TOWARD THE OPTIMAL PRECONDITIONED EIGENSOLVER: LOCALLY OPTIMAL BLOCK PRECONDITIONED CONJUGATE GRADIENT METHOD

ANDREW V. KNYAZEV*

Abstract. We describe new algorithms of the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) Method for symmetric eigenvalue problems, based on a local optimization of a three-term recurrence. To be able to compare numerically different methods in the class, with different preconditioners, we suggest a common system of model tests, using random preconditioners and initial guesses. As the "ideal" control algorithm, we propose the standard preconditioned conjugate gradient method for finding an eigenvector as an element of the null–space of the corresponding homogeneous system of linear equations under the assumption that the eigenvalue is known. We recommend that every new preconditioned eigensolver be compared with this "ideal" algorithm on our model test problems in terms of the speed of convergence, costs of every iterations and memory requirements. We provide such comparison for our LOBPCG Method. Numerical results establish that our algorithm is practically as efficient as the "ideal" algorithm when the same preconditioner is used in both methods. We also show numerically that the LOBPCG Method provides approximations to first eigenpairs of about the same quality as those by the much more expensive global optimization method on the same generalized block Krylov subspace. Finally, direct numerical comparisons with the Jacobi–Davidson method show that our method is more robust and converges almost two times faster.

Key words. symmetric eigenvalue problems, preconditioning, conjugate gradient methods, the Lanczos method

AMS(MOS) subject classifications. 65F15, 65N25

1. Introduction.

We consider the *real* symmetric case for simplicity. The complex self-adjoint case can be treated similarly.

We assume that an inner product (x, y) of real-valued column vectors x and y is given, or defined by a user. Typically, the standard inner product, $(x, y) = x^T y$, is used. The norm $\|x\|$ is defined using the standard equality $\|x\|^2 = (x, x)$.

We call a real square matrix A *symmetric with respect to the inner product* $(\cdot, \cdot)$, or simply *symmetric*, and write $A = A^*$, if

$$(Ax, y) = (x, Ay), \text{ for arbitrary vectors } x,\ y,$$

and *positive definite with respect to the inner product* $(\cdot, \cdot)$, or simply *positive definite*, and write $A > 0$, if

$$(Ax, x) > 0, \text{ for an arbitrary vector } x \neq 0.$$

When the standard inner product $x^T y$ is used, these definitions become the standard ones for real matrices, e.g., $A^* = A^T$.

We consider a generalized *symmetric definite* eigenvalue problem of the form $(A - \lambda B)x = 0$ with real symmetric n-by-n matrices A and B, assuming that A is positive definite. That describes a regular matrix pencil $A - \lambda B$ with a discrete spectrum (set of eigenvalues λ). It is well known that such generalized eigenvalue problem has all real eigenvalues λ_i and corresponding (right) eigenvectors x_i, satisfying $(A - \lambda_i B)x_i = 0$, can be chosen orthogonal in the following sense:

$$(x_i, Ax_j) = (x_i, Bx_j) = 0,\ i \neq j.$$

Some λ_i may be infinite, in which case we really consider the pencil $\mu A - B$ with corresponding eigenvalues $\mu_i = 0$.

In a more general case, when neither A, no B is positive definite, but real constants α and β are explicitly known such that $\alpha A + \beta B$ is positive definite, the definite pencil $(\alpha A + \beta B) - \lambda B$ should be considered to satisfy our assumptions.

In some applications, the matrix B is simply the identity, $B = I$, and then we have the standard symmetric eigenvalue problem with matrix A, which has n real positive eigenvalues

$$0 < \lambda_{\min} = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n = \lambda_{\max}.$$

*Department of Mathematics, University of Colorado at Denver, P.O. Box 173364, Campus Box 170, Denver, CO 80217-3364. *e-mail*: andrew.knyazev@cudenver.edu *WWW URL*: http://www- math.cudenver.edu/~aknyazev.

In this case, we consider the problem of computing the smallest m eigenvalues λ_1 through λ_m with the desired accuracy, and corresponding eigenvectors. For the eigenvalues that are clustered together, the user may choose to compute the associated *invariant subspace*, i.e. the space spanned by the corresponding eigenvectors, since in this case the individual eigenvectors can be very ill-conditioned, while the invariant subspace may be less so.

In general, when $B \neq I$, the pencil $A-\lambda B$ has n real, some possibly infinite, eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. If B is nonsingular, all eigenvalues are finite. If B is positive semi-definite, all eigenvalues are positive, and we consider the problem of computing the smallest m eigenvalues of the pencil $A-\lambda B$. When B is indefinite, it is convenient to consider the pencil $\mu A - B$ with eigenvalues

$$\mu = \frac{1}{\lambda},\ \mu_{\min} = \mu_n \leq \cdots \leq \mu_1 = \mu_{\max},$$

where we want to compute the largest m eigenvalues, $\mu_1, \ldots \mu_m$. The problem of computing m smallest eigenvalues μ_i can be reformulated as in the previous case by replacing B with $-B$.

An important class of eigenproblems is the class of *mesh eigenproblems*, arising from discretizations of boundary value problems with self-adjoint differential operators of mathematical physics. Such problems appear, e.g., in structural mechanics, where it is usual to call A a *stiffness* matrix, and B a *mass* matrix. A mass matrix is usually positive definite, but in some applications, e.g., in buckling, the matrix B is only nonnegative, or even indefinite, while A is positive definite.

Typical properties of mesh eigenproblems are well known, e.g., [15]. We just want to highlight, that the desired eigenpairs of the matrix pencil $B-\mu A$ are rarely needed with high accuracy as the pencil itself is just an approximation of the original continuous problem and the approximation error may not be small in practice. It means that the typical ratio of the number of iterations and the number of unknowns should be small when a preconditioner is of a reasonable quality. For that reason, we are not interested in the present paper in such properties of eigensolvers, e.g., in super-linear convergence, which could be only observed after large number of steps.

In the rest of the paper, for brevity, we deal with the pencil $B-\mu A$ only. With $B = I$ and $\lambda = 1/\mu$, our results and arguments are readily applied for the pencil $A - \lambda I$.

The paper is organized as follows.

In Section 2, we introduce, follow [14, 15], preconditioning for eigenvalue solvers and give general definitions of preconditioned single-vector and block, or simultaneous, iterative eigensolvers. We describe the global optimization method on the corresponding generalized Krylov subspace. No efficient algorithm is presently known to perform a global optimization of the Rayleigh quotient on the generalized Krylov subspace. We shall show numerically in Section 8, however, that the method we suggest in Section 4 provides approximations often quite close to those of the global optimization, for a small fraction of the cost.

In Section 3, we outline the "ideal" control algorithm, namely, the standard preconditioned conjugate gradient method for finding an eigenvector as an element of the null–space of the corresponding homogeneous system of linear equations under assumption that the eigenvalue is known. The algorithm cannot, of course, be used in practice as it requires knowledge of the extreme eigenvalue, but it seems to be a perfect choice as a benchmark for preconditioned eigensolvers for computing the extreme eigenpair.

In Section 4, we describe, in some details, a new algorithm of the Locally Optimal Preconditioned Conjugate Gradient Method, based on the local optimization of the three-term recurrence suggested by the author in [12, 14, 15]. In the original algorithm of [12], the three-term recurrence contains the current eigenvector approximation, the current preconditioned residual, and the previous eigenvector approximation. As the current eigenvector approximation and the previous eigenvector approximation get closer in the process of iterations, special measures need to be used in the algorithm to overcome the potential instability. In our new algorithm, the three-term recurrence contains the current eigenvector approximation, the current preconditioned residual, and the implicitly computed difference of the current and previous eigenvector approximations. Such choice resolves instability issues and allows us to write a much simpler and more efficient code.

We present block versions of the method, the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) Methods for symmetric eigenvalue problems, in Section 5.

To be able to compare numerically different methods in the class, with different preconditioners, we suggest, in Section 6, a common system of model tests, using random preconditioners and initial guesses. We recommend that every new preconditioned eigensolver for computing the extreme eigenpair be compared with our "ideal" algorithm on our model test problems in terms of the speed of convergence, costs of every iterations, and memory requirements. We also recommend a comparison with the global optimization method in terms of accuracy.

We provide such comparison for our LOBPCG Method in Sections 7 and 8. Numerical results of Section 7 establish that our algorithm is practically as efficient as the "ideal" algorithm when the same preconditioner is used in both methods. In Section 8 we show numerically that the block version of our method comes close to finding the global optimum of the Rayleigh quotient on the corresponding generalized block Krylov subspace.

Section 9 contains an informal discussion of block Davidson's method. We describe a nonstandard restart strategy that makes block Davidson's method a generalization of our LOBPCG method. We argue, however, that such generalization may not be beneficial for symmetric eigenvalue problem.

In Section 10, we compare directly our method with the version of the Jacobi–Davidson method [8] for $B = I$. No MATLAB code of the Jacobi–Davidson method for a generalized eigenvalue problem is currently publicly available. We find that our method is much more robust and typically converges almost two times faster. This is not very surprising as the MATLAB version of the Jacobi–Davidson method available to us for testing is not apparently optimized for symmetric eigenvalue problems, while our method takes full advantage of the symmetry by using a three-term recurrence. The only disadvantage of our method is that it requires several (minimum two, sometimes three, up to four) applications of the matrix A on every iteration step, because of the use of the Rayleigh–Ritz method and orthogonalization in the A-based scalar product. At the same time, the Jacobi–Davidson method applies A only once on every iteration.

Finally, Section 11 contains references to some relevant software written by the author.

The main results of the paper were presented by the author at the MiniSymposium on Very Large Eigenvalue Problems at the Fifth U.S. National Congress on Computational Mechanics, University of Colorado at Boulder, 1999.

2. Preconditioning for eigenvalue problems. First, we briefly review the traditional approach for large symmetric generalized eigenproblems, based on using traditional methods, e.g., the Lanczos method, for a shifted-and-inverted operator

$$(B - \nu A)^{-1} A.$$

It typically lets us quickly compute the eigenvalues closest to the shift ν, assuming that this operation may be implemented with an efficient factorization of $B - \nu A$. However, for very large problems such factorizations are usually too expensive.

If B is efficiently factorizable, e.g., $B = I$, so that we can multiply vectors by AB^{-1}, or $B^{-1}A$, we take $\nu = 0$. In this case, a single iteration may not be expensive, but eigenvalues μ close to zero are usually not of practical interest, and the convergence for eigenvalues of interest is often very slow.

Thus, the traditional approach is inefficient for very large mesh eigenproblems. Preconditioning is the key for significant improvement of the performance as it allows one to find a path between Scylla of expensive factorizations and Charybdis of slow convergence.

Preconditioned methods are designed to handle the case when the only operation we can perform with matrices A and B of the pencil is multiplication of a vector by A and B. To accelerate the convergence, we introduce a *preconditioner* T.

In many engineering applications, preconditioned iterative solvers for linear systems $Ax = b$ are already available, and efficient preconditioners $T \approx A^{-1}$ are constructed. It is important to realize that the same preconditioner T can be used to solve an eigenvalue problem $Ax = \lambda x$ and $Bx = \mu Ax$.

We assume that the preconditioner T is *symmetric positive definite*. As A is also symmetric positive definite, there exist positive constants $\delta_1 \geq \delta_0 > 0$ such that

$$\delta_0 (T^{-1}x, x) \leq (Ax, x) \leq \delta_1 (T^{-1}x, x). \tag{2.1}$$

The ratio δ_1/δ_0 can be viewed as the spectral condition number $\kappa(TA)$ of the preconditioned matrix TA and measures how well the preconditioner T approximates, up to a scaling, the matrix A^{-1}. A smaller ratio δ_1/δ_0 usually ensures faster convergence.

We want to highlight that the assumption we just made on T is essential for the theory, see [14], but may not be an actual limitation in numerical computations for some methods. In particular, our own method of Section 5 is quite robust in practice with respect to the choice of the preconditioner, even when the assumptions above are not satisfied; see Figure 5.3 below as an example of using an indefinite preconditioner.

We first define, following [14], a preconditioned singe-vector iterative solver for the pencil $B-\mu A$, as a generalized polynomial method of the following kind:

$$x^{(k)} = P_k(TA, TB)x^{(0)}, \tag{2.2}$$

where P_k is a polynomial of the k-th degree of two independent variables, $x^{(0)}$ is an initial guess, and T is a fixed preconditioner.

We only need to choose a polynomial, either *a priory*, or during the process of iterations, and use a recursive formula which leads to an iterative scheme. For an approximation $\mu^{(i)}$ $(\lambda^{(i)})$ to an eigenvalue of the pencil $B-\mu A$ $(A-\lambda B)$ for a given eigenvector approximation $x^{(i)}$ the Rayleigh quotient $\mu(x)$ $(\lambda(x))$ defined as

$$\mu(x^{(i)}) = \frac{(x^{(i)}, Bx^{(i)})}{(x^{(i)}, Ax^{(i)})} \quad \left(\lambda(x^{(i)}) = \frac{(x^{(i)}, Ax^{(i)})}{(x^{(i)}, Bx^{(i)})}\right). \tag{2.3}$$

is typically used.

ALGORITHM 2.1 : **A general Preconditioned Eigensolver.**
Input: *a starting vector $x^{(0)}$, devices to compute: Ax, Bx, and Tx for a given vector x, and the vector inner product (x,y).*

1. *Start: select $x^{(0)}$.*
2. *Iterate k steps to compute $x^{(k)} = P_k(TA,TB)x^{(0)}$.*
3. *Compute $\mu^{(k)} = (x^{(k)}, Bx^{(k)})/(x^{(k)}, Ax^{(k)})$.*

Output: *the approximations $\mu^{(k)}$ and $x^{(k)}$ to the largest eigenvalue μ_1 and its corresponding eigenvector.*

Thus, we have a complete description of the class of preconditioned eigensolvers for the pencil $B-\mu A$, see Algorithm 2.1. With $B=I$ and $\lambda = 1/\mu$, we obtain a similar algorithm for $A-\lambda I$.

Let us now define the generalized Krylov subspace:

$$K_k\left(TA, TB, x^{(0)}\right) = \left\{P_k(TA,TB)x^{(0)}\right\}, \tag{2.4}$$

where P_k runs through the set of all polynomials of the k-th degree of two independent variables and $x^{(0)}$ is a fixed initial vector. In particular,

$$K_2\left(TA, TB, x^{(0)}\right) = \text{span}\left\{x^{(0)}, TAx^{(0)}, TBx^{(0)}, (TA)^2x^{(0)}, TATBx^{(0)}, TBTAx^{(0)}, (TB)^2x^{(0)}\right\}.$$

We notice that in our definition (2.2) of the preconditioned eigensolver

$$x^{(k)} \in K_k\left(TA, TB, x^{(0)}\right).$$

Having definition (2.2) of the whole class of preconditioned eigensolvers, we can introduce, as in [14], the *global optimization* method for computing the first eigenpair simply by maximizing the Rayleigh quotient $\mu(x)$ on the Krylov subspace (2.4):

$$x_o^{(k)} = \text{argmin}_{x \in K_k\left(TA, TB, x^{(0)}\right)} \mu(x). \tag{2.5}$$

We want to highlight that an efficient algorithm of finding $x_o^{(k)}$, e.g., based on short-term recurrences, it is not presently known, and that the number of vectors in the basis of the Krylov subspace (2.4) grows exponentially, which makes the method very expensive in practice, similarly to the situation with Davidson's method, see discussion in [14, 15], unless restarts are used. Therefore, the global optimization method (2.5) is optimal only in the sense that it provides the global maximum of the Rayleigh quotient on the Krylov subspace, but may not be optimal if we also count computational costs.

For block methods, we introduce the generalized block Krylov subspace:

$$K_k\left(TA, TB, X^{(0)}\right) = \operatorname{span}\left\{P_k(TA, TB)x_j^{(0)},\ j = 1, \ldots, m\right\}, \tag{2.6}$$

where P_k runs through the set of all polynomials of the k-th degree of two independent variables and $X^{(0)} = \operatorname{span}\left\{x_j^{(0)},\ j = 1, \ldots, m\right\}$.

A general preconditioned block eigensolver is a generalization of Algorithm 2.1 with a single vector being replaced with several ones. Using the Rayleigh–Ritz method is typical for block methods, see [14, 15].

Here, we only want to define the *block global optimization* method, Algorithm 2.2, as we use it later in our numerical experiments.

ALGORITHM 2.2 : **The Block Globally Optimal Preconditioned Eigensolver.**
Input: *m starting vectors $x_1^{(0)}, \ldots x_m^{(0)}$,*
devices to compute: Ax, Bx, and Tx for a given vector x,
and the vector inner product (x, y).

1. *Start: select $x_j^{(0)},\ j = 1, \ldots, m$.*
2. *Iterate to compute the basis of the generalized block Krylov subspace (2.6).*
3. *Use the Rayleigh–Ritz method for the pencil $B - \mu A$ in the subspace to compute the Ritz values $\mu_j^{(k)}$ and the corresponding Ritz vectors $x_j^{(k)}$.*

Output: *the approximations $\mu_j^{(k)}$ and $x_j^{(k)}$ to the largest eigenvalues μ_j and corresponding eigenvectors.*

In our code of the block global optimization method, we do not even try to minimize computation costs, and simply compute recursively

$$K_{k+1} = K_k + TAK_k + TBK_k, \tag{2.7}$$

followed by complete orthogonalization. The only purpose of the code is to provide a comparison, in terms of accuracy, for the actual block method we suggest in Section 5.

3. The "Ideal" Preconditioned Conjugate Gradient Method. In this section, we outline the "ideal" control algorithm, namely, the standard preconditioned conjugate gradient method for finding an eigenvector, corresponding to the minimal eigenvalue, as an element of the null–space of the corresponding homogeneous system of linear equations under the assumption that the eigenvalue is known.

Let us *suppose* that the minimal eigenvalue λ_1 is *already known*, and we just need to compute the corresponding eigenvector x_1, an element of the null–space of the homogeneous system of linear equations

$$(A - \lambda_1 B)x_1 = 0,$$

where the matrix of the system is symmetric and nonnegative definite. What would be an *ideal* preconditioned method of computing x_1? As such, we choose the *standard preconditioned CG method*. It is well-known that a CG method can be used to compute a nonzero element of the null–space of a homogeneous system of linear equations with symmetric and nonnegative definite matrix if a nonzero initial guess is used. This ideal method *cannot be used in practice* as it requires knowledge of the eigenvalue.

We suggest using the method as a *control* in numerical comparison with practical eigenvalue solvers, in particular, with PCG eigensolvers, e.g., with our Locally Optimal PCG method. If an eigensolver finds the eigenvector u_1 with the same accuracy and costs as the ideal method, we have reasons to call such an eigensolver "optimal" for computing the eigenvector u_1.

For our numerical tests, we write a code PCGNULL.m, which is a slightly modified version of the built-in MATLAB code PCG.m of the standard PCG method to cover solution of homogeneous systems of linear equations with symmetric and nonnegative definite system matrices.

To be consistent with the code PCG.m, we measure the error as the Euclidean norm of the residual, i.e. $\|(A-\lambda_1 B)x^{(i)}\|/\|x^{(i)}\|$ in PCGNULL and $\|(A-\lambda^{(i)}B)x^{(i)}\|/\|x^{(i)}\|$ in our code, when we compare them.

4. The Preconditioned Conjugate Gradient Methods. In this section, we propose a new version of the *Locally Optimal Preconditioned Conjugate Gradient Method* [12].

In [12], the author suggested the following preconditioned CG method for the pencil $B-\mu A$:

$$x^{(i+1)} = w^{(i)} + \tau^{(i)}x^{(i)} + \gamma^{(i)}x^{(i-1)},\ w^{(i)} = T(Bx^{(i)} - \mu^{(i)}Ax^{(i)}),\ \mu^{(i)} = \mu(x^{(i)}),\ \gamma^{(0)} = 0, \tag{4.1}$$

with scalar iteration parameters $\tau^{(i)}$ and $\gamma^{(i)}$ chosen using an idea of *local optimality* [12], namely, select $\tau^{(i)}$ and $\gamma^{(i)}$ that maximize the Rayleigh quotient $\mu(x^{(i+1)})$ by using the Rayleigh–Ritz method. As the current eigenvector approximation $x^{(i)}$ and the previous eigenvector approximation $x^{(i-1)}$ are getting closer to each other in the process of iterations, special measures need to be used in the algorithm to overcome the potential instability. In our new algorithm, the three-term recurrence contains the current eigenvector approximation, the current preconditioned residual, and the implicitly computed difference between the current and the previous eigenvector approximations:

$$\begin{array}{rcrcl} x^{(i+1)} & = & w^{(i)} + \tau^{(i)}x^{(i)} & + & \gamma^{(i)}p^{(i)},\ w^{(i)} = T(Bx^{(i)} - \mu^{(i)}Ax^{(i)}), \\ p^{(i+1)} & = & w^{(i)} & + & \gamma^{(i)}p^{(i)},\ p^{(0)} = 0,\ \mu^{(i)} = \mu(x^{(i)}). \end{array} \tag{4.2}$$

with scalar iteration parameters $\tau^{(i)}$ and $\gamma^{(i)}$ chosen using the idea of *local optimality* as above, namely, select $\tau^{(i)}$ and $\gamma^{(i)}$ that maximize the Rayleigh quotient $\mu(x^{(i+1)})$ by using the Rayleigh–Ritz method. We see that

$$p^{(i+1)} = x^{(i+1)} - \tau^{(i)}x^{(i)},$$

thus,

$$x^{(i+1)} \in \text{Span}\,\{w^{(i)},\ x^{(i)},\ p^{(i)}\} = \text{Span}\,\{w^{(i)},\ x^{(i)},\ x^{(i-1)}\},$$

therefore, the new formula (4.2) is mathematically equivalent to the previous one, (4.1).

We describe the actual algorithm of the method given by (4.2) as Algorithm 4.1.

ALGORITHM 4.1 : **The Locally Optimal Preconditioned Conjugate Gradient Method.**

Input: *devices to compute: Ax, Bx and Tx for a given vector x, the vector inner product (x, y), and a starting vector $x^{(0)}$.*

1. *Start: select $x^{(0)}$ and set $p^{(0)} = 0$.*
2. *Iterate: For $i = 0, \ldots,$ Until Convergence Do:*
3. $\mu^{(i)} := (x^{(i)}, B\,x^{(i)})/(x^{(i)}, A\,x^{(i)})$
4. $r := B\,x^{(i)} - \mu^{(i)} A x^{(i)}$
5. $w^{(i)} := Tr$
6. *Use the Rayleigh–Ritz method for the pencil $B - \mu A$ on the trial subspace Span $\{w^{(i)}, x^{(i)}, p^{(i)}\}$*
7. $x^{(i+1)} := w^{(i)} + \tau^{(i)} x^{(i)} + \gamma^{(i)} p^{(i)}$, *(the Ritz vector corresponding to the maximal Ritz value)*
8. $p^{(i+1)} := w^{(i)} + \gamma^{(i)} p^{(i)}$
9. *EndDo*

Output: *the approximations $\mu^{(k)}$ and $x^{(k)}$ to the largest eigenvalue μ_1 and its corresponding eigenvector.*

Let us also mention a possibility to avoid using the residual in method (4.1) by splitting it into two vectors:

$$x^{(i+1)} \in \text{Span}\left\{TAx^{(i)}, TBx^{(i)}, x^{(i)}, x^{(i-1)}\right\}. \tag{4.3}$$

In this new method, the trial subspace is enlarged, which may lead to somewhat faster convergence. However, we need to apply the preconditioner two times now on every iteration. We do not have any numerical results for method (4.3), but, in our opinion, it will not be more efficient for computing the extreme eigenpair than our favorite method, Algorithm 4.1.

Other known CG methods for eigenproblems, e.g., [24, 25, 10, 26, 9, 7, 1], starting from Bradbury, Fletcher (1966) [3], are usually constructed as general CG minimization methods, applied to the Rayleigh quotient. They are typically based on (now standard for linear systems) two linked two-term recurrences,

$$p^{(i)} = -w^{(i)} + \beta^{(i)} p^{(i-1)},\ x^{(i+1)} = x^{(i)} + \alpha^{(i)} p^{(i)},$$

where $\alpha^{(i)}$ is chosen using a line search to minimize the Rayleigh quotient of $x^{(i+1)}$, which leads to a quadratic equation for $\alpha^{(i)}$, but $\beta^{(i)}$ is computed to make directions $p^{(i)}$ to be conjugate in some sense. These methods *do not utilize* the specific property of the Rayleigh quotient that the local minimization of the Rayleigh quotient can be cheaply carried out using the Rayleigh–Ritz method not just in two dimensions, for line search for finding $\alpha^{(i)}$, but in three and larger dimensional subspaces as well.

The locally optimal version of the preconditioned conjugate gradient method trivially converges not slower than the preconditioned steepest ascent in terms of the maximizing the Rayleigh quotient, thus, we can use known and well-developed convergence theory of the latter method, e.g., [12]. Our numerical comparison in [14, 15] shows that the preconditioned conjugate gradient method converges much faster in practice than the preconditioned steepest ascent. A 10-fold increase of δ_1/δ_0 leads to the increase of number of iterations, 10-fold for the preconditioned steepest ascent, but only about 3-fold for the preconditioned conjugate gradient method, exactly as we would expect for a genuine PCG solver.

We have collected the dominant costs per iteration for Algorithm 4.1 in terms of storage, in Table 4.1, and floating point operations, in Table 4.2.

We note that, for ill-conditioned problems and when a high accuracy is required, even our new choice of the basis $w^{(i)}, x^{(i)}, p^{(i)}$ of the trial subspace of the Rayleigh–Ritz method may lead to ill-conditioned 3-by-3 matrices, which makes necessary orthogonalization prior to the use of the Rayleigh–Ritz method. As by our assumptions matrix B may not be positive definite, there is no other option except to use the A-based scalar product for the orthogonalization. This typically increases the cost of iterations, but makes the algorithm more robust.

We present block versions of Algorithm 4.1 in the next section.

item	*storage*
residual	1 n-vector
approximate eigenvector	2 n-vectors
temporary vectors	$3-5$ n-vectors

TABLE 4.1: **Storage requirements for Preconditioned Conjugate Gradient Method.**

action	*major cost of one iteration*
Rayleigh–Ritz method	9 dots and 2 matrix-vector products, i.e. Bx and Ax
residual r	2 matrix-vector products, i.e. Bx and Ax, 1 update
preconditioned residual w	one application of the preconditioner T
approximate eigenvector	1 update

TABLE 4.2: **Computational costs for Preconditioned Conjugate Gradient Method.**

5. Preconditioned Simultaneous Iterations. A well known idea of using *Simultaneous, or Block Iterations* provides an important improvement over singe-vector methods, and permits us to compute an $m > 1$ dimensional invariant subspace, rather than one eigenvector at a time. It can also serve as an acceleration technique over single-vector methods on parallel computers, as convergence for extreme eigenvalues usually increases with the size of the block, and every step can be naturally implemented on wide varieties of multiprocessor computers.

A block version of the Locally Optimal Preconditioned Conjugate Gradient Method [12] was suggested in [14, 15]:

$$x_j^{(i+1)} \in \text{Span}\left\{x_1^{(i-1)},\ x_1^{(i)},\ T(B-\mu_1^{(i)}A)x_1^{(i)},\ldots,\ x_m^{(i-1)},\ x_m^{(i)},\ T(B-\mu_m^{(i)}A)x_m^{(i)}\right\}. \tag{5.1}$$

where $x_j^{(i+1)}$ is computed as the j-th Ritz vector. It shares the same problem, as the single-vector version of [12] discussed in the previous section, of having close vectors in the trial subspace.

Our new block algorithm is a straightforward generalization of the singe-vector Algorithm 4.1 and is combined with the Rayleigh–Ritz procedure. Here we present two different variants of the algorithm. They differ in the way that the Rayleigh–Ritz method is used. The first version is mathematically equivalent (without round-off errors) to that of [14, 15], but uses directions $p_j^{(i)}$ instead of $x_j^{(i-1)}$, similar to that of Algorithm 4.1.

ALGORITHM 5.1 : **The Block Locally Optimal Preconditioned Conjugate Gradient Method I.**

Input: *m starting vectors $x_1^{(0)}, \dots x_m^{(0)}$,*
devices to compute: Ax, Bx and Tx for a given vector x,
and the vector inner product (x, y).

1. *Start: select $x_j^{(0)}$, and set $p_j^{(0)} = 0$, $j = 1, \dots, m$.*
2. *Iterate: For $i = 0, \dots,$ Until Convergence Do:*
3. $\mu_j^{(i)} := (x_j^{(i)}, B\, x_j^{(i)})/(x_j^{(i)}, A\, x_j^{(i)})$, $j = 1, \dots, m$
4. $r_j := B\, x_j^{(i)} - \mu_j^{(i)} A x_j^{(i)}$, $j = 1, \dots, m$
5. $w_j^{(i)} := T r_j$, $j = 1, \dots, m$
6. *Use the Rayleigh–Ritz method for the pencil $B - \mu A$ on the trial subspace Span $\{w_1^{(i)}, \dots, w_m^{(i)}, x_1^{(i)}, \dots, x_m^{(i)}, p_1^{(i)}, \dots, p_m^{(i)}\}$,*
7. $x_j^{(i+1)} := \sum_{k=1,\dots,m} \alpha_k^{(i)} w_k^{(i)} + \tau_k^{(i)} x_k^{(i)} + \gamma_k^{(i)} p_k^{(i)}$,
(the j-th Ritz vector corresponding to the j-th largest Ritz value), $j = 1, \dots, m$
8. $p_j^{(i+1)} := \sum_{k=1,\dots,m} \alpha_k^{(i)} w_k^{(i)} + \gamma_k^{(i)} p_k^{(i)}$
9. *EndDo*

Output: *the approximations $\mu^{(k)}$ and $x^{(k)}$ to the largest eigenvalue μ_1 and its corresponding eigenvector.*

Here, the column-vector

$$\left(\alpha_1^{(i)}, \dots, \alpha_m^{(i)},\ \tau_1^{(i)}, \dots, \tau_m^{(i)}\ \gamma_1^{(i)}, \dots, \gamma_m^{(i)}\right)^T$$

is the the j-th eigenvector corresponding to the j-th largest eigenvalue of the $3m$-by-$3m$ eigenvalue problem of the Rayleigh–Ritz method, so it should have had an index j also, but we run out of space for indexes.

We observe that

$$p_j^{(i+1)} = x_j^{(i+1)} - \sum_{k=1,\dots,m} \tau_k^{(i)} x_k^{(i)},$$

thus,

$$\begin{aligned} x_j^{(i+1)} &\in \text{Span}\,\{w_1^{(i)}, \dots, w_m^{(i)}, x_1^{(i)}, \dots, x_m^{(i)}, p_1^{(i)}, \dots, p_m^{(i)}\} \\ &= \text{Span}\,\{w_1^{(i)}, \dots, w_m^{(i)}, x_1^{(i)}, \dots, x_m^{(i)}, x_1^{(i-1)}, \dots, x_m^{(i-1)}\}, \end{aligned}$$

and, indeed, the new Algorithm 5.1 is mathematically equivalent to method (5.1).

EXAMPLE 5.1. *Let us consider the problem of computing first eigenpairs of the standard five-point finite-difference approximation of the Laplacian in a $[-1, 1] \times [-1, 1]$ square on a uniform mesh with the step $1/10$, such that the total number of the unknowns is 361. The initial approximations are random. No preconditioning is used in the first test, i.e. $T = I$. We plot on Figures 5.1 and 5.2 on a semi-log scale relative errors in this example defined as*

$$\|Ax_j^{(i)} - \lambda_j^{(i)} x_j^{(i)}\|_A / \|x_j^{(i)}\|_A,$$

where we use the standard notation of an A-based norm, $\|\cdot\|_A^2 = (\cdot, A\cdot)$.

Figures 5.1 and 5.2 show a super-linear convergence. We see that a larger block size in the method improves convergence in this example. We also observe different convergence speed for different eigenpairs on Figure 5.2 – the convergence is faster for smaller eigenvalues λ, in particular, λ_1 converges first.

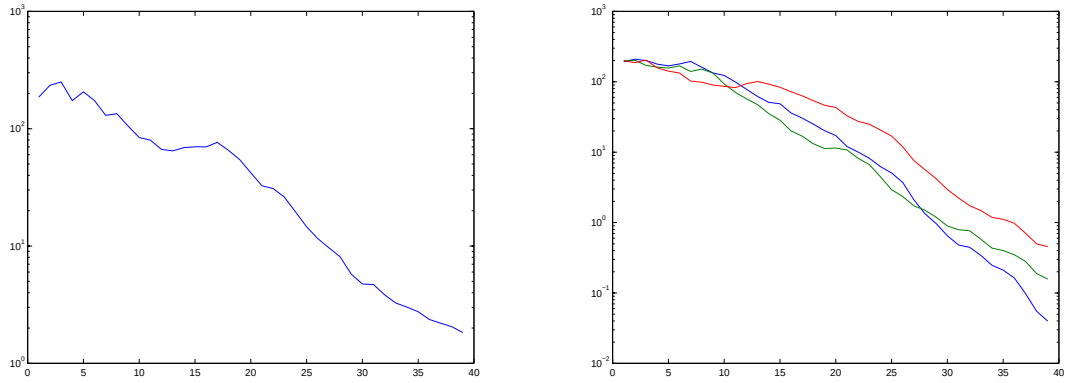

FIG. 5.1: *Errors for the Poisson operator in a square without preconditioning for one (left) and three (right) first eigenpairs*

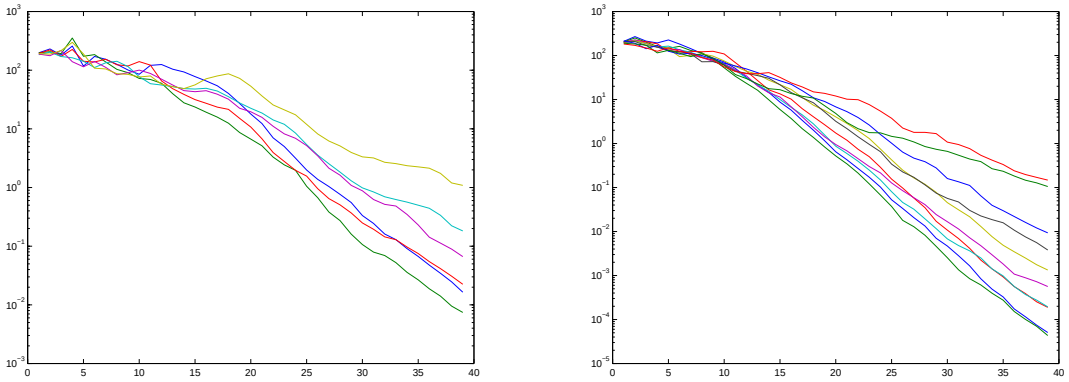

FIG. 5.2: *Errors for the Poisson operator in a square without preconditioning for six (left) and ten (right) first eigenpairs*

In the second test, we use a preconditioner $T = A - \nu I$ *with the shift* $\nu = 20$ *which is approximately in the middle of the group of the first ten eigenvalues that we compute starting with random initial approximations. We plot on Figure 5.3 on a semi-log scale relative residuals now defined as*

$$\|Ax_j^{(i)} - \lambda_j^{(i)} x_j^{(i)}\| / \|x_j^{(i)}\|,$$

using the Euclidian norm. Figure 5.3 shows a superior convergence of the preconditioning. Now, we have a better convergence for eigenvalues close to the shift.

The example thus illustrates that our method works without preconditioning and may even work with an indefinite preconditioner.

EXAMPLE 5.2. *As our next example, we solve a similar eigenvalue problem, but in the L-shaped domain — a union of three unit squares, with a mesh, uniform in both directions, with the step* $1/8$*, such that the total number of unknowns is 161. A specialized domain-decomposition without overlap method for such problem is suggested in [17]. Our numerically computed eigenvalues are consistent with those found in [17].*

We compare the performance of the method without preconditioning and with preconditioning based on an incomplete Choleski decomposition of A with a drop tolerance of 10^{-3}*. We plot on Figure 5.4 relative errors in this example defined as*

$$\|Ax_j^{(i)} - \lambda_j^{(i)} x_j^{(i)}\| / \|x_j^{(i)}\|,$$

using the Euclidian norm. We see that preconditioning leads to approximately quadruple acceleration.

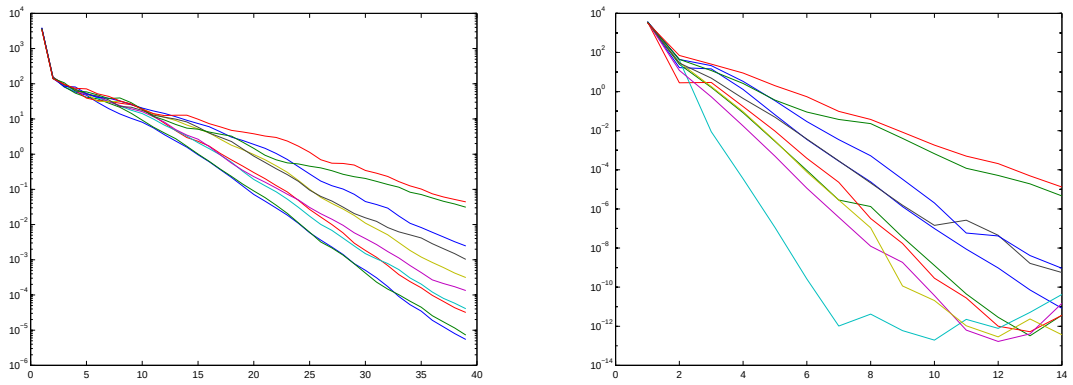

FIG. 5.3: *Errors for the Laplacian operator in a square without preconditioning (left) and with an indefinite preconditioner(right)*

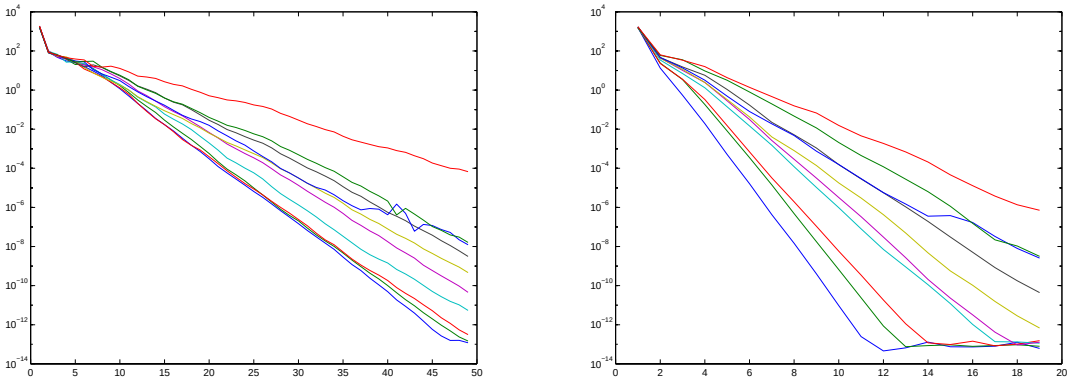

FIG. 5.4: *Residuals for the Laplacian operator in the L-shaped domain with no preconditioning (left) and an ILU preconditioning (right)*

The actual MATLAB code LOBPCG of Algorithm 5.1 we wrote uses the basis of the trial subspace exactly the way it appears in Algorithm 5.1 until this choice leads to ill-conditioned Gramm matrices in the Rayleigh–Ritz method. When such an ill-conditioning shows up, we perform a selected orthogonalization. If this does not fix the problem, as a last resort we apply a complete orthogonalization prior to the Rayleigh–Ritz method. By our assumptions, A may be the only positive definite matrix of the pencil, thus, the A-based scalar product is used for the orthogonalization.

A potential disadvantage of Algorithm 5.1 can manifest itself when the number m of eigenpairs of interest is large as we need to form $3m$-by-$3m$ matrices and solve the corresponding eigenvalue problem of the size $3m$ in the Rayleigh–Ritz method on every step. It seems that vectors $w_j^{(i)}$ and $p_j^{(i)}$ may not be always helpful as basis vectors of the trial subspace of the Rayleigh–Ritz method to improve the approximation of $x_k^{(i+1)}$ for $j \neq k$. And adding unnecessary vectors in the trial subspace increases computational costs. Even more importantly, it may make the algorithm less stable as the $3m$-by-$3m$ eigenvalue problem of the Rayleigh–Ritz method is likely to inherit ill-conditioning of the original eigenvalue problem when m is large.

In our second variant, Algorithm 5.2, we apply the Rayleigh–Ritz method in two stages: firstly, as in Algorithm 4.1, for individual indexes j and, secondly, we include only the minimal set of vectors, namely, just current approximations to different eigenvectors into the trial subspace.

ALGORITHM 5.2 : **The Block Locally Optimal Preconditioned Conjugate Gradient Method II.**
Input: *m starting vectors* $x_1^{(0)}, \dots x_m^{(0)}$,
devices to compute: Ax, Bx *and* Tx *for a given vector* x,
and the vector inner product (x, y),

1. *Start: select* $x_j^{(0)}$, *and set* $p_j^{(0)} = 0,\ j = 1, \dots, m.$
2. *Iterate: For* $i = 0, \dots,$ *Until Convergence Do:*
3. $\mu_j^{(i)} := (x_j^{(i)}, B\, x_j^{(i)})/(x_j^{(i)}, A\, x_j^{(i)}),\ j = 1, \dots, m$
4. $r_j := B\, x_j^{(i)} - \mu_j^{(i)} A x_j^{(i)},\ j = 1, \dots, m$
5. $w_j^{(i)} := Tr_j,\ j = 1, \dots, m$
6. *Use the Rayleigh–Ritz method* m *times for the pencil* $B - \mu A$
on the trial subspaces
Span $\{w_j^{(i)},\ x_j^{(i)},\ p_j^{(i)}\},\ j = 1, \dots, m,$
7. $\widehat{x}_j^{(i+1)} := w_j^{(i)} + \tau_j^{(i)} x_j^{(i)} + \gamma_j^{(i)} p_j^{(i)},$
(the Ritz vector corresponding to the maximal Ritz value), $j = 1, \dots, m$
8. $p_j^{(i+1)} := w_j^{(i)} + \gamma_j^{(i)} p_j^{(i)},$
9. *Use the Rayleigh–Ritz method for the pencil* $B - \mu A$ *on the trial subspaces*
$\{\widehat{x}_1^{(i+1)}, \dots, \widehat{x}_m^{(i+1)}\},$
10. $x_j^{(i+1)} :=$ *the* j*-th Ritz vector corresponding to the* j*-th*
largest Ritz value, $j = 1, \dots, m$
11. *EndDo*

Output: *the approximations* $\mu^{(k)}$ *and* $x^{(k)}$ *to the largest eigenvalue* μ_1 *and its corresponding eigenvector.*

Thus, on the first stage of an iteration of Algorithm 5.2 we solve m three-dimensional eigenproblems, and on the second stage we solve one m-dimensional eigenproblem. But this m-dimensional eigenproblem is constructed using approximate eigenvectors, corresponding to extreme eigenvalues, as a basis of the trial subspace. Therefore, this eigenvalue problem should not be ill-conditioned and no orthogonalization would be required.

Algorithm 5.2 is not yet implemented in a code. It will be incorporated into a future revision of our LOBPCG.

There is no theory available yet to predict accurately the speed of convergence of our new algorithms. However, by analogy with known convergence theory of preconditioned CG system solvers, and having in mind results of [13] on the Rayleigh–Ritz method, we expect convergence of norms of residuals to be asymptotically linear with the ratio

$$q = \left(\frac{1 - \sqrt{\xi_j}}{1 + \sqrt{\xi_j}} \right),\ \xi_j = \frac{\delta_0}{\delta_1} \frac{\mu_j - \mu_{m+1}}{\mu_j - \mu_{\min}}. \tag{5.2}$$

Thus, convergence of $\mu_j^{(i)}$ to μ_j should be linear with the ratio q^2.

All our numerical tests, see some selected results in Sections 7 and 8 below, support our expectation of having (at least) the linear convergence with the ratio (5.2).

Finally, we would like to remind the reader that in [14, 15] we demonstrate numerically that our method is much faster than that of [4]. The latter method, however, has an advantage that a rigorous convergence rate estimate is obtained [4].

As theoretical investigation and comparison of preconditioned eigensolvers is quite tedious ([4] provides a perfect example of this), a possibility of a fair numerical comparison becomes even more important than usual. In the next section, we suggest a numerical benchmark for preconditioned eigensolvers.

6. Model Test Problems with Random Preconditioners. To be able to compare numerically different methods in the class with different preconditioners, we suggest the following system of model tests, with random preconditioners and initial guesses, to be used for benchmarking.

For simplicity, we take the mass matrix $B = I$.

The stiffness matrix A is in our model tests a diagonal matrix with the minimal entry 1 and the maximal entry 10^{10}, therefore, all eigenvalues μ of the pencil $B - \mu A$ lie on the semi-closed interval $(0, 1]$. We are interested in finding a group of the largest eigenvalues and corresponding eigenvectors. In most of the tests of the present paper, we compute only one, the largest eigenvalue $\mu = 1$.

For preconditioned eigensolvers, we expect that the convergence does not slow down when the condition number of A gets larger [11, 12, 4, 14, 15], with a properly chosen preconditioner. Because of that, we simply use a fixed large value, 10^{10}, for the maximal entry of A. Our code seems to work robustly for condition number of A as large as 10^{16}.

The gap between computed and excluded parts of the spectrum is known to play an important role in the convergence speed. It seems necessary to fix the gap within a series of tests. We do it with the gap ranging from 1 to 0.01 in different series. A small value of the gap may, or may not, lead to slow convergence, depending on several factors, first of all, on distribution of eigenvalues in the excluded part of the spectrum close to the desired part. That makes comparison of different methods somewhat unreliable when the gap is small.

It is also necessary to choose a distribution of eigenvalues. The desired eigenvalues, if there is more then one, are distributed randomly on a given interval as their distribution should not affect performance significantly. In the rest of the spectrum, the distribution of eigenvalues does not noticeably affect the speed of convergence in our tests for a general preconditioner. If, however, the preconditioner commutes with A, e.g., $T = I$, or $T = A^{-1}$, we do observe a strong influence of the distribution on convergence. For such cases, we choose a distribution that mimics that of an ordinary differential equation of the forth order, but with the given maximal entry 10^{10} (see above). The initial guess is fixed for every run of the actual and the control codes, but is changed for every new run as a vector with random entries, chosen from a normal distribution with mean zero and variance one.

The preconditioner T is also fixed for every run of the actual and the control programs, but is modified for every new run as a random symmetric positive definite matrix with the fixed value of $\kappa(TA) = \delta_1/\delta_0$ of (2.1). We construct T as follows.

First, we chose a diagonal matrix D with random diagonal entries, chosen from a uniform distribution on the interval $(0, 1)$. Then we find minimal $minD$ and maximal $maxD$ values of D and do a linear scaling

$$D = 1 + (D - minD)/(maxD - minD) * (\kappa - 1),$$

where $\kappa = \kappa(TA)$ is the desired fixed value of the spectral condition number of TA. That makes diagonal entries of D uniformly distributed on the interval $(1, \kappa)$ with minimal and maximal values exactly at the end points of the interval, thus, the condition number of D equals exactly κ.

Second, we chose a square matrix with random entries, chosen from a normal distribution with mean zero and variance one, and perform the standard orthogonalization procedure on it. That produces a random orthogonal matrix Q. We now scale it $S = QA^{-1/2}$ keeping in mind that Q is orthogonal and A is diagonal, therefore, $S^T = A^{-1/2}Q^{-1}$

Finally, we form $T = S^T DS$. The matrix T is clearly symmetric. Moreover, the diagonal entries of D and columns of Q are eigenpairs of the matrix

$$A^{1/2}TA^{1/2} = Q^{-1}DQ,$$

which completes our argument.

There are two reasons of using random preconditioners for our model test problems. First of all, it is a natural choice when we solve eigenvalue problems for diagonal matrices. It allows us to make a fair numerical comparison of different eigensolvers and gives a simple opportunity to check that the best method in the class consistently outperforms other methods independently of the choice of the preconditioner, for a fixed value of κ.

The size of the problem varies from 1000–4000.

The upper bound, 4000, is simply determined by our computer resources. The algorithm above of constructing a random preconditioner is quite expensive and leads to a full matrix T. The total cost grows cubically with the size of the problem. In the next revision of the code we plan to develop and program a cheaper algorithm of constructing the preconditioner. We find in our tests that small problems may lead to unreliable conclusions when the number of iterations is large enough. Namely, in some tests, depending on distribution of excluded eigenvalues of the original pencil, we observed a super-linear convergence of our methods when the total number of steps was more than 30% of the size of the problem. However, in practical applications of interest, eigenvalue problems are so large that the number of steps should not usually exceed 20% of the size of the problem, taking also into account that the high accuracy of desired eigenpairs of the algebraic pencil is rarely needed as the pencil itself is just an approximation of the original continuous problem and the approximation error may not be small. Thus, one should not count in practice on having a super-linear convergence; and we try to rule such a possibility out by choosing the size of the problem large enough.

We recommend every new preconditioned eigensolver be compared with our "ideal" algorithm on our model test problems in terms of the speed of convergence, costs of every iterations and memory requirements. We provide such comparison for our LOBPCG Method in the next section.

7. Numerical Results: LOBPCG vs. NULLPCG. Here, we solve a model eigenvalue problem with $\lambda_1 = 1,\ \lambda_2 = 2$ and we take the condition number of A to be 10^{10}.

For simplicity, the mass matrix equals the identity, $B = I$.

We compute only the first eigenpair, i.e. the smallest eigenvalue $\lambda_1 = 1$ and the corresponding eigenvector. Thus, we set the block size $m = 1$ in our LOBPCG Algorithm 5.1.

The initial guess is fixed for every run of the actual and the control codes, but is modified for every new run randomly. The preconditioner T is also fixed for every run of the actual and the control programs, but is modified for every new run as a random symmetric positive definite matrix with the fixed value of the condition number $\kappa(TA)$ (see above). We vary $\kappa(TA)$ in different series of tests.

The straight green line corresponds to linear convergence with the residual reduction rate

$$q = \left(\frac{1-\sqrt{\xi}}{1+\sqrt{\xi}}\right),\ \xi = \frac{1}{\kappa(TA)}\left(1-\frac{\lambda_1}{\lambda_2}\right).$$

predicted by the standard theory for the PCGNULL, cf. (5.2) To be consistent with MATLAB's build-in code PCG.m, we measure the error as the Euclidean norm of the residual, i.e. $\|(A-\lambda_1 B)x^{(i)}\|/\|x^{(i)}\|$ in PCGNULL and $\|(A-\lambda^{(i)}B)x^{(i)}\|/\|x^{(i)}\|$ in our code. With these definitions, norms of the residuals in the our code are artificially higher on the first couple of iterations in our tests, which should be ignored.

The average slope is the most important. We observe on Figures 7.1, 7.2, and 7.3,that the average residual reduction rate is about the same for the "ideal" method, PCGNULL, and for our LOBPCG, and is quite close to the theoretical prediction. For all the figures, convergence history lines for every method are tightly bundled together, with the bundle for our LOBPCG (colored red in the electronic version of the paper) consistently a bit lower than the bundle for the PCGNULL. Thus, our code converges essentially as fast as the "ideal" method.

Let us now compare computational costs of a single iteration. PCGNULL involves one application of the preconditioner T, and two multiplications of A and a vector. LOBPCG has exactly the same major cost, except that for very ill-conditioned problems and when a very high accuracy is required, to increase stability some redundancy is incorporated (revision 3.2.5, 1999/12/19), and the matrix A needs to be multiplied three times instead of the regular two.

We also compare the total number of floating point operations (FLOPS) and the central processor unit (CPU) time. The two methods are, again, quite similar. For a problem size $n = 300$ they take about the same number of FLOPS and CPU time. When the size of the problem grows, our method somehow performs better, e.g., for $n = 500$ PCGNULL is about twice slower and needs 30 percent more FLOPS then our code LOBPCG, while for $n = 1000$ it is three times slower and needs 40 percent more FLOPS. We do not have a complete explanation of this fact; let us, however, remind the reader that in our model tests A is a diagonal matrix, which apparently hides the potential extra costs of LOBPCG.

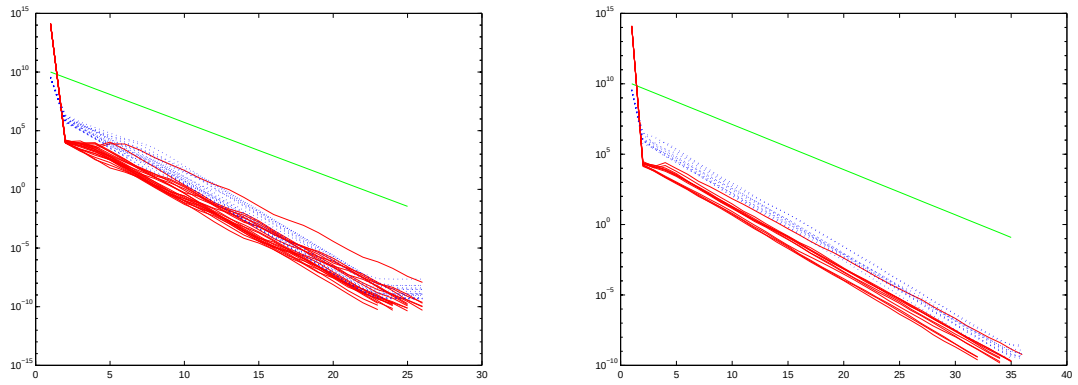

FIG. 7.1: **LOBPCG vs. Ideal, $\kappa(TA) = 2$ (left) and $\kappa(TA) = 4$ (right)**

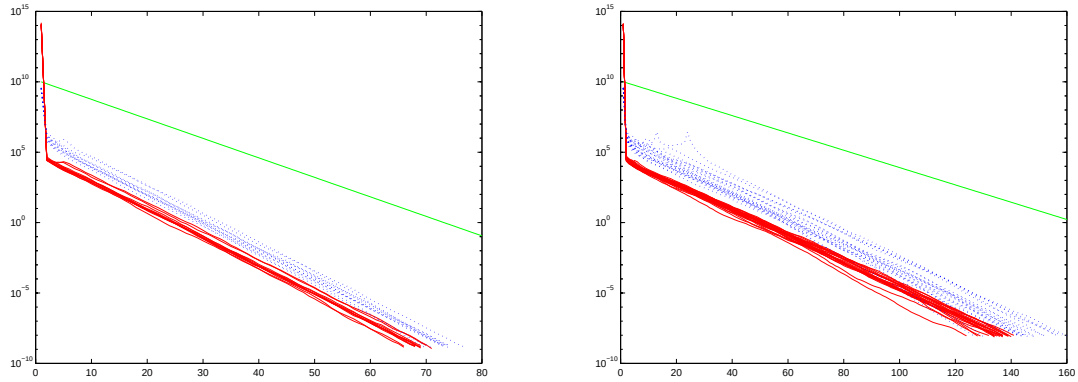

FIG. 7.2: **LOBPCG vs. Ideal, $\kappa(TA) = 20$ (left) and $\kappa(TA) = 100$ (right)**

In terms of memory use, both methods are similar, as they require only several vectors, but no large matrices, to be stored.

To conclude, numerical results establish that our algorithm is practically as efficient as the "ideal" algorithm when preconditioners and initial approximations are the same in both methods.

8. Numerical Results: LOBPCG vs. GLOBALMIN. In the previous section, we compare our LOBPCG with the benchmark based on PCGNULL and show that LOBPCG is practically the optimal preconditioned method for finding the extreme eigenvalue and the corresponding eigenvector. LOBPCG can also be used, of course, for computing a group of extreme eigenvalues, when the block size $m > 1$.

What benchmark do we advocate for preconditioned eigensolvers for finding several eigenpairs? We do not have an answer to this question as satisfactory as for a single extreme eigenpair, because we are not able yet to suggest a convincing "ideal" (in terms of speed and costs) solver. We do have, however, the block globally optimal solver, Algorithm 2.2.

Let us highlight again that the number of vectors in the basis of the block Krylov subspace (2.6) grows exponentially, which makes Algorithm 2.2 very expensive. On the other hand, it provides the global optimization of the Rayleigh quotient on the block Krylov subspace and, thus, can be used for numerical comparison with actual block preconditioned eigensolvers to check if they provide approximations close to those of the global optimization.

We write a MATLAB code of Algorithm 2.2, called GLOBALMIN, using recursion (2.7), followed by complete orthogonalization.

We test LOBPCG vs. GLOBALMIN on model problems described in Section 6 with $n = 3000$.

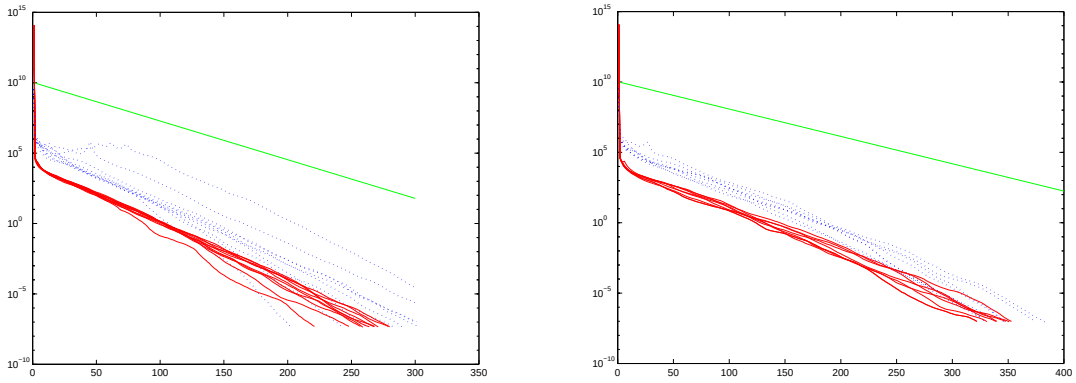

FIG. 7.3: **LOPCG vs. Ideal,** $\kappa(TA) = 500$ **(left) and** $\kappa(TA) = 1000$ **(right)**

The gap between computed and excluded parts of the spectrum is one. The condition number of A is chosen to be 10^8 as GLOBALMIN fails in some tests for larger condition numbers.

We find in the present section that putting only one run on a figure is more illustrative as LOBPCG and GLOBALMIN produce very similar results, but they change greatly with different random initial guesses. We remove the initial value of the error from all pictures as it is typically too large to fit the chosen scale. LOBPCG is presented by a solid (and red in a color version of the paper) line. The straight (and green) line corresponds, as in Section 7, to the average error reduction predicted by (5.2).

The residual-based error is measured as $\|(A - \lambda_j^{(i)} B)x_j^{(i)}\|/\|x_j^{(i)}\|$ on a corresponding Ritz vector $x_j^{(i)}$. The eigenvalue error is simply measured as the difference of the Ritz value $\lambda_j^{(i)}$ and the corresponding eigenvalue λ_j. Both methods, LOBPCG and GLOBALMIN, should monotonically decrease the eigenvalue error. GLOBALMIN provides the global minimum of the Rayleigh quotient, therefore, the eigenvalue error of GLOBALMIN should be always not larger than that of LOBPCG.

We first compare, on Figures 8.1 and 8.2, errors just for the smallest eigenvalue for the LOBPCG and GLOBALMIN both with block size one as in Section 7. Figures 8.1 and 8.2 display errors for the same problems as Figure 7.1, but we also add the eigenvalue error. We highlight again that dimension of the generalized Krylov subspace global (2.4) grows exponentially with the number of iterations. For numerical tests presented on Figures 8.1 and 8.2, typical dimensions are:

$$3, 7, 15, 31, 63, 127, 255, 511,$$

e.g., on the last, eighth iteration, GLOBALMIN minimizes the Rayleigh quotient on a trial subspace of dimension 511.

For all other figures of the Section, Figures 8.3, 8.4, 8.5, 8.6, and 8.7, we compare the error for the third smallest eigenvalue for the LOBPCG and GLOBALMIN, both with block size three. In these experiments, dimensions of the block generalized Krylov subspace global (2.6) typically are:

$$9, 21, 45, 93, 189, 381, 765.$$

As our code GLOBALMIN is based on complete orthogonalization that filters out possible linearly dependent vectors in the trial subspace, in different tests we observe slightly different dimensions.

The trial subspace in GLOBALMIN is getting large enough, about $10\% - -20\%$ of the size of the problem, to lead to a super-linear convergence of GLOBALMIN when $\kappa(TA)$ is not too large, see Figures 8.1, 8.3, and 8.4. We believe that this effect is artificially created by the fact that our problem is of a small size, only 3000, and should be disregarded. Our computer resources do not allow us to solve larger problems.

We first observe that LOBPCG and GLOBALMIN produce almost the same approximations on the first two steps. Even more importantly, by comparing the slopes on the figures, we come to the conclusion that our LOBPCG provides approximations close to those of the Global Optimization Method on the generalized block Krylov subspace and has a similar convergence speed.

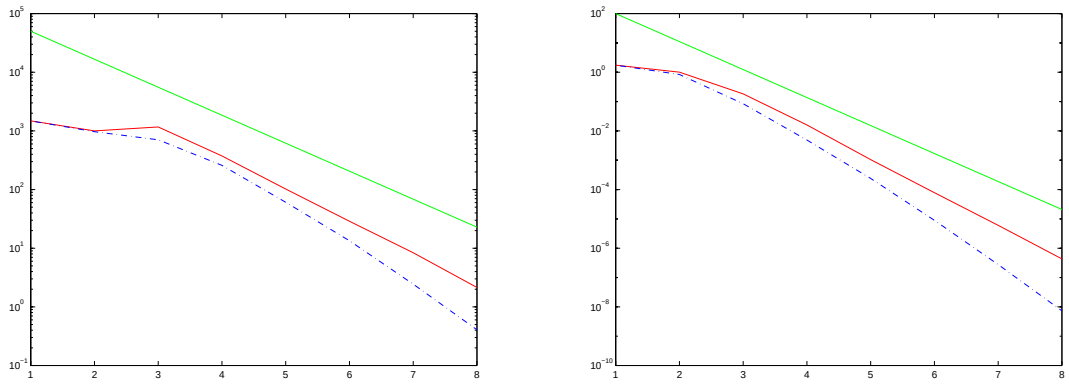

FIG. 8.1: *LOBPCG vs. GLOBALMIN, $\kappa(TA) = 2$, $bsize = 1$ residual (left) and lambda (right)*

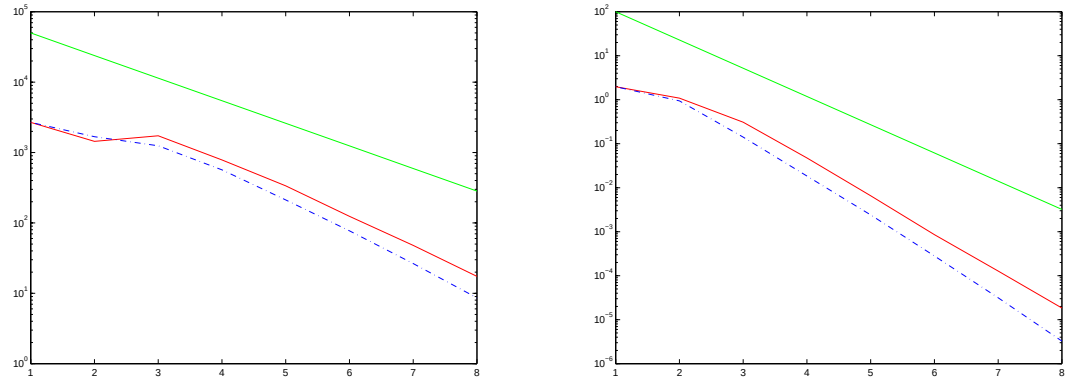

FIG. 8.2: *LOBPCG vs. GLOBALMIN, $\kappa(TA) = 4$, $bsize = 1$ residual (left) and lambda (right)*

9. LOBPCG vs. Davidson's method. The discussion above allows us to make some conclusions on LOBPCG vs. Davidson's method, though we do not have a numerical comparison. Block Davidson's method without restarts can be presented, cf. [5, 20, 23], as the Rayleigh–Ritz method on the trial subspace spanned on vectors:

$$\begin{array}{ll} \{ & x_1^{(0)},\ T(B-\mu_1^{(0)}A)x_1^{(0)},\ T(B-\mu_1^{(1)}A)x_1^{(1)},\ldots,\ T(B-\mu_1^{(i)}A)x_1^{(i)},\ldots \\ & x_m^{(0)},\ T(B-\mu_m^{(0)}A)x_m^{(0)},\ T(B-\mu_m^{(1)}A)x_m^{(1)},\ldots,\ T(B-\mu_m^{(i)}A)x_m^{(i)}\ \}, \end{array} \tag{9.1}$$

and $x_j^{(i+1)}$ is computed as the j-th Ritz vector. All vectors (9.1) are in the block generalized Krylov subspace (2.6) (assuming a fixed preconditioner), so such defined block Davidson's method cannot converges faster than the GLOBALMIN. But our LOBPCG converges with about the same rate as the GLOBALMIN. Therefore, we can expect that LOBPCG is more efficient that Davidson's method as the former should not converge much slower, but is significantly less expensive, than the latter.

To make Davidson's method more competitive with our LOBPCG, one needs to restart after every k steps in the following special way: the Rayleigh–Ritz method is now used on the trial subspace spanned by vectors

$$\begin{array}{ll} \{ & x_1^{(i-1)},\ x_1^{(i)},\ T(B-\mu_1^{(i)}A)x_1^{(i)},\ T(B-\mu_1^{(i+1)}A)x_1^{(i+1)},\ldots,\ T(B-\mu_1^{(i+k)}A)x_1^{(i+k)},\ldots, \\ & x_m^{(i-1)},\ x_m^{(i)},\ T(B-\mu_m^{(i)}A)x_m^{(i)},\ T(B-\mu_m^{(i+1)}A)x_m^{(i+1)},\ldots,\ T(B-\mu_m^{(i+k)}A)x_m^{(i+k)}\ \}, \end{array} \tag{9.2}$$

and $x_j^{(i+k+1)}$ is computed as the j-th Ritz vector. The new trial subspace is still a subset of the block generalized Krylov subspace (2.6), but its dimension does not depend on the number of iterations any longer.

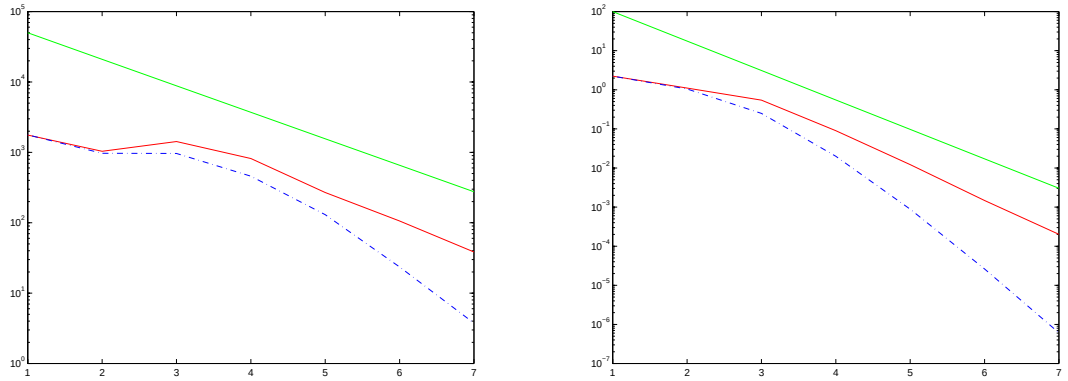

FIG. 8.3: *LOBPCG vs. GLOBALMIN,* $\kappa(TA) = 2$*,* $bsize = 3$ *residual (left) and lambda (right)*

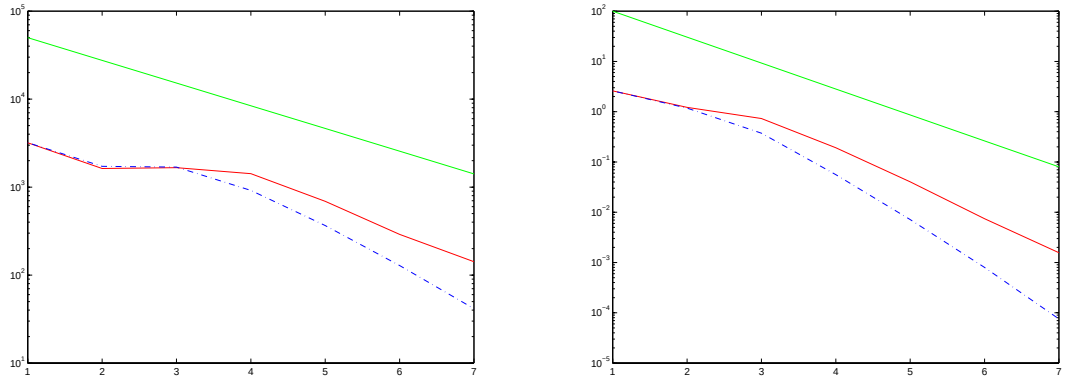

FIG. 8.4: *LOBPCG vs. GLOBALMIN,* $\kappa(TA) = 4$*,* $bsize = 3$ *residual (left) and lambda (right)*

Compare to a naive way of restarts, we have extra vectors, $x_j^{(i-1)}$, $j = 1, \ldots, m$, in the basis of the trial subspace, which we expect to make method (9.2) much faster. We now notice that Davidson's method based on (9.2) with $k = 0$ coincide with our earlier method (5.1). Thus, our Algorithm 5.1 can be viewed as a specially restarted at every step block Davidson's method.

Is there any benefit to use block Davidson's method, based on (9.2) with $k > 0$? In our opinion, for symmetric eigenproblems, the answer seems negative. We expect methods with $k = 0$ and $k > 0$ to be quite similar to each other in terms of speed of convergence, as the method with $k = 0$ already provides approximations practically close to those of the global optimization method on the block Krylov subspace. At the same time, method (9.2) with $k > 0$ will be less stable for ill-conditioned problems simply because the dimension of its trial subspace for the Rayleigh–Ritz method is larger. A direct numerical comparison is yet to be done.

10. Numerical Results: LOBPCG vs. JDQR. Here, we present results of direct numerical comparison of our method LOBPCG with the MATLAB code JDQR.m of the Jacobi–Davidson method [8], written by Gerard Sleijpen, which is publicly available on the Internet.

As in the previous two sections, the comparison is made using model eigenvalue problems with $B = I$ (a revision of JDQR code for generalized eigenproblems is not yet available anyway) and random preconditioners, suggested in Section 6. JDQR is used with the default tolerance. The number of iterations of our LOBPCG is chosen to match the accuracy of eigenvector approximations provided by the JDQR. We measure the accuracy as the angle between computed and exact invariant subspaces in the two-norm.

First, we find that JDQR is not as robust as our method with respect to ill-conditioning of the

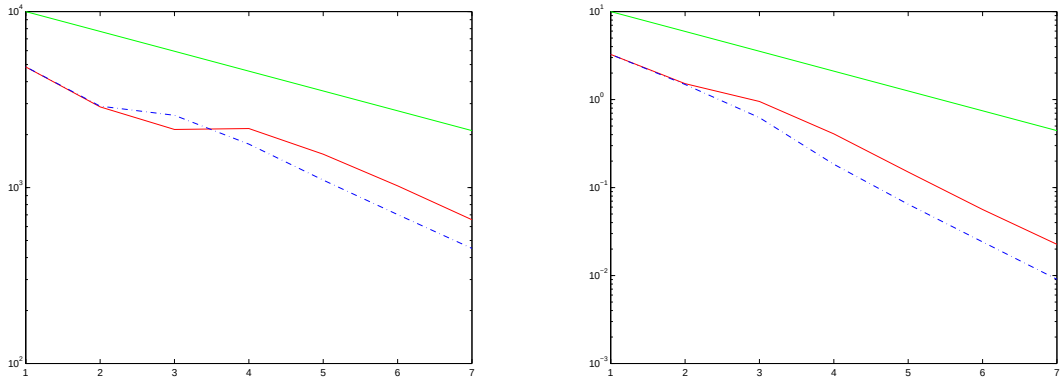

FIG. 8.5: *LOBPCG vs. GLOBALMIN,* $\kappa(TA) = 20$*,* $bsize = 3$ *residual (left) and lambda (right)*

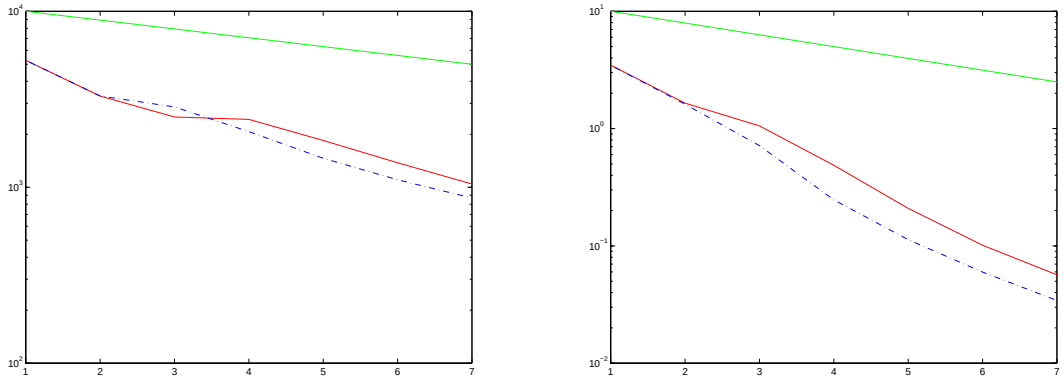

FIG. 8.6: *LOBPCG vs. GLOBALMIN,* $\kappa(TA) = 100$*,* $bsize = 3$ *residual (left) and lambda (right)*

matrix A and the number of required eigenpairs. JDQR consistently fails to find even one eigenpair for condition number of A above 10^8. JDQR becomes even more sensitive to ill-conditioning when we increase the number of required eigenpairs. With $cond(A) = 10^6$, JDQR typically fails to compute all ten required eigenpairs in another series of tests, and in some of the tests outputs only one eigenpair out of ten. Attempts to compute forty eigenpairs using JDQR even with $cond(A) = 10$ produce no more than sixteen eigenpairs.

JDQR does not handle random initial guess very well. In some tests it converges to the second eigenpair instead of the desired first one, with the smallest eigenvalue λ.

Our LOBPCG is much more robust and successfully computes all required eigenpairs in all tests mentioned above without any troubles.

LOBPCG typically converges about one and a half – two times faster than JDQR if we count the number of iterations as the number of times the preconditioner is invoked. This is not very surprising as the MATLAB version of the Jacobi–Davidson method available to us for testing can be used for nonsymmetric eigenproblems as well and is not apparently optimized for symmetric eigenvalue problems, while our method takes full advantage of the symmetry by using a three-term recurrence.

The only disadvantage of our algorithm compare to JDQR is that it requires several (minimum two, typically three, or sometimes even four) applications of the matrix A of the original pencil on every iteration step, because of the use of the Rayleigh–Ritz method and possibly orthogonalization in the A-based scalar product. At the same time, the Jacobi–Davidson method applies A only once on every iteration.

Both methods are scalable, as expected, with respect to the size of the problem and the quality of the preconditioner used.

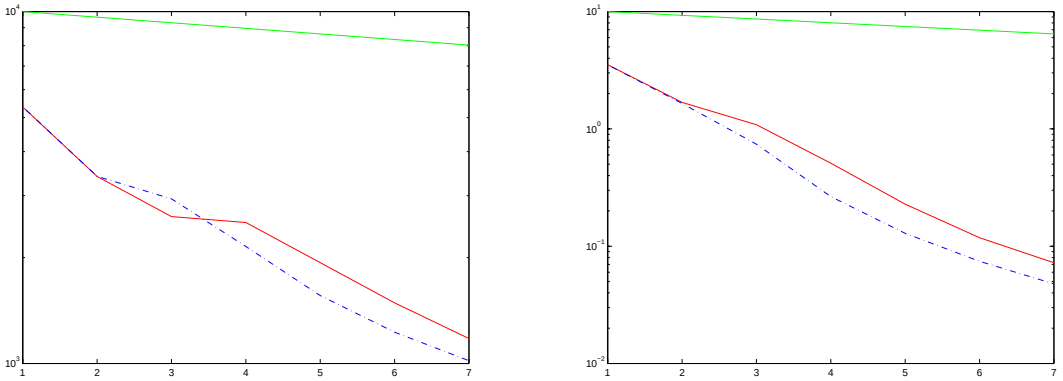

FIG. 8.7: *LOBPCG vs. GLOBALMIN,* $\kappa(TA) = 1000$, $bsize = 3$ *residual (left) and lambda (right)*

11. Availability of Software for the Preconditioned Eigensolvers. The Internet page
`http:// www-math.cudenver.edu/~aknyazev/ research/ eigensolvers/`
is maintained by the author. It contains an interactive discussion section, a bibliography page, and links to some software. In particular, the following software, written by the author, is available:

- MATLAB implementations of the block steepest ascent, the block global minimization method, and the block conjugate gradient methods, used for numerical experiments in the present paper.
- MATLAB codes for constructing matrices A and T suggested for benchmarking in Section 6.
- A Fortran-77 code of the method suggested in [16, 12, 17], which computes eigenvalues of the Laplacian in the L-shaped domain on a uniform mesh, using preconditioned domain decomposition Lanczos-type method.

12. Conclusion. Let us formulate here the main points of the present paper:

- The suggested Locally Optimal PCG eigensolver is a *genuine* CG method.
- Numerical results establish that our algorithm LOBPCG is practically as efficient as the "ideal" algorithm for computing an extreme eigenpair and provides approximations close to those of the Global Optimization Method on the generalized block Krylov subspace.
- The written MATLAB code of the Locally Optimal Block PCG eigensolver is *robust* and *fast*. It may be used for *extremely large* and *poorly conditioned* symmetric eigenvalue problems with an arbitrary symmetric positive definite preconditioner.
- Our method is much more robust and typically converges about one and a half – two times faster than the Jacobi–Davidson method, but may require somewhat more applications of the matrix A per iteration.
- A user can apply a favorite symmetric preconditioner for linear systems as the preconditioner in our Locally Optimal Block PCG eigensolver as well.
- We provide a system of test problems with random preconditioners that we suggest to use for benchmarking. Every new preconditioned solver for finding an extreme eigenpair should be compared with the "ideal" algorithm in terms of the speed of convergence, costs of every iterations and memory requirements. As a number of publications on different preconditioned eigenvalue solvers and their applications, e.g., just recent papers [22, 27, 28, 19, 18, 2, 21, 6], keeps growing rapidly, a need for such benchmarking becomes evident.

Acknowledgments. The author thanks several CU-Denver graduate students: Rico Argentati, Sean Jenson, Chan-Chai Aniwathananon, and Saulo Oliveira, who participated in writing and testing a MATLAB code ORTHA.m that was used for numerical tests of the present paper.

REFERENCES

[1] L. Bergamaschi, G. Gambolati, and G. Pini, *Asymptotic convergence of conjugate gradient methods for the partial symmetric eigenproblem*, Numer. Linear Algebra Appl., 4 (1997), pp. 69–84.

[2] L. Borges and S. Oliveira, *A parallel Davidson-type algorithm for several eigenvalues*, J. Comput. Phys., 144 (1998), pp. 727–748.

[3] W. W. Bradbury and R. Fletcher, *New iterative methods for solution of the eigenproblem*, Numer. Math., 9 (1966), pp. 259–267.

[4] J. H. Bramble, J. E. Pasciak, and A. V. Knyazev, *A subspace preconditioning algorithm for eigenvector/eigenvalue computation*, Adv. Comput. Math., 6 (1996), pp. 159–189.

[5] E. R. Davidson, *Matrix eigenvector methods*, in Methods in Computational Molecular Physics, G. H. F. Direcksen and S. Wilson, eds., Reidel, Boston, 1983, pp. 95–113.

[6] D. C. Dobson, *An efficient method for band structure calculations in 2D photonic crystals*, J. Comput. Phys., 149 (1999), pp. 363–376.

[7] Y. T. Feng and D. R. J. Owen, *Conjugate gradient methods for solving the smallest eigenpair of large symmetric eigenvalue problems*, Internat. J. Numer. Methods Engrg., 39 (1996), pp. 2209–2229.

[8] 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 (1999), pp. 94–125 (electronic).

[9] G. Gambolati, G. Pini, and M. Putti, *Nested iterations for symmetric eigenproblems*, SIAM J. Sci. Comput., 16 (1995), pp. 173–191.

[10] G. Gambolati, F. Sartoretto, and P. Florian, *An orthogonal accelerated deflation technique for large symmetric eigenproblems*, Comput. Methods Appl. Mech. Engrg., 94 (1992), pp. 13–23.

[11] A. V. Knyazev, *Convergence rate estimates for iterative methods for mesh symmetric eigenvalue problem*, Soviet J. Numerical Analysis and Math. Modelling, 2 (1987), pp. 371–396.

[12] ———, *A preconditioned conjugate gradient method for eigenvalue problems and its implementation in a subspace*, in International Ser. Numerical Mathematics, v. 96, Eigenwertaufgaben in Natur- und Ingenieurwissenschaften und ihre numerische Behandlung, Oberwolfach, 1990., Basel, 1991, Birkhauser, pp. 143–154.

[13] A. V. Knyazev, *New estimates for Ritz vectors*, Math. Comp., 66 (1997), pp. 985–995.

[14] A. V. Knyazev, *Preconditioned eigensolvers - an oxymoron?*, ETNA, 7 (1998), pp. 104–123.

[15] ———, *Preconditioned eigensolvers: practical algorithms*, in Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, SIAM, 2000. Accepted. An extended version published as a technical report UCD-CCM 143, 1999, at the Center for Computational Mathematics, University of Colorado at Denver.

[16] A. V. Knyazev and A. L. Skorokhodov, *Preconditioned iterative methods in subspace for solving linear systems with indefinite coefficient matrices and eigenvalue problems*, Soviet J. Numerical Analysis and Math. Modelling, 4 (1989), pp. 283–310.

[17] ———, *The preconditioned gradient-type iterative methods in a subspace for partial generalized symmetric eigenvalue problem*, SIAM J. Numerical Analysis, 31 (1994), pp. 1226–1239.

[18] C. Liu and J.-F. Lee, *Jacobi-Davidson algorithm and its application to modeling RF/microwave detection circuits*, Comput. Methods Appl. Mech. Engrg., 169 (1999), pp. 359–375.

[19] S. H. Lui, H. B. Keller, and T. W. C. Kwok, *Homotopy method for the large, sparse, real nonsymmetric eigenvalue problem*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 312–333.

[20] R. B. Morgan, *Davidson's method and preconditioning for generalized eigenvalue problems*, J. Comp. Phys., 89 (1990), pp. 241–245.

[21] B. G. Pfrommer, J. Demmel, and H. Simon, *Unconstrained energy functionals for electronic structure calculations*, J. Comput. Phys., 150 (1999), pp. 287–298.

[22] A. Ruhe, *Rational Krylov: a practical algorithm for large sparse nonsymmetric matrix pencils*, SIAM J. Sci. Comput., 19 (1998), pp. 1535–1551 (electronic).

[23] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Halsted Press, New York, 1992.

[24] E. Suetomi and H. Sekimoto, *Conjugate gradient like methods and their application to eigenvalue problems for neutron diffusion equation*, Annals of Nuclear Energy, 18 (1991), p. 205.

[25] A. E. T. Arias and S. Smith, *Curvature in conjugate gradient eigenvalue computation with applications*, in Proceedings of the 1994 SIAM Applied Linear Algebra Conference, J. Lewis, ed., Philadelphia, 1994, SIAM, pp. 233–238.

[26] H. Yang, *Conjugate gradient methods for the Rayleigh quotient minimization of generalized eigenvalue problems*, Computing, 51 (1993), pp. 79–94.

[27] T. Zhang, G. H. Golub, and K. H. Law, *Subspace iterative methods for eigenvalue problems*, Linear Algebra Appl., 294 (1999), pp. 239–258.

[28] T. Zhang, K. H. Law, and G. H. Golub, *On the homotopy method for perturbed symmetric generalized eigenvalue problems*, SIAM J. Sci. Comput., 19 (1998), pp. 1625–1645 (electronic).