Technical Report WM-CS-2006-02

College of
William & Mary
Department of Computer Science

WM-CS-2006-02

Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part II: Seeking many eigenvalues

Andreas Stathopoulos

June 2006

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 we 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arguments and on our models we argue that any eigenmethod with $O(1)$ basis size, preconditioned or not, will be superseded asymptotically by Lanczos type methods that use $O(nev)$ vectors in the basis.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 [13, 51, 56, 30, 29, 50, 41, 54], 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. In 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 even these Lanczos properties do not hold. 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, the best way to approach the problem is from the non linearity stemming from both eigenvalue and eigenvector being unknown. In [54], we approached the problem from two non-linear viewpoints: the inexact Newton, and the limited memory quasi-Newton [40]. The two methods we developed, the JDQMR and the GD+k, can be related to the above respective viewpoints, which allowed 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

*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).

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, 54]. 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. 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 a review of the state-of-the-art in preconditioned eigenmethods, 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 provide 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 [35].

Third, we conduct an extensive set of experiments, with and without preconditioning, and compare 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. Current state-of-the-art in eigenmethods.

Throughout this paper we assume we are seeking the *nev* smallest eigenvalues λ_i, and their corresponding eigenvectors $\mathbf{x}_i$, $i = 1,\dots,nev$ of a symmetric matrix A, of dimension N.

2.1. Lanczos based methods.

It should be no surprise that Lanczos is still considered a state-of-the-art method. 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 [59, 45, 21, 8]. For finding one eigenvalue Lanczos methods have now been superseded by Jacobi-Davidson methods, not only because of preconditioning, but also because of nearly optimal convergence that does not require unlimited storage. For many eigenvalues, however, the ability of unrestarted Krylov spaces to capture large parts of the extremes of the spectrum of A is 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 in the order of tens of thousands, and solve for *nev* of its eigenpairs.

In theory, Lanczos can only obtain one copy of a multiple eigenvalue, although in the presence of floating point arithmetic additional ones will appear one after the other [45]. Block versions of Lanczos have been proposed as a robust alternative [7, 19], but they increase the total number of matrix vector operations (matvecs). Still block methods have computational advantages on systems with memory hierarchies. Currently, there is no clear understanding of when and how large a block size to use, although the consensus seems to agree on small values of 2 to 4. For some recent talks with extensive bibliography on block methods see [22].

In the rest of the paper, we do not consider unrestarted Lanczos methods, because they do not satisfy our limited memory criterion, and they cannot use preconditioning. 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. Inversion can be prohibitive for matrices of several million in size, but more importantly, it cannot apply to problems where the operator is given implicitly as a function.

To reduce storage and iteration costs, the Lanczos method can be restarted through the implicit restarting technique [53]. The technique is mathematically equivalent to thick restarting in Davidson type methods [37, 58]. 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/Lanczos methods [33], and it has become the default eigensolver in many applications.

Restarting impairs the optimality of Lanczos convergence. Implicit (or thick) restarting with *nev* (or more) vectors discards less information at restart, thus improving convergence, but it 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 $\mathbf{x}^T A\mathbf{x}/\mathbf{x}^T\mathbf{x}$ [13]. 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 [54], 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 can be applied on the Grassmann manifold (to enforce normalization of the eigenvectors), which is equivalent to classical Rayleigh Quotient Iteration (RQI) [13]. It is well known that when using an inner iterative method to solve the linear system at every step, converging beyond some level of accuracy, increases the overall number of matrix vector multiplications, and hence time. Inexact Newton methods attempt to balance good convergence of the outer Newton method with an inexact solution of the Hessian equation. However, when the linear system in RQI is solved to lower accuracy, RQI ceases to be equivalent to inexact Newton, and it may not converge. This has attracted a lot of attention in the literature [46, 31, 52, 20, 50].

We believe a more appropriate representative of the inexact Newton minimization for the eigenvalue problem is the Jacobi-Davidson method [51]. 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 Newton iterate is then $\mathbf{u}^{(m+1)} = \mathbf{u}^{(m)} + \mathbf{t}^{(m)}$. There are two important differences from RQI. First, 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 methods [9, 38, 57]. The second difference from RQI is that JD applies the pseudoinverse of the Hessian to the residual, which is the gradient of the Rayleigh quotient, not to $\mathbf{u}^{(m)}$. Some theoretical differences between Newton variants are discussed in [1]. Here, we focus on their computational ramifications.

The Generalized Davidson (GD) method obtains the next iterate as $\mathbf{t}^{(m)} = K^{-1}\mathbf{r}^{(m)}$, where the preconditioner K approximates $(A-\eta I)$. Although K can be thought of as an approximate solution to eq. (2.1), we follow prevalent nomenclature, and refer to GD as the application of a given preconditioner to the residual.

JD is 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, 36, 25]. This can be particularly beneficial, especially when looking for more than one eigenpair. Note, however, that without inner iterations, $\mathbf{t}^{(m)} = -\mathbf{r}^{(m)}$, and JD becomes subspace accelerated steepest descent, or equivalently the Lanczos method. With restarting and no inner iterations, JD is mathematically equivalent to IRL.

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 eigenresidual was monitored inexpensively through a scalar recurrence. In [54], we proposed JDQMR that extends JDCG by using symmetric QMR [15] as the inner method. The advantages are: (a) The smooth convergence of sQMR allows for a set of 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 Grassman manifold to compute directly the invariant subspace (see [48] and [2] for a more recent review). Practically, however, the Grassman 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 [49, 5, 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.

A block JD method, however, is not required for $nev > 1$. In our experience with block JDQMR and block JDBSYM [16], the single vector versions outperform their block counterparts both in execution time and matvecs. However, single vector JD methods may converge to the required eigenvalues out of order, and thus are prone to misconvergence. In some occasions, a small block size was required for JDBSYM to converge in the presence of exact multiplicities. Our experience agrees with the prevailing practice of not using block methods, except for robustness.

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, 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 on the Grassman manifold, which has given rise to many NLCG eigenmethod variants [13]. However, 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) [12, 28], and seems to consistently outperform other NLCG type methods. For numerical stability, the basis can be kept orthonormal, or $\mathbf{u}^{(m)} - \mathbf{u}^{(m-1)}$ can be used instead of $\mathbf{u}^{(m-1)}$. The latter, when used with multivectors $\mathbf{u}^{(m)}, \mathbf{r}^{(m)}$ is the well known LOBPCG method [30].

Because of the non-linearity of the eigenproblem, neither NLCG nor LOBPCG can be optimal. Quasi-Newton methods use the NLCG vector iterates to construct incrementally an approximation to the Hessian, and therefore they almost always converge faster than NLCG [18]. In the context of eigenvalue problems, if all the iterates are considered, certain forms of quasi-Newton are equivalent to unrestarted Lanczos. With restarting, however, IRL loses the single important direction ($\mathbf{u}^{(m-1)}$) that offers the excellent convergence to NLCG and LOBPCG. Therefore, the appropriate way to restart methods such as JD, GD and even Lanczos, would be by subspace acceleration of the LOBPCG recurrence. This was first observed in [39] for the Davidson method, although under a different viewpoint. In [56] we offered a theoretical justification, and an efficient implementation that combined this technique with thick restarting for the GD. In [54], we noted the connection of our method, which we call GD+k, to quasi-Newton and in particular to the limited memory BFGS method [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 from step $m-1$: $\mathbf{u}_i^{(m-1)}$. An orthonormal basis for this set of $m_{min}+k$ vectors, which can be computed in negligible time, becomes the restarted basis. A JD+k implementation is identical.

If the GD/JD method is block, with block size b, it is advisable to keep k $\geq b$, to maintain good convergence for all block vectors. Note also that the special case of block GD(b, $3b$)+b is equivalent mathematically to LOBPCG with the same block size. As we showed in [54], convergence of the GD+k is appreciably faster than LOBPCG for one eigenpair, even with small subspace acceleration, and 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. In [54] we argued that most eigenvalue methods can be implemented using the basic iterative framework of GD. Algorithm 2.1 depicts a version of the basic GD+k algorithm for finding *nev* smallest eigenpairs $(\lambda_i, \mathbf{x}_i)$ of a real symmetric matrix, A.

What characterizes Algorithm 2.1 is its flexibility, It allows for complete freedom on how to expand the space, how to extract approximations from it, and how to restart it. The price for this flexibility is that, at every step, it needs to compute eigenresiduals, orthogonalize new vectors against all current ones in the basis V, and maintain a work array for $W = AV$. Algorithm 2.1, as presented here, implements GD+k, with Rayleigh-Ritz and locking to find more than m_{min} eigenpairs. The implementation of +k restarting is shown at steps numbered with decimal points. Note that 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 the thick restarted GD(m_{min}+k, m_{max}). In our extensive experience with this scheme, k=1 or 2 is always sufficient, obviating the use of larger m_{min}, hence being less expensive.

Step (28) is the one differentiating between most eigenmethods. When the algorithm returns $\mathbf{t}^{(m)} = \mathbf{r}^{(m)}$ (and with k=0), it is equivalent to an expensive IRL implementation. However, if the matrix vector operation is costly, the nearly optimal convergence of GD+1 could be preferable. When the preconditioner is applied directly on the residual we have the classical GD+k method. By considering vectors as multivectors, a block GD implementation [34, 55, 17] yields the equivalents of block Lanczos [45], subspace iteration (without preconditioning) [6], and preconditioned subspace iteration [4], while the GD(b,$3b$)+b is a numerically stable implementation of the LOBPCG [30, 54, 24]. Variations such as GD(b, mb)+b are also plausible.

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 the residual. The pseudoinverse of such a preconditioner can be written as:

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

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 [14, 51, 4] 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.

Let $A_{\eta,u^{(m)}}$ denote the projected matrix operator in the correction eq. (2.1), and $K_{u^{(m)}}$ the projected preconditioner. Our JDQMR algorithm uses the GD+k as the underlying outer method, and at step (28) calls the symmetric, right preconditioned QMR with $A_{\eta,u^{(m)}}$, $K_{u^{(m)}}$, 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* $\theta_1 \leq \theta_2 \leq \ldots \leq \theta_m$
(9) $\mathbf{u}^{(m)} = V s_1$, $\theta^{(m)} = \theta_1$, $\mathbf{w}^{(m)} = W s_1$
(10) $\mathbf{r}^{(m)} = \mathbf{w}^{(m)} - \theta^{(m)} \mathbf{u}^{(m)}$
(11) **while** $\|\mathbf{r}^{(m)}\| \leq tol$
(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) $\mathbf{v}_i = V s_{i+1}$, $\mathbf{w}_i = W s_{i+1}$
(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** $m \geq m_{max}$ *then*
(21.0) *Orthogonalize* $s_i^{old},\ i = 1, \ldots, k$ *among themselves and against* $s_i,\ i = 1, \ldots, m_{min}$
(21.1) *Compute* $H_{sub} = s^{oldT} H s^{old}$
(21.2) *Set* $s = [s_1, \ldots, s_{m_{min}}, s_1^{old}, \ldots, s_k^{old}]$
(22) $H = 0$
(23) **for** $i = 2, \ldots, m_{min} + k$
(24) $\mathbf{v}_i = V s_i$, $\mathbf{w}_i = W s_i$, $H_{ii} = \theta_i$
(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) $H(m_{min} + 1 : m_{min} + k, m_{min} + 1 : m_{min} + k) = H_{sub}$
(26.1) $m = m_{min} + k$
(27) **end if**
(28) *Precondition the residual* $\mathbf{t}^{(m)} = Prec(\mathbf{r}^{(m)})$
(29) **end while**

and right hand side $-\mathbf{r}^{(m)}$. Algorithm 2.2 shows our sQMR algorithm. The scalar recurrences for monitoring the eigenvalue residual, and the dynamic stopping criteria are shown at steps numbered with decimal points.

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 the 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.

ALGORITHM 2.2. *Symmetric QMR for JDQMR with adaptive stopping criteria*

Input: $A_{\eta,u^{(m)}}, K_{u^{(m)}}, -\mathbf{r}^{(m)}, maxiter$

Output: t_k

(1) $\mathbf{t}_0^{(m)} = 0, \delta_0 = 0, \mathbf{r}_0 = -\mathbf{r}^{(m)}, \mathbf{d}_0 = K_{u^{(m)}}^{-1}\mathbf{r}_0$

(2) $g_0 = \|\mathbf{r}_0\|, \Theta_0 = 0, \rho_0 = \mathbf{r}_0^T\mathbf{d}_0$

(2.0) $\mathrm{B}_0 = \Delta_0 = \Gamma_0 = \Phi_0 = \Psi_0 = 0$

(3) **if** $(maxiter = 0)$, $\mathbf{t}_0 = \mathbf{d}_0$, **return**

(4) **for** $k = 1, \ldots, maxiter$

(5) $\mathbf{w} = A_{\eta,u^{(m)}}\mathbf{d}_{k-1}$

(6) $\sigma_{k-1} = \mathbf{d}_{k-1}^T\mathbf{w}$, **if** $(\sigma_{k-1} = 0)$, **return**

(7) $\alpha_{k-1} = \frac{\rho_{k-1}}{\sigma_{k-1}}$

(8) $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_{k-1}\mathbf{w}$

(9) $\Theta_k = \frac{\|\mathbf{r}_k\|}{g_{k-1}}, c_k = \frac{1}{\sqrt{1+\Theta_k^2}}, g_k = g_{k-1}\Theta_k c_k$

(10) $\delta_k = (c_k^2\Theta_{k-1}^2)\delta_{k-1} + (c_k^2\alpha_{k-1})\mathbf{d}_{k-1}$

(11) $\mathbf{t}_k = \mathbf{t}_{k-1} + \delta_k$

(12) **if** $(\rho_{k-1} = 0)$, **return**

(12.0) $\gamma_k = c_k^2\Theta_{k-1}^2,\ \xi_k = c_k^2\alpha_{k-1}, f = 1 + \|\mathbf{t}_k\|^2$

(12.1) $\Psi_k = \gamma_k(\Psi_{k-1} + \Phi_{k-1})$

(12.2) $\Phi_k = \gamma_k^2\Phi_{k-1} + \xi_k^2\sigma_{k-1}$

(12.3) $\Gamma_k = \Gamma_{k-1} + 2\Psi_k + \Phi_k$

(12.4) $\Delta_k = \gamma_k\Delta_{k-1} - \xi_k\rho_{k-1}$

(12.5) $\mathrm{B}_k = \mathrm{B}_{k-1} + \Delta_k$

(12.6) $p = \left(\theta^{(m)} - \eta + 2\mathrm{B}_k + \Gamma_k\right)/f$

(12.7) $\theta_k^{(m+1)} = \eta + p$

(12.8) $p_k = (\theta^{(m)} - \eta + \mathrm{B}_k)^2/f - p^2$

(12.9) $r_k^{(m+1)} = \sqrt{g_k^2/f + p_k}$

(12.10) **if** *(*$r_k^{(m+1)}$ *not real),* $r_k^{(m+1)} = \sqrt{g_k^2/f}$

(12.11) **if** $\Big(g_k \le r_k^{(m+1)}\max(0.99\sqrt{f}, \sqrt{g_k/g_{k-1}})$ **or** *(* $\theta_k^{(m+1)} > \theta_{k-1}^{(m+1)}$*)*
or *(*$r_k^{(m+1)} < 0.1 r_0$*)* **or** *(* $g_k < \varepsilon_{inn}$ *)* **or** $(r_k^{(m+1)} < \varepsilon_{inn})\Big)$
then return *the correction* $\mathbf{t}_k$.

(13) $\mathbf{w} = K_{u^{(m)}}^{-1}\mathbf{r}_k, \rho_k = \mathbf{r}_k^T\mathbf{w}, \beta_k = \frac{\rho_k}{\rho_{k-1}}$

(14) $\mathbf{d}_k = \mathbf{w} + \beta_k\mathbf{d}_{k-1}$

(15) **end for**

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,

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}

they could arise. The same problems cannot occur with the converged eigenvectors X if the matrix vector multiplication with A is kept orthogonal to X. 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 for any matrix B the projector:

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

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:

$$PAP(PKP)^+ = PAP_{K^{-1}}K^{-1} \tag{3.3}$$

$$= PAK^{-1}P_{K^{-1}}^T. \tag{3.4}$$

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.

Eqs. (3.3,3.4) suggest 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 $PAK^{-1}P_{K^{-1}}^T$ and $AK^{-1}P_{K^{-1}}^T$ 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 $PAK^{-1}P$.

First, we discuss cases without a 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 $K = I$), the QMR starts with the residual $\mathbf{r}^{(m)} \perp Q$ and thus all its iterates will stay in $Q^\perp$. Floating point arithmetic, and the fact that 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., $tol \geq \|A\|\sqrt{\varepsilon_{\text{machine}}}$). 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 by fixing the concern, as in case (101), it is still not competitive as we show below.

Second, we discuss cases with a P on the left of A, which 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 particular (100) does not require the right projector, making it also 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:

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

□

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):

$$\kappa_{111} = \|PAP(PKP)^{+}\|\|PKP(PAP)^{+}\| = \|PAK^{-1}P^{T}_{K^{-1}}\|\|PKA^{-1}P^{T}_{A^{-1}}\| \tag{3.5}$$

$$\kappa_{101} = \|PAPK^{-1}P\|\|(PK^{-1}P)^{+}(PAP)^{+}\| = \|PAPK^{-1}P\|\|P_KKPA^{-1}P^{T}_{A^{-1}}\| \tag{3.6}$$

$$\kappa_{100} = \|PAK^{-1}P\|\|(PAK^{-1}P)^{+}\| = \|PAK^{-1}P\|\|P_{KA^{-1}}KA^{-1}\|. \tag{3.7}$$

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.

First, we assume that Q are exact eigenvectors, leading to $P_{KA^{-1}} = P_K, P^T_{A^{-1}} = P, PA = AP$, and to the following simplifications:

$$\kappa_{111} = \|AK^{-1}P^T_{K^{-1}}\|\|PKA^{-1}P\| \tag{3.8}$$

$$\kappa_{101} = \kappa_{100} = \|PAK^{-1}P\|\|P_K KA^{-1}\|. \tag{3.9}$$

Equation (3.9) was expected. 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:

$$\begin{aligned}\kappa_{100} &= \|PAK^{-1}P\|\|PKA^{-1}P - R_K(Q^TKQ)^{-1}Q^TKPA^{-1}\| \\ &= \|PAK^{-1}P\|\|PKA^{-1}P - R_K(Q^TKQ)^{-1}R_K^TA^{-1}P\|.\end{aligned} \tag{3.10}$$

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

$$\kappa_{111} = \|PAK^{-1}P - APR_{K^{-1}}(Q^TK^{-1}Q)^{-1}R^T_{K^{-1}}\|\|PKA^{-1}P\|. \tag{3.11}$$

Thus, each norm factor in κ_{100} and κ_{111} differs from each other by a term which is proportional to the square of the residual. Depending on what part of the K spectrum is approximated by Q, and how well, κ_{100} may be better than κ_{111} and vice versa. However, the better the preconditioner is, in the sense of approximating A, the closer the two condition numbers are, as it is suggested by the $O(\|R_K\|^2)$ and $O(\|R_{K^{-1}}\|^2)$ terms. In extensive experiments we have noticed negligible differences between cases (111) and (100).

Second, we consider the extreme case where $K = A$, but Q is inaccurate. In this 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 $R^T_{A^{-1}} = Q^TA^{-1} - (Q^TA^{-1}Q)Q^T$, and consider the norms for κ_{101}:

$$\|PAPK^{-1}P\| = \|P(A - AQQ^T)A^{-1}P\| = \|P(I - (R + QM)Q^TA^{-1}P\| \tag{3.12}$$

$$= \|P - P(R_A + Q(Q^TAQ))(R^T_{A^{-1}} + (Q^TA^{-1}Q)Q^T)P\| \tag{3.13}$$

$$= \|P - R_AR^T_{A^{-1}}\| \tag{3.14}$$

$$\|P_KKPA^{-1}P^T_{A^{-1}}\| = \|P + R_A(Q^TAQ)^{-1}(Q^TA^{-1}Q)^{-1}R^T_{A^{-1}}\| \tag{3.15}$$

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

In the general case, when both Q and K are not accurate, it is hard to quantify the relative merits of κ_{111} and κ_{100}. The results in eqs. (3.10, 3.11) suggest the rather counter intuitive measure of using the expensive method (111) when the preconditioner is not very accurate. In our extensive experiments, we never found this necessary when the eigenvectors were computed accurately. In fact, method (100) consistently gave lower iteration numbers. With decreasing eigenvector accuracy we have found one matrix where case (111) proved beneficial. 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. Predicting the performance of iterative methods, especially relatively to each other, is a challenging task. 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. As modeling and experimentation are complementary in this process, we first describe our experimentation platform, methods, and problems, that will be used in the following sections.

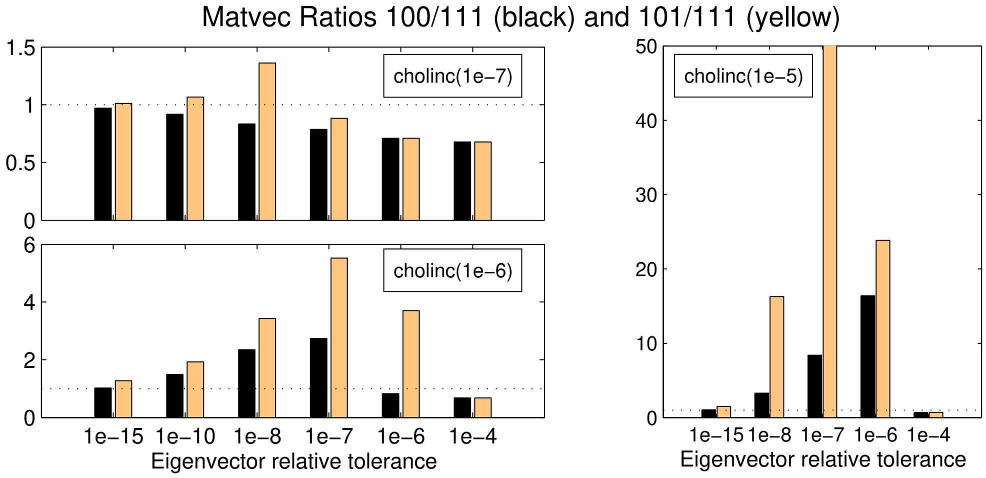

FIG. 3.1. *We look for 20 lowest eigenpairs of the matrix PLAT362 from Harwell Boeing. We show the ratios of the number of matrix-vector products (including the preconditioner application) of method (100) over (111) (black bars), and (101) over (111) (yellow bars). The three boxes depict results from 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* $\sqrt{\varepsilon_{machine}}$*) 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 the (111).*

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 [35]. With appropriate choice of parameters, PRIMME transforms to any eigenvalue method as described in section 2.3. Here, 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 parameters 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 Frobenious norm of A.

The only other symmetric Jacobi-Davidson implementation available is JDBSYM [16]. It is also written in C, it is a block method, but it stops the inner iteration using the scheme proposed in [51, 11]. 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 we 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.

TABLE 4.1
The matrices used in the experiments, their size, non-zero 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

Finally, we compare against BLOPEX, a recent 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 [10] 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.

4.2. JDQMR-000 vs ARPACK. For large *nev*, the number of matrix vector operations per eigenvalue found by ARPACK is expected to decrease rapidly with *nev*, because of the effectiveness of the large Lanczos basis. In contrast, the number of matvecs per new 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 with *nev* 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. Also interestingly, fitting the ratios to a power law curve results in the model:

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

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

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 $\sum_{k=nev}^{2nev} 8kN = 3/2 nev^2 8N$ flops for (re-)orthogonalization, and $2nev^2 N$ 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

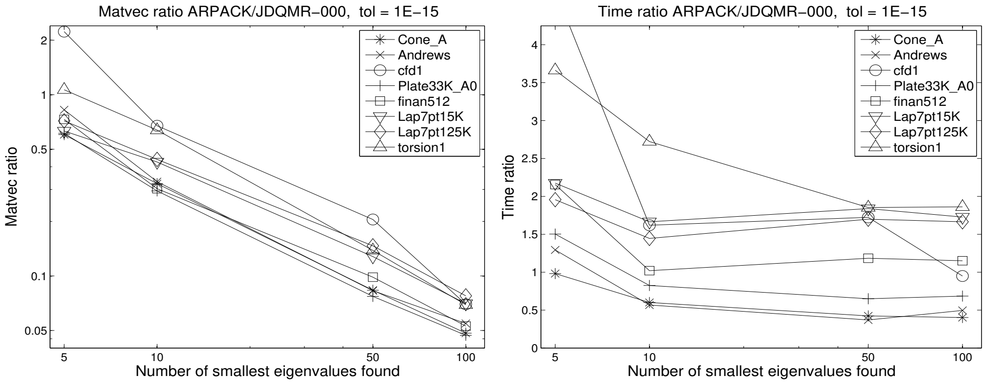

FIG. 4.1. *Relative performance of ARPACK over JDQMR-000 for finding nev smallest eigenvalues of 8 matrices. The left graph shows the matvec ratios. The right graph shows time ratios.*

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 $l = 1,\dots,nev$ 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 $l = 1,\dots,nev$. First note that each eigenvalue is obtained with roughly the same number of outer iterations, because the outer iteration is an inexact Newton method [54]. 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 employed, these outer JD costs do not scale with *nev* so they are not included in c_g. If $f = c_g/c_a$, then $f < 1$ and it only approaches 1 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:

$$r = \frac{MV_{jdqmr}\ (c_q + OP + c_g\ nev/k_{inn})}{MV_{arp}\ (c_a nev + OP)} = \frac{C\ c_q + C\ OP}{c_a + OP/nev} + \frac{C\ nev\ c_g/k_{inn}}{c_a + OP/nev}. \tag{4.2}$$

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

$$r \approx C\ \frac{c_q + OP}{c_a} + \frac{f\ C\ nev}{k_{inn}}. \tag{4.3}$$

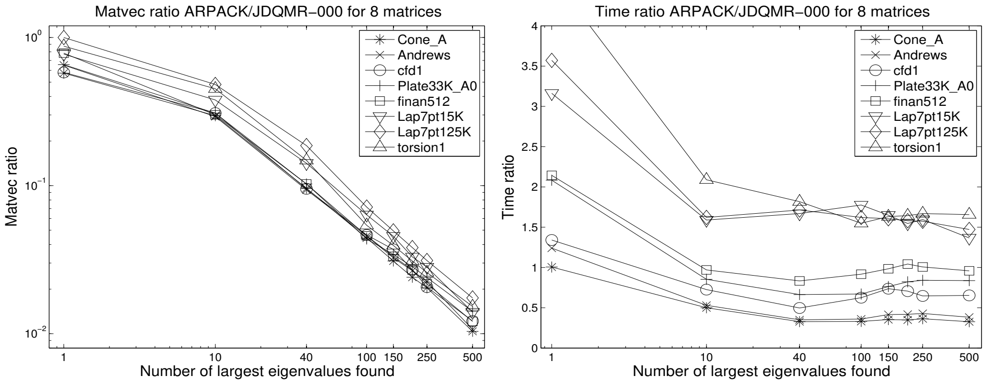

FIG. 4.2. *Relative performance of ARPACK over JDQMR-000 for finding nev largest eigenvalues for tol =1e-15. The left graph shows the matvec ratios. The right graph shows time ratios.*

The first summand in eq. (4.3) describes the dependence of the ratio on the cost of applying the operator. If $OP > c_q(c_a/(c_q\,C)-1)$, the first summand, and thus the ratio, is always greater than 1. In our flop model, $c_q = 21N$ and $c_a/c_q \approx 0.67$. Considering the least competitive of our experiments, i.e., $C = 0.25$, we have that ARPACK is *asymptotically* faster than JDQMR-000 when $OP > 1.68c_q = 35\,N$. Yet, for many sparse matrices stemming from finite difference or finite element discretizations of PDEs, the operator cost is smaller than $35N$. Moreover, C is often 0.2 or .15 allowing operators with cost of $50N$ and $70N$ respectively. For these important cases the first summand will always favor JDQMR-000.

The second summand in eq. (4.3) quantifies the effect of the orthogonalization during the outer iteration. Initially, when only some of the eigenvectors have converged, JDQMR-000 spends most of its time in the inner iteration. As more eigenvectors converge, the outer step becomes increasingly expensive and will eventually dominate. The crossover point where ARPACK becomes better is reached when:

$$nev = k_{inn}/(fC) - k_{inn}(c_q + OP)/(fc_a). \tag{4.4}$$

Some intuition can be gained by a few extreme examples. In the least competitive case, $C = 0.25$, $m_{min} = 10$, and $f = 0.35$. The second summand remains less than 0.8 for $nev < 91$. Figure 4.3 shows an actual experiment with the Lap7pt15K matrix. 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 $C = 0.1$, $m_{min} = 63$. Also, because of the 7 point, 3-D Laplacian, the operator cost is $14N$. With these values, the model becomes $r = 0.25 + 0.00056\ nev$, dictating that JDQMR-000 is faster than ARPACK for $nev < 1350$, 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 $40 < nev < 500$.

When seeking only a few eigenvalues, JDQMR-000 is always faster than ARPACK, not so much because it avoids the projectors, but mainly because of the nearly optimal convergence for one eigenpair, and the cheaper QMR iteration ($21N$ vs $14N\ nev$). First observe that in such cases, the second summand is negligible. Second, consider the first summand without the asymptotic approximation for the denominator:

$$r \approx C\ \frac{c_q + OP}{c_a + OP/nev}.$$

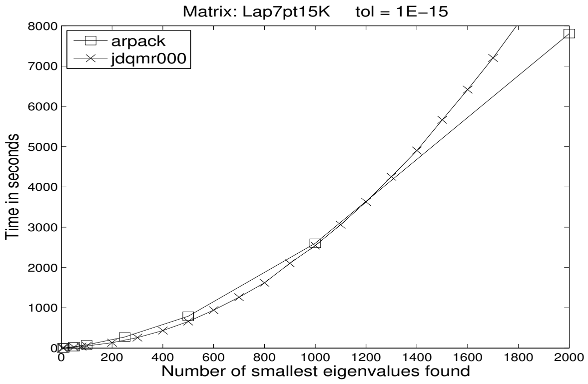

FIG. 4.3. *Time for finding nev smallest eigenvalues for ARPACK and JDQMR-000. Although much faster initially, JDQMR-000 succumbs to orthogonalization costs of the outer step at* 1200

This shows that r is a monotonically increasing function of *nev*, which originally is less than 1. Indeed, $r \to Cnev$ as $OP \to \infty$, so according to the model, JDQMR-000 should always be faster for $nev < 5$, regardless of operator cost. In practice, the model is not accurate for small values *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.

4.3. Preconditioned JDQMR-100 vs unpreconditioned JDQMR-000. The convergence behavior of JDQMR-000 depends highly on the preconditioner. With a "good" preconditioner, QMR can converge fast on the indefinite correction equation, obviating the projection with locked eigenvectors. However, as the shift in the correction equation moves inside the spectrum, the preconditioner, which usually does not change, may not be equally effective for different shifts. In our experiments, we have found instances where preconditioned JDQMR-000 converged fast to many extreme eigenvalues, only to slow down appreciably on the next unconverged one. For this reason, we do not consider the preconditioned JDQMR-000 further.

First, we relate JDQMR-100 to unpreconditioned JDQMR-000. Depending on the preconditioner, JDQMR-100 reduces the number of matvecs over the unpreconditioned JDQMR-000 by a factor $f_p < 1$. Note that the number of matvecs per eigenvalue found may increase with *nev* for JDQMR-100, implying that the preconditioner loses some of its effectiveness inside the spectrum. A commensurate increase in the 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 $k_{inn}^{100} = f_p k_{inn}^{000}$.

Each JDQMR-100 inner iteration involves a projection with locked eigenvectors ($c_p\, nev$), and a more expensive operator (OP_p). 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. However, we should not include the term $c_g nev$ for this, but only $c_p nev$, because an inner projection has already been counted for this iteration. Hence, we arrive at the model for the time ratio of the two methods:

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

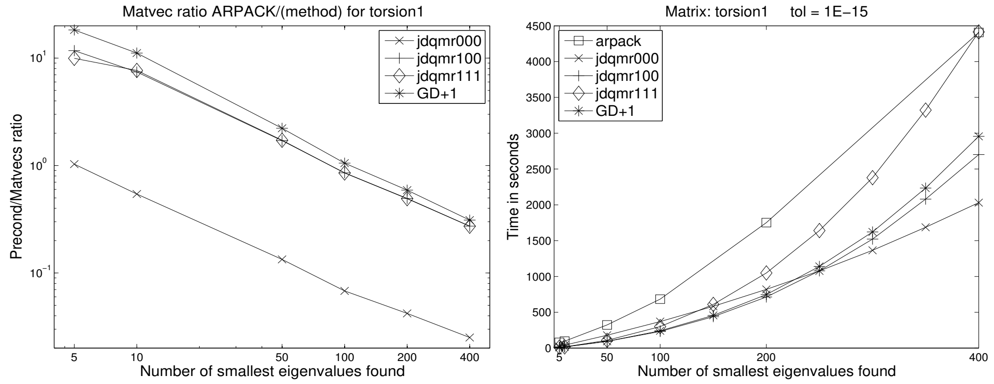

FIG. 4.4. *Matvec ratio of ARPACK over five methods (left graph) and time (right graph) for finding nev smallest eigenvalues. All methods use ILUT(40,1e-3) preconditioning, except for ARPACK and JDQMR-000. Methods that avoid projections are better, even unpreconditioned.*

$$= \frac{f_p(c_q + OP_p) + f_p c_p nev + c_p \; nev/k_{inn}^{000})}{(c_q + OP + c_g \; nev/k_{inn}^{000})}. \tag{4.5}$$

For small *nev* the first part of the nominator dominates the ratio and thus JDQMR-100 delivers the benefits of preconditioning. For large *nev*, the orthogonalization factor $f_p c_p nev$ starts to dominate, and the preconditioned JDQMR-100 becomes increasingly more expensive than the unpreconditioned JDQMR-000. The approximate crossover point is when the asymptotically important terms of the nominator match the denominator, i.e., $f_p c_p nev = c_q + OP + c_p \; nev/k_{inn}^{000}$, or

$$nev = (c_q + OP)/(c_p f_p - c_p/k_{inn}^{000}). \tag{4.6}$$

Intuitively, the more effective the preconditioner and costlier the matrix vector operation, the more eigenvalues JDQMR-100 can find faster. Figure 4.4 shows an example of this behavior for the torsion1 matrix. Using the measured values $OP = 10N, f_P = 1/12$, we compute the crossover point as $nev = 229$, which is very close to the experimental value.

4.4. Preconditioned JDQMR-100 vs ARPACK. Multiplying eq. (4.2) with eq. (4.5) we obtain a time ratio for JDQMR-100 over ARPACK as follows:

$$C f_p \, \frac{c_q + OP_p}{c_a} + C\frac{f}{2} nev(f_p + \frac{1}{k_{inn}^{000}}). \tag{4.7}$$

Setting this ratio to 1 and solving for *nev*, yields the crossover point beyond which ARPACK becomes better:

$$nev = \left(1 - C f_p \, \frac{c_q + OP_p}{c_a}\right) / \left(C\frac{f}{2}(f_p + \frac{1}{k_{inn}^{000}})\right). \tag{4.8}$$

In the example in Figure 4.4, the preconditioned JDQMR-100 has not crossed over the ARPACK curve for $nev = 400$, but at that value their growth rates have become similar. According to the model, the curves will cross at $nev = 565$. The figure also shows the time curve for the two-projection JDQMR-111 method crossing 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

TABLE 4.2
The crossover cost in flops of the matrix and the preconditioner beyond which GD+1 becomes faster than JDQMR-100.

	k_{inn} :	10	30	∞
$\gamma = 1.5$	$nev = 10$	$337N$	$381N$	$403N$
	$nev = 100$	$462N$	$543N$	$584N$
$\gamma = 2$	$nev = 10$	$137N$	$166N$	$181N$
	$nev = 100$	$100N$	$154N$	$181N$

unpreconditioned JDQMR-000 (if the matrix operator is inexpensive), or ARPACK (if the matrix operator is expensive). The crossover for that switch, however, 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 [54], the ratio of the number of matvecs of JDQMR-100 over GD+1 is typically a constant, $\gamma \in [1, 2.4]$, and usually closer to 1.5. Because of their 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* (such as residual computation and restarting).

In [54], we presented an accurate cost model based on flop counts for GD+1 and JDQMR. Using the same accounting method as in section 4.2 we can extended it to $nev > 1$. According to this, the remaining costs for GD+1 are given by $c_{gr} = 11.4 m_{max} N + 18N$, and for the outer iteration of JDQMR-100 are given by $c_{qr} = 11.4 m_{max} N - 3N$. Hence, the time ratio can be modeled as:

$$\begin{aligned} r &= \frac{\text{Time}_{JDQMR100}}{\text{Time}_{GD+1}} = \gamma \frac{(c_q + OP_p + c_p nev + (c_{qr} + c_p\ nev)/k_{inn})}{(c_{gr} + c_g\ nev + OP_p)} \\ &\longrightarrow \gamma \frac{c_p}{c_g}\left(1 + \frac{1}{k_{inn}}\right) = \frac{\gamma}{2}\left(1 + \frac{1}{k_{inn}}\right). \end{aligned}$$

GD+1 is asymptotically faster than JDQMR-100, if $\gamma \geq 2$, or if the number of inner iterations of JDQMR-100 are less than $k_{inn} < \gamma/(2-\gamma)$. In practice, for typical $\gamma \approx 1.5$, JDQMR-100 needs only three inner steps to be asymptotically faster than GD+1. However, to achieve the above limit, *nev* must be 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*, ARPACK could be faster so the asymptotic comparison is less relevant. In Figure 4.4, in the presence of 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:

$$OP_p < \frac{1}{\gamma - 1}\left(223.2 - \gamma(21 + 202.2/k_{inn}) + (4 - 2\gamma - 2\gamma/k_{inn}) nev\right) N. \tag{4.9}$$

Table 4.2 shows the cost for OP_p (i.e., applying both the matrix and the preconditioner) beyond which GD+1 becomes faster than JDQMR-100. 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) 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 JDBSYM 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: 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.

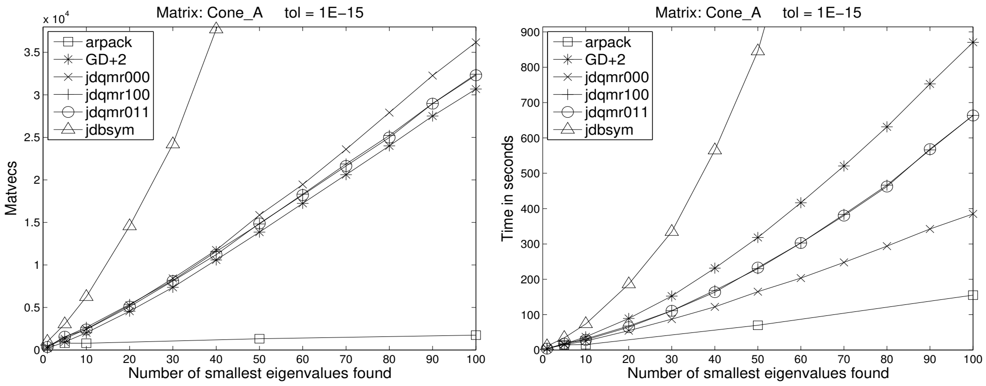

FIG. 5.1. *Matvecs (left graph) and time (right graph) of six methods for nev smallest eigenvalues.*

In Figure 5.2, all JD/GD methods converge very similarly including JDBSYM, which means that its stopping criteria work well in this case. On the other hand, the dynamic criteria in JDQMR work always, giving the method a robustness rivaled only by GD+k. Interestingly, most methods and particularly JDQMR-000 are better than ARPACK up to 50 eigenvalues, but for $nev = 100$, ARPACK uses a much larger basis which captures some part of the spectrum that smaller bases could not. ARPACK takes fewer matvecs to find 100 eigenvalues than 50, and it matches the time of JDQMR-000. This spectrum dependent behavior is hard to model in general.

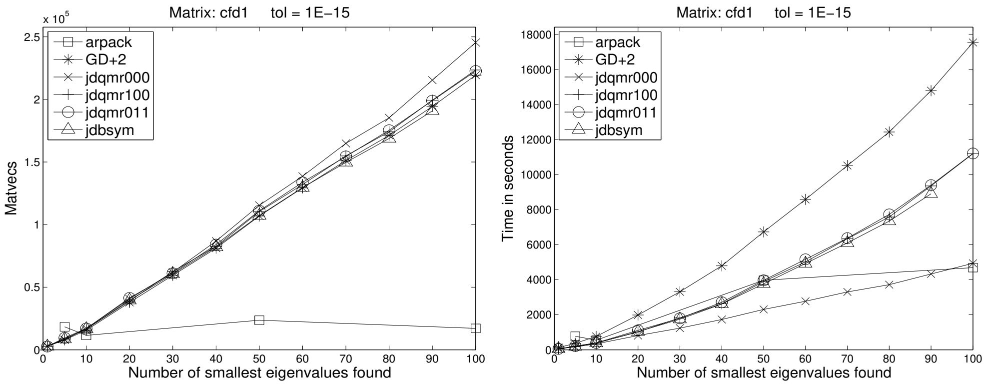

FIG. 5.2. *Matvecs (left graph) and time (right graph) of six methods for nev smallest eigenvalues.*

In Figure 5.3, a sparser matrix, finan512, is considered and the 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.

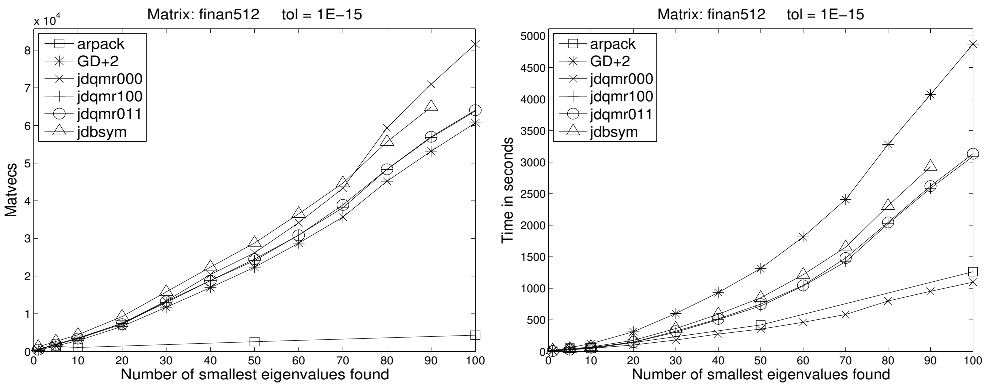

FIG. 5.3. *Matvecs (left graph) and time (right graph) of six methods for nev smallest eigenvalues.*

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. As with finan512, the sparsity of this Laplacian makes JDQMR-000 significantly faster than all other methods. Also, in all four examples, we see GD+2 always taking the minimum number of matvecs among JD methods, yet it loses time-wise because of its more expensive iteration. Results for 100 smallest eigenvalues from all matrices were summarized in Figure 4.1.

In the previous examples BLOPEX could not reach the required tol, hence it is not reported. 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

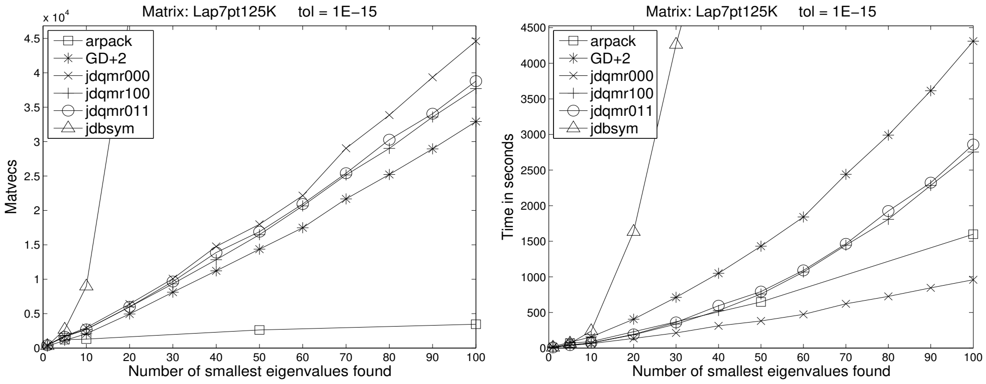

FIG. 5.4. *Matvecs (left graph) and time (right graph) of six methods for nev smallest eigenvalues.*

5.1).

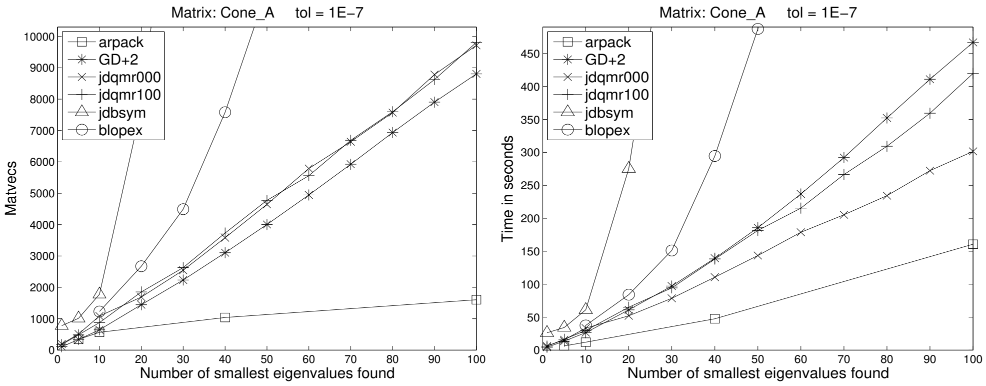

FIG. 5.5. *Matvecs (left) and time (right) of six methods with tol=1e-7, for smallest eigenvalues.*

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 threshold, still computing almost all 100 eigenpairs in 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.

Figures 5.7 and 5.8 report similar results 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 Figures 5.9 and 5.10 we give sample results from three matrices. It turns our that for torsion1 the largest side of the spectrum is even harder then the smallest side. BLOPEX does converge in this case, but it is much slower than both ARPACK and JDQMR-000. Because of the sparsity and the diffi-

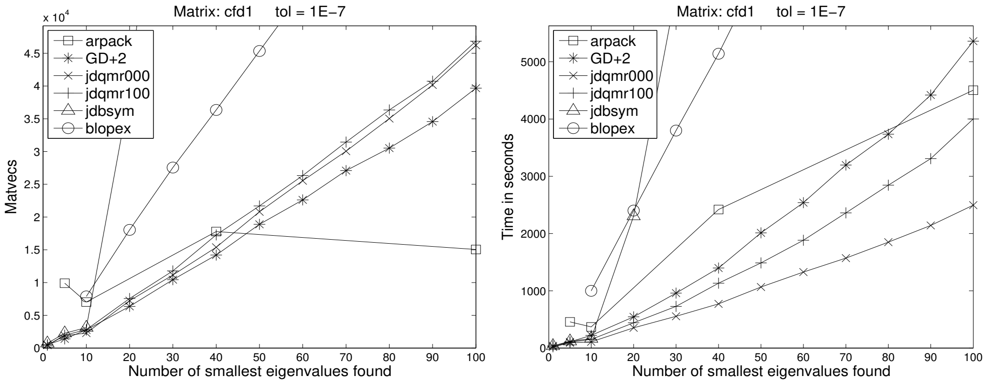

FIG. 5.6. *Matvecs (left) and time (right) of six methods with tol=1e-7, for smallest eigenvalues.*

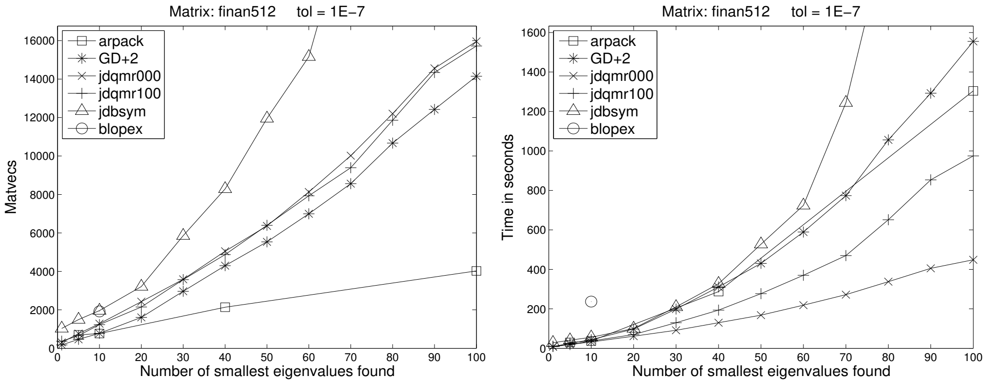

FIG. 5.7. *Matvecs (left) and time (right) of six methods with tol=1e-7, for smallest eigenvalues.*

culty of this case JDQMR-000 will not meet the ARPACK curve for several thousand more eigenvalues. For Plate33K_A0, the largest side is slightly easier to compute, but BLOPEX is still not competitive. For Cone_A, the largest side is much easier. While BLOPEX is quite competitive with JDQMR-000 in terms of matvecs up to 100 eigenvalues, 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.

5.2. With preconditioning. For our preconditioning experiments, we use the the ILUT preconditioner from the SPARSKIT library [47]. Choosing the appropriate preconditioner is matrix-specific and usually an art in itself. A wrong choice could significantly impair the convergence of an iterative method. 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 5.11, 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. Notice that all JDQMR-100/111/011 variants converge identically, but the 111 takes more time for larger *nev*. JDBSYM also improves but not as much as JDQMR-100. Because of large fill-in, the

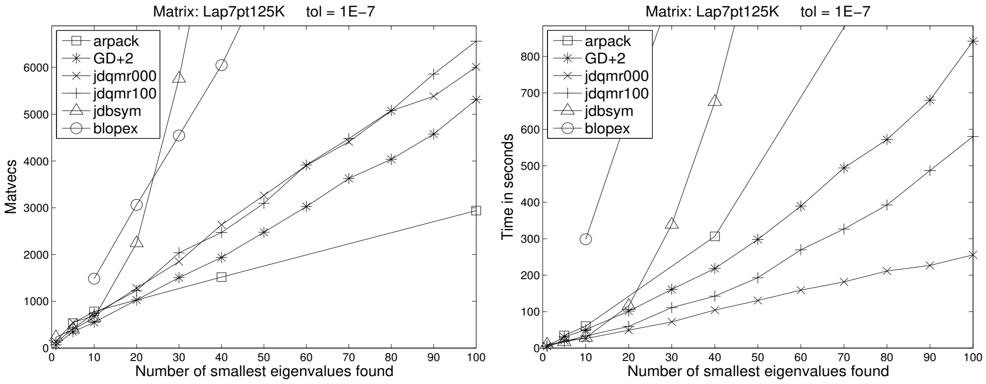

FIG. 5.8. *Matvecs (left) and time (right) of six methods with tol=1e-7, for smallest eigenvalues.*

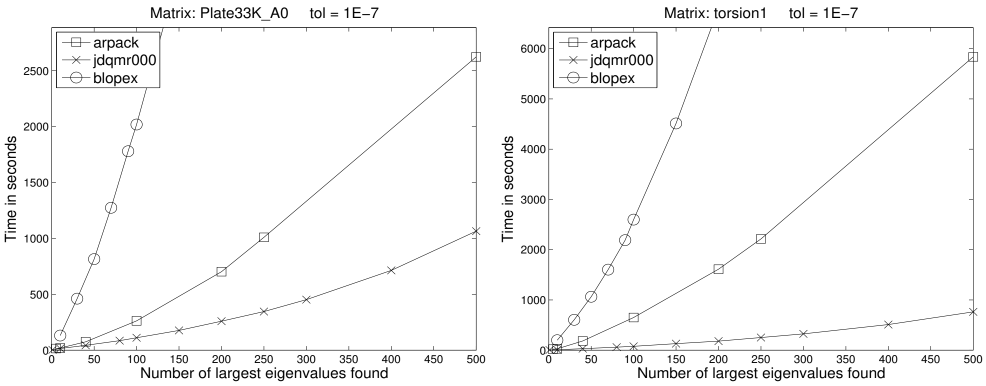

FIG. 5.9. *Times for three methods on two matrices, tol=1e-7, for nev largest eigenvalues.*

computed ILUT factors are rather 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 5.12, 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.

In Figure 5.13, the ILUT(10,1e-6) of the shifted Plate33K_A0 matrix $A+1.5e6I$ is computed. This preconditioner has a smaller fill-in, and while still effective, it is much less expensive. Therefore all methods except for JDQMR-111 converge faster than ARPACK. Interestingly, JDBSYM stagnated before 10 eigenvalues converge.

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

Figure 5.15 shows timing results from Cone_A with the same preconditioner as before, and from the matrix Andrews shifted as ($A-$1e-6I). For Cone_A, BLOPEX is marginally

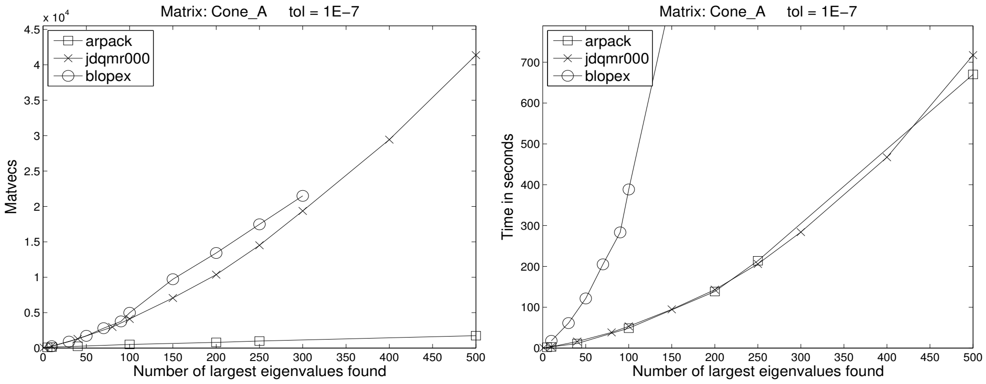

FIG. 5.10. *Matvecs (left) and time (right) for three methods, tol=1e-7, for nev largest eigenvalues.*

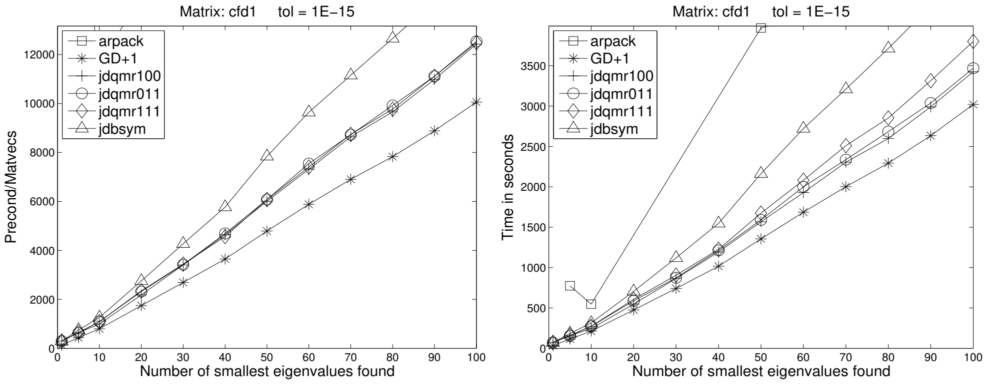

FIG. 5.11. *Matvecs (left) and time (right) with ILUT(80,1e-4) preconditioner. Smallest nev.*

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 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.

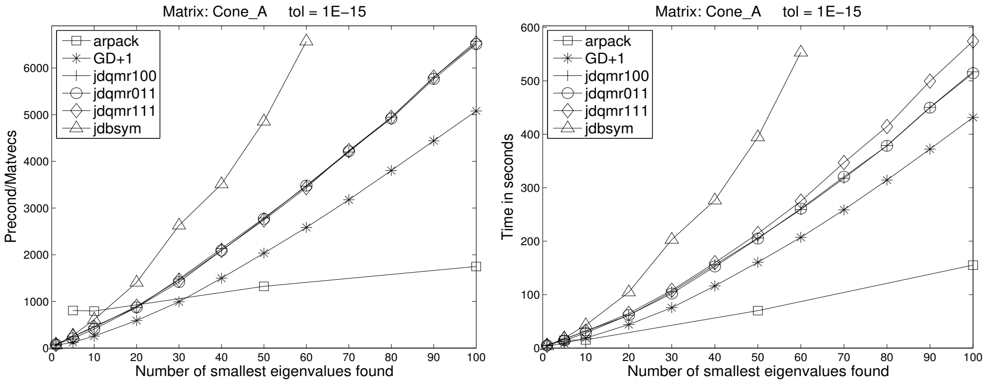

FIG. 5.12. *Matvecs (left) and time (right) with ILUT(40,1e-6) preconditioner. Smallest nev.*

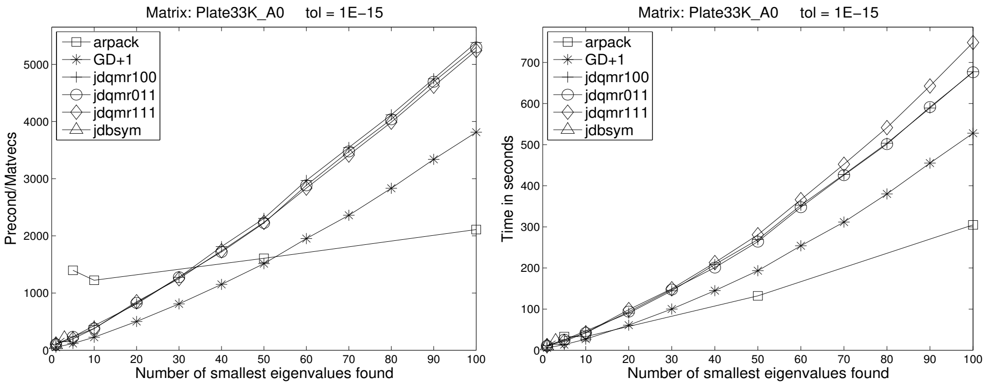

FIG. 5.13. *Matvecs (left) and time (right) with ILUT(10,1e-6) preconditioner. Smallest nev.*

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.

REFERENCES

[1] P.-A. Absil, C. G. Baker, and K. A. Gallivan. A truncated-CG style method for symmetric generalized eigen-

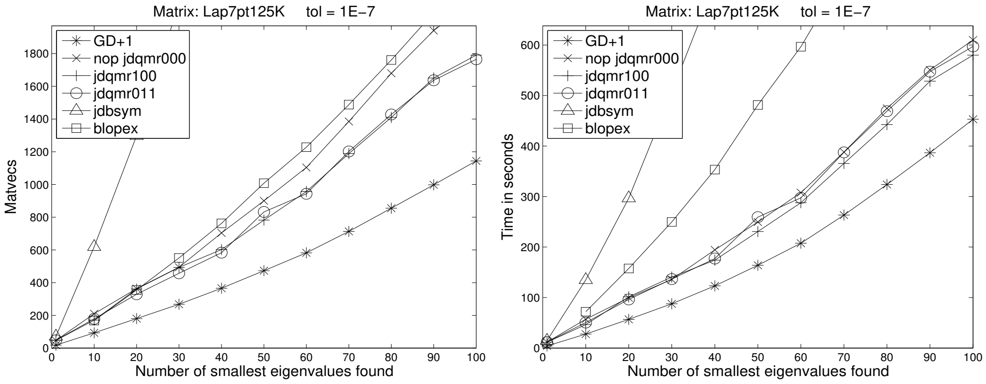

FIG. 5.14. *Matvecs (left) and time (right) with ILUT(20,1e-6). Smallest nev and tol=1e-7.*

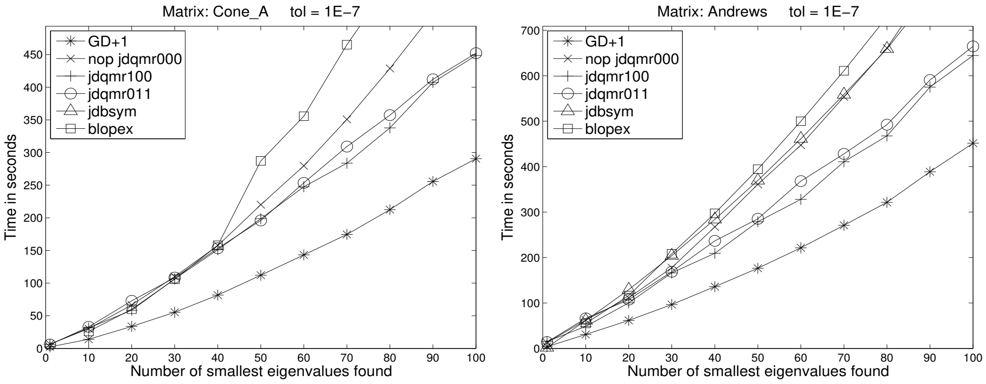

FIG. 5.15. *Times for two matrices, smallest nev, and tol=1e-7. ILUT(40, 1e-6) for both matrices.*

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

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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, 19.

[11] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.

[12] E. G. D'yakonov. Iteration methods in eigenvalue problems. *Math. Notes*, 34:945–953, 1983.

[13] Alan Edelman, Tomás A. Arias, and Steven T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2):303–353, 1999.
[14] D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst. Jacobi-Davidson style QR and QZ algorithms for the partial reduction of matrix pencils. *SIAM J. Sci. Comput.*, 20(1), 1998.
[15] 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.
[16] R. Geus. JDBSYM. http://people.web.psi.ch/geus/software.html.
[17] Roman Geus. *The Jacobi-Davidson algorithm for solving large sparse symmetric eigenvalue problems with application to the design of accelerator cavities*. PhD thesis, ETH, 2002. Thesis. No. 14734.
[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] U. Hetmaniuk and R. B. Lehoucq. Basis selection in LOBPCG.
[25] 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.
[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. 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] Yu-Ling Lai, Kun-Yi Lin, and Wen-Wei 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 USERS GUIDE: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, Philadelphia, PA, 1998.
[34] B. Liu. Numerical algorithms in chemistry: Algebraic methods, eds. c. moler and i. shavitt. Technical Report LBL-8158, Lawrence Berkeley Laboratory, 1978.
[35] J. R. McCombs and A. Stathopoulos. PRIMME: PReconditioned Iterative Multimethod Eigensolver. http://www.cs.wm.edu/ andreas/software/.
[36] R. B. Morgan. Computing interior eigenvalues of large matrices. *Lin. Alg. Appl.*, 154–156:289–309, 1991.
[37] R. B. Morgan. On restarting the Arnoldi method for large nonsymmetric eigenvalue problems. *Math. Comput.*, 65:1213–1230, 1996.
[38] 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.
[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] Yvan Notay. Combination of Jacobi-Davidson and conjugate gradients for the partial symmetric eigenproblem. *Numerical Linear Algebra with Applications*, 9:21–44, 2002.
[42] Yvan Notay. Is Jacobi-Davidson faster than Davidson? *SIAM J. Matrix Anal. Appl.*, 26(2):533–543, 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] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, Philadelphia, PA, 1998.
[46] Axel Ruhe and T. Wiberg. The method of conjugate gradients used in inverse iteration. *BIT*, 12(4):543–554, 1972.

[47] Yousef 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 E. Gallopoulos. An iterative method for nonsymmetric systems with multiple right-hand side. *SIAM J. Sci. Comput.*, 16(4), 1995.

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

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

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

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

[57] 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.

[58] 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.

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