NEARLY OPTIMAL PRECONDITIONED METHODS FOR HERMITIAN EIGENPROBLEMS UNDER LIMITED MEMORY. PART II: SEEKING MANY EIGENVALUES *

ANDREAS STATHOPOULOS † AND JAMES R. MCCOMBS †

Abstract. In a recent companion paper, we proposed two methods, GD+k and JDQMR, as nearly optimal methods for finding one eigenpair of a real symmetric matrix. In this paper, we seek nearly optimal methods for a large number, *nev*, of eigenpairs that work with a search space whose size is $O(1)$, independent from *nev*. The motivation is twofold: avoid the additional $O(nevN)$ storage and the $O(nev^2N)$ iteration costs. First, we provide an analysis of the oblique projectors required in the Jacobi-Davidson method and identify ways to avoid them during the inner iterations, either completely or partially. Second, we develop a comprehensive set of performance models for GD+k, Jacobi-Davidson type methods, and ARPACK. Based both on theoretical arguments and on our models we argue that any eigenmethod with $O(1)$ basis size, preconditioned or not, will be superseded asymptotically by Lanczos type methods that use $O(nev)$ vectors in the basis. However, this may not happen until $nev > O(1000)$. Third, we perform an extensive set of experiments with our methods and against other state-of-the-art software that validate our models and confirm our GD+k and JDQMR methods as nearly optimal within the class of O(1) basis size methods.

1. Introduction. The numerical solution of large, sparse, Hermitian or real symmetric eigenvalue problems is one of the most computationally intensive tasks in a variety of applications. The challenge is twofold; First, the matrix size, N, is routinely more than a million, while an order of a billion has also been tried [43]. Second, many applications, including electronic structure calculations, require the computation of hundreds or even thousands of extreme eigenpairs. Often the number of required eigenpairs, *nev*, is described as a small percentage of the problem size. In such cases, orthogonalization of *nev* vectors, an $O(nev^2N)$ task, becomes $O(N^3)$, making the scaling to larger problem sizes practically infeasible.

Iterative methods are the only means of addressing these large problems. Yet, iterative methods may converge slowly, especially as the problem size grows, and must store the iteration vectors for computing eigenvector approximations. Beyond challenges in execution time, the storage demands of these applications can be staggering. Over the last decade, iterative methods have been developed [14, 51, 54, 30, 29, 49, 41, 57] that can use effectively the large arsenal of preconditioners for linear systems and converge nearly optimally to an eigenpair under limited memory requirements.

The quest for optimality under limited memory is a natural one. On symmetric linear systems, Krylov methods such as Conjugate Gradient (CG) achieve optimal convergence through a three term recurrence. For eigenvalue problems, the Lanczos method can produce the optimal space through a three term recurrence, but the vectors must be stored or recomputed. With preconditioning, traditional Lanczos does not apply directly. Restarting techniques can be employed so that approximations are obtained from a search space of limited size, but at the expense of convergence.

When seeking one eigenpair, a useful way to approach the problem is by considering it as a nonlinear problem, where both the eigenvalue and eigenvector are unknown. In [57], we studied the problem from two nonlinear viewpoints: the inexact Newton, and the limited memory quasi-Newton [40]. The two methods we have developed, the JDQMR and the GD+k, can be related to the above respective viewpoints, which al-

*Work supported by National Science Foundation grants: ITR/DMR 0325218, ITR/AP-0112727.

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

lowed us to argue for their near-optimal convergence. The former is a Jacobi-Davidson (JD) method where the inexact inner iteration is stopped dynamically and optimally. The latter is a Generalized Davidson method with a recurrence based restarting that delivers convergence extremely close to, and sometimes indistinguishable from the optimal method (i.e., without restarting).

When seeking many eigenpairs, it is an open question whether optimality can be achieved under limited memory. If one eigenvalue is known exactly, the corresponding eigenvector can be obtained optimally through a CG iteration [30, 57]. If *nev* eigenvalues are known, one may think that the analogue optimality is to run *nev* separate CG iterations. This is the approach taken by most limited memory, preconditioned eigensolvers for small *nev* values. Yet, it is clearly suboptimal because a method that stores the CG iterates from each run would converge in much fewer iterations. For example, when the number of CG iterations is $O(N)$, the former approach takes $O(nev\ N)$, while an unrestarted Lanczos would take no more than N.

In this paper, we focus on methods that do not allow their memory requirements to grow unbounded. We further distinguish between two "classes" of these methods: The first, allows the size of the search space to be a small multiple of *nev*, typically $2nev$ or $1.5nev$. This class concedes the $O(nev^2)$ factor in the hope of much fewer iterations. The second class restricts the search space to a constant size (e.g., 20 vectors) and uses locking to obtain all *nev* eigenvectors. Although the number of iterations increases, orthogonalization and iteration costs decrease dramatically. This memory-based classification is slightly different from the one in [5] but it captures better the performance characteristics of solvers for large *nev*. The holy grail in this area has been to obtain *nev* eigenpairs in linear to *nev* scaling.

We make three contributions in this paper. First, after discussing the merits of current state-of-the-art methods for finding many eigenvalues, we analyze the effects of projecting against converged eigenvectors in the correction equation of the JD method, and propose variants that avoid both the extra storage and the second orthogonalization. In particular, the unpreconditioned JDQMR-000 variant requires no orthogonalization during inner iterations and thus has the potential of overcoming the $O(nev^2)$ scaling barrier.

Second, we perform a detailed complexity analysis of the two classes of methods and, based also on experimental observations, we develop a complete set of models that describe the relative performance between JD variants, GD+k, and implicitly restarted Lanczos as implemented in ARPACK. The analysis not only reveals the relative asymptotic behavior, but also provides performance crossover points between methods that can be used for dynamic method selection within a multimethod software, such as PRIMME that we have recently developed [34].

Third, we conduct an extensive set of experiments, with and without preconditioning, which complement the ones in [5] by comparing our GD+k and JDQMR variants against software that implements current state-of-the-art methods such as JDBSYM, BLOPEX, and ARPACK. The experiments show that our proposed JDQMR variants and GD+k are clearly the best methods in their class, and confirm the crossover points and the asymptotic behaviors from our models. They also demonstrate the almost linear scaling of the JDQMR-000 variant without preconditioning up to several hundreds of eigenvalues.

2. State-of-the-art eigenmethods for many eigenvalues. Throughout this paper we assume we seek the *nev* algebraically smallest eigenvalues λ_i and their corresponding eigenvectors $\mathbf{x}_i,\ i = 1, \ldots, nev$, of a symmetric matrix A of size N.

2.1. Lanczos based methods. It should be no surprise that unrestarted Lanczos is still considered a state-of-the-art method without preconditioning. It is the optimal method in terms of number of iterations, for one or more eigenvalues, it is theoretically well understood, and it has several efficient, highly robust implementations [58, 45, 21, 9]. For finding one eigenvalue Jacobi-Davidson methods are now preferred to Lanczos, not only because of preconditioning, but also because of nearly optimal convergence that does not require unlimited storage [57]. For many eigenvalues, however, the ability of unrestarted Krylov spaces to capture large parts of the extremes of the spectrum of A remains unparalleled. This optimal convergence comes with several drawbacks. Frequent selective and/or partial orthogonalizations are required to avoid dealing with ghost eigenvalues. Also, besides storing or recomputing the vector iterates, the method needs to store a tridiagonal matrix of size equal to the number of iterations, which can be tens of thousands, and solve for nev of its eigenpairs. Finally, Lanczos has difficulty obtaining multiple eigenvalues. Block versions of Lanczos have been proposed as a robust and cache efficient alternative [8, 19] but usually increase the total execution time (see [22] for a recent survey).

In the rest of the paper, we do not consider unrestarted Lanczos methods, because they do not satisfy our limited memory criterion and cannot use preconditioning without some form of inner-outer iteration [57, 38]. In doing so, we realize that there could be cases where the unrestarted methods are faster than the alternatives we present. In addition, we do not discuss the powerful shift-and-invert Lanczos [45], because it requires the exact inversion of the matrix.

To reduce storage and iteration costs, the Lanczos method can be restarted through the implicit restarting [53] or the equivalent thick restarting [36, 56, 59] techniques. When the maximum size m_{max} for the search space is reached, the nev required Ritz vectors are computed and replace the search space. The ARPACK software is a high quality, efficient implementation of the implicitly restarted Arnoldi and Lanczos methods [33], and it has become the default eigensolver in many applications.

Restarting impairs the optimality of Lanczos. Implicit (or thick) restarting with nev (or more) vectors improves convergence but may not be sufficient with small basis sizes. Indeed, it is known that implicitly restarted Lanczos (IRL) is a polynomially accelerated simultaneous iteration with a polynomial of degree $m_{max} - nev$ [32]. Hence, inner-outer methods can be more effective for small nev. On the other hand, ARPACK requires that $m_{max} > nev$ (and typically $m_{max} = 2nev$), which implies that for really large nev the algorithm approaches again an unrestarted Lanczos with full orthogonalization; with its convergence benefits but high iteration costs.

2.2. Newton approaches. We can view the eigenvalue problem as a constrained minimization problem for minimizing the Rayleigh quotient $\mathbf{x}^T A\mathbf{x}$ on the unit sphere or equivalently minimizing $\mathbf{x}^T A\mathbf{x}/\mathbf{x}^T\mathbf{x}$ [14]. For many eigenpairs, the same formulation applies for minimizing the trace of a block of vectors [48], working with nev-dimensional spaces [2]. As we discussed in [57], most eigenmethods can be interpreted through the inexact Newton viewpoint or the quasi-Newton viewpoint.

2.2.1. The inexact Newton approach. The exact Newton method for eigenproblems, applied on the Grassmann manifold to enforce normalization of the eigenvectors, is equivalent to Rayleigh Quotient Iteration (RQI) [14]. It is well known that solving the Hessian equation to full accuracy is not needed at every Newton step. Inexact Newton methods attempt to balance good convergence of the outer Newton method with an inexact solution of the Hessian equation. Extending inexact Newton methods to RQI has attracted a lot of attention in the literature [46, 31, 52, 20, 49].

In [57] we argued that a more appropriate representative of the inexact Newton minimization for eigenvalue problems is the Jacobi-Davidson method [51]. See also [1] for a theoretical discussion on Newton variants. Given an approximate eigenvector $\mathbf{u}^{(m)}$ and its Ritz value $\theta^{(m)}$, the JD method obtains an approximation to the eigenvector error by solving approximately the correction equation:

$$(I - \mathbf{u}^{(m)}\mathbf{u}^{(m)T})(A - \eta I)(I - \mathbf{u}^{(m)}\mathbf{u}^{(m)T})\mathbf{t}^{(m)} = -\mathbf{r}^{(m)} = \theta^{(m)}\mathbf{u}^{(m)} - A\mathbf{u}^{(m)}, \tag{2.1}$$

where η is a shift close to the wanted eigenvalue. The next (inexact) Newton iterate is then $\mathbf{u}^{(m+1)} = \mathbf{u}^{(m)} + \mathbf{t}^{(m)}$. The pseudoinverse of the Hessian is considered to avoid the singularity when $\eta \approx \lambda$, and also to avoid yielding back $\mathbf{t}^{(m)} = \mathbf{u}^{(m)}$ when the equation is solved accurately with $\eta = \theta^{(m)}$. The latter problem could cause stagnation in the classical or the Generalized Davidson (GD) methods [10, 37, 55, 42, 61]. The GD method obtains the next iterate as $\mathbf{t}^{(m)} = K^{-1}\mathbf{r}^{(m)}$, where the preconditioner K approximates $(A - \eta I)$ directly, not as an iterative solver.

JD and GD are typically used with subspace acceleration, where the iterates $\mathbf{t}^{(m)}$ are accumulated in a search space from which eigenvector approximations are extracted through Rayleigh-Ritz or some other projection technique [44, 35, 25]. This is particularly beneficial, especially when looking for more than one eigenpair.

The challenge in JD is to identify the optimal accuracy to solve each correction iteration. In [41] Notay proposed a dynamic stopping criterion based on monitoring the growing disparity in convergence rates between the eigenvalue residual and linear system residual of CG. The norm of the eigenvalue residual was monitored inexpensively through a scalar recurrence (see [24] for recent extensions). In [57] we proposed JDQMR that extends JDCG by using symmetric QMR [16] as the inner method. The advantages are: (a) The smooth convergence of sQMR allows for robust and efficient stopping criteria. (b) It can handle indefinite correction equations, which is important also for large *nev*. (c) sQMR, unlike MINRES, can use indefinite preconditioners, which are often needed for interior eigenproblems. We also argued that JDQMR type methods cannot converge more than three times slower than the optimal method, and usually are significantly less than two times slower. Coupled with the very low sQMR costs, JDQMR has proven one the fastest and most robust methods for $nev = 1$.

When seeking many eigenvalues, the Newton method can be applied on the *nev* dimensional Grassmann manifold to compute directly the invariant subspace (see [48] and [2] for a more recent review). Practically, however, the Grassmann RQI approach proposed in [2] is simply a block JD method. To see this, note first that the JD search space is kept orthonormal; second, a Rayleigh-Ritz is performed at every outer step, and therefore the Sylvester equations stemming from the Newton approach are decoupled as *nev* independent correction equations for each vector in the block. The open computational question is how to solve these *nev* linear systems most efficiently, and whether to use a block method at all.

The problem is highly related to ongoing linear systems research for multiple right hand sides [50, 7, 26, 23]. It could be argued that seed methods that build one, large Krylov space, and reuse it to solve all the *nev* equations are the best choice. In our case, the disadvantage of seed methods is that they need to store this large space; exactly what we try to avoid by restarting the outer JD iteration, and by using short term recurrence methods for the correction equation. Moreover, the multiple right hand sides are not related in any way, so it is unclear how much one system can benefit from the Krylov space of another.

Block methods that solve all the correction equations simultaneously do not consistently improve the overall runtime. In some occasions, however, when the eigenvalues are multiple or highly clustered, certain JD implementations may benefit from a small block size, because the fast, targeted convergence of Newton could cause eigenvalues to converge out of order, and thus the required ones to be missed.

With large *nev*, however, the near optimal convergence of the JDQMR has to be repeated *nev* times; much like the *nev* independent CGs we mentioned in the introduction. In this case, and assuming a very small block size, the role of a larger subspace acceleration is to obtain better approximations for nearby eigenpairs while JD converges to the targeted eigenpair. Although the convergence rate of QMR cannot improve further, increasing the basis size gives increasingly better initial guesses for the eigenpairs to be targeted next. In this paper, we avoid this continuum of choices and focus only on constant, limited memory basis sizes.

2.2.2. The quasi-Newton approach. An alternative to Newton is the use of the nonlinear Conjugate Gradient (NLCG) method for eigenproblems [14]. It is natural to consider a method that minimizes the Rayleigh quotient on the whole space $L = \left\{\mathbf{u}^{(m-1)}, \mathbf{u}^{(m)}, \mathbf{r}^{(m)}\right\}$, instead of only along one search direction. The method:

$$\mathbf{u}^{(m+1)} = \text{RayleighRitz}\left(\{\mathbf{u}^{(m-1)}, \mathbf{u}^{(m)}, \mathbf{r}^{(m)}\}\right),\ m > 1, \tag{2.2}$$

is often called locally optimal Conjugate Gradient (LOCG) [13, 28], and seems to consistently outperform other NLCG type methods. For numerical stability, $\mathbf{u}^{(m)} - \tau^{(m)}\mathbf{u}^{(m-1)}$, for some weight $\tau^{(m)}$, can be used instead of $\mathbf{u}^{(m-1)}$. When used with multivectors $\mathbf{u}^{(m)}, \mathbf{r}^{(m)}$ this is the well known LOBPCG method [30].

LOCG can be accelerated by minimizing over the subspace of more iterates $\mathbf{u}^{(m)}$. When the subspace reaches its maximum size we must restart. If we restart through IRL, however, we lose the single important direction ($\mathbf{u}^{(m-1)}$) that offers the excellent convergence to LOCG. This was first observed in [39] for the Davidson method, although under a different viewpoint. In [54] we offered a theoretical justification and an efficient implementation that combined this technique with thick restarting for the GD. In [57], we noted the connection of our method, which we call GD+k, to quasi-Newton and in particular to the limited memory BFGS method [18, 40].

GD(m_{min}, m_{max})+k uses a basis of maximum size m_{max}. When m_{max} is reached, we compute the m_{min} smallest Ritz vectors, $\mathbf{u}_i^{(m)}$, and also k of the corresponding Ritz vectors $\mathbf{u}_i^{(m-1)}$ from step $m-1$. An orthonormal basis for this set of m_{min}+k vectors becomes the restarted basis. If the block size in GD/JD is b, it is advisable to keep k $\geq b$ to maintain good convergence for all block vectors. The special case of block GD(b, 3b)+b is equivalent mathematically to LOBPCG with the same block size. In [57], we showed that convergence of the GD+k for one eigenvalue is often indistinguishable from the optimal method. Yet, higher iteration costs than JDQMR make it less competitive for very sparse operators.

When seeking many eigenvalues, we have never found the use of a block size $b > 1$ beneficial, even with a small subspace acceleration. Then, as with JDQMR, each eigenpair has to be targeted and converged independently. The role of subspace acceleration is perhaps more important for GD+k than for JDQMR, as it provides both its local optimal convergence and the initial guesses for nearby eigenvalues.

2.3. The GD+k and the JDQMR algorithms. The GD algorithm can serve as the basic framework for implementing both JDQMR and GD+k. Algorithm 2.1 depicts a version of the basic GD+k algorithm for finding *nev* smallest eigenpairs. It implements Rayleigh-Ritz and locking and is the basis for the performance models

ALGORITHM 2.1. *The Generalized Davidson(m_{min}, m_{max})+k algorithm*

(1) *start with* $\mathbf{v}_0$ *starting vector*
(2) $\mathbf{t}^{(0)} = \mathbf{v}_0,\ l = m = nmv = 0,\ X = [\ ]$
(3) **while** $l < nev$ **and** $nmv < max\_num\_matvecs$
(5)   *Orthonormalize* $\mathbf{t}^{(m)}$ *against* $\mathbf{v}_i, i = 1,\ldots,m$ *and* $\mathbf{x}_i, i = 1,\ldots,l$
(6) $m = m + 1$, $nmv = nmv + 1$, $\mathbf{v}_m = \mathbf{t}^{(m-1)}$, $\mathbf{w}_m = A\mathbf{v}_m$
(7) $H_{i,m} = \mathbf{v}_i^T \mathbf{w}_m$ *for* $i = 1,\ldots,m$
(7.0) $s_i^{old} = s_i,\ i = 1,\ldots,m$
(8)   *compute eigendecomposition* $H = S\Theta S^T$ *with* $\theta_1 \le \theta_2 \le \ldots \le \theta_m$
(9) $\mathbf{u}^{(m)} = Vs_1$, $\theta^{(m)} = \theta_1$, $\mathbf{w}^{(m)} = Ws_1$
(10) $\mathbf{r}^{(m)} = \mathbf{w}^{(m)} - \theta^{(m)}\mathbf{u}^{(m)}$
(11)   **while** $\|\mathbf{r}^{(m)}\| \le tol$
(12) $\lambda_{l+1} = \theta^{(m)}$, $X = [X, \mathbf{u}^{(m)}]$, $l = l + 1$
(13)   **if** $(l = nev)$ **stop**
(14) $m = m - 1$, $H = 0$
(15)   **for** $i = 1,\ldots,m$
(16) $\mathbf{v}_i = Vs_{i+1}$, $\mathbf{w}_i = Ws_{i+1}$
(17) $H_{ii} = \theta_{i+1}$, $s_i = e_i$, $\theta_i = \theta_{i+1}$
(18)   **end for**
(19) $\mathbf{u}^{(m)} = \mathbf{v}_1$, $\theta^{(m)} = \theta_1$, $\mathbf{r}^{(m)} = w_1 - \theta^{(m)}\mathbf{u}^{(m)}$
(20)   **end while**
(21)   **if** $m \ge m_{max}$ *then*
(21.0)   *Orthogonalize* $s_i^{old},\ i = 1,\ldots,\mathrm{k}$ *among themselves*
            and against $s_i,\ i = 1,\ldots,m_{min}$
(21.1)   *Compute* $H_{sub} = s^{oldT} H s^{old}$
(21.2)   *Set* $s = [s_1,\ldots,s_{m_{min}}, s_1^{old},\ldots,s_{\mathrm{k}}^{old}]$
(22) $H = 0$
(23)   **for** $i = 2,\ldots,m_{min} + \mathrm{k}$
(24) $\mathbf{v}_i = Vs_i$, $\mathbf{w}_i = Ws_i$, $H_{ii} = \theta_i$
(25)   **end for**
(26) $\mathbf{v}_1 = \mathbf{u}^{(m)}$, $\mathbf{w}_1 = \mathbf{w}^{(m)}$, $H_{11} = \theta^{(m)}$, $m = m_{min}$
(26.0) $H(m_{min} + 1 : m_{min} + \mathrm{k}, m_{min} + 1 : m_{min} + \mathrm{k}) = H_{sub}$
(26.1) $m = m_{min} + \mathrm{k}$
(27)   **end if**
(28)   *Precondition the residual* $\mathbf{t}^{(m)} = Prec(\mathbf{r}^{(m)})$
(29) **end while**

in Section 4. The implementation of +k restarting is shown at steps numbered with decimal points. Both steps (21.0) and (21.1) apply on vectors of size m_{max}, and therefore the cost of the GD(m_{min}, m_{max})+k implementation is the same as that of GD(m_{min}+k, m_{max}). In our experience, k=1 or 2 is always sufficient, obviating the use of larger m_{min}, hence being less expensive. Although not explored here, Algorithm 2.1 can be viewed as a block method by implementing vectors as multivectors.

Step (28) differentiates between eigenmethods. By returning $\mathbf{t}^{(m)} = \mathbf{r}^{(m)}$ or the preconditioned residual, $\mathbf{t}^{(m)} = K^{-1}\mathbf{r}^{(m)}$, we have the GD+k method. Step (28) can also return a JD correction vector. Without inner iterations, a preconditioner K can be inverted orthogonally to the space $Q = [X, \mathbf{u}^{(m)}]$ and applied to $\mathbf{r}^{(m)}$. Assuming

$(Q^T K^{-1} Q)$ is invertible, the pseudoinverse of this preconditioner can be written as:

$$\left((I - QQ^T)K(I - QQ^T)\right)^+ = (I - K^{-1}Q(Q^T K^{-1} Q)^{-1} Q^T) K^{-1} (I - QQ^T) \tag{2.3}$$
$$= K^{-1}(I - Q(Q^T K^{-1} Q)^{-1} Q^T K^{-1})(I - QQ^T). \tag{2.4}$$

When this preconditioner is used in an iterative method on eq. (2.1), a significant extra storage for $K^{-1}Q$ is required to avoid doubling the number of preconditioning operations. In [15, 51, 6] it is shown that the JD method can be implemented with one projection with Q per iteration. In the following section we show that this may not be sufficient with right preconditioning and high convergence tolerance. Our JDQMR algorithm uses the GD+k as the underlying outer method and, at step (28), calls the symmetric, right preconditioned QMR as described in detail in [57].

3. Avoiding the JD oblique projectors. When eigenvectors converge in the Jacobi-Davidson method, they are locked in the array X, and the algorithm targets the next higher eigenpair. As the shift in the correction equation moves inside the spectrum the equation becomes increasingly indefinite. Although this is not prohibitive for the QMR solver, it usually is unnecessarily expensive, as QMR has to rebuilt the lower eigendirections in X sufficiently so that it can converge to the correction which should be orthogonal to X. The JD authors have observed that it is always beneficial to project X from the correction equation. They also mention that the preconditioner needs also to be projected in the same way, so that the image of its operation matches the domain of the projected matrix A. This reasoning, however, is not of practical use because the vectors can be considered embedded in $\Re^N$. A better approach would be to consider the effect of these projectors on the conditioning of the correction equation, and whether it justifies the significant additional expense; either a second preconditioning operation per iteration, or an array that stores $K^{-1}X$.

There is little doubt that the skew projection of eq. (2.3) should be used for the Ritz vector $\mathbf{u}^{(m)}$. First, there is a constant additional cost to QMR, which is independent from *nev*. More importantly, an extremely accurate preconditioner may cause the problems seen in Davidson. A recent analysis by Notay [42] shows that, although such problems are rare, they could arise. The Davidson problem is specific to the vector $\mathbf{u}^{(m)}$ and cannot occur if orthogonality to the converged eigenvectors X is not enforced in the correction equation. However, the effectiveness of the preconditioner may change. To simplify notation in this section, we assume that the matrix A and the preconditioner K encapsulate both the shifting with η and the projection (2.3) for $\mathbf{u}^{(m)}$, and focus on $Q = X$. Define also the projector for any matrix B for which $Q^T BQ$ is not singular:

$$P_B = (I - BQ(Q^T BQ)^{-1} Q^T), \tag{3.1}$$

and note that if $P = (I - QQ^T)$, in addition to eqs. (2.3, 2.4) it holds:

$$PP_B BP = P_B B = BP_B^T = PBP_B^T P. \tag{3.2}$$

Based on the above, the appropriate formulation for right preconditioning of the JD projected operator in the correction equation is:

$$PAP(PKP)^+ = PAP_{K^{-1}} K^{-1} \tag{3.3}$$
$$= PAK^{-1} P_{K^{-1}}^T. \tag{3.4}$$

Table 3.1

Projection alternatives to the classical Jacobi-Davidson correction equation (with right preconditioning). The 0/1 string characterizes whether there is a projection on the left of A, on the right of A, and whether the right projection is skew projection or not. Theoretically Jacobi-Davidson corresponds to (111) although it is typically implemented as (011).

(Left Skew Right)	Operator	(Left Skew Right)	Operator
111	$PAP_{K^{-1}}K^{-1}$	011	$AP_{K^{-1}}K^{-1}$
101	$PAPK^{-1}$	001	APK^{-1}
100	PAK^{-1}	000	AK^{-1}

Computationally the above formulation is expensive because at every QMR step it requires two orthogonalizations with Q, from both left and right of A, and a backsolve with the factors of the small matrix $(Q^T K^{-1} Q)^{-1}$. Moreover, to avoid an extra preconditioning operation per iteration, the vectors $K^{-1}Q$ must be stored, a very unreasonable storage requirement for large number of eigenvalues. Naturally, the question is whether some projectors can be avoided, in their skew form or altogether.

Eq. (3.3) suggests several variants of a projected operator based on whether we operate with a projector on the left and/or on the right of A, and whether we relax the requirement for a right skew projector, replacing it with P. Table 3.1 summarizes the variants. Note that the cases $PAK^{-1}P^T_{K^{-1}}$ and $AK^{-1}P^T_{K^{-1}}$ (from eq. (3.4)) are equivalent to (111) and (011) respectively, so they are not included. Additionally, when the QMR iterates are orthogonal to Q, a P projection on the right of the preconditioner is unnecessary, hence we have not included the case $PAK^{-1}P$. Finally, we do not include the variant $AK^{-1}P$ because it maintains no orthogonality of the QMR iterates. We discuss each column of the table separately.

3.1. No P projection on the left of A. Case (000) does not work orthogonally to Q, and it is expected to require a sharply increasing number of inner iterations as more eigenvalues are locked, especially with less effective preconditioners. There is a notable exception. When K has the same eigenvectors as A (e.g., if K is a polynomial of A or simply $K = I$), QMR starts with the residual $\mathbf{r}^{(m)} \perp Q$ and thus all its iterates will stay in $Q^{\perp}$. This property is sometimes exploited in methods with polynomial acceleration [60] (see also [4] in the context of an a priori known null space in Maxwell's equations). Floating point arithmetic and the fact that the eigenvectors in Q are converged to tol, not to machine precision, will eventually introduce Q components that QMR will have to remove by additional iterations. However, a few additional iterations is a small price to pay for removing the limiting scalability factor $O(nev^2 N)$ of orthogonalization. In all our experiments, unpreconditioned JDQMR-000 (no projections) achieves an almost linear scaling with nev, both in convergence and in time.

Case (011) is the recommended way to implement right preconditioning in JD. For any vector v, we have $P_{K^{-1}}K^{-1}v \in Q^{\perp}$, and thus $AP_{K^{-1}}K^{-1}v \in Q^{\perp}$, requiring no additional left orthogonalization. This formulation, however, may have disadvantages when the required relative convergence tolerance is much larger than machine precision (e.g., $tol \geq \|A\| \sqrt{\epsilon_{\text{machine}}}$). In such cases, large Q components may appear early in the QMR iterations, delaying its convergence. Finally, we note that the same concern holds for the (001) case, which we do not examine further because even if we fixed this concern, for example by using case (101), it would still not be competitive for the reason described in Section 3.2.2.

3.2. P projection on the left of A. Cases that project with P on the left of A guarantee that all QMR iterates will be exactly in $Q^{\perp}$, regardless of how close Q is to an A-invariant subspace. Case (111) implements accurately the JD projection, but it is very expensive and storage demanding. Cases (101) and (100) approximate equations (3.3) and (3.4) respectively by replacing the skew projectors with P, thus avoiding the additional storage. In addition, (100) does not require the right projector, so it is the least expensive method that can maintain the QMR iterates in $Q^{\perp}$. If conditioning of the matrix in (100) does not differ much from the matrix in (111), case (100) should be preferred.

To examine the condition numbers of the projected matrices, we note first that when B has a non trivial null space, its condition number is defined in its range as $\kappa(B) = \|B\|\|B^+\|$. We also need the following elementary property for pseudoinverses.

PROPOSITION 3.1. *If symmetric matrices A, B have the same null space, then:*

$$(AB)^+ = B^+A^+.$$

Proof. The pseudoinverse of a matrix A is defined as $A^+ = V\Sigma^{-1}U^T$, where $A = U\Sigma V^T$ is the economy size SVD of A which consists only of the non zero singular values and vectors. Similarly, the pseudoinverse for $B = YMZ^T$ is $B = ZM^{-1}Y^T$. Let $X \in \Re^{N,k}$, $k < N$, be an orthonormal basis for the range of A and B. Because A, B are symmetric, their SVD can be described as $A = U\Sigma(XQ_A)^T$ and $B = XQ_BMZ^T$, for some orthogonal matrices $Q_A, Q_B \in \Re^{k,k}$. By substituting $Y = XQ_B$ and $V = XQ_A$ and using a known property of pseudoinverses, we have:

$$\begin{aligned}(AB)^+ &= (U\Sigma V^TYMZ^T)^+ = Z(\Sigma V^TYM)^{-1}U^T \\ &= ZM^{-1}Y^TY(V^TY)^{-1}\Sigma^{-1}U^T = B^+Y(V^TY)^{-1}\Sigma^{-1}U^T \\ &= B^+XQ_B(Q_A^TQ_B)^{-1}\Sigma^{-1}U^T = B^+XQ_A\Sigma^{-1}U^T = B^+A^+.\end{aligned}$$

□

Using Proposition 3.1 with a two norm and eqs. (3.3, 3.4), we can derive the condition numbers for cases (111), (101), and (100):

$$\begin{aligned}\kappa_{111} &= \|PAP(PKP)^+\|\|PKP(PAP)^+\| = \|PAK^{-1}P^T_{K^{-1}}\|\|PKA^{-1}P^T_{A^{-1}}\| \\ \kappa_{101} &= \|PAPK^{-1}P\|\|(PK^{-1}P)^+(PAP)^+\| = \|PAPK^{-1}P\|\|P_KKPA^{-1}P^T_{A^{-1}}\| \\ \kappa_{100} &= \|PAK^{-1}P\|\|(PAK^{-1}P)^+\| = \|PAK^{-1}P\|\|P_{KA^{-1}}KA^{-1}\|.\end{aligned}$$

There are obvious similarities between the condition numbers, but no one emerges as the default winner. The effect of the projectors $P^T_{K^{-1}}, P^T_{A^{-1}}$ and $P_{KA^{-1}}$ depends on how close A, K, K^{-1}, KA^{-1} are to Q-invariant. We therefore examine the following two special, extreme cases.

3.2.1. Special case: Q are exact eigenvectors of A. This assumption leads to $PA = AP$, $P_{KA^{-1}} = P_K$, $P^T_{A^{-1}} = P$, and, by considering eq. (3.2), to the following simplifications:

$$\begin{aligned}\kappa_{111} &= \|PAK^{-1}P^T_{K^{-1}}\|\|PKA^{-1}P\|, \\ \kappa_{100} &= \kappa_{101} = \|PAK^{-1}P\|\|P_KKA^{-1}P\|.\end{aligned}$$

The differences between κ_{111} and κ_{100} may be better understood if we consider how close Q is to an invariant subspace of K. If we denote as $R_K = KQ - Q(Q^TKQ)$ the residual of Q in K, we have $P_K = P - R_K(Q^TKQ)^{-1}Q^T$, and κ_{100} becomes:

$$\begin{aligned}\kappa_{100} &= \|PAK^{-1}P\|\|PKA^{-1}P - R_K(Q^TKQ)^{-1}Q^TKPA^{-1}\| \\ &= \|PAK^{-1}P\|\|PKA^{-1}P - R_K(Q^TKQ)^{-1}R_K^TA^{-1}\|.\end{aligned} \tag{3.5}$$

Similarly, with $R_{K^{-1}} = K^{-1}Q - Q(Q^TK^{-1}Q)$ and $P_{K^{-1}}^T = P - Q(Q^TK^{-1}Q)^{-1}R_{K^{-1}}$,

$$\kappa_{111} = \|PAK^{-1}P - AR_{K^{-1}}(Q^TK^{-1}Q)^{-1}R_{K^{-1}}^T\|\|PKA^{-1}P\|. \tag{3.6}$$

Moreover, let $\delta_0 = \|R_K(Q^TKQ)^{-1}R_K^TA^{-1}\|$ and $\delta_1 = \|AR_{K^{-1}}(Q^TK^{-1}Q)^{-1}R_{K^{-1}}^T\|$. Then, $\delta_0 = O(\|R_K\|^2)$ and $\delta_1 = O(\|R_{K^{-1}}\|^2)$, and through some basic algebra we can derive:

$$|\kappa_{111} - \kappa_{000}| \le \delta_0\|P_KKA^{-1}P\| + \delta_1\|PAK^{-1}P_{K^{-1}}^T\| = O(\|R_K\|^2 + \|R_{K^{-1}}\|^2). \tag{3.7}$$

Eq. (3.7) implies that the differences between the two methods disappear with better preconditioners, when Q becomes close to K-invariant (depending on the squares of the residuals). When Q is far from K-invariant the two condition numbers may not be close. However, there is no reason why κ_{111} should be smaller than κ_{100}. If we assume, according to eqs. (3.5, 3.6), that smaller residuals may lead to smaller condition numbers, then obtaining $\|R_K\| \ll \|R_{K^{-1}}\|$ would depend on the norms of $\|K\|$ and $\|K^{-1}\|$ and on what part of the K spectrum is approximated by Q and how well. In extensive experiments on practical cases we have noticed negligible differences between cases (111) and (100).

3.2.2. Special case: Q inexact eigenvectors but $K = A$. In this extreme case, $\kappa_{111} = \kappa_{100} = 1$, but κ_{101} can be much worse. To see why, let the residuals $R_A = AQ - Q(Q^TAQ)$ and $R_{A^{-1}}^T = Q^TA^{-1} - (Q^TA^{-1}Q)Q^T$, and consider the norms for κ_{101}:

$$\begin{aligned}\|PAPK^{-1}P\| &= \|P(A - AQQ^T)A^{-1}P\| = \|P(I - (R + QM)Q^TA^{-1}P\| \\ &= \|P - P(R_A + Q(Q^TAQ))(R_{A^{-1}}^T + (Q^TA^{-1}Q)Q^T)P\| \\ &= \|P - R_AR_{A^{-1}}^T\|, \\ \|P_KKPA^{-1}P_{A^{-1}}^T\| &= \|P + R_A(Q^TAQ)^{-1}(Q^TA^{-1}Q)^{-1}R_{A^{-1}}^T\|.\end{aligned}$$

Obviously, interleaving an inaccurate eigenprojector between A and K can cause κ_{101} to be far from 1, despite the double orthogonalization expense. The same effects remain, only exaggerated, if the left projection is avoided as in the (001) case.

3.2.3. General case. When both Q and K are not accurate, it is hard to quantify the relative merits of κ_{111} and κ_{100}. However, the Q are not random vectors but approximate eigenvectors to a certain threshold (typically at least as good as single precision). Therefore, we expect our results of Section 3.2.1 to apply in general. In particular, eqs. (3.5, 3.6) suggest that method (100) should be preferred over the expensive (111) with good preconditioners. This is in contrast to the situation for the Ritz vector $\mathbf{u}^{(m)}$, where skew projectors are especially needed when $K \approx A$.

In our extensive experiments, we never found it necessary to use method (111) when the eigenvectors were computed relatively accurately. In fact, method (100) consistently gave lower iteration numbers. Despite the lack of theoretical assurance, we have observed similar behavior even with smaller eigenvector accuracy (10^{-7} relative tolerance in the next Section). Situations can arise, however, where (111) could

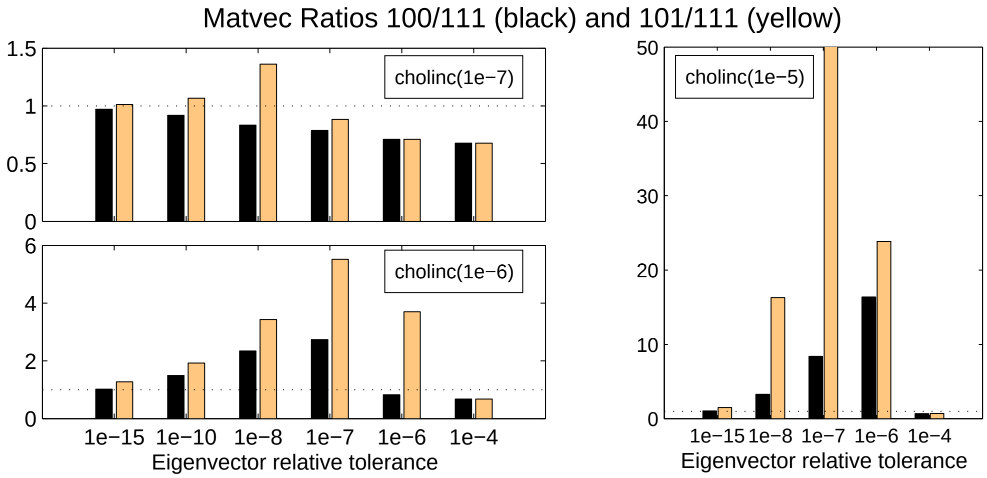

FIG. 3.1. *We look for 20 lowest eigenpairs of the matrix PLAT362 from Harwell Boeing. We show the ratios of the number of matrix-vector products (including the preconditioner application) of method (100) over (111) (black bars), and (101) over (111) (yellow bars). The three boxes depict results from preconditioning with the Matlab* `cholinc`, *with thresholds* `1e-7, 1e-6, 1e-5` *respectively. In each box, the x-axis varies the relative tolerance for computed eigenvectors, i.e., the residual of Q is less than $\|A\|_F tol$. The example was picked because it shows two possible anomalies. First, decreasing both eigenvector accuracy (around $\sqrt{\epsilon_{machine}}$) and the quality of the preconditioner could necessitate the use of (111). Second, case (101) shows a much higher sensitivity than (100) to the accuracy of the eigenvalues. Even in this "anomaly" case, with very good or very bad eigenvector accuracy, method (100) takes less iterations than (111).*

be beneficial. We have been able to identify only one such case. Figure 3.1 shows the results from this "anomaly" case. Even so, method (100) is still competitive and often better than (111) except for a small range of carefully picked parameters. The figure also demonstrates the disadvantages of method (101).

4. Modeling the relative asymptotic behavior between methods. The challenge in relative modeling is that practical models must include information about time complexity, type of operations and their performance on certain hardware, cost of the operators (matvec and preconditioner), but most importantly they must capture the relative convergence behavior of the methods. Insight on the latter arises from lengthy experimentation with the methods on a variety of problems. In the following, we do not separate the experimentation from the modeling process as the two are complementary. Instead, for each pair of methods, we first model their observed relative convergence behavior. Then, we couple it with time complexity models to predict their relative asymptotic behavior for large *nev*. Because the meaning of iteration differs among methods, we normalize all models as cost per operator application (that includes matvec and preconditioner, if available).

4.1. Experimentation environment. In our experiments we use actual production codes, not prototypes, that implement state-of-the-art methods. Our GD+k and JDQMR methods are implemented in C, in the package PRIMME: PReconditioned Iterative MultiMethod Eigensolver [34]. From various methods available in PRIMME, we compare only GD+k and the following JDQMR variants (following notation from the previous section): JDQMR-000, JDQMR-100, JDQMR-011, JDQMR-111. With the exception of forcing these projector configurations, the default param-

TABLE 4.1
The matrices used in the experiments, their size, nonzero elements per row, and their source.

Matrix	N	nz/row	Source	Matrix	N	nz/row	Source
torsion1	40000	4.94	UF	Cone_A	22032	65.04	FEAP
Andrews	60000	12.67	UF	Plate33K_A0	39366	23.22	FEAP
cfd1	70656	25.84	UF	Lap7pt15K	12167	6.74	SKIT
finan512	74752	7.99	UF	Lap7pt125K	110592	6.88	SKIT

eters provided by PRIMME are used in all the experiments. These defaults include the use of locking, block size of 1, $m_{min} = 6$, $m_{max} = 18$, k=1 with preconditioning and k=2 without it. The methods converge when the residual norm of each of the nev required eigenpairs is less than $\|A\|_F tol$, where $\|A\|_F$ is the Frobenius norm of A.

The only other symmetric Jacobi-Davidson implementation available is JDBSYM [17]. It is also written in C, it is a block method, but it stops the inner iteration using the scheme proposed in [51, 12]. JDBSYM provides several choices of inner iterative methods. We opt for the symmetric QMR to facilitate a comparison with JDQMR. We use the same m_{min}, m_{max} for the JD basis, block size of 1, a maximum number of 200 inner iterations, TOLDECAY = 1.5, symmetric preconditioning OPTYPE, and strategy = 0. In certain cases strategy = 1 was necessary to achieve convergence. JDBSYM finds only eigenvalues closest to a shift τ, not extreme ones. In all our tests, we provide τ as a small, left perturbation of the precomputed λ_1 and let JDBSYM switch to using the Ritz values as shifts when EPS_TR $= 10^{-3}\|A\|_F/\sqrt{N}$. Convergence is declared when all residual norms fall below $\|A\|_F tol$.

From the class of Lanczos methods we use the ARPACK software [33]. The basis size for ARPACK is chosen as $\max(40, 2nev)$ for $nev < 400$ and $1.5nev$ for $nev \geq 400$. The tolerance tol is provided directly to ARPACK.

Finally, we compare against BLOPEX, a C implementation of LOBPCG [27]. We chose to implement a wrapper around BLOPEX(b) that uses locking to compute nev eigenvalues a block, b, at a time. After some experimentation, we found $b = 10$ to be the best choice for most problems. For large nev, BLOPEX(nev) was several times slower than BLOPEX(10). We ask for convergence tolerance of $\|A\|_F tol$.

All methods start with the same random initial guess. We have run experiments for two different tolerances: tol = 1e-15 and tol = 1e-7. For BLOPEX we only report results for tol = 1e-7, as it could not produce results with the lower tolerance. We run experiments on an Apple G5 with 1 GB of memory and two 2GHz processors, each with 512 MB L2 cache. The C codes are compiled using the gcc-4.0.0 compiler with -O3 flag, and the ARPACK is compiled with the g77 compiler. We link with the Apple vecLib library that includes optimized versions of BLAS/LAPACK libraries.

We use eight matrix problems, six from the University of Florida [11] and the FEAP [3] collections, and two standard 7-point 3D Laplacian matrices generated by SPARSKIT [47] with zero Dirichlet boundary conditions. The matrix sizes are far beyond toy problems, yet they make finding 100s or 1000s eigenvalues tractable on a workstation. They also represent different applications, operators, and sparsities. The smallest side of the spectrum is hard to obtain for all matrices, while the largest side is easier for several of them. Finally, Table 4.2 provides a look-up reference for the parameters used in our modeling, their meaning, and their typical range of values.

4.2. JDQMR-000 vs ARPACK. Because the size of the ARPACK basis increases with nev so does its effectiveness. Thus, the number of matrix vector opera-

TABLE 4.2
The parameters used to model the various methods, their meaning, and range of values.

Name	Description	Value
$OP(OP_p)$	Cost of the matvec (matvec+preconditioner)	
c_a	Arpack iteration cost per eigenvalue	$14N$
c_g	Cost of reorthogonalizing locked vectors in GD	$4N$
c_p	Cost of projecting the locked vectors in QMR	$2N$
c_q	Symmetric QMR cost per step	$21N$
c_{qr}, c_{gr}	Outer iteration costs for JDQMR/GD+1	(see Section 4.5)
C	$MV_{jdqmr000}/(MV_{arp}nev)$	$[0.1,\ 0.25]$
f	$f = c_g/c_a$	$4/14$
f_p	$f_p = MV_{jdqmr100}/MV_{jdqmr000}$	
k_{inn}^{xyz}	Number of inner iterations in JDQMR-xyz	> 1
γ	$\gamma = MV_{jdqmr100}/MV_{GD+1}$	$[1.1,\ 2.4]$

tions per eigenvalue found by ARPACK is expected to decrease rapidly with nev. In contrast, the number of matvecs per eigenvalue found for JDQMR-000 is expected to be at least constant or increase slightly for highly interior eigenpairs because of the loss of implicit orthogonality during inner iterations. Although it is difficult to model this decrease/increase of matvecs for the two methods, surprisingly, we can relate empirically the ratio of their number of matvecs. The left graph in Figure 4.1 shows that for sufficiently large nev, the matvec ratio between ARPACK and JDQMR follows a power law. Interestingly, fitting the ratios to a power law curve results in the model:

$$MV_{jdqmr000} = C\ nev\ MV_{arp}, \tag{4.1}$$

with $0.1 < C < 0.25$. The same observations hold for the experiments in Figure 4.2.

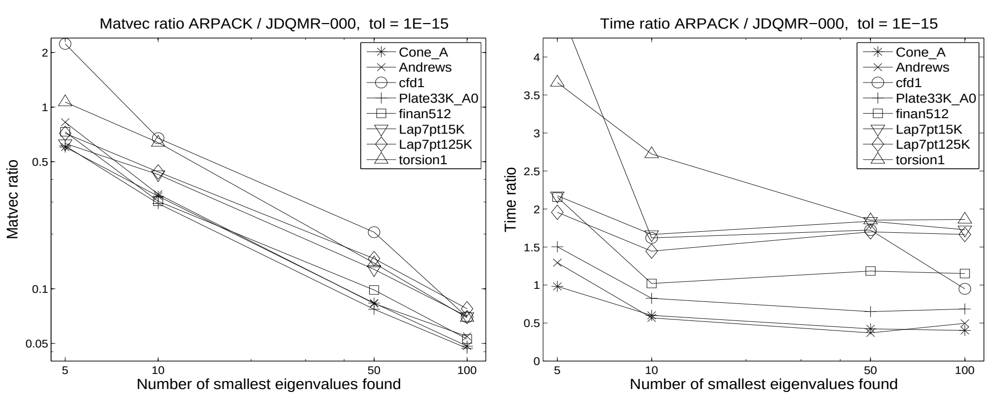

FIG. 4.1. *Relative performance of ARPACK over JDQMR-000 for finding nev smallest eigenvalues of 8 matrices. The left graph shows the matvec ratios. The right graph shows time ratios.*

ARPACK obtains this excellent asymptotic convergence with nev, at the expense of increasing orthogonalization and restarting costs. Based solely on flop counts as a measure of performance and considering only higher order terms and their constants, we have that between restarts ARPACK requires $\sum_{k=nev}^{2nev} 8kN = 3/2nev^2 8N$ flops for (re-)orthogonalization and $2nev^2N$ flops for restarting. Averaging over the nev steps between restarts, we obtain $14nevN$ flops per matrix-vector operation. Let $c_a = 14N$

be the constant capturing these costs independently from nev. This facilitates a more general model, as the constants in the flop counts may change depending on implementation and hardware platform. Letting OP denote the cost of the matrix vector operator, the approximate cost function per step for ARPACK is:

$$ArpPerMV = c_a\ nev + OP.$$

For JDQMR-000 the cost of the inner iteration is that of the QMR method,

$$InnerPerMV = c_q + OP.$$

If flop count is used as a performance measure, $c_q = 21N$. The outer JD step, which includes orthogonalization against all $l = 1, \ldots, nev$ converged eigenvalues, occurs only every k_{inn} inner iterations. To avoid counting the matvec of the outer step twice, we include it in the cost of the inner method. Then, we have the following approximate cost model per operator application:

$$OuterPerMV = c_g\ nev/k_{inn}.$$

To provide insight on the size of the constant c_g we can use an accounting method to analyze the average cost per step over all $l = 1, \ldots, nev$. Because the outer iteration is an inexact Newton method, each eigenvalue is obtained with roughly the same number of outer iterations. Then, If m_{op} is the total number of matvecs, one eigenvalue is found every $m_{op}/(k_{inn}nev)$ outer iterations. Therefore, the total orthogonalization cost throughout the execution of the method is $m_{op}/(k_{inn}nev)\sum_{l=0}^{nev-1} 8lN = m_{op}4(nev-1)N/k_{inn}$. Averaging over all m_{op} matvecs, we obtain: $c_g = 4N$.

The outer JD step incurs also costs for orthogonalization of the basis, restarting, and residual computation. These can be twice as expensive as the ARPACK costs, if the basis sizes were the same. But with locking, these outer JD costs do not scale with nev so they are not included in c_g. Thus, asymptotically, $f = c_g/c_a = 4/14 < 1$, but f increases for small nev.

Considering the ratio of the overall JDQMR-000 and ARPACK costs and substituting the matvec model from eq. (4.1), we have:

$$r_{arp}^{j000} = \frac{MV_{jdqmr000}(c_q + OP + c_g\ nev/k_{inn})}{MV_{arp}\ (c_a nev + OP)} = \frac{C(c_q + OP)}{c_a + OP/nev} + \frac{Cnev\ c_g/k_{inn}}{c_a + OP/nev}. \tag{4.2}$$

Asymptotically, for large nev and not too dense operators, we have $OP/nev \ll c_a$, and the ratio can be approximated as:

$$r_{arp}^{j000} \approx C\ \frac{c_q + OP}{c_a} + \frac{f\ C\ nev}{k_{inn}}. \tag{4.3}$$

The first summand in eq. (4.3) determines when JDQMR-000 cannot be faster than ARPACK, regardless of k_{inn}, because of too expensive an operator. The first summand, and thus the ratio, is always greater than 1, if $OP > c_a/C - c_q$. In our experiments $C \in [0.1, 0.25]$, *so JDQMR-000 can be faster than ARPACK only if* $OP < 119N$ *or* $OP < 35N$ (for $C = 0.1$ or $C = 0.25$ respectively). Yet, for many important problems stemming from finite difference or finite element discretizations of PDEs the matrices are sufficiently sparse to favor JDQMR-000.

Eq. (4.3) states that even for those sufficiently sparse matrices, *ARPACK will eventually outperform JDQMR-000 for large enough nev.* Initially, when only some

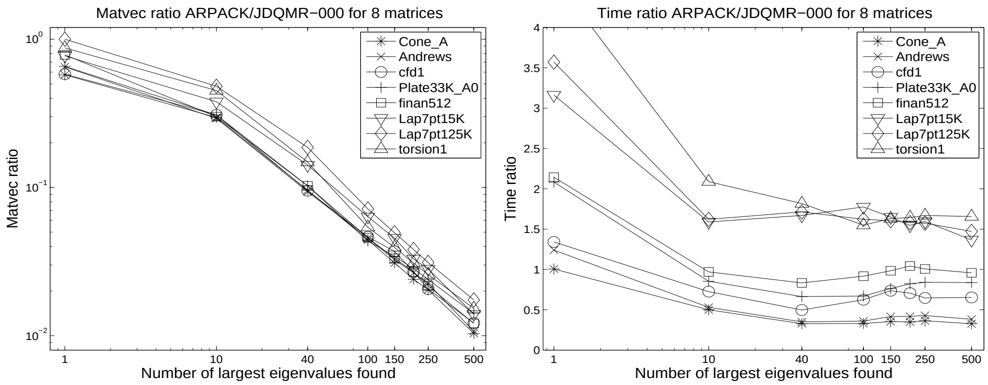

FIG. 4.2. *Relative performance of ARPACK over JDQMR-000 for finding nev largest eigenvalues for tol =1e-15. The left graph shows the matvec ratios. The right graph shows time ratios.*

of the eigenvectors have converged, JDQMR-000 spends most of its time in the inner iteration. As more eigenvectors converge, the orthogonalization of the outer step becomes increasingly expensive and will eventually dominate.

Some intuition can be gained by two extreme examples. First, consider a hypothetical case with the least competitive choice of parameters for JDQMR, $C = 0.25$ and $k_{inn} = 10$. The second summand of eq. (4.3) remains less than 0.8 for $nev < 112$. Second, consider an actual experiment with the Lap7pt15K matrix, as shown in Figure 4.3. This case is favorable for JDQMR-000, but it also demonstrates how well the model captures the relative behavior of the two methods. For this case, we empirically measure $C = 0.1$, $k_{inn} = 50$. Also, because of the 7 point, 3-D Laplacian, the operator cost is $14N$. With these values, the model becomes $r = 0.25 + 0.00057\ nev$, dictating that JDQMR-000 is faster than ARPACK for $nev < 1313$, which is close to the crossover point in the experiment. Moreover, for smaller nev values, the first summand dominates, and this is the reason why in Figures 4.1 and 4.2 the time ratio of the two methods for this matrix looks constant for $40 < nev < 500$.

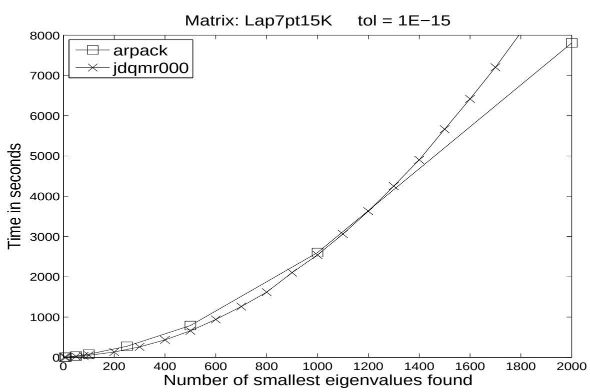

FIG. 4.3. *Time for finding nev smallest eigenvalues for ARPACK and JDQMR-000. Although much faster initially, JDQMR-000 succumbs to orthogonalization costs of the outer step at* 1100.

When seeking only a few eigenvalues, JDQMR-000 is always faster than ARPACK, not so much because it avoids the projectors, but because of the nearly optimal convergence for one eigenpair and the cheaper QMR iteration ($21N$ vs $14N\ nev$). In eq. (4.2) the second summand is negligible if $nev \ll k_{inn}$, and so $r_{arp}^{j000} < 1$ for $nev = 1$, and r_{arp}^{j000} increases monotonically with nev. Then, $r_{arp}^{j000} \to Cnev$ as $OP \to \infty$ so, assuming a worst case $C = .25$, JDQMR-000 should always be faster for $nev < 4$, regardless of operator cost. In practice, this model is not meant to be accurate for small nev, but the intuition it provides agrees with all the experiments we have performed. Moreover, for very high cost operators, GD+k provides a more efficient choice than JDQMR.

In summary, JDQMR-000 is faster than ARPACK for sufficiently sparse matrices and up to a certain nev, which can be $O(100s)$. Beyond that nev, using a larger basis with ARPACK is preferable. For very small nev, JDQMR-000 is always preferable.

4.3. Preconditioned JDQMR-100 vs unpreconditioned JDQMR-000. With a "good" preconditioner, JDQMR-000 can converge fast on the indefinite correction equation, similarly to an exact shift and invert, obviating the projection with locked eigenvectors. With less accurate preconditioners, our results have been inconclusive. JDQMR-000 would sometimes converge fast to many extreme eigenvalues, only to slow down appreciably, and unpredictably, on the next unconverged one. For this reason, we do not consider the preconditioned JDQMR-000 further.

Depending on the preconditioner, JDQMR-100 reduces the number of matvecs over unpreconditioned JDQMR-000 by a factor $f_p < 1$: $MV_{jdqmr100} = f_p MV_{jdqmr000}$. The number of matvecs per eigenvalue found by JDQMR-100 may increase with nev because as the shift in the correction equation moves inside the spectrum, the preconditioner, which usually does not change, may not be equally effective. A commensurate increase in matvecs of the unpreconditioned JDQMR-000, however, keeps the f_p roughly constant for different nev. This is depicted for a sample matrix in Figure 4.4, where JDQMR-100 requires 12 times less matvecs than the unpreconditioned JDQMR-000 for any nev. Moreover, since the number of outer iterations does not vary substantially, we assume that $k_{inn}^{100} = f_p k_{inn}^{000}$.

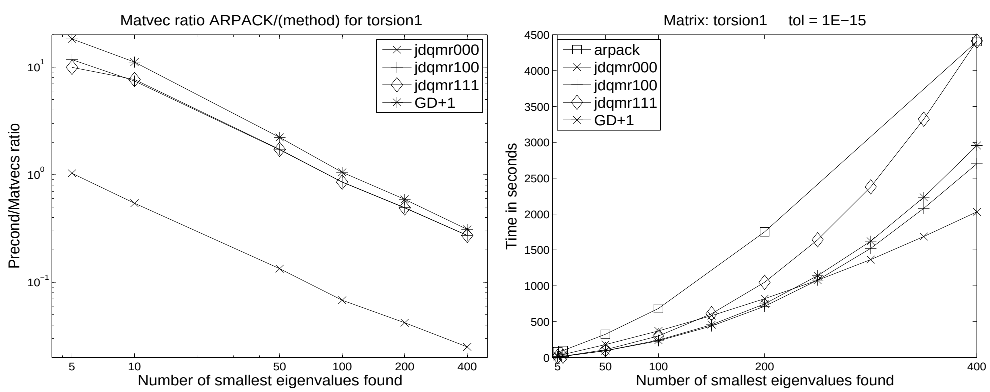

FIG. 4.4. *Matvec ratio of ARPACK over five methods (left graph) and time (right graph) for finding nev smallest eigenvalues. All methods use ILUT(40,1e-3) preconditioning, except for ARPACK and JDQMR-000. Methods that avoid projections are better, even unpreconditioned.*

In addition to the c_q QMR cost, each JDQMR-100 inner iteration involves a projection with locked eigenvectors ($c_p\ nev$) and an operator OP_p which includes matvec and preconditioning. In our flop count model, $c_p = 2N$, which is half of c_g, because we do not re-orthogonalize when applying the projector. Re-orthogonalization against locked eigenvectors occurs at every outer step for a total of $c_g nev$ cost. Hence, we arrive at the model for the time ratio of the two methods:

$$r_{j000}^{j100} = \frac{MV_{jdqmr100}\ (c_q + OP_p + c_p nev + c_g\ nev/k_{inn}^{100})}{MV_{jdqmr000}\ (c_q + OP + c_g\ nev/k_{inn}^{000})}$$

$$(4.4) \qquad = \frac{f_p(c_q + OP_p + c_p nev) + c_g\ nev/k_{inn}^{000}}{c_q + OP + c_g\ nev/k_{inn}^{000}}$$

For small nev and for preconditioners that are not disproportionally expensive, the ratio is close to f_p, and JDQMR-100 delivers the benefits of preconditioning. For large nev, orthogonalization becomes dominant: $r_{j000}^{j100} \to f_p + f_p k_{inn}^{000}/2 = f_p + k_{inn}^{100}/2$, so since $k_{inn}^{100} > 1$, *JDQMR-000 is always better asymptotically than JDQMR-100.*

Intuitively, the more effective and relatively cheaper the preconditioner, the more eigenvalues JDQMR-100 finds faster before JDQMR-000 becomes beneficial. Figure 4.4 shows an example of this behavior for the torsion1 matrix.

4.4. Preconditioned JDQMR-100 vs ARPACK. Multiplying eq. (4.2) with eq. (4.4) we obtain a time ratio for JDQMR-100 over ARPACK as follows:

$$(4.5) \qquad r_{arp}^{j100} = Cf_p\ \frac{c_q + OP_p}{c_a} + Cfnev(\frac{f_p}{2} + \frac{1}{k_{inn}^{000}}),$$

where $f = c_g/c_a$ as previously, and OP/nev is negligible. In the example in Figure 4.4, the preconditioned JDQMR-100 has not crossed the ARPACK curve for $nev = 400$, but at that value their growth rates have become similar. For this example, we empirically obtained $f_p = 1/12$, $k_{inn}^{000} = 45$, and $C = 0.1$. Then, according to the model the curves will cross at $nev = 511$, although it is the approximate asymptotic behavior which is of interest here and not the exact crossover point. The curve for the two-projection JDQMR-111 already crosses the ARPACK curve at $nev = 400$.

The last two models suggest that *the benefits of preconditioned Jacobi-Davidson methods do not extend to large numbers of eigenvalues*, for which it may be preferable to switch to unpreconditioned JDQMR-000 (if the matrix operator is inexpensive), or ARPACK (if the matrix operator is expensive). The crossover for that switch, however, depends on the preconditioner and may be a very large nev number.

4.5. JDQMR-100 vs GD+1. We conclude this section with a discussion on the relative performance behavior of GD+1 and JDQMR-100. As we observed in all our experiments and conjectured in [57], the ratio of matvecs $\gamma = MV_{jdqmr1000}/MV_{GD+1}$ is typically $\gamma \in [1, 2.4]$ and usually closer to 1.5. Because of similar costs, to compare the two methods for small nev, we must complement the asymptotic model with the outer JD costs that do not depend on nev (residual computation and restarting).

We use the accounting method of Section 4.2 to extend the flop-count cost model in [57] to $nev > 1$. The remaining costs that do not depend on nev are given for GD+1 by $c_{gr} = (11.4m_{max} + 18)N$, and for the outer iteration of JDQMR-100 by $c_{qr} = (11.4m_{max} - 3)N$. Hence, the time ratio can be modeled as:

$$r_{GD+1}^{j100} = \gamma \frac{(c_q + OP_p + c_p nev + (c_{qr} + c_g\ nev)/k_{inn})}{(c_{gr} + c_g\ nev + OP_p)} \longrightarrow \gamma \left(\frac{1}{2} + \frac{1}{k_{inn}}\right).$$

JDQMR-100 is faster than GD+1 asymptotically if $\gamma \in [1, 2)$ and $k_{inn} > 2\gamma/(2 - \gamma)$, which is the most typical case. For example, for $\gamma = 1.5$ and $k_{inn} = 10$, JDQMR-100 is 10% faster than GD+1 while with $k_{inn} = \infty$ it can be up to 25% faster. If $\gamma > 2$,

Table 4.3

JDQMR-100 vs GD+1: The crossover cost in flops of the combined matrix and preconditioner (OP_p). JDQMR-100 is faster for OP_p costs less than the given values. GD+1 is faster otherwise. For a certain γ and k_{inn} we also provide the crossover as a function of nev.

	k_{inn} :	10	30	∞
$\gamma = 1.5$	$OP_P <$	$(323 + 0.8nev)N$	$(363 + 1.6nev)N$	$(383 + 2nev)N$
	$nev = 10$	$331N$	$379N$	$403N$
	$nev = 100$	$403N$	$523N$	$583N$
$\gamma = 2$	$OP_P <$	$(141 - 0.8nev)N$	$(168 - 0.3nev)N$	$(181 - 8\frac{nev}{k_{inn}})N$
	$nev = 10$	$133N$	$165N$	$181N$
	$nev = 100$	$61N$	$141N$	$181N$

then GD+1 is always faster. It is important to note that in either case, *the methods are away from each other by at most a factor of two.*

This asymptotic performance comparison is only valid for *nev* that are significantly larger than OP_p/N, which includes the cost of both the matrix and the preconditioning operators and thus can be very expensive. Moreover, in the realm of such large *nev*, JDQMR-000 is always better than JDQMR-100, and ARPACK is eventually faster than JDQMR-000, so the asymptotic comparison is less relevant. In Figure 4.4, using an expensive preconditioner, the two methods are very close, with GD+1 winning slightly for $nev < 150$, and JDQMR-100 winning slightly for $nev > 200$.

For small *nev*, the winner is determined by the cost of the operator and the number of inner iterations that JDQMR-100 performs. Considering our flop model for the above constants, including the c_{gr} and c_{qr}, and a typical basis size $m_{max} = 18$, we can obtain *the condition under which JDQMR-100 is faster than GD+1:*

$$OP_p < \frac{1}{\gamma - 1}\left(223.2 - \gamma(21 + 202.2/k_{inn}) + (4 - 2\gamma - 4\gamma/k_{inn})nev\right) N.$$

Table 4.3 shows the cost for OP_p beyond which GD+1 becomes faster than JDQMR-100. Despite the asymptotic behavior, with expensive preconditioners the case for GD+1 is compelling even for 100 eigenvalues. Our experiments in the next section use an ILUT preconditioner with a rather large fill-in, and as expected the GD+1 is the fastest method.

We emphasize that we do not advocate the use of flop-counts in the models. They serve only to obtain qualitative information on the trade-offs between methods. Yet, our models are built on top of much more general cost variables (such as c_g or c_q) and operator costs (OP, OP_p) that depend on the implementation and hardware platform. More importantly, they can be measured not only a priori but also at run-time, thus enabling a dynamic and possibly autonomic method selection that depends on the problem at hand. This is a direction of future research.

5. Comparisons with other methods. In this section we provide further experiments with and without preconditioning, for high and low tolerances, and for both sides of the spectrum. We have already demonstrated the relative asymptotic strengths between GD+k, JDQMR-100 and JDQMR-000. The goals of the following experiments are: (a) to show that by avoiding the oblique projector in JDQMR-100, convergence is almost identical to (and sometimes better than) JDQMR-011 and JDQMR-111, but in significant less time and/or storage, (b) to compare with JDB-SYM and BLOPEX, and (c) to confirm some modeling predictions in various cases.

5.1. Without preconditioning. We look for the smallest 100 eigenvalues, and ask for *tol*=1e-15. In Figure 5.1 the left graph shows the linear scaling of matvecs with the number of eigenvalues found for all JDQMR and GD+k methods, except for a slight increase in JDQMR-000. JDBSYM has problems scaling to many eigenvalues for this matrix. The right graph, shows the quadratic $O(nev^2)$ behavior in methods with projections, while JDQMR-000 achieves (at least up to 100 eigenvalues) what was designed for: almost linear scaling with nev. The matrix, however, has 65 elements per row, yielding an expensive $130N$ operator. Thus, the significant reduction in matvecs from the large ARPACK basis is hard to match. Notice also that JDQMR-100 and JDQMR-011 converge identically.

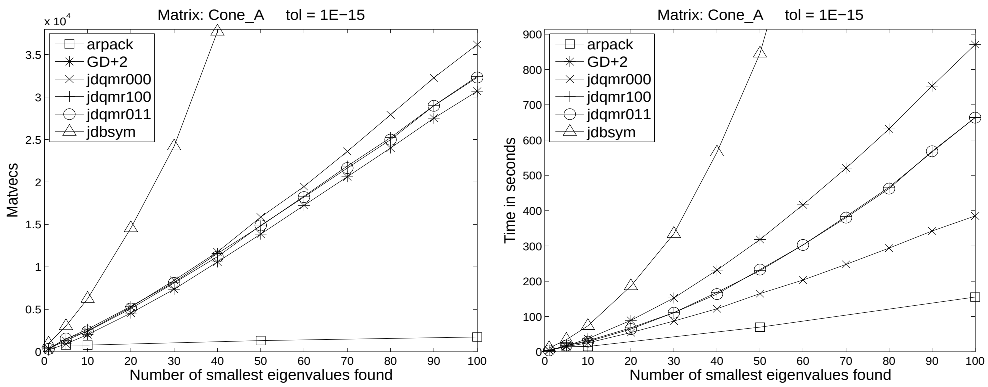

FIG. 5.1. *Matvecs (left graph) and time (right graph) of six methods for nev smallest eigenvalues.*

In Figure 5.2, all JD/GD methods converge very similarly, which means that the stopping criteria of JDBSYM work well in this case. Most methods, and particularly JDQMR-000, are better than ARPACK up to 50 eigenvalues, but for $nev = 100$ the much larger ARPACK basis captures some part of the spectrum that smaller bases could not. ARPACK takes fewer matvecs to find 100 eigenpairs than 50, matching the time of JDQMR-000. This spectrum dependent behavior is hard to model in general.

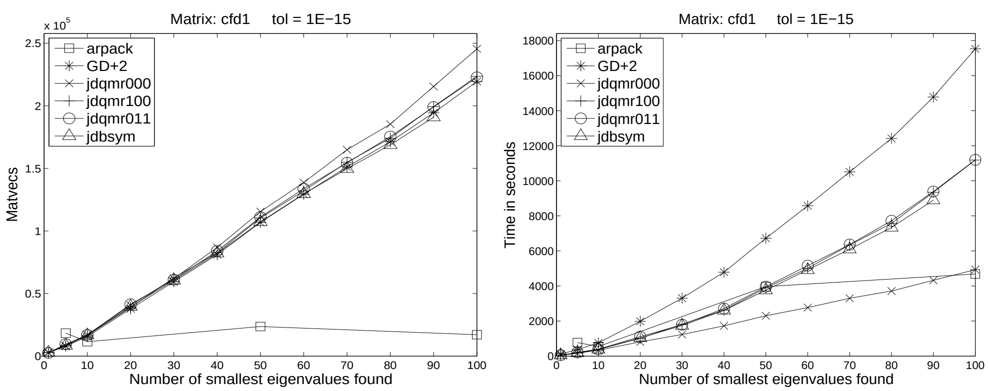

FIG. 5.2. *Matvecs (left graph) and time (right graph) of six methods for nev smallest eigenvalues.*

In Figure 5.3, a sparser matrix, finan512, is considered so JDQMR-000 is several times faster than any other JD/GD method and better than ARPACK, exactly as predicted by our model. Again JDBSYM is only slightly slower than JDQMR-100 and JDQMR-011. The last two are again identical.

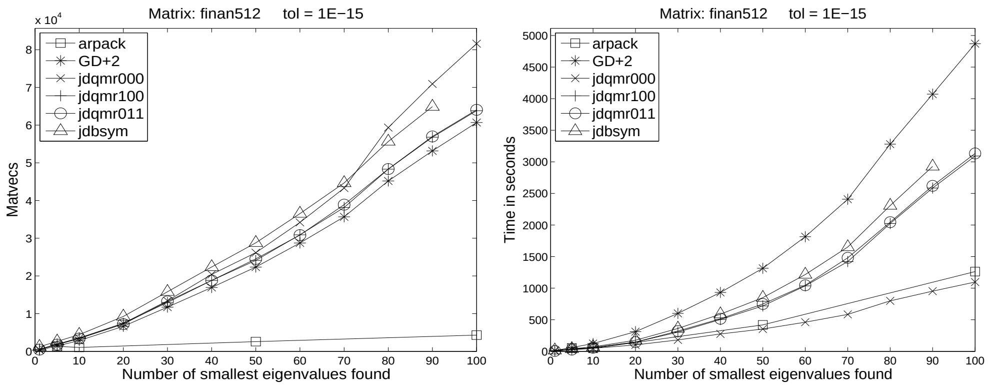

FIG. 5.3. *Matvecs (left graph) and time (right graph) of six methods for nev smallest eigenvalues.*

In Figure 5.4, we consider the largest matrix in the group, whose eigenvalues are all of multiplicity 3 or 6. JDBSYM cannot converge in tractable time for this matrix with strategy = 1, and strategy = 0 performed worse. The sparsity of this Laplacian makes JDQMR-000 significantly faster than all other methods. In all four examples, GD+2 offers the fastest convergence among JD methods in terms of matvecs but not execution time because of its more expensive iteration. Results for 100 smallest eigenvalues from all matrices were summarized in Figure 4.1.

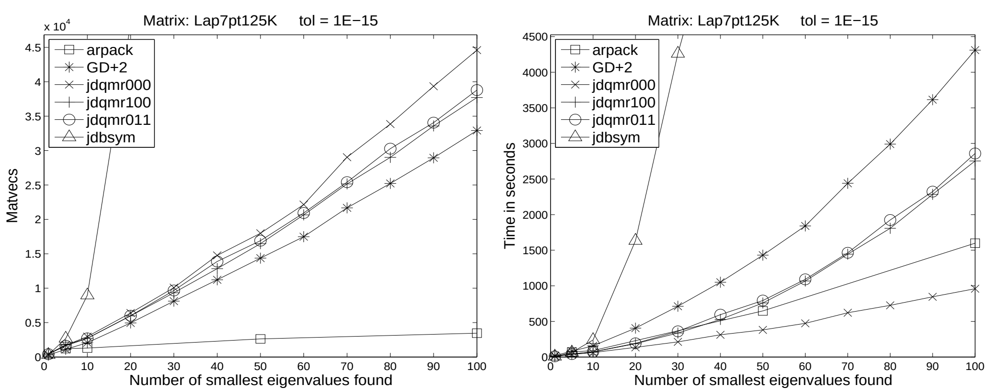

FIG. 5.4. *Matvecs (left graph) and time (right graph) of six methods for nev smallest eigenvalues.*

In the previous examples BLOPEX could not reach the required *tol*. Figure 5.5 shows results from the Cone_A matrix, but with *tol*=1e-7. As with *tol* =1e-15, JDBSYM does not converge well for this matrix. BLOPEX with block size of 10 not only is slower than GD+2/JDQMR methods, but its convergence does not scale linearly with *nev*. For this matrix, the relative behavior of the rest of the methods is similar to *tol* =1e-15 (Figure 5.1).

In Figure 5.6 we observe a significant deterioration of the performance of ARPACK over the *tol* =1e-15 case in Figure 5.2. Closer scrutiny of the iterations between the two figures reveals they are about the same. ARPACK does not benefit from the higher

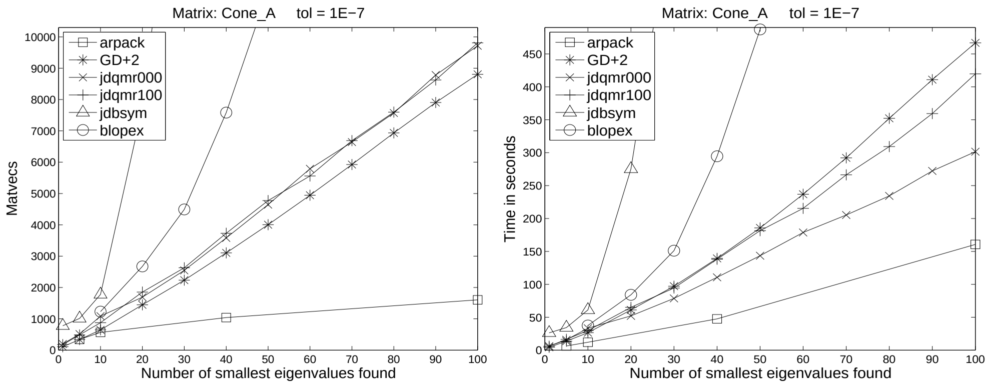

FIG. 5.5. *Matvecs (left) and time (right) of six methods with tol=1e-7, for smallest eigenvalues.*

threshold, still computing almost all 100 eigenpairs to full accuracy. We observed this behavior of ARPACK with high tolerances in the majority of our experiments. Surprisingly, JDBSYM is much slower for *tol*=1e-7 than with full accuracy (compare with Figure 5.2). BLOPEX now scales linearly with *nev*, but it is significantly worse than any GD+k/JDQMR method, even for the smallest 10 eigenpairs.

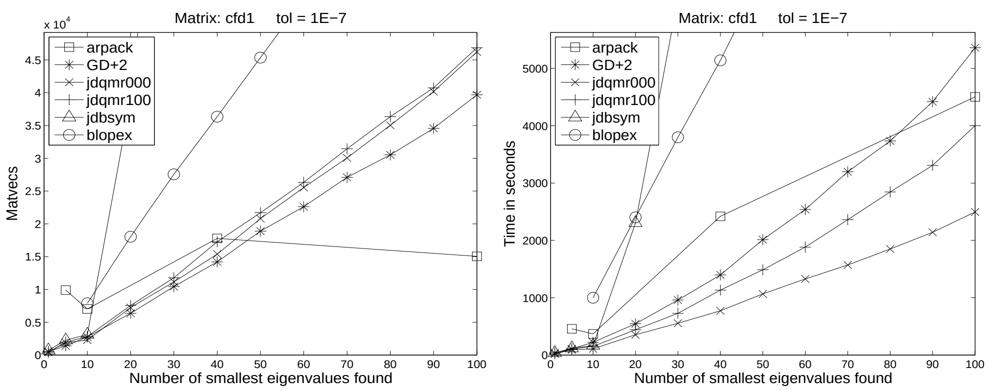

FIG. 5.6. *Matvecs (left) and time (right) of six methods with tol=1e-7, for smallest eigenvalues.*

Figure 5.7 reports similar results in execution time for the finan512 and Lap7pt125K matrices. For the finan512, BLOPEX fails to converge to more than 10 eigenpairs, and JDBSYM is again far slower than the *tol* =1e-15 case. The JDQMR and GD+k methods are consistent both in robustness and their relative behavior.

To see whether BLOPEX would perform better on easier spectra, we run experiments with *tol*=1e-7, seeking 500 largest eigenvalues of the matrices. In Figure 5.8 we give sample results from two matrices. For Plate33K_A0, the largest side of the spectrum is slightly easier to compute, but BLOPEX is still not competitive. For Cone_A, the largest side is much easier. In terms of matvecs (not shown in the figure) BLOPEX is competitive with JDQMR-000 up to 100 eigenvalues, then convergence deteriorates slowly, and the method fails for the 350-th eigenvalue. In terms of time, the orthogonalization penalty is apparent on the BLOPEX method.

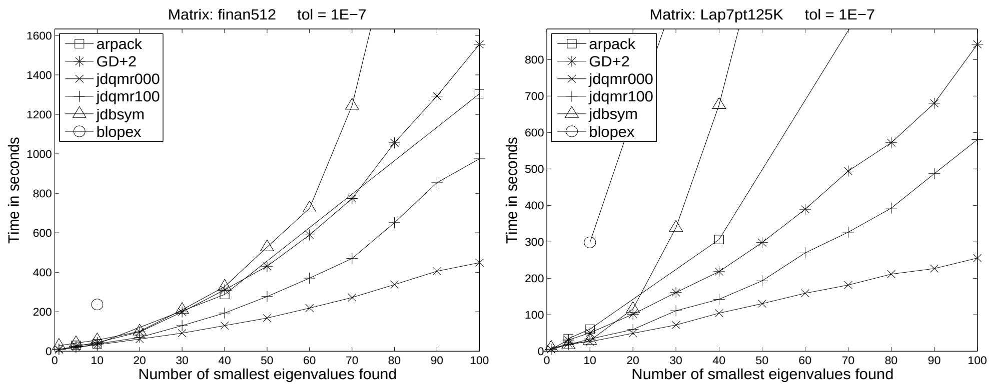

FIG. 5.7. *Execution times of six methods on two matrices with tol=1e-7, for smallest eigenvalues.*

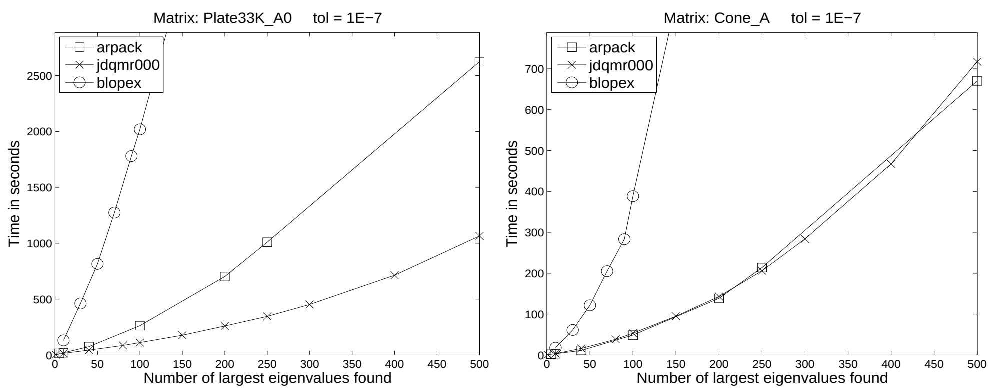

FIG. 5.8. *Times for three methods on two matrices, tol=1e-7, for nev largest eigenvalues.*

5.2. With preconditioning. For our preconditioning experiments, we use the the ILUT preconditioner from the SPARSKIT library [47]. We have observed that although matvecs always decreased with ILUT, for some matrices execution time did not. Next, we report results from the three matrices on which unpreconditioned JDQMR/GD+k methods were slower than the ARPACK method. For these three cases, ILUT resulted in gains. Also, for the ILUT factorization to complete successfully, we had to shift the matrices slightly away from singularity.

In Figure 5.9, the ILUT(80,1e-4) of the shifted cfd1 matrix $A+0.01I$ is computed. All preconditioned methods improve and become much better than ARPACK. All JDQMR-100/111/011 variants converge identically, but the 111 takes more time for large *nev*. JDBSYM also improves but less than JDQMR-100. Because of large fill-in, the ILUT preconditioner is expensive and therefore the method with smallest matvecs wins, i.e., GD+1. The number of matvecs for ARPACK is large and out of scale.

In Figure 5.10, the ILUT(40,1e-6) of the shifted Cone_A matrix $A + 3e6I$ is computed. Although preconditioning reduces matvecs appreciably for JD methods, they still grow linearly with *nev*. As a consequence, the time graph for this experiment shows that ARPACK is faster than any other method beyond $nev = 10$. Observations about the relative behavior of the other methods hold in this example too. Experiments with the Plate33K_A0 matrix have shown similar convergence behaviors and

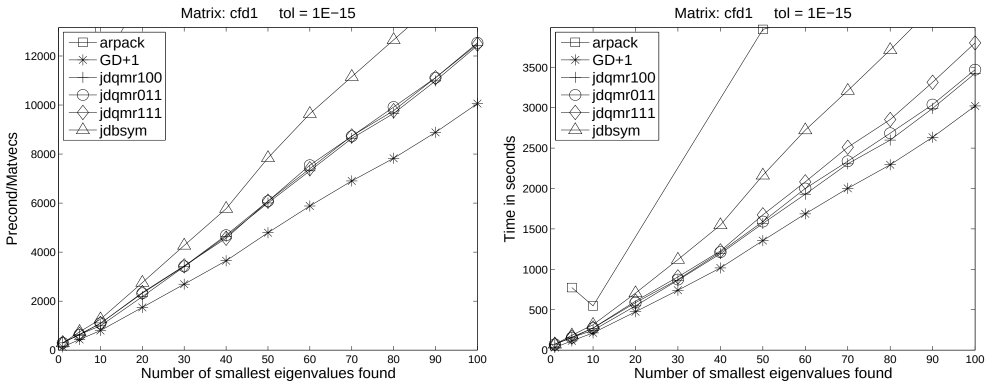

FIG. 5.9. *Matvecs (left) and time (right) with ILUT(80,1e-4) preconditioner. Smallest nev.*

crossover points, so they are not reported.

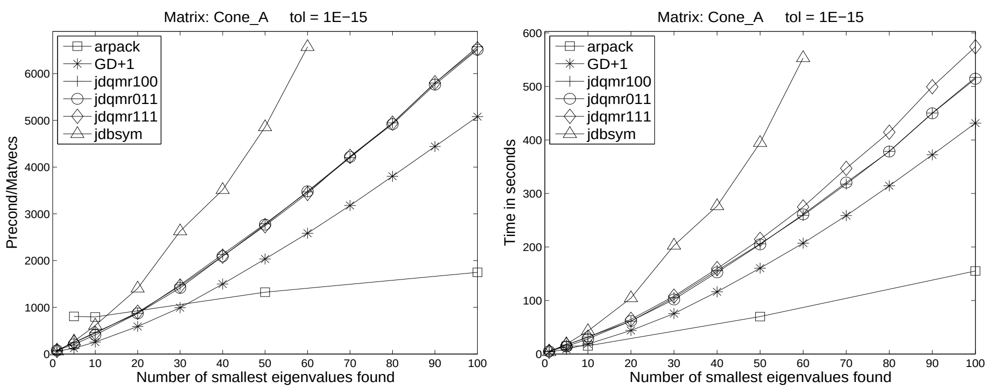

FIG. 5.10. *Matvecs (left) and time (right) with ILUT(40,1e-6) preconditioner. Smallest nev.*

Despite the use of preconditioning, BLOPEX was not able to reach convergence to *tol* =1e-15 in any of the previous cases. Therefore, we conclude this section with three experiments that include preconditioning but use a higher tolerance, 1e-7.

In Figure 5.11, with an ILUT(20,1e-6) of the unshifted Laplacian, neither JDBSYM nor BLOPEX were competitive. The graphs also include the unpreconditioned JDQMR-000, the time of which is identical to preconditioned JDQMR-100.

Figure 5.12 shows timing results from Cone_A with the same preconditioner as before and from the matrix Andrews shifted as $(A-1\text{e-}6I)$. For Cone_A, BLOPEX is marginally faster than JDQMR-100 for 10 eigenvalues, but its convergence deteriorates slowly beyond that and more rapidly after $nev = 40$. JDBSYM converged very slowly for this example. For the Andrews matrix, BLOPEX and JDBSYM are competitive with JDQMR-100 for 10 eigenvalues, but they grow appreciably slower with *nev*, following the curve of the unpreconditioned JDQMR-000. In all our preconditioning experiments, the heavier operators made GD+1 the fastest method.

6. Conclusions. In our research we seek nearly optimal methods for obtaining one or many eigenpairs of Hermitian or real symmetric matrices under limited memory. In an earlier, companion paper we have identified GD+k and JDQMR as two nearly

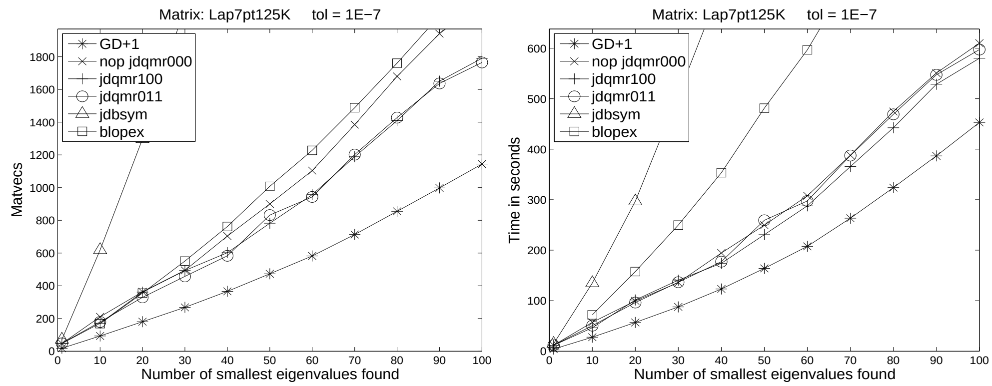

FIG. 5.11. *Matvecs (left) and time (right) with ILUT(20,1e-6). Smallest nev and tol=1e-7.*

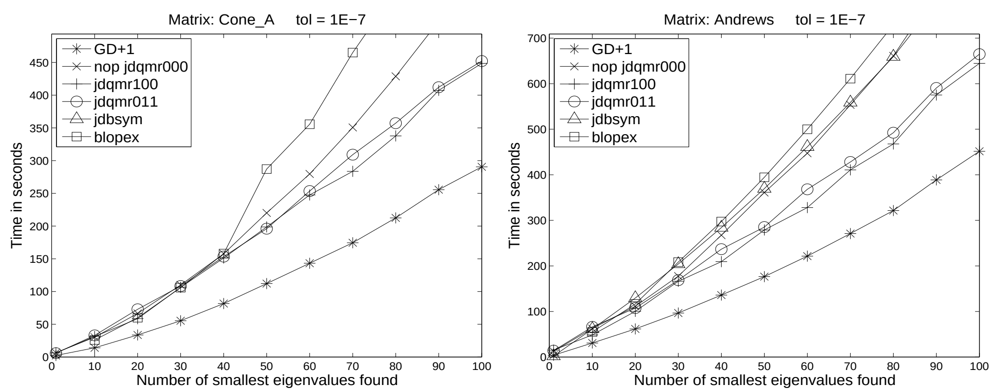

FIG. 5.12. *Times for two matrices, smallest nev, and tol=1e-7. ILUT(40, 1e-6) for both matrices.*

optimal methods with complementary strengths for obtaining one eigenvalue. In this paper, we offer arguments why the same near optimality is not achievable when looking for many eigenpairs.

We have also offered ways to alleviate the quadratic scaling bottleneck stemming from the orthogonalization against converged eigenvectors. Specifically, without preconditioning, our JDQMR-000 method performs no projections during the inner JD iteration, and thus achieves nearly linear scaling with *nev* up to the point where *nev* becomes much larger than the number of inner iterations performed, shifting the bottleneck to the outer step. JDQMR-000 outperforms all other JD/GD variants, and for sparse enough matrices even outperforms ARPACK up to 1000s of eigenvalues. With preconditioning, our analysis suggests that the oblique projection in the JD correction equation is unnecessary for converged eigenvectors. The resulting method, JDQMR-100, is faster and requires half the storage of other JD variants.

Based on a detailed complexity analysis and extensive experimental observations, we have developed a comprehensive set of models that describe the relative performance between JD variants, GD+k, and ARPACK. The asymptotic analysis has provided the following valuable insight: *Methods that use $O(1)$ basis size become slower asymptotically than methods that use $O(nev)$ basis size, regardless of preconditioning.* In practice, $O(1)$ basis sizes with or without preconditioning can be used extremely effectively up to 100s or 1000s of eigenvalues (as our JDQMR-000 shows). Our models provide also performance crossover points between methods that can be used for dynamic, and possibly autonomic method selection within a multimethod software, such as PRIMME.

Finally, we have provided a large set of experiments on a variety of problems and conditions, and compared against state-of-the-art methods and software, such as JDBSYM and BLOPEX. Invariably, our GD+k and JDQMR methods are far more efficient and robust, and seem to be nearly optimal within the class of $O(1)$ basis size.

Acknowledgment. The comments from the referees have contributed to a clearer and more concise presentation. We thank M. Hochstenbach for pointing out some connections with a method in [4].

REFERENCES

[1] P.-A. Absil, C. G. Baker, and K. A. Gallivan. A truncated-CG style method for symmetric generalized eigenvalue problems. *J. Comput. Appl. Math.*, 189(1–2):274–285, 2006.
[2] P.-A. Absil, R. Mahony, R. Sepulchre, and P. Van Dooren. A grassmann-rayleigh quotient iteration for computing invariant subspaces. *SIAM Review*, 44(1):57–73, 2002.
[3] 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.
[4] P. Arbenz, R. Geus, and S. Adam. Solving maxwell eigenvalue problems for accelerating cavities. *Phys. Rev. ST Accel. Beams*, 4:022001, 2001.
[5] P. Arbenz, U. L. Hetmaniuk, R. B. Lehoucq, and R. S. Tuminaro. A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods. *International Journal of Numerical Methods in Engineering*, 64:204–236, 2005.
[6] 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.
[7] Tony F. Chan and W.L. Wan. Analysis of projection methods for solving linear systems with multiple right-hand sides. *SIAM J. Sci. Comput.*, 18:1698–1721, 1997.
[8] J. Cullum and W.E. Donath. A block Lanczos algorithm for computing the q algebraically largest eigenvalues and a corresponding eigenspace of large, sparse, symmetric matrices. In *Proc. 1974 IEEE Conference on Decision and Control*, pages 505–509, 1974.
[9] J. Cullum and R. A. Willoughby. *Lanczos algorithms for large symmetric eigenvalue computations*, volume 2: Programs of *Progress in Scientific Computing; v. 4*. Birkhauser, Boston, 1985.
[10] E. R. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *J. Comput. Phys.*, 17:87–94, 1975.
[11] T. Davis. University of florida sparse matrix collection. Technical report, University of Florida. NA Digest, vol. 92, no. 42, October 16, 1994, NA Digest, vol. 96, no. 28, July 23, 1996, and NA Digest, vol. 97, no. 23, June 7, 1997.
[12] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.
[13] E. G. D'yakonov. Iteration methods in eigenvalue problems. *Math. Notes*, 34:945–953, 1983.
[14] A. Edelman, T. A. Arias, and S. T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2):303–353, 1999.
[15] D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst. Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils. *SIAM J. Sci. Comput.*, 20(1), 1998.
[16] R. W. Freund and N. M. Nachtigal. A new Krylov-subspace method for symmetric indefinite linear systems. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1994.
[17] R. Geus. JDBSYM. http://people.web.psi.ch/geus/software.html.
[18] P. H. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1986.
[19] 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.
[20] G. H. Golub and Q. Ye. Inexact inverse iteration for generalized eigenvalue problems. *BIT*, 40(4):671–684, 2000.
[21] 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.

[22] Martin H. Gutknecht. Block Krylov space solvers: A survey. http://www.sam.math.ethz.ch/ mhg/talks/bkss.pdf.

[23] Martin H. Gutknecht. Block krylov space methods for linear systems with multiple right-hand sides: an introduction. In *Modern Mathematical Models, Methods and Algorithms for Real World Systems*. Anamaya Publishers, New Delhi, India, 2006.

[24] Michiel E. Hochstenbach and Yvan Notay. Controlling inner iterations in the Jacobi–Davidson method. CASA report 07-01, Department of Mathematics, TU Eindhoven, The Netherlands, January 2007. Submitted.

[25] Z. Jia. A refined iterative algorithm based on the block Arnoldi process for large unsymmetric eigenproblems. *Linear Algebra and Its Applications*, 270:171–189, 1998.

[26] M. Kilmer and E. dr Sturler. Recycling subspace information for diffuse optical tomography. *SIAM J. Sci. Comput.*, 27(6):2140–2166, 2006.

[27] A. V. Knyazev. BLOPEX. http://www-math.cudenver.edu/ aknyazev/software/BLOPEX.

[28] A. V. Knyazev. Convergence rate estimates for iteratime methods for symmetric eigenvalue problems and its implementation in a subspace. *International Ser. Numerical Mathematics*, 96:143–154, 1991. Eigenwertaufgaben in Natur- und Ingenieurwissenschaften und ihre numerische Behandlung, Oberwolfach, 1990.

[29] A. V. Knyazev. Preconditioned eigensolvers - an oxymoron? *Electr. Trans. Numer. Anal.*, 7:104–123, 1998.

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

[31] Y.-L. Lai, K.-Y. Lin, and W.-W. Lin. An inexact inverse iteration for large sparse eigenvalue problems. *Num. Lin. Alg. Appl.*, 4:425–437, 1997.

[32] R. B. Lehoucq. Implicitly restarted arnoldi methods and subspace iteration. *SIAM J. Matrix Anal. Appl.*, 23(2):551–562, 2001.

[33] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK User's guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, Philadelphia, PA, 1998.

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

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

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

[37] 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.

[38] R. B. Morgan and D. S. Scott. Preconditioning the Lanczos algorithm for sparse symmetric eigenvalue problems. *SIAM J. Sci. Comput.*, 14:585–593, 1993.

[39] C. W. Murray, S. C. Racine, and E. R. Davidson. Improved algorithms for the lowest eigenvalues and associated eigenvectors of large matrices. *J. Comput. Phys.*, 103(2):382–389, 1992.

[40] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer-Verlag, New York, 1999.

[41] Y. Notay. Combination of Jacobi-Davidson and conjugate gradients for the partial symmetric eigenproblem. *Numerical Linear Algebra with Applications*, 9:21–44, 2002.

[42] Y. Notay. Is Jacobi-Davidson faster than Davidson? *SIAM J. Matrix Anal. Appl.*, 26(2):522–543, 2005.

[43] J. Olsen, P. Jørgensen, and J. Simons. Passing the one-billion limit in full configuration-interaction (FCI) calculations. *Chem. Phys. Lett.*, 169(6):463–472, 1990.

[44] 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.

[45] B. N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, Philadelphia, PA, 1998.

[46] A. Ruhe and T. Wiberg. The method of conjugate gradients used in inverse iteration. *BIT*, 12(4):543–554, 1972.

[47] Y. Saad. SPARSKIT: A basic toolkit for sparse matrix computations. Technical Report 90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990. Software currently available at <ftp://ftp.cs.umn.edu/dept/sparse/>.

[48] A. Sameh and Z. Tong. The trace minimization method for the symmetric generalized eigenvalue problem. *J. Comput. Appl. Math.*, 123:155–175, 2000.

[49] V. Simoncini and L. Eldén. Inexact Rayleigh quotient-type methods for eigenvalue computations. *BIT Numerical Mathematics*, 42(1):159–182, 2002.

[50] V. Simoncini and E. Gallopoulos. An iterative method for nonsymmetric systems with multiple

right-hand side. *SIAM J. Sci. Comput.*, 16(4), 1995.

[51] 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.

[52] P. Smit and M. H. C. Paardekooper. The effects of inexact solvers in algorithms for symmetric eigenvalue problems. *Linear Algebra Appl.*, 287(1-3):337–357, 1999. Special issue celebrating the 60th birthday of Ludwig Elsner.

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

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

[55] A. Stathopoulos, Y. Saad, and C. F. Fischer. Robust preconditioning of large, sparse, symmetric eigenvalue problems. *Journal of Computational and Applied Mathematics*, 64:197–215, 1995.

[56] 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.

[57] Andreas Stathopoulos. Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part I: Seeking one eigenvalue. *SIAM Journal on Scientific Computing*, 29(2):481–514, 2007.

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

[59] K. Wu and H. D. Simon. Thick-restart Lanczos method for symmetric eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 22(2):602–616, 2000.

[60] Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky. Self-Consistent-Field calculations using Chebyshev-filtered subspace iteration. *J. Compu. Phys.*, 219:172–184, 2006.

[61] Yunkai Zhou. Studies on Jacobi-Davidson, Rayleigh quotient iteration, inverse iteration generalized Davidson and Newton updates. *Numerical Linear Algebra with Applications*, 13(8):621–642, 2006.