Locking issues for finding a large number of eigenvectors of Hermitian matrices ⋆

Andreas Stathopoulos

Department of Computer Science, College of William and Mary, Williamsburg, Virginia 23187-8795

Abstract

Locking is a popular deflation technique followed by many eigensolvers, where a converged eigenvector is frozen and removed from the iteration search space. Other deflation techniques that do not modify the matrix have less favorable numerical properties, so the alternative to locking is not to perform deflation at all. Without deflation, which we refer to as non-locking, converged eigenvectors are kept in the search space. One of the goals of this paper is to determine when locking is computationally preferable, and for which eigensolvers. Our primary goal, however, is to address a subtle numerical, but not floating point, problem that arises with locking. The problem stems from the fact that converged eigenpairs are only accurate to a specified tolerance Tol, so if they are locked, they may impede convergence to Tol accuracy for some subsequent eigenvector. Although the problem is rare, the resulting stagnation is a liability for general purpose software. We provide a theoretical explanation of the problem, and based on it we derive an algorithm that resolves the problem, is easy to implement, and incurs minimal additional costs.

Key words: locking, soft locking, eigenvalues, large number of eigenvalues, convergence accuracy

1 Introduction

Hermitian eigenvalue problems remain some of the most computationally intensive application kernels. Because their eigenvalues are well conditioned,

⋆ Work supported by National Science Foundation grants: (ITR/DMR 0325218) and (ITR/AP-0112727)

Email address: `andreas@cs.wm.edu` (Andreas Stathopoulos).

and their theoretical properties well understood, we can solve problems of very large size. This ability has spurred even further increases in the accuracy of engineering models, which in turn requires the solution of matrix problems of unprecedented size. Even more challenging is the need to compute hundreds and often thousands of eigenvalues and their corresponding eigenvectors.

Eigenvalue iterative methods for such problem sizes are based predominantly on projection methods. These methods build a sequence of subspaces, called search spaces, from which they extract the approximations to the eigenvalues and eigenvectors through the RR or some other projection technique [22,20,10,34]. For the RR, the approximations are referred to as Ritz values and Ritz vectors. Krylov subspaces are a common and effective choice of search space. In particular, when seeking a large number of eigenvalues that are not tightly clustered, unrestarted Lanczos [13,5] is difficult to beat, while for a few eigenvalues, implicitly restarted Lanczos (IRA) and Shift and Invert Lanczos [27,16,9] may be preferable. In the presence of multiplicities or tightly clustered eigenvalues, subspace iteration and block methods (e.g., LOPSI, block Lanczos) [33,8], or methods that use preconditioning to build non-Krylov spaces (e.g., Davidson, Generalized Davidson, Jacobi Davidson, LOBCPG) are usually more effective [6,21,26,12].

There are two main ways that eigensolvers deal with converged eigenpairs. The first flags an eigenvector as converged and leaves it in the search space with no other special treatment. Some flag checking may be needed for eigensolvers that work specifically on unconverged eigenvectors (e.g., Davidson type solvers). The second way deflates a converged eigenvector x from the search space so that all work is performed in the orthogonal complement of x. The latter approach is often called *locking*, as the converged x is frozen and removed (locked) from the search space [3,4,28,15]. In exact arithmetic, locking is equivalent to working with the deflated matrix $(I-xx^*)A(I-xx^*)$, but locking does not modify the matrix and it is also a numerically stable way to implement deflation. For this reason, other forms of deflation (such as Wielandt deflation or polynomial filtering) have been discouraged for iterative methods [34,25]. The former approach, which involves no deflation, is sometimes referred to as "soft locking" [11], but because there is no locking involved, we simply call it non-locking.

There are trade offs between using or avoiding locking, but the differences do not show until a relatively large number of eigenvalues (*nev*) is required. Convergence is generally better without locking, because the subspace size must be larger than *nev*, but the computational costs of Rayleigh Ritz, restarting, and the computation of residuals increase rapidly with the subspace size. This implies that the use of locking becomes necessary for eigensolvers that make frequent use of these kernels (e.g., Davidson, LOBPCG). At the same time, locking may reduce the storage requirements of a method, because it works with a small basis. A computational cost benefit analysis has not been carried out, and this is one of the goals in this paper.

Our primary goal is to address a more important, yet quite subtle numerical problem that is caused by locking. Although the problem has not been discussed in the eigenvalue literature, it is sometimes observed by practitioners. It stems from the fact that converged eigenpairs are only accurate to a specified tolerance Tol, so their orthogonal complement is not an accurate invariant subspace. Thus, it is possible that convergence to a subsequent eigenvector to Tol accuracy is impeded by locked vectors, causing any iterative solver to stagnate. A similar problem is sometimes observed in block iterative methods for linear systems of equations with multiple right hand sides, as several systems converge and their solutions are deflated [14]. Although the problem is quite rare, it is a big liability for any eigenvalue software that is general purpose, or used within a production code. Yet, we are not aware of any attempts to theoretically understand this problem and address it appropriately at the software level.

In this paper, we provide theory that shows that the problem is not fundamental, but related to the implementation of locking and of iterative solvers. Specifically, we provide a computable bound on how far from Tol the residual norm of an approximate eigenvector can be, and we show that the missing eigenvector components are almost entirely in the deflated space. Thus, a Rayleigh Ritz procedure with a basis that includes both the locked vectors and the yet unconverged eigenvector resolves the problem. We also outline the modifications to iterative solvers that address this problem with minimal additional computational costs.

2 Notation

We consider the problem of finding nev eigenvalues and the corresponding eigenvectors of a Hermitian matrix A of size $N \times N$: $Ax_i = \lambda_i x_i,\ i = 1, \ldots, nev$. Usually extreme eigenpairs are sought but our discussion applies to eigenvalues obtained in any order. A Ritz value and its Ritz vector obtained through the Rayleigh Ritz procedure on some basis V is denoted as (θ, u). We also use the acronym RR for Rayleigh Ritz. Convergence requires that the 2 norm of the residual, $r = Au - \theta u$, is less than Tol. With locking, Q and Θ denote the matrix of all converged Ritz vectors and the diagonal matrix of their Ritz values, respectively, and R the matrix of their corresponding residuals. We will also be using the orthogonal projector $P = I - QQ^*$. By $\|\cdot\|$ we denote the 2 norm, and $\|\cdot\|_F$ the Frobenious norm.

3 Iterative solvers and locking

To better describe the computational and numerical issues of locking, we present a generic form of an eigenvalue iterative solver. We have already seen the basic ingredients: build a subspace, or more accurately a basis for a subspace, and then extract the approximations from the subspace through the RR procedure. Krylov subspaces are the most common, but often a preconditioner can be used to create a subspace enriched in the required directions. For a broad survey of eigensolvers see [3] and for some more recent advances see [29,1]. In this paper we focus only on the way these eigensolvers handle the converged eigenpairs.

Typically, eigensolvers test convergence by checking the norm of the residual of the Ritz vectors. Depending on the algorithm, this test may occur at every step (e.g., (Jacobi-)Davidson, LOBPCG), or every certain number of steps (e.g., ARPACK). When the residual and the test are computed infrequently, non-locking is a natural choice. When an eigenvector converges, a counter k is increased and the method continues until $k > nev$. As the RR has to produce approximations to all nev eigenvectors, the maximum basis size has to be $maxSize > nev$. On the other hand, for methods that require the most recent Ritz vectors and their residuals at every step (e.g, (Jacobi-)Davidson), there is little sense in continuing iterations with the residual of a converged eigenvector. Similar concerns apply also to block methods, if some vectors in the block have converged. These methods can still be implemented without locking, but more book-keeping is required so that converged eigenvectors are not targeted. Traditionally, such methods have been used with locking.

Table 1 depicts two implementations of a generic eigensolver; one without locking (left) and one with locking (right). These eigensolvers build a basis of maximum size $maxSize$, at which step they restart with the most recent approximations to the the sought eigenvectors. The RR and convergence tests are applied every m steps. Without locking, the converged eigenvectors are simply kept in V, and participate in subsequent RR procedures. Thus, they are expected to improve as more information is accumulated in V.

For the purposes of this paper, most known eigensolvers can be represented in these generic forms. For example, ARPACK corresponds to the non-locking case with $m = maxSize$, and v_{new} coming from a Krylov space, Generalized Davidson corresponds to non-locking with $m = 1$, and v_{new} being the preconditioned residual, while Jacobi-Davidson is usually implemented as a locking method, with $m = 1$, and v_{new} the vector returned by some inner solver. Note also that v_{new} and u_{k+1} could be multivectors so that the solvers capture block methods. For simplicity, however, we consider only the single vector case.

Table 1
Generic iterative eigensolvers based on projection methods, without locking (left) or with locking (right). Matlab-like notation is followed. By $V = (I - uu^*)V$ we mean that u is removed from V, and V is reassigned to a basis of size: $size(V) - 1$.

Generic solver without locking	**Generic solver with locking**
$k = 0$, $V = [\ ]$	$k = 0$, $V = [\ ]$, $Q = [\ ]$
while $k \leq nev$	while $k \leq nev$
for $i = 1 : m$	for $i = 1 : m$
$V = [V, v_{new}]$	$V = [V, (I - QQ^*)v_{new}]$
end	end
$(\theta, u)_{k+1}$ = Rayleigh Ritz (V)	$(\theta, u)_{k+1}$ = Rayleigh Ritz (V),
$r_{k+1} = Au_{k+1} - \theta_{k+1}u_{k+1}$	$r_{k+1} = Au_{k+1} - \theta_{k+1}u_{k+1}$
if ($\Vert r_{k+1} \Vert < Tol$)	if ($\Vert r_{k+1} \Vert < Tol$)
$k = k + 1$	$Q = [Q, u_{k+1}]$; $V = (I - uu^*)V$
	$k = k + 1$
if $(size(V) = maxSize)$	if $(size(V) = maxSize)$
restart $V = [u_1, \ldots, u_{k+1}]$	restart $V = [u_1]$
end	end

4 Computational trade offs for locking

The effect of the basis size on the convergence, in terms of iterations, of a method is hard to quantify. Qualitatively, larger bases imply faster convergence, especially for difficult problems without good preconditioners. But the convergence benefits wane beyond a certain basis size which depends on the problem and on nev. Non-locking algorithms require that $maxSize > nev$, therefore for large nev they usually converge in fewer iterations than locking algorithms that use $maxSize \ll nev$. If the matrix-vector and/or preconditioning operator are very expensive (much more expensive than $O(nevN)$ as we see later), non-locking algorithms should be preferable. For sparse matrices, however, the smaller computational costs per iteration of locking methods may outweigh the faster convergence of non-locking methods.

Ideally, one would like to identify cross-over points for nev and $maxSize$ where non-locking becomes faster than locking. Because of the difficulty to quantify the convergence effects, we present only the difference in the per-step computational costs of the two methods for large nev. We do not consider the cost of the matrix-vector and preconditioning operators as these are common to both locking and non-locking. In the following complexity analysis, we include coefficients for the largest order terms, and consider single vector operations of cost $O(N)$ (i.e., BLAS level 1 kernels) the building blocks for all computations.

Non-locking costs. Typically $maxSize$ is a small multiple of nev or a small number of vectors larger than nev. We use the former case because it is much more common (e.g., ARPACK suggests $maxSize \geq 2nev$), but we retain the variable $maxSize$ to show some effects of the latter case.

The RR procedure is applied every m steps and it involves the solution of an eigenvalue problem of size $j \leq maxSize$. Thus, the computational cost per step (matvec operation) is $O(maxSize^3/m)$. RR also involves the recombination of the basis V to compute the Ritz vector u_{k+1} and its residual. The total cost for these operations between two successive restarts is $O(\sum_j 2 * j * N, \quad j = nev : m : maxSize)$. Performing the summation and averaging over $(maxSize - nev)$ steps, we obtain the average cost per step for RR: $O(maxSize^3/m + (nev + maxSize) * N/m)$.

Non-locking algorithms restart with at least the number of converged eigenpairs, and therefore the restarting cost is $O(nev * maxSize * N)$. Averaged over the $(maxSize - nev)$ steps, we have: $O(nev * maxSize * N/(maxSize - nev))$ per step. This demonstrates the inefficiency of the case $maxSize - nev = O(1)$ as restarting costs become $O(nev * maxSize * N)$ or $O(nev^2 * N)$ per step! When $maxSize = 2 * nev$, the average restarting cost is $O(nev * N)$ per step.

Each v_{new} vector is orthogonalized against the current basis V, which includes $j = nev, \ldots, maxSize$ vectors, for a cost of $O(2 * j * N)$. Summing and averaging over the $(maxSize - nev)$ steps between two successive restarts, the cost per step for orthogonalization is $O((maxSize + nev) * N)$.

Therefore, the total average cost per step for non-locking is:

$$\begin{aligned} O(\; & maxSize^3/m + (nev + maxSize) * N/m + \\ & nev * maxSize * N/(maxSize - nev) + (maxSize + nev) * N). \end{aligned}$$

When $m = 1$, as in the case of Generalized Davidson, and using the common $maxSize = 2 * nev$, the average non-locking cost per step becomes:

$$NonLock_{m=1} = O(8 * nev^3 + 8 * nev * N). \tag{1}$$

When $m = maxSize$, as in the case of ARPACK, the costs per step become:

$$NonLock_{m=maxSize} = O(4 * nev^2 + 6.5 * nev * N). \tag{2}$$

Because Davidson type methods are more expensive per step than ARPACK, they rely on the effectiveness of the preconditioner to converge in fewer iterations. These per-step-costs do not include the matrix-vector and, if available, the preconditioning operation. Although these constitute the dominant costs for small nev, for large nev ($O(100–1000)$) it takes an extremely expensive operator to match the above iteration costs.

Locking costs. With locking, $maxSize$ is a small constant (20 is a common recommendation), and for the purpose of this asymptotic analysis, m is also constant. Therefore, the costs of RR, of computing the Ritz vector and the residual, of restarting, as well as the orthogonalization costs against V do not depend on nev, and thus can all be considered $O(N)$ per step.

The orthogonalization at every step against all k currently locked eigenvectors, involves a cost of $O(2 * k * N)$, $k = 1, \ldots, nev$. Assuming that each eigenvalue converges at about the same number of steps, a simple averaging argument results in the average cost per step: $O(nev * N)$ Then, the total average cost per step for locking techniques becomes:

$$Locking = O(N + nev * N) \tag{3}$$

For large nev, locking methods have a significant advantage over non-locking ones in terms of per-iteration costs (at least a factor of 6 in our models). Moreover, locking methods take about the same number of iterations to converge for each eigenvalue, as they treat each one as an independent problem. After a few eigenpairs have converged, we can estimate accurately the expected total execution time. On the other hand, non-locking methods may take far fewer iterations. If we have an estimate of their relative convergence rates, the above models can determine which method should be preferable, and for how large nev. For small nev (e.g., 10 or 20) the two techniques have similar complexities, and the winner is determined primarily by their the relative convergence characteristics. For Davidson-like, LOBPCG, or subspace iteration methods, where RR, residual, and (re-)orthogonalization are required at every step, non-locking involves unreasonably high computational costs for large nev. This justifies the use of locking in most implementations of these methods.

We conclude this section with a storage consideration. Many methods require more than one vector to be stored for each basis vector. For example, Davidson requires $2 * maxSize$, while LOBPCG requires $2 * maxSize$ and $maxSize = 3 * nev$. Locking versions of these methods still require the additional space but for a very small constant $maxSize$. For many applications, this storage reduction is a much more important consideration than computational costs.

5 A numerical problem with locking

The premise of locking, or deflation in general, is that the locked vector x is an accurate eigenvector, so that the rest of the eigenvectors can be found in the orthogonal complement of x. In practice, however, eigensolvers converge

Table 2
Harwell-Boeing matrices that caused the convergence problem for Generalized Davidson. There are three experiments: GD without preconditioning, GD with cholinc($A, 1e-3$), and GD with the almost accurate inverse cholinc($A, 1e-13$). Entries without a dash denote the index of the first eigenpair that the code stagnated while trying to obtain it. For example, for matrix BCSSTK06, GD without preconditioning had locked 312 eigenvectors, but failed to converge to the 313th.

Matrix	Size	No precond eval index	cholinc(A,1e-3) eval index	cholinc(A,1e-13) eval index
494BUS	494	-	283	-
BCSSTK06	420	313	319	-
NOS6	675	293	-	-
NOS7	729	-	177	-
GR3030	900	-	-	252

to a residual tolerance Tol, and x is only an approximate eigenvector. This raises the question of whether other eigenvectors can also be found to Tol residual accuracy. Most eigensolvers that implement locking presume a positive answer to the question. In spite of this, practitioners have long observed rare, but real world cases where some eigenvectors could not be obtained. Almost invariably, the scenario involves many locked eigenvectors and low accuracy (high Tol value). Unfortunately, occurrences of this problem have been mainly anecdotal, and not well documented.

To show that this problem can indeed occur, we ran our Generalized Davidson (GD) algorithm with locking, which is implemented in Matlab [29], on all the matrices of the symmetric Harwell-Boeing collection [7]. Obviously we only consider matrices where GD could converge. We used a basis size of $maxSize = 20$, a restart size of 10, and we tried to compute 350 smallest eigenpairs. We were not able to observe the above problem for small residual tolerances Tol, until $Tol \approx \|A\|_F \sqrt{\epsilon_{machine}}$. Table 2 shows five matrices for which the problem occurred when $Tol = \|A\|_F 1e-5$. The table records the eigenvalue index at which the code stagnated, unable to converge to the required accuracy. Evidently, the problem is rare and requires extenuating circumstances (in all these cases, a substantial part of the spectrum was computed). Nevertheless, it occurred both with and without preconditioning.

In all the above cases, the problematic Ritz vector reached an accuracy very close to Tol, but it could never reach below Tol. A reasonable explanation is that the inaccuracies of the k converged Ritz vectors (at the Tol level) somehow conspire to impede convergence for the $k+1$ eigenvector.

It should not be surprising that non-locking methods do not suffer from this problem. Converged eigenvectors, that are not locked but participate in the RR, keep improving beyond the required accuracy Tol, at a rate which depends on both the solver and the problem. Thus, those eigenvectors that are obtained earlier by a solver, tend to have better accuracy at the end. Even if the required accuracy is poorly chosen, stagnation or extra iterations will improve the converged eigenpairs up to the accuracy necessary to allow the computation of the $k+1$ eigenvector.

Although it is rare, the above problem undermines the reliability of any numerical software that implements locking. Stagnation can prove extremely costly in many applications, while it is not an option in critical applications. The first question is how to identify the problem. Observed stagnation is not a good criterion, because it may be due to various other reasons, including asking for more accuracy than can be numerically attained, clustering of eigenvalues, or simply bad initial guesses. However, although many of these problems can be identified, and sometimes treated appropriately, we are not aware of any techniques that identify the locking problem.

Moreover, once identified there seems to be very little we can do to cure it. The required accuracy cannot be achieved regardless of how long we iterate, it is too late to improve on the locked vectors, and we may not be able to single out which locked vectors cause the problem.

5.1 Some ad hoc approaches

If locked vectors with residual norms converged to Tol accuracy can impede convergence of future residuals to the same accuracy, a first response would be to use a better accuracy than Tol. For example, the choice $Tol/\sqrt{nev}$ is suggested by Lemma 1 in the following section. However, it is easy to see that we have not solved the problem, just restated it at a lower threshold. The same reasons that caused the problem at the Tol level could also cause the stagnation at the $Tol/\sqrt{nev}$ level. In practice, we have observed that more accurate Tol (better than square root of machine precision) tend to avoid the problem. However, the lack of guarantees does not justify the significant additional expense. More practically, we cannot predict when this problem occurs so that we can choose a new Tol a priori.

One approach that has often been used is to require higher tolerance to outer eigenvalues and decrease the accuracy for the inner eigenvalues [31] (which are usually obtained last). In a way, this tries to mimic the resulting accuracies of non-locking, with the innermost eigenvalue residuals having the required Tol. However, there is no explanation why this should work better, and in most of the problems of Table 2 this approach failed to resolve the problem.

Moreover, it is not clear how to progressively reduce the accuracy, and whether the additional iterations are justified.

Computational scientists faced with this problem have noticed that an application of the RR procedure over the space that includes both the locked vectors Q and the search space V improves the stagnating eigenvector and allows it to achieve the required accuracy. The consensus is that, although expensive, this is the only way to deal with the problem [17,24]. As we show next, this problem can be understood theoretically and corrected appropriately.

6 Theoretical approach

We assume exact arithmetic, so that all convergence problems that might occur are due to the inaccurate convergence of locked eigenvectors. An investigation of the influence of floating point errors in this situation is necessary but beyond the scope of our current work.

Lemma 1 *Let $P = I - QQ^*$, the orthogonal projector against the computed approximate k eigenvectors Q. Let also $R = AQ - Q\Theta$ be the matrix of residuals for each converged Ritz pair (θ_i, Q_i). Let $r = Au - \theta u$ denote the residual of the current Ritz pair (θ, u), and define the deflated residual $r_d = Pr$. Then,*

$$r = r_d + QR^*u, \quad \|r\| \leq \|r_d\| + \|R\|, \quad \|r\|^2 = \|r_d\|^2 + \|R^*u\|^2. \tag{4}$$

PROOF. Because A is Hermitian, $Q^*A = \Theta Q^* + R^*$. Using $u \perp Q$, $\|u\| = \|Q\| = 1$, we have: $r_d = P(Au - \theta u) = (Au - \theta u) - Q(Q^*Au) = r - QR^*u$. Then, $\|r\| = \|r_d + QR^*u\| \leq \|r_d\| + \|R\|$. Finally, note that $r_d \perp QR^*u$. $\square$

Any eigenvalue algorithm that uses locking works orthogonally to the locked eigenvectors (i.e., on $Q^\perp$). Equivalently we can consider the algorithm working on the matrix PAP, but ignoring the zero eigenvalues. For Krylov methods in exact arithmetic an initial guess in $Q^\perp$ is sufficient to ignore zero eigenvalues, because powers of PAP remove the null space components from iteration vectors. Therefore, convergence of the eigenmethod to an eigenpair of PAP implies that $\|r_d\| \to 0$, and from Lemma 1 the upper bound for the residual is $\|r\| \leq \|R\| \leq \sqrt{k}\, Tol$. For most matrices and small number of locked eigenvectors (k), the residual r regularly achieves a norm less than Tol. However, the bound does not preclude cases where the residual norm can be as large as $\sqrt{k}\, Tol$. In such cases, no matter how rare, a method that expects $\|r\|$ to converge to less than Tol will stagnate.

The question, therefore, is how to detect those instances, and how to resolve the problem. Detection follows when $\|r_d\| \ll \|r\|$, but it is unclear how smaller $\|r_d\|$ should be. Near convergence, and if the eigenvalue is simple, u does not vary much between steps, so we expect $\|r\| \to \|R^* u\|$. If $\|R^* u\| < Tol$, the residual r can achieve the required tolerance so we should let the algorithm continue. If $\|R^* u\| \geq Tol$, there is no hope to get $\|r\| < Tol$, even if we let the algorithm converge to $r_d = 0$. Thus, a first approach to detect this problem checks whether $\|r_d\|$ is sufficiently converged, but $\beta = \|r - r_d\| \geq Tol$. Obviously the test need not be performed when $\|r\| \geq \sqrt{k}\, Tol$:

if $\|r\| < Tol$, Lock u as converged, **break**
if $\|r\| < \sqrt{k}\, Tol$
    Compute $\|r_d\| = \|Pr\|$ and $\beta = \|r - r_d\| = \sqrt{\|r\|^2 - \|r_d\|^2}$
    if ($\beta > Tol$ and $\|r_d\| < Tol$) then declare Locking Problem.

We now address how to deal with the locking problem when it arises. The source of this problem is that the required k-th eigenvector x has components in Q that the algorithm working on matrix PAP cannot identify. We can follow the traditional suggestion and perform a RR with a larger basis that includes the converged vectors Q and the basis V. During this step, the algorithm is equivalent to its non-locking version, and therefore the new Ritz vector, w, will contain the x components that were hidden in Q. The algorithm then resumes in locking mode, targeting w, if additional convergence is needed, or the next Ritz pair. Although this approach recovers x, it does not guarantee that the Locking Problem will not resurface for future Ritz vectors $(k+1, \ldots, nev)$. It is possible therefore, that this expensive RR may have to be repeated frequently, giving the locking algorithm the computational cost of its non-locking counterpart, but not its convergence benefits.

A different strategy is to declare the problematic Ritz pair (θ, u) "practically converged", lock it, and continue with the $k+1$ eigenpair. At first, it seems that such a strategy would return solutions that do not satisfy the accuracy requirements posed by the user. However, the x components missing from u are mainly in Q, and therefore if we performed one RR procedure over all the nev locked vectors at the end of the algorithm, we should obtain all eigenvectors at the required accuracy. The above intuitive argument can be supported theoretically, but with a subtle difference stemming from the fact the method in PAP converges close to, but not exactly Px.

Theorem 2 *Let (λ, x), $\|x\| = 1$, be the exact eigenpair approximated by (θ, u). Let $u_\infty = \lim_{r_d \to 0} u$, and $\theta_\infty = \lim_{r_d \to 0} \theta$ be the eigenvector and eigenvalue of PAP, respectively, to which u and θ converge in the locking algorithm. Denote by $\gamma_d = \min_i |\lambda - \Theta_{ii}|$, the gap of the required eigenvalue from all locked eigenvalues, and $\gamma_p = \min_{\mu_i \neq \theta_\infty} |\theta_\infty - \mu_i|$ the gap of eigenvalue θ_∞ from*

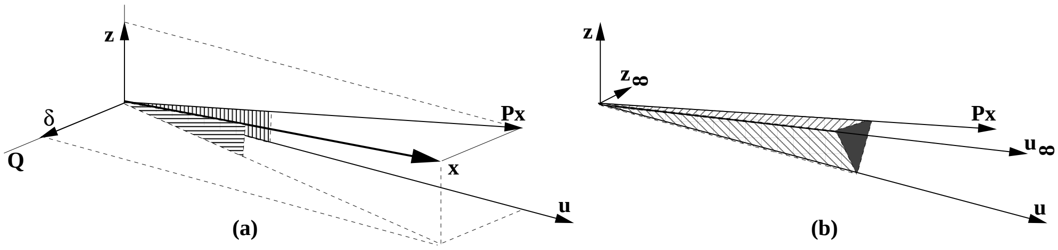

Fig. 1. The orthogonal decomposition of the eigenvector x onto the locked eigenvectors Q, the Ritz vector u, and the yet undiscovered components z. In the limit, the locking method still is missing z_∞, which however is very small.

the rest of the eigenvalues μ_i of PAP. Then,

$$\sin(u_\infty, Px) \leq \|R\|^2/(\gamma_p\gamma_d) \leq k\ Tol^2/(\gamma_p\gamma_d).$$

PROOF. Assume that λ is a single eigenvalue, so that there is a unique orthogonal decomposition $x = \delta + \alpha u + z$, with $\delta \in span(Q)$, and $z \perp span([Q, u])$. Then $Px = \alpha u + z$, and $1 = \|\delta\|^2 + \|z\|^2 + \alpha^2$. This is depicted in Figure 1(a).

To bound $\|\delta\| = \|QQ^*x\| = \|Q^*x\|$ note that $Q^*A = \Theta Q^* + R^* \Rightarrow Q^*Ax = \lambda Q^*x = \Theta Q^*x + R^*x$. Then,

$$\|\delta\| = \|Q^*x\| = \|(\lambda I - \Theta)^{-1}R^*x\| \leq \|(\lambda I - \Theta)^{-1}\|\|R\| \leq \|R\|/\gamma_d.$$

Also, $Ax = \lambda x \Leftrightarrow A(Px + \delta) = \lambda(Px + \delta) \Rightarrow PAPPx - \lambda Px = -PA\delta = -P(Q\Theta + R)Q^*x = -PRQ^*x$. Let $\tilde{\mu} = x^*PAPx/(x^*Px)$. One of the properties of the Rayleigh quotient is that $\|Aw - w(w^*Aw)/w^*w\| \leq \|Aw - \mu w\|, \forall\mu$. Then, using the classical bound on the angle between Px and the exact eigenvector u_∞ of PAP (Theorem 11.7.1 in [23]), we have:

$$\begin{aligned}\sin(u_\infty, Px) &\leq \|PAPPx - \tilde{\mu}Px\|/\gamma_p \leq \|PAPPx - \lambda Px\|/\gamma_p \\ &= \|PRQ^*x\|/\gamma_p \leq \|R\|\|\delta\|/\gamma_p \leq \|R\|^2/(\gamma_p\gamma_d) \leq kTol^2/(\gamma_p\gamma_d).\end{aligned}$$

□

Thus, if the algorithm converges to a sufficiently small $\|r_d\|$, x is very close to the space $[Q, u]$ and could be obtained accurately through RR. The bound $\|R\| \leq \sqrt{k}\ Tol$, although sharp, is usually pessimistic. In practice, we observe $\|R\| = O(\beta) = O(\|r - r_d\|)$ which is often much less than $\sqrt{k}\ Tol$, but more importantly, it provides a computable approximation at runtime:

$$\sin(u_\infty, Px) \leq \beta^2/(\gamma_p\gamma_d). \tag{5}$$

Some discussion on the presence of gaps is in order. First, because eigenvalues have a much smaller error than eigenvectors, we can assume that they are well approximated by the Ritz values. Thus, the gap for the sought eigenvalue in the spectrum of A is its minimum distance from the locked and the yet unlocked eigenvalues: $\gamma = \min(\gamma_d, \gamma_p)$. Let w be the Ritz vector from the RR on $[Q, u]$, and r_w its residual. The algorithm must conform to the user requirement of $\|r_w\| < Tol$, which implies $\sin(w, x) \leq \|r_w\|/\gamma \leq Tol/\gamma$. Theorem 2 says that for "practically converged" vectors, the RR can attain an angle on the order of $kTol^2/(\gamma_p\gamma_d) \leq kTol^2/\gamma^2 \ll Tol/\gamma$, which satisfies the user requirement. For this angle, we can estimate, not bound, the $\|r_w\|$ as:

$$\|r_w\| = O(\|R\|^2\gamma/(\gamma_p\gamma_d)) = O(\beta^2\gamma/(\gamma_p\gamma_d)). \tag{6}$$

Theorem 2 can provide also a heuristic for when to stop converging the $\|r_d\|$. Because $\sin(x, [Q, u]) \leq \sin(u, Px)$, we focus again on $\sin(u, Px)$. As shown in Figure 1(b), the positive angles between the three vectors Px, u, u_∞ satisfy the following inequality: $\angle(u, Px) \leq \angle(u_\infty, Px) + \angle(u_\infty, u)$, with equality achieved only when Px, u, u_∞ are co-planar, and u_∞ is the middle vector. Near convergence the three angles are very small, far less than $\pi/4$, and therefore their cosines are very close to 1, and sin() is monotonic. Hence,

$$\begin{aligned}
\sin(u, Px) &\leq \sin\left(\angle(u_\infty, Px) + \angle(u_\infty, u)\right) \\
&= \sin(u_\infty, Px)\cos(u_\infty, u) + \cos(u_\infty, Px)\sin(u_\infty, u) \\
&< \sin(u_\infty, Px) + \sin(u_\infty, u) < kTol^2/(\gamma_p\gamma_d) + \|r_d\|/\gamma_p.
\end{aligned}$$

The user requirement $\sin(u, Px) \leq Tol/\gamma$ surely holds if we require instead that the bound is less than Tol/γ. Then we can solve for an appropriate stopping threshold for $\|r_d\|$:

$$\|r_d\| \leq Tol\ \gamma_p/\gamma - Tol^2 k/\gamma_d. \tag{7}$$

Intuitively, when the locking problem arises in highly clustered eigenvalues and/or when a large number of eigenvectors have been locked, it is advisable to iterate the r_d to better accuracy. Practical implementations must ensure that threshold (7) is in the interval $[\epsilon_{machine}\|A\|, Tol)$, although in practice, it is usually only slightly less than Tol. For large k, the bound for $\|R\|$ may lead to too low a threshold, and it is more efficient to use the approximation:

$$\|r_d\| \leq Tol\ \gamma_p/\gamma - \beta^2/\gamma_d. \tag{8}$$

Yet, solving to better accuracy through threshold (7) may avoid locking problems for subsequent Ritz vectors. This justifies the additional few iterations, as the locking problem is not expected to arise frequently.

Table 3
Algorithm for identifying the locking problem. It replaces the convergence test of any eigensolver with locking.

Originally $E = \sqrt{k}\ Tol$
Convergence test at each iteration
  if $\|r\| < Tol$, Lock u as converged, **break**
  if $\|r\| < E$
    Compute $\|r_d\| = \|Pr\|$, $\beta = \sqrt{\|r\|^2 - \|r_d\|^2}$, and $E = \sqrt{Tol^2 + \beta^2}$
    Compute $\gamma_p, \gamma_d, \gamma$ from current Ritz values
    if ($\beta > Tol$ **and** $\|r_d\| < Tol\ \gamma_p/\gamma - Tol^2 k/\gamma_d$)
      Lock u as "practically converged"
      restore $E = \sqrt{k+1}\ Tol$, **break**

Finally, we point out that for methods like Davidson, or subspace iteration, where the residual is updated at every step, we want to avoid performing the expensive residual projection Pr at every step after $\|r\| < \sqrt{k}\,Tol$, but only after $\|r\|$ is close to its limit $\beta_\infty = \|R^* u_\infty\|$. Note that β_∞ can be estimated quite early, as u is very close to u_∞: $\beta_\infty = \|R^*(u_\infty - u) + R^* u\| \leq O(\|R\|\|r_d\|/\gamma_p) + \beta$. Eventually, $\|r_d\| < Tol$ and because of eq. (4), we can start checking $\|r_d\|$ again after $\|r\| = (\|r_d\|^2 + \beta^2)^{1/2} < (Tol^2 + \beta^2)^{1/2}$. One could also monitor the change in the angles of u iterates, and recompute $\|r_d\|$ when needed, but these details are beyond the current scope. We are now ready to present in Table 3 our algorithm for efficiently dealing with the locking problem.

7 Numerical experiments

We present numerical experiments with our Matlab GD code, and with our PRIMME eigenvalue multi-method package, which is implemented in C [19], to demonstrate that our theory agrees with observed numerical behavior. We also explore how the maximum basis size, and different methods affect the occurrence of the locking problem. We use two of the matrices in Table 2: BCSSTK06 and NOS6, and one additional Harwell-Boeing matrix, 1138BUS, of size 1138. For all matrices we try to find as many eigenvalues as we can with residual norm tolerance $\|A\|_F 10^{-7}$. For the Matlab experiments we use Matlab 6 on an Apple G4 computer, while for PRIMME experiments we compile with gcc-4.0.0 and g77 with option -O3, on an Apple G5, dual processor computer.

In the first experiment we focus on the matrix BCSSTK06. Our Matlab GD code with locking, without preconditioning, a maximum basis size 20, and a restart size 10, stagnates trying to obtain the 217-th eigenvalue. Figure 2 shows the convergence history before and during stagnation for five values: $\|r\|$, $\|r_d\|$, $sin(u_\infty, Px)$, $sin(u_\infty, u)$, and $\|r_w\|$ which is the norm of the residual resulting from the RR on $[Q, u]$. Their behavior is exactly as described by

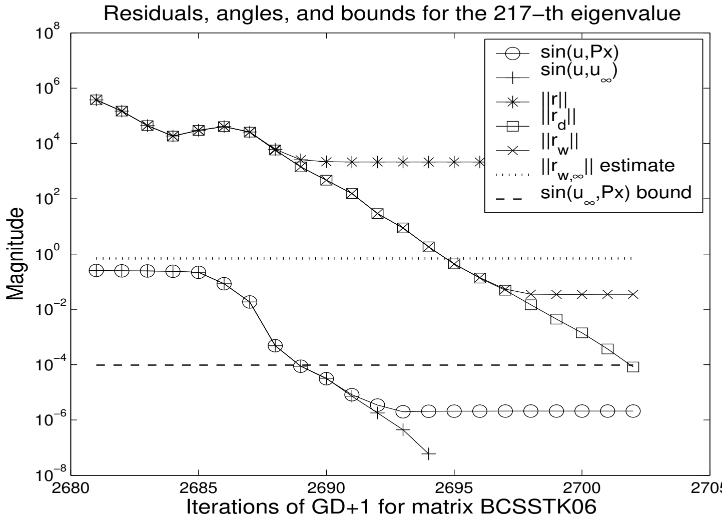

Fig. 2. Matrix BCSSTK06. The norm of the residual $\|r\|$ of the Generalized Davidson stagnates while trying to obtain the 217-th eigenvalue. As expected, $\|r_d\|$, the residual of the PAP eigenproblem, converges to zero. However, the residual of the Ritz vector from applying the RR to the $[Q, u]$ basis, does not converge to zero $\|r_w\| \rightarrow \|r_{w,\infty}\| \neq 0$. Similarly, although u converges to an eigenvector of PAP, u_∞, the eigenvector u_∞ cannot fully recover the complement of the required eigenvector x in $Q^\perp : Px$. Eqs. (5) and (6) provide the bound for $sin(u_\infty, Px)$ and the estimate for $\|r_{w,\infty}\|$ respectively.

theory. Specifically, note that u_∞, the eigenvector of PAP, does not fully match Px, but their angle is sufficiently small. Theorem 2 provides a quite accurate bound for this angle, and a means to compute a good estimate for $\|r_w\|$ at the limit.

Figure 3 shows the convergence curves for $\|r\|$ and $\|r_d\|$ of the GD method from the PRIMME eigensolver, again for BCSSTK06 and the 217-th eigenvalue. In the left figure we run PRIMME with the Algorithm 3 turned off. Again $\|r\|$ stagnates while $\|r_d\| \rightarrow 0$. We also plot $\beta = \|r - r_d\|$ which shows that the problem appears when $\beta > Tol$. It also shows that β is relatively constant near the locking problem, hence its value can be used to estimate $\|R\|$. The right graph shows the convergence of GD with the Algorithm 3 turned on. The algorithm correctly detects the problem and declares the eigenpair "practically converged". Surprisingly, also the 218-the eigenvalue was declared practically converged as the $\beta > Tol$ shows. The tight clustering of these two eigenvalues must have triggered this locking problem.

In the next set of experiments, we use four methods from PRIMME: GD, GD+1 which is an extension of GD that uses a locally optimal restarting [32,29], JDQMR-000 which is Jacobi-Davidson that uses QMR with optimal stopping criteria as an inner solver, but without any projections against Q during the inner iterations [29,30], and JDQMR-100 which is identical to JDQMR-

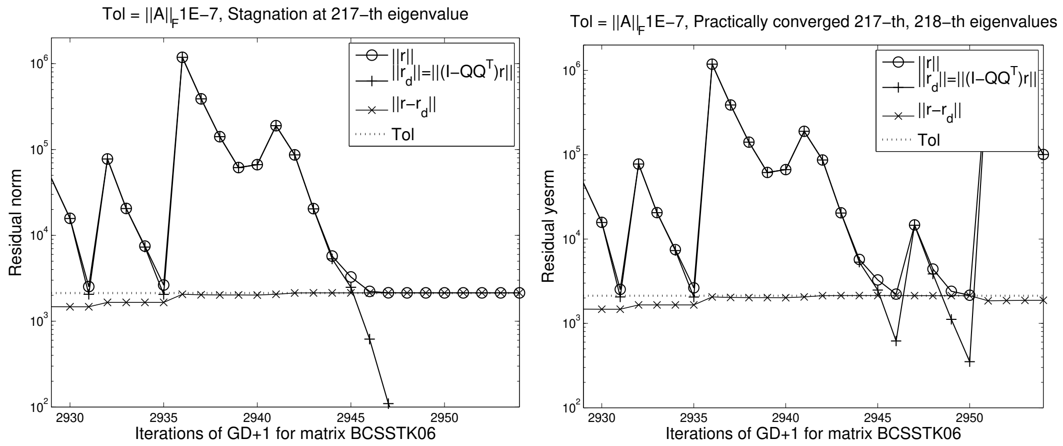

Fig. 3. Matrix BCSSTK06. GD from the PRIMME eigensolver with the detection Algorithm 3 turned off (left graph), and turned on (right graph). In addition to $\|r\|$, $\|r_d\|$ we show the role of $\beta = \|r - r_d\|$ in detecting the locking problem ($\beta > Tol$), and that it is relative constant near the locking problem (so $\beta \approx \|R\|$).

000 except that inner iteration vectors are projected against Q [30]. All methods use locking, and we vary the values for maximum basis size ($maxSize$) and the restart size ($minSize$). We experiment on NOS6, BCSSTK06, and 1138BUS. Results from GD and GD+1 methods are shown in Table 4, and from JDQMR-000 and JDQMR-100 in Table 5. As in Table 2, we report the eigenvalue index for which the method first demonstrates the locking problem.

First, we observe that regardless of method and matrix, increasing the basis size results in the locking problem appearing much later. There are two reasons for this. First, because all PRIMME methods lock converged eigenpairs only at restart, converged eigenvectors stay in the basis and improve a little further than Tol from the time they are declared converged to the time the are locked. Second, larger basis sizes offer subspace acceleration advantages [30], i.e., approximate not only the targeted eigenvector but also nearby ones. Therefore it is less likely that large missing components of nearby eigenvectors are locked away in Q. Nevertheless, although our matrices are of small size, the trend in the results seems to suggest that for any constant basis size, there is a large enough nev for which the locking problem will appear.

Our second observation from these tables is that methods that converge in the fewest iterations (not necessarily in smallest time), GD+1 and JDQMR-100 in particular, tend to experience locking problems later than the other methods. The reason may be that fast convergence during the last step may improve eigenvectors well below the Tol threshold. A third observation from Table 5 is that the JDQMR-100, which orthogonalizes against Q at every step of the QMR inner method, is the least prone to get into a locking problem. This orthogonalization is too expensive, however, especially without preconditioning where the JDQMR-000 offers similar convergence, but most importantly it

Table 4
Index of the first eigenvalue that demonstrated a locking problem with the GD and GD+1 methods from the PRIMME package. We look for N smallest eigenvalues for three matrices, under different basis sizes. A dash means that the method found all the eigenvalues without locking problems. Residual tolerance is $\|A\|_F 10^{-7}$.

	GD method			GD+1 method		
($minSize$,	Matrices			Matrices		
$maxSize$)	NOS6	BCSSTK06	1138BUS	NOS6	BCSSTK06	1138BUS
(3,6)	99	77	128	138	203	41
(5,10)	137	97	181	134	215	545
(7,15)	143	217	273	93	-	616
(9,18)	143	216	415	143	216	632
(20,40)	240	-	-	-	-	749
(80,160)	479	-	-	-	-	-

Table 5
As in Table 4, the index of the first eigenvalue that demonstrated a locking problem with the JDQMR-000 and JDQMR-100 methods from the PRIMME package.

	JDQMR-000 method			JDQMR-100 method		
($minSize$,	Matrices			Matrices		
$maxSize$)	NOS6	BCSSTK06	1138BUS	NOS6	BCSSTK06	1138BUS
(3,6)	87	96	32	226	278	319
(5,10)	94	100	457	-	100	-
(7,15)	146	101	616	256	-	-
(9,18)	143	372	649	255	-	-
(15,30)	-	-	646	235	-	-

does not eliminate the vulnerability to the problem (see NOS6 case). With the Algorithm 3 switched on, all PRIMME methods detected the locking problems in all cases, and were able to compute all N eigenvalues for each matrix.

Finally, to show that these locking problems are not artificially created by asking for a huge portion of the spectrum of very small matrices, we present a few examples of much larger matrices where our algorithm has detected and resolved successfully locking problems. In these examples, all algorithms use $minSize = 6$ and $maxSize = 18$, and attempt to find 1000 eigenvalues. The most common example is the 3-D, 7 point Laplacian, with Dirichlet boundary conditions on the unit cube, with a $25{\times}25{\times}25$ uniform finite difference grid. The GD+1 method without detection on, stagnated on the 748-th eigenvalue.

The JDQMR-000 method, with Algorithm 3 turned on, identified "practically converged" eigenpairs at indices: 591, 686, 692, 699. Locking problems were also detected with JDQMR-000 for matrices Cone_A of size 22032 and Plate33K_A0 of size 39366, both from the FEAP collection [2]. The corresponding "practically converged" indices were: 492, 532, 758, 843, 894, 951, 954, 956, for Cone_A, and 731, 774, 928, 930, 965, for Plate33K_A0.

A surprising observation came from two runs with the same parameters as before, but for residual threshold close to machine precision: $\|A\|_F 10^{-15}$. The JDQMR-000 detected "practical convergence" for eigenvalue indices: 495, 925, 926, 927, for the 3-D Laplacian, and for eigenvalue index 571 for the Cone_A matrix. These were the only two instances we have observed the locking problem with such a high accuracy. Yet, they demonstrate that dealing with this problem is essential for robustness in eigenvalue software.

8 Implementation issues

As shown in Table 3, the additions required to implement the algorithm in any method are minimal. A variable can be maintained that keeps track of whether or how many eigenpairs have "practically" converged. If the problem has occurred at least for one eigenpair, the code has to call the RR procedure after all *nev* eigenpairs have been computed.

Computationally, there are only a few more orthogonalizations with the locked vectors Q per eigenvector sought. There is one orthogonalization when $\|r\|$ goes below $\sqrt{k}Tol$ for the first time, and a small number of them from $\|r\| < E$ until r_d converges. One step of Gram-Schmidt without re-orthogonalization is always enough because r is almost orthogonal to Q.

Interestingly, the RR procedure that uses all *nev* eigenvectors returned by the solver can be implemented with no additional matrix vector products, if we update the projected matrix Q^*AQ at every step. To see this, consider the matrix Q^*AQ at some step of the solver, and assume that the Ritz pair (θ, u) has just been declared converged (actually or "practically"). Then the matrix Q^*AQ needs to be updated as follows:

$$[Q, u]^* A[Q, u] = \begin{bmatrix} Q^*AQ & Q^*Au \\ (Q^*Au)^* & u^*Au \end{bmatrix}.$$

Our algorithm in Table 3 ensures that $r_d = (I - QQ^*)r$ is computed always for the vector that just converged. Because $Q^*u = 0$, we have $r_d = (Au - \theta u) -$

QQ^*Au, and the required column vector (Q^*Au) is a byproduct of the Gram-Schmidt process. Thus, if we use the $k+1$ column of the above projection matrix as storage for (Q^*Au), the update is completely automatic. When our algorithm determines that convergence occurred, it will also append $\theta = u^*Au$ at the diagonal element of the Q^*AQ matrix. By induction, the Q^*AQ matrix of all the locked vectors is built correctly, with no additional operations.

The final RR procedure will solve an $nev \times nev$ eigenvalue problem, and it will produce a set of new, slightly better, approximate eigenvalues, and a matrix of coefficients $Y \in \Re^{nev,nev}$, that yield the new approximate eigenvectors $W = QY$. The eigendecomposition and the linear recombination of the Ritz vectors are the only expenses, which cannot be avoided.

The only disadvantage of this implementation is that it does not provide the final residuals of the W vectors. Although all residual norms are expected to be less than Tol, in practice it is possible that an eigenvalue might have been missed by the solver. In that case, the final RR could have accumulated enough information in Q to identify the missing pair, but not at the required accuracy. This problem, however, does not reflect a shortcoming of our approach but of any iterative eigenvalue algorithm. In [18], we have provided an algorithmic solution to missing eigenvalues, by repeatedly calling the solver until no further eigenvalues can be identified. Our so called Iterative Validation of Eigensolvers (IVE) approach meshes well with the algorithm proposed in this paper, because it requires locking and an a-posteriori RR application.

8.1 A hybrid locking, non-locking approach

Non-locking eigensolvers do not have the locking problem because converged Ritz vectors keep improving as they remain in the basis. This suggests a hybrid locking approach, where an eigenvector x_k is not locked immediately when it converges, but stays in the basis until the eigenvector x_{k+j} converges. This strategy maintains a pipeline of j converged Ritz vectors in the basis V, allowing them some additional time to improve beyond the Tol accuracy, before they are locked out. When $j = 0$ we have traditional locking, and when $j = nev$ we have non-locking.

In practice, the value of j must be kept small enough to avoid the large computational and storage costs of non-locking, but large enough to allow sufficient improvements to converged eigenvectors. The optimal value for j is not easy to find, and it depends not only on the algorithm and the problem, but also on the initial guess. In general, although the implementation of the RR post processing is still needed as a safe guard, the hybrid approach may reduce the frequency that the post processing is needed. The PRIMME implementation that locks eigenvectors only at restart is reminiscent of this hybrid approach.

9 Relation to other approaches

An alternative to applying the RR procedure once at the end is to apply it at the iteration that the problem arises. The benefit is that the RR is applied on fewer vectors and it is thus cheaper. According to computational folklore, practitioners have resorted to this alternative as their codes could not identify the problem, and thus they had to stop the code, apply the RR, and then restart for computing the rest of the required eigenvalues. However, this approach does not guarantee that the problem will not re-occur for some of the remaining eigenvalues. Because of the possibility of such a repeated expense, and because such problems are more likely to occur after most eigenvalues have already been found, we do not consider this alternative further.

Our theory also explains why in some cases the cascading accuracy approach has been reported to provide a viable alternative. In fact, we can provide a formula that guides the cascading, and ensures that the problem is avoided. According to Lemma 1, when $k = 1$, i.e., one eigenvector is locked with residual accuracy $\|r_1\|$, the second eigenvector can also be obtained with residual norm $\|r_1\|$. To obtain the third residual to accuracy Tol, it is sufficient that both $\|r_1\|, \|r_2\| < Tol/\sqrt{2}$. By induction we can show that to obtain $\|r_k\| < Tol$, earlier eigenvectors must be obtained with accuracy $\|r_i\| < Tol/\sqrt{2^{k-i}}$ for $1 < i \leq k$. However, such a strategy is unreasonable, as it requires an exponentially decreasing accuracy toward the extremes of the spectrum. The linear cascading strategy in [31] did not resolve the problem in most of our experiments.

10 Conclusions

Locking is a stable form of deflation, where an eigenvector x is removed from the search space of an eigensolver and all subsequent vector operations are performed orthogonally to x. In our earlier research [18], we have shown that locking provides a better mechanism than non-locking for identifying eigenvalues that are highly clustered or of very high multiplicity. In this paper, we analyzed the computational and storage costs of locking versus non-locking, and showed that for large numbers of eigenvalues and for Davidson-type methods, locking is almost always beneficial. However, convergence may vary between locking and non-locking methods.

We have also identified a subtle numerical, but not floating point, problem with locking. Specifically, a large number of locked, approximate eigenvectors, that have converged to Tol residual accuracy, may impede convergence to Tol accuracy for some subsequent eigenvector. Although the problem is rare, the resulting stagnation is a liability for general purpose software. We have provided a theoretical explanation of the problem that helped us derive an easy to implement algorithm that identifies the problem correctly, and resolves it with minimal additional costs. We also presented a hybrid locking, non-locking approach that reduces the occurrence of the problem, at no additional costs. Our methods proved effective in the all occurrences of the problem we have identified.

Acknowledgments

The author would like to thank Julien Langou, and the anonymous referee for very constructive comments.

References

[1] P.-A. Absil, C. G. Baker, and K. A. Gallivan. A truncated-cg style method for symmetric generalized eigenvalue problem. *Journal of Computational and Applied Mathematics, submitted*.

[2] M. F. Adams. Evaluation of three unstructured multigrid methods on 3d finite element problems in solid mechanics. *International Journal for Numerical Methods in Engineerin*, 55:519–534, 2002.

[3] 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.

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

[5] J. Cullum and R. A. Willoughby. *Lanczos algorithms for large symmetric eigenvalue computations*, volume 1: Theory of *Progress in Scientific Computing; v. 3*. Birkhauser, Boston, 1985.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] X. Jia. A refined iterative algorithm based on the block Arnoldi process for large unsymmetric eigenproblems. *Linear Algebra and Its Applications*, 270:171–189, 1997.

[11] A. Knyazev. Hard and soft locking in iterative methods for symmetric eigenvalue problems. In *Eighth Copper Mountain Conference on Iterative Methods*, 2004.

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

[13] 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.

[14] J. Langou. For a few iterations less. In *Eighth Copper Mountain Conference on Iterative Methods*, 2004.

[15] 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.

[16] 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.

[17] John G. Lewis. private communication, 2004.

[18] J. R. McCombs and A. Stathopoulos. Iterative validation of eigensolvers: A scheme for improving the reliability of hermitian eigenvalue solvers. *submitted*, (Tech. report WM-CS-2005-02).

[19] J. R. McCombs and A. Stathopoulos. PRIMME: PReconditioned Iterative Multimethod Eigensolver. http://www.cs.wm.edu/ andreas/software/.

[20] R. B. Morgan. Computing interior eigenvalues of large matrices. *Lin. Alg. Appl.*, 154–156:289–309, 1991.

[21] 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.

[22] C. C. Paige, B. N. Parlett, and Van der Vorst. Approximate solutions and eigenvalue bounds from Krylov spaces. *Num. Lin. Alg. Appl.*, 2:115–133, 1995.

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

[24] Beresford N. Parlett. private communication, 2004.

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

[26] 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.

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

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

[29] A. Stathopoulos. Nearly optimal preconditioned methods for hermitian eigenproblems under limited memory. Part i: Seeking one eigenvalue. *submitted*, (Tech. report WM-CS-2005-03).

[30] A. Stathopoulos and J. R. McCombs. Nearly optimal preconditioned methods for hermitian eigenproblems under limited memory. Part ii: Seeking many eigenvalues. *submitted*, (Tech. report WM-CS-2006-02).

[31] A. Stathopoulos, Serdar Öğüt, Y. Saad, J. R. Chelikowsky, and Hanchul Kim. Parallel methods and tools for predicting material properties. *Computing in Science and Engineering*, 2(4):19–32, 2000.

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

[33] 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.

[34] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Claredon Press, Oxford, England, 1965.