACE matrix is closer to a real world problem. We seek the 19 algebraically smallest eigenvalues, many of which come in multiple pairs. In particular the 18th and 19th eigenvalues are a multiple pair. We give results only for $\tau =$1e-9, and for $mxk = 1,2$ as IVE will not go beyond the maximum observed multiplicity. Table 4.10 shows the results of the GD method, which is identical to GD+k but without the CG-type restarting. An initial block size of 1 misses one eigenvalue, which is easily restored by IVE and in shorter time than to run an initial block size of 2. Table 4.11 for GD+k shows

Table 4.9 ARPACK/DIAG						
	Initial runs			IVE run		
τ	evals	MV	Time	evals	Total MV	Total time
1e-16	5	305990	5004	8	309889	5092
1e-8	5	27678	443	8	30885	515
1e-7	2	4167	68	2	4256	70

TABLE 4.9

ARPACK on matrix DIAG. As in Table 4.5, IVE is able to achieve the required robustness on the DIAG matrix for the two smallest tolerances, and with minimal additional overhead. The need of a larger block size is evident for such pathological cases, and for large tolerances (τ =1e-7).

Table 4.10A: GD/LAPLACE Initial runs	
k	τ = 1e-9
1	18, 10812, 116.9
2	19, 37497, 391.8

Table 4.1B: GD/LAPLACE IVE runs	
mxk	τ = 1e-9
1	3944, 49.9
2	22186, 254.9
Best total MV/times for Initial + IVE run	
	14756, 166.8
k,mxk	1, 1

TABLE 4.10

GD on matrix LAPLACE. Table A shows results for computing the 19 smallest eigenvalues, and Table B shows the subsequent IVE results. We follow the format of Table 4.2, testing only one tolerance, and not reporting the nevFound in IVE, which finds all eigenvalues. We test block sizes up to the maximum multiplicity of two. IVE restores robustness, and improves time in all cases.

that even when the method does not miss an eigenvalue, the validation step is not expensive relatively. Table 4.12 shows similar results with JDQMR, where IVE restores robustness in better or similar time as the most robust initial run. Finally Table 4.13 demonstrates the robustness of IVE with ARPACK, even with only a few IVE features enabled.

5. Conclusions. Hermitian, iterative eigenvalue solvers cannot guarantee, without factorizing the matrix, that the eigenvalues they compute are the required ones. Eigenvalues that are tightly clustered or of high algebraic multiplicity can be missed by any type of eigensolver. There are many techniques to alleviate this: ask for more than the required eigenpairs, and to better than sufficient accuracy, or use a block size larger than the maximum multiplicity sought. Choosing these parameters, however, requires a-priori knowledge about the problem, and increasing them beyond a certain value slows convergence and reduces cache performance. Alternatively, locking converged eigenvectors, and restarting the solver with new random vectors improves robustness, also at the cost of convergence. Typically, robust choices lead to unnecessarily slow codes especially for the easier parts of the required spectrum.

Our *a posteriori* Iterative Validation of Eigensolvers is essentially a wrapper around any given eigensolver that automates all the above choices of techniques, except for accuracy, to provide robustness when this is needed. Relying on the synergy of these techniques, IVE keeps calling the solver to find additional eigenpairs, until no missed eigenvalues can be identified. IVE works both with single-vector and block solvers, requiring no changes to the solver.

There are two key ideas in IVE. The first is the repeated Rayleigh Ritz minimization in the space orthogonal to all converged eigenvectors, each time starting with at least one random initial guess. The second is an automatic way to increase the block size based on the largest multiplicity computed thus far, hence increasing the chances of finding missed eigenvalues of similar multiplicity.

Both key ideas can be very expensive during the initial run of any eigensolver, but as an a posteriori technique they focus only on the problematic part of the spectrum, and only

Table 4.11A: GD+k/LAPLACE Initial runs	
k	$\tau = 1\text{e-}9$
1	19, 5062, 62.7
2	19, 7852, 86.4

Table 4.11B: GD+k/LAPLACE IVE runs	
mxk	$\tau = 1\text{e-}9$
1	654, 8.6
2	1180, 14.9

TABLE 4.11

GD+k on matrix LAPLACE. Similarly to Table 4.10, Table A shows results for initial runs and Table B for various IVE runs. Best performance is not included because the initial run does not miss eigenvalues. Yet, the additional IVE expense is not significant.

Table 4.12A: JDQMR/LAPLACE Initial runs	
k	$\tau = 1\text{e-}9$
1	18, 5360, 18.7
2	19, 7144, 25.0

Table 4.12B: JDQMR/LAPLACE IVE runs	
mxk	$\tau = 1\text{e-}9$
1	1213, 4.7
2	2779, 11.2
Best total MV/times for Initial + IVE run	
	6573, 23.4
k,mxk	1, 1

TABLE 4.12

JDQMR on matrix LAPLACE. Similarly to Table 4.10, Table A shows results for initial runs and Table B for various IVE runs, including the best performance obtained. Robustness is restored for a smaller or comparable cost.

if needed. Moreover, the ability to resolve multiple or clustered eigenvalues during a post processing phase, frees the user to employ in the solver the block size that gives the best cache performance, and the tolerance required by the problem. Even when the block size is limited to modest values in IVE, the synergetic effects of the rest of the techniques can deliver the desired robustness. Our experiments support both the robustness of IVE on some very hard problems, and its ability to decouple robustness from performance choices.

As future work, we plan to investigate the applicability and effectiveness of IVE for interior eigenvalues, and for non-hermitian eigenproblems. The major obstacle is that the Rayleigh-Ritz procedure does not provide monotonic convergence toward interior eigenvalues. Thus, the existence of a Ritz value that is closer to a value σ than any previously computed eigenvalue is does not imply that an eigenvalue was missed. A harmonic-Ritz procedure, which may provide a solution to this problem, unfortunately does not extend to non-hermitian problems.

Acknowledgements. We thank the referees for helping us shape a better paper.

REFERENCES

[1] J. Baglama, D. Calvetti, and L. Reichel. IRBL: An implicitly restarted block lanczos method for large-scale hermitian eigenproblems. *SIAM J. Sci. Comput.*, 24(5):1650–1677, 2003.

[2] J. Baglama, D. Calvetti, and L. Reichel. IRBLEIGS: A MATLAB program for computing a few eigenpairs of a large sparse hermitian matrix. *ACM Transaction on Mathematical Software*, 29(5):337–348, 2003.

[3] Z. Bai, D. Day, and Q. Ye. ABLE: An adaptive block lanczos method for non-Hermitian eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 20(4):1060–1082, 1999.

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

[5] M. Clint and A. Jennings. The evaluation of eigenvalues and eigenvectors of a real symmetric matrix by simultaneous iteration. 13:68–80, 1970.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, second edition, 2003.

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

[8] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H.A. van der Vorst. *Numerical Linear Algebra for High Performance Computers*. SIAM, Philadelphia, PA, 1998.

[9] Iain Duff, Roger G. Grimes, and John G. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection (Release I). Technical Report TR/PA/92/86, CERFACS, October 1992.

Table 4.1 ARPACK/LAPLACE						
	Initial runs			IVE run		
τ	evals	MV	Time	evals	Total MV	Total time
1e-9	16	2874	72	19	6042	137

TABLE 4.13

ARPACK on matrix DIAG. As in Table 4.5, IVE is able to achieve the required robustness on the LAPLACE matrix. Note that ARPACK missed 3 eigenvalues despite a relatively low tolerance.

[10] Roman Geus. *The Jacobi-Davidson algorithm for solving large sparse symmetric eigenvalue problems with application to the design of accelerator cavities*. PhD thesis, ETH, 2002. Thesis. No. 14734.

[11] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, MD 21211, 1989.

[12] G. H. Golub and R. Underwood. The block Lanczos method for computing eigenvalues. In J. R. Rice, editor, *Mathematical Software III*, pages 361–377, New York, 1977. Academic Press.

[13] R. G. Grimes, J. G. Lewis, and H. D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM J. Matrix Anal. Appl.*, 15(1):228–272, 1994.

[14] Christopher Hsu and James Demmel. Effects of block size on the block Lanczos algorithm. Technical report, Department of Mathematics, U.C. Berkeley, June, 2003.

[15] A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient method. *SIAM J. Sci. Comput.*, 23(2):517–541, 2001.

[16] Arno B. J. Kuijlaars. Convergence analysis of krylov subspace iterations with methods from potential theory. *SIAM Review*, 48:3–40, 2006.

[17] C. Lanczos. An iterative method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Nur. Stand.*, 45:255–282, 1950.

[18] R. B. Lehoucq and D. C. Sorensen. Deflation techniques for an implicitly restarted arnoldi iteration. *SIAM J. Matrix Anal. Appl.*, 17(4):789–821, 1996.

[19] 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, PA, 1998.

[20] J. R. McCombs and A. Stathopoulos. Technical report.

[21] R. B. Morgan. A harmonic restarted Arnoldi algorithm for calculating eigenvalues and determining multiplicity. *LAA, to appear*.

[22] R. B. Morgan. On restarting the Arnoldi method for large nonsymmetric eigenvalue problems. *Math. Comput.*, 65:1213–1230, 1996.

[23] R. B. Morgan and D. S. Scott. Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices. *SIAM J. Sci. Comput.*, 7:817–825, 1986.

[24] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, Philadelphia, PA, 1998.

[25] Heinz Ruthishauser. Simultaneous iteration method for symmetric matrices. *Numer. Math.*, 16:205–223, 1970.

[26] Yousef Saad. On the rate of convergence of the Lanczos and the block-Lanczos methods. *SIAM J. Numer. Anal.*, 17:687–706, 1980.

[27] Yousef Saad. *Numerical methods for large eigenvalue problems*. Manchester University Press, 1992.

[28] G. L. G. Sleijpen, A. G. L. Booten, D. R. Fokkema, and H. A. van der Vorst. Jacobi-davidson type methods for generalized eigenproblems and polynomial eigenproblems. *BIT*, 36(3):595–633, 1996.

[29] 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(2):401–425, 1996.

[30] D. C. Sorensen. Implicit application of polynomial filters in a K-step Arnoldi method. *SIAM J. Matrix Anal. Appl.*, 13(1):357–385, 1992.

[31] D. C. Sorensen. Deflation for implicitly restarted Arnoldi methods. Technical Report Tech. Rep. TR98-12, Department of Computational and Applied Mathematics, Rice University, 1998.

[32] A. Stathopoulos. Nearly optimal preconditioned methods for hermitian eigenproblems under limited memory. Part i: Seeking one eigenvalue. Technical Report Tech. report WM-CS-2005-03.

[33] A. Stathopoulos and C. F. Fischer. A Davidson program for finding a few selected extreme eigenpairs of a large, sparse, real, symmetric matrix. *Computer Physics Communications*, 79(2):268–290, 1994.

[34] A. Stathopoulos and Y. Saad. Restarting techniques for (Jacobi-)Davidson symmetric eigenvalue methods. *Electr. Trans. Numer. Alg.*, 7:163–181, 1998.

[35] A. Stathopoulos, Y. Saad, and K. Wu. Dynamic thick restarting of the Davidson, and the implicitly restarted Arnoldi methods. *SIAM J. Sci. Comput.*, 19(1):227–245, 1998.

[36] W. J. Stewart and A. Jennings. ALGORITHM 570: LOPSI a simultaneous iteration method for real matrices. *ACM Transactions on Mathematical Software*, 7(2):230–232, 1981.

[37] Tianruo Yang. Theoretical error bounds on the convergence of the Lanczos and block-Lanczos methods. *Computers and Mathematics with Applications*, 38:19–38, 1999.