TRACE-PENALTY MINIMIZATION FOR LARGE-SCALE EIGENSPACE COMPUTATION

ZAIWEN WEN[†], CHAO YANG[‡], XIN LIU[§], AND YIN ZHANG[¶]

Abstract. In a block algorithm for computing a relatively high-dimensional eigenspaces of large sparse symmetric matrices, the Rayleigh-Ritz (RR) procedure often constitutes a major bottleneck. Although dense eigenvalue calculations for subproblems in RR steps can be parallelized to a certain level, their parallel scalability, which is limited by some inherent sequential steps, is lower than dense matrix-matrix multiplications. The primary motivation of this paper is to develop a methodology that reduces the use of the RR procedure in exchange for matrix-matrix multiplications. We propose an unconstrained trace-penalty minimization model and establish its equivalence to the eigenvalue problem. With a suitably chosen penalty parameter, this model possesses far fewer undesirable full-rank stationary points than the classic trace minimization model. More importantly, it enables us to deploy algorithms that makes heavy use of dense matrix-matrix multiplications. Although the proposed algorithm does not necessarily reduce the total number of arithmetic operations, it leverages highly optimized operations on modern high performance computers to achieve parallel scalability. Numerical results based on a preliminary implementation, parallelized using OpenMP, show that our approach is promising.

Key words. eigenvalue computation, exact quadratic penalty approach, gradient methods

AMS subject classification. 15A18, 65F15, 65K05, 90C06

1. Introduction.

Eigenvalue and eigenvector calculation is a fundamental computational problem with extraordinarily wide-ranging applications. In the past several decades, a great deal of progress has been made in the development of efficient algorithms and solvers for various types of eigenvalue problems. Iterative methods are usually preferred for solving large-scale problems because of their ability to take advantage of sparsity or other structures existing in the matrices of interest. When a few eigenpairs are needed, the task of sparse matrix-vector multiplications (SpMVs), which can often be performed efficiently on sequential computers, usually constitutes the dominant computational cost. However, as the number of desired eigenpairs increases, the computational costs in an iterative eigensolver can shift to other linear algebra operations. Furthermore, when the eigenvalue computation is carried out on parallel computers with many processors, making SpMV and other dense linear algebra operations scalable is extremely important.

The difficulty we face here is not just a computational complexity issue. On modern multicore computers, memory access pattern and communication overhead must also be taken into account. Although standard Krylov subspace iterative algorithms such as the Lanczos algorithm can be parallelized through the parallelization of the sparse matrix vector multiplications (SpMV) and other dense linear algebra operations, the amount of parallelism is limited because SpMVs must be done in sequence in these algorithms, and each SpMV can only make effective use of a limited number of processing units in general. The dense linear

[†]Beijing International Center for Mathematical Research (BICMR), Peking University, Beijing, CHINA (wenzw@math.pku.edu.cn). Research supported in part by NSFC grants 11101274, 11322109 and 91330202.

[‡]Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, UNITED STATES (cyang@lbl.gov). Support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences) under award number DE-SC0008666.

[§]State Key Laboratory of Scientific and Engineering Computing, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, CHINA (liuxin@lsec.cc.ac.cn). Research supported in part by NSFC grant 11101409 and 10831006, and the National Center for Mathematics and Interdisciplinary Sciences, CAS.

[¶]Department of Computational and Applied Mathematics, Rice University, Houston, UNITED STATES (yzhang@rice.edu). Research supported in part by NSF Grant DMS-0811188, ONR Grant N00014-08-1-1101, and NSF Grant DMS-1115950.

algebra operations are limited to level-2 BLAS operations. These issues are also pertinent to the Jacobi-Davidson (JD) type of algorithms in which eigenvalues are computed one at a time. Block methods, such as the locally optimal block preconditioned conjugate gradient (LOBPCG) algorithm [8], the block Krylov-Schur algorithm [22] and the Chebyshev-Davidson algorithm [20, 21], are more scalable because more concurrency can be exploited in multiplying a sparse matrix with a block of vectors and there are opportunities to use level-3 BLAS to perform some of the dense linear algebra operations.

However, there are two types of operations that can potentially become bottlenecks in a block method. One is the construction and/or maintenance of orthonormal bases for subspaces from which approximate eigenvalues and eigenvectors are extracted at each iteration. This type of operations is often carried out through either a Gram-Schmidt procedure (e.g. in the Lanczos algorithms) or a QR factorization. Another potentially high-cost procedure is the extraction of k pairs of eigenvalue and eigenvector approximations from a subspace of dimension $p \geq k$. This procedure involves solving a p-dimensional dense eigenvalue problem and assembling the so-called Ritz vectors which are approximate eigenvectors in the original space. Because the Ritz vectors are mutally orthonormal, this procedure can sometimes be viewed as a way to construct an orthonormal basis. However, in this paper, we will distinguish RR from standard basis orthonormalization. The later can be made efficient by using a proper algorithm whose computation is dominated by level-3 BLAS (BLAS3), whereas the need to solve a dense eigenvalue problem in the RR procedure makes the performance optimization of RR difficult. When the number of desired eigenvalues k is small, the cost of solving a $k \times k$ dense eigenvalue problem is minor or even negligible relative to other computations. However, when k increases to a moderate portion of the matrix dimension n, solving the dense eigenvalue problem can represent a significant, or even dominant, portion of the overall cost. This is partly due to the fact that algorithms for computing eigenvalues and eigenvectors of a projected dense matrix are not easy to parallelize. Although parallel algorithms are available in multi-thread LAPACK [1] libraries for shared-memory parallel computers and in the ScaLAPACK [3] library for distributed-memory parallel computers, the parallel efficiency of these algorithms is often limited to a relatively small number of processors or cores. When a large number of processing units are involved, the thread or communication overhead can be significant.

In this paper, we present a block algorithm for computing k algebraically smallest eigenvalues of a real symmetric matrix $A \in \mathbb{R}^{n\times n}$ and their corresponding eigenvectors, though the same methodology can easily be applied to compute the largest eigenvalues and to compute eigenvalues of complex Hermitian matrices. Our approach reformulates the trace minimization model for eigenvalue problems. It is well known that the invariant subspace associated with a set of k algebraically smallest eigenvalues of A yields an optimal solution to the following trace minimization problem with orthogonality constraints

$$\min_{X\in\mathbb{R}^{n\times k}} \operatorname{tr}(X^{\mathrm{T}}AX), \text{ s.t. } X^{\mathrm{T}}X = I. \tag{1.1}$$

A main theoretical result of this paper is to establish an equivalence relationship between problem (1.1) and the following unconstrained optimization problem

$$\min_{X\in\mathbb{R}^{n\times k}} f_\mu(X) := \frac{1}{2}\operatorname{tr}(X^{\mathrm{T}}AX) + \frac{\mu}{4}\|X^{\mathrm{T}}X - I\|_F^2, \tag{1.2}$$

when the penalty parameter $\mu > 0$ takes suitable finite values. As is well recognized, the objective function in (1.2) is the classic quadratic (or Courant) penalty function [4, 11, 16] for the constrained problem (1.1). Generally speaking, the classic quadratic penalty model approaches the original constrained problem only as the penalty parameter μ goes to infinity.

However, we show that problem (1.2) is essentially equivalent to (1.1) in terms of finding an optimal eigenspace and it excludes all full-rank non-optimal stationary points when the penalty parameter μ is appropriately chosen.

We will call the approach of solving model (1.2) *trace-penalty minimization*. A key difference between trace minimization model (1.2), which can be solved by a number of algorithms such as the LOBPCG and the TRACEMIN algorithm proposed in [13], and trace-penalty minimization is that explicit orthogonality of X is no longer required in the latter, which immediately opens up the possibility of doing far fewer RR steps including far fewer orthogonalizations and other RR-related operations. In exchange, as will be demonstrated later, more dense matrix-matrix multiplications are performed (to a less extent, also more SpMV). A major potential advantage of replacing RR steps by dense matrix-matrix multiplications is that the latter operations have much better parallel scalability and are highly optimized for modern high performance computers. In addition, one could incorporate preconditioning into trace-penalty minimization in a straightforward manner.

We consider applying gradient-type methods to the trace-penalty minimization problem (1.2). These methods can often quickly reach the vicinity of an optimal solution and produce a moderately accurate approximation. In many applications, rough or moderately accurate approximations are often sufficient. One of such instances is when solving a nonlinear eigenvalue problem, one approximately solves a sequence of linearized eigenvalue problems one after another (such as in solving the Kohn-Sham equation in electronic structure calculation by "self-consistent field" iterations [12]). Once good estimates are at hand, there exist a number of techniques that can perform further refinements to obtain a higher accuracy. For the proposed trace-penalty minimization problem (1.2), we have experimented with various algorithmic options in Matlab, developed a Fortran implementation and parallelized it using OpenMP. Preliminary numerical comparison with some of the existing approaches shows that our approach is promising.

The rest of this paper is organized as follows. We analyze the trace-penalty minimization model in Section 2. Our algorithms and several implementation details are discussed in Section 3. The cost analysis and comparison are provided in Section 4. Numerical results are reported in Section 5. Finally, we conclude the paper in Section 5.

2. Trace-Penalty Minimization: Model Analysis.

2.1. Equivalence Between Trace and Trace-Penalty Minimizations.

For a given real symmetric matrix $A = A^T \in \mathbb{R}^{n\times n}$, an eigenvalue decomposition of A is defined as

$$A = Q_n \Lambda_n Q_n^{\mathrm{T}}, \tag{2.1}$$

where, for any integer $i \in [1, n]$,

$$Q_i = [q_1,\ q_2,\ \ldots, q_i] \in \mathbb{R}^{n\times i}, \quad \Lambda_i = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_i) \in \mathbb{R}^{i\times i}, \tag{2.2}$$

so that $Q_i^{\mathrm{T}} Q_i = I \in \mathbb{R}^{i\times i}$ and Λ_i is diagonal. The columns $q_1, \ldots, q_n$ of Q_n are eigenvectors of A associated with eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$, respectively, which are assumed to be in an ascending order,

$$\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n.$$

We note the non-uniqueness of eigenvalue decomposition (2.1). One could not only alter the signs of eigenvectors, but also choose different unit eigenvectors associated with eigenvalues of multiplicity greater than one. For convenience, we will treat (2.1) as a generic form of decomposition that represents all possible alternatives.

Given a positive integer $k \leq n$, it is well known that the eigenvector matrix Q_k is a solution to the trace minimization problem (1.1). As is stated in the introduction, instead of solving (1.1) directly, we propose to solve the trace-penalty minimization problem (1.2). These two problems are said to be *equivalent* if each of its global minimizers spans a k-dimensional eigenspace associated with k smallest eigenvalues of A. The main theoretical result is that (1.2) is equivalent to (1.1) when the penalty parameter μ satisfies $\mu > \max(0, \lambda_k)$. That is, μ does not need to be set to an extremely large value, which could cause numerical difficulty in solving the minimization problem.

THEOREM 2.1. *Problem (1.2) is equivalent to (1.1) if and only if*

$$\mu > \max(0, \lambda_k). \tag{2.3}$$

Specifically, any global minimizer $\hat{X}$ of (1.2) *has a singular-value decomposition of the form:*

$$\hat{X} = Q_k(I - \Lambda_k/\mu)^{1/2} V^{\mathrm{T}} \tag{2.4}$$

where Q_k and Λ_k are defined as in (2.2)*, and $V \in \mathbb{R}^{k\times k}$ is any orthogonal matrix.*

The proof of Theorem 2.1 is given in the Appendix A.1. In the next theorem, we show that the trace-penalty minimization model (1.2) can have far fewer undesirable, full-rank stationary points than the trace minimization model (1.1). Hence, when the penalty parameter is suitably chosen, one could reasonably argue that from an optimization point of view trace-penalty minimization is theoretically more desirable than trace minimization.

THEOREM 2.2. *If $\mu \in (\max(0, \lambda_k), \lambda_n)$, then $f_\mu(X)$ has no local maxima, nor local minima other than the global minimum attained by $\hat{X}$ defined in (2.4). Moreover, if $\mu \in (\max(0, \lambda_k), \lambda_{k+p})$ where λ_{k+p} is the smallest eigenvalue greater than λ_k, then all k-dimensional stationary points of $f_\mu(X)$ must be global minimizers.*

The proof of Theorem 2.2 is given in the Appendix A.2.

2.2. Optimality Conditions and the Condition Number of the Hessian. The first-order necessary condition for trace-penalty minimization (1.2) is simply

$$\nabla f_\mu(X) = AX + \mu X(X^{\mathrm{T}}X - I) = 0. \tag{2.5}$$

Given any approximate solution X of (1.2). Let $Y(X)$ be any orthonormal basis of the subspace spanned by columns of X. The violation of the first-order necessary conditions of the trace minimization (1.1) can be measured by the Frobenious norm of the residual

$$R(X) \triangleq AY(X) - Y(X)\left(Y(X)^{\mathrm{T}}AY(X)\right).$$

It can be shown that

$$\|R(X)\|_F \leq \sigma_{\min}^{-1}(X)\,\|\nabla f_\mu(X)\|_F,$$

Moreover, for any global minimizer $\hat{X}$ and any $\epsilon > 0$, there exists $\delta > 0$ such that whenever $\|X - \hat{X}\|_F \leq \delta$,

$$\|R(X)\|_F \leq \frac{1+\epsilon}{\sqrt{1-\lambda_k/\mu}}\|\nabla f_\mu(X)\|_F. \tag{2.6}$$

The Fréchet derivative of ∇f_μ at X, or the Hessian of $f_\mu(X)$, has the operator form

$$\nabla^2 f_\mu(X)(S) = AS + \mu S(X^{\mathrm{T}}X - I) + \mu X(S^{\mathrm{T}}X + X^{\mathrm{T}}S). \tag{2.7}$$

The condition number of the Hessian at a global minimizer $\hat{X}$ of (1.2) is defined as

$$\kappa(\nabla^2 f_\mu(\hat{X})) = \lambda_{\max}(\nabla^2 f_\mu(\hat{X}))/\lambda_{\min}(\nabla^2 f_\mu(\hat{X})),$$

where $\lambda_{\max}(\cdot)$ (or $\lambda_{\min}(\cdot)$) stands for the largest (or the smallest) eigenvalue of the referred matrix. Obviously, κ is infinity when the involved matrix is singular.

If $k = 1$ and $\lambda_1 < \lambda_2$, according to Theorem 2.1 there exists exactly two isolated global minimizers for (1.2) at which the Hessian of f_μ is nonsingular. If $\lambda_1 = \lambda_2$, however, the Hessian of f_μ becomes singular throughout the solution set since the multiplicity is greater than one. In the case of $k > 1$, it follows from Theorem 2.1 that there is no isolated global minimizer. Hence, the Hessian at any solution is singular. The following result examines the restricted condition number of the Hessian outside of the optimal eigenspace.

LEMMA 2.3. *Let $k > 1$, $\mu > \max(0, \lambda_k)$ and $\hat{X}$ be any global minimizer of f_μ in* (1.2). *Then*

$$\kappa\left(\nabla^2 f_\mu(\hat{X})\Big|_{Q_k^\perp}\right) \triangleq \frac{\max\limits_{S \in Q_k^\perp} \left\{\mathrm{tr}(S^{\mathrm{T}} \nabla^2 f_\mu(\hat{X})(S))\right\}}{\min\limits_{S \in Q_k^\perp} \left\{\mathrm{tr}(S^{\mathrm{T}} \nabla^2 f_\mu(\hat{X})(S))\right\}} = \frac{\lambda_n - \lambda_1}{\lambda_{k+1} - \lambda_k}, \tag{2.8}$$

where $Q_k^\perp = \{S \in \mathbb{R}^{n\times k} \mid \mathrm{tr}(S^{\mathrm{T}}S) = 1, S^{\mathrm{T}}Q_k = 0\}$.

The proof of Lemma 2.3 is given in Appendix A.3. Not surprisingly, one can expect some difficulty arising from the existence of a narrow gap $\lambda_{k+1} - \lambda_k$ (also known as a cluster at a critical location) relative to the total spectrum length $\lambda_n - \lambda_1$. This difficulty represents a common challenge to eigensolvers in general. However, since the extreme cases are attained by rank-one matrices (see the proof in Appendix A.3), the worst-case conditioning in (2.8) is unlikely to happen in practice as converging iterates remain full rank near a solution.

2.3. Extensions. It is not difficult to see that our analysis in this section, as well as the algorithmic framework described in the next section, can be extended to the generalized eigenvalue problem:

$$\min_{X \in \mathbb{R}^{n\times k}} \mathrm{tr}(X^{\mathrm{T}}AX), \text{ s.t. } X^{\mathrm{T}}BX = I, \tag{2.9}$$

where B is symmetric and positive definite. In this case, the trace-penalty minimization model is simply

$$\min_{X \in \mathbb{R}^{n\times k}} f_\mu(X) := \frac{1}{2}\mathrm{tr}(X^{\mathrm{T}}AX) + \frac{\mu}{4}\|X^{\mathrm{T}}BX - I\|_F^2. \tag{2.10}$$

Another useful extension is to find eigenvectors in the orthogonal complement of the column-space of a given U such that $U^{\mathrm{T}}U = I$, that is:

$$\min_{X \in \mathbb{R}^{n\times k}} \mathrm{tr}(X^{\mathrm{T}}AX), \text{ s.t. } X^{\mathrm{T}}BX = I, \;\; U^{\mathrm{T}}X = 0. \tag{2.11}$$

The variation (2.11) can arise from a deflation procedure where U is constructed from already converged eigenvectors. The trace-penalty minimization model corresponding to (2.11) is

$$\min_{X \in \mathbb{R}^{n\times k}} f_\mu(X) := \frac{1}{2}\mathrm{tr}(X^{\mathrm{T}}AX) + \frac{\mu}{4}\|X^{\mathrm{T}}BX - I\|_F^2, \text{ s.t. } U^{\mathrm{T}}X = 0. \tag{2.12}$$

3. Algorithmic Framework.

3.1. Gradient Methods for Trace-Penalty Minimization. The model of trace-penalty minimization proposed in the previous section is an unconstrained nonconvex minimization problem. There are many well-studied approaches for this problem, such as the steepest descent gradient, the conjugate gradient, the Newton's and Quasi-Newton methods. Considering the scale of the eigenvalue computation of interest, in this paper we focus on the gradient-type methods of the form:

$$X^{j+1} = X^j - \alpha^j \nabla f_\mu(X^j), \tag{3.1}$$

where the superscript j denotes the j-th iteration and α^j is the step size.

Although the penalty function may have multiple stationary points, Theorem 2.2 shows that when μ is chosen slightly above $\lambda_k \geq 0$, then rank-k stationary points are likely to be all global minimizers. The following proposition suggests that the iterations generated by (3.1) will most likely remain full rank.

PROPOSITION 3.1. *Let X^{j+1} be generated by* (3.1) *from a full-rank iterate X^j. Then X^{j+1} is rank-deficient only if $1/\alpha^j$ is one of the k generalized eigenvalues of the problem:*

$$[(X^j)^{\mathrm{T}} \nabla f_\mu(X^j)]u = \lambda[(X^j)^{\mathrm{T}}(X^j)]u. \tag{3.2}$$

On the other hand, if $\alpha^j < \sigma_{\min}(X^j)/\|\nabla f_\mu(X^j)\|_2$, then X^{j+1} is of full rank.

The proof of Proposition 3.1 is given in the Appendix A.4.

We next present a strategy for choosing α^j using a Barzilai-Borwein (BB) size [2]. Let

$$S^j := X^j - X^{j-1} \quad \text{and} \quad Y^j = \nabla f_\mu(X^j) - \nabla f_\mu(X^{j-1}). \tag{3.3}$$

The BB step size is

$$\alpha^j_{\mathrm{BB1}} = \mathrm{tr}((S^j)^{\mathrm{T}} Y^j)\|Y^j\|_{\mathrm{F}}^2/ \quad \text{or} \quad \alpha^j_{\mathrm{BB2}} = \|S^j\|_{\mathrm{F}}^2/\mathrm{tr}((S^j)^{\mathrm{T}} Y^j). \tag{3.4}$$

Since $S^j = \alpha^{j-1}\nabla f_\mu(X^{j-1})$, the computation of the BB step sizes only requires storing one intermediate matrix Y^j in (3.3). When n and k are huge or when storage becomes a critical factor, one can still compute a so-called partial BB step size by using (3.4) but with a pre-selected small subset of columns of both S^j and Y^j, making the storage of an extra Y-matrix unnecessary. A simple heuristic line search scheme that we will use is to shorten the step size, whenever necessary, by back-tracking $\alpha^j = \alpha\delta^h$, where α is one of the BB step sizes in (3.4), $\delta \in (0, 1)$ and h is the smallest positive integer satisfying the condition

$$f_\mu(X^j - \alpha\delta^h \nabla f^j_\mu) \leq 2 f_\mu(X^j). \tag{3.5}$$

It is known that certain global convergence properties can be guaranteed in theory by more elaborate line search conditions such as non-monotone line search conditions in [5, 6, 19]. We have found, however, that on our trace-penalty function $f_\mu(X)$ condition (3.5) has performed efficiently and reliably.

At the end of trace-penalty minimization, a Rayleigh-Ritz (RR) step is necessary to compute Ritz-pairs as approximations to eigenpairs. Specifically, in our context the RR step corresponding to a given matrix $X \in \mathbb{R}^{n\times k}$ is defined by the following steps.

1. Orthogonalize and normalize X to obtain U so that $U^{\mathrm{T}}U = I$.
2. Compute the projection $U^{\mathrm{T}}AU$ and its eigenvalue decomposition $V^{\mathrm{T}}\Sigma V$.
3. Assemble the Ritz-pairs into the matrix-pair (Y, Σ) where $Y = UV$.

For convenience, we will refer the above RR procedure as a map $(Y, \Sigma) = \mathrm{RR}(X)$.

In Algorithm 1 below, we specify a basic version of a method for trace-penalty minimization, called "EigPen-B", which uses the first BB step formula in (3.4), the simple line search condition (3.5), and a termination rule

$$\|\nabla f_\mu(X^j)\|_{\mathrm{F}} \le \epsilon, \tag{3.6}$$

where $\epsilon > 0$ is a prescribed tolerance.

Algorithm 1: Eigenspace by Penalty – basic version (EigPen-B)

```
Initialize $X^0 \in \mathbb{R}^{n\times k}$ and estimate $\mu \in (\lambda_k, \lambda_n)$. Set $\epsilon, \delta \in (0,1)$ and $j = 0$.
Compute initial step $\alpha = \|X^0\|_F / \|\nabla f_\mu(X^0)\|_F$ .
while $\|\nabla f_\mu(X^j)\|_F > \epsilon$ do
    compute the smallest natural number $h$ so that $\alpha^j = \alpha\delta^h$ satisfying (3.5);
    update $X^{j+1} = X^j - \alpha^j \nabla f_\mu(X^j)$;
    compute $\alpha$ using the first formula in (3.4);
    increment $j$ and continue.
Execute the RR procedure $(X, \Sigma) = \mathrm{RR}(X^j)$.
```

The memory requirement of Algorithm 1 is as follows. Four n by k matrices, X, AX, $\nabla f_\mu(X)$ and Y defined in (3.3), are required. As mentioned earlier, the need for storing Y can be essentially eliminated if partial BB step sizes are computed (without obvious performance degradation in our experiments).

3.2. Enhancement by Restarting. Algorithm EigPen-B often works quite well in practice. However, a typical behavior of gradient methods is that they can reduce the objective function rather rapidly at an initial stage, but the amount of reduction can become extremely small as iterates get closer to a solution. In trace-penalty minimization, it has been observed that restarting the gradient method with a modified X can usually help accelerate convergence and achieve a higher accuracy more quickly. In this subsection, we describe a restarting strategy for trace-penalty minimization that utilizes more than one RR step. In addition to accelerating convergence, the restarting strategy provides a more reliable termination procedure by examining more than one set of Ritz-pairs.

We now demonstrate how RR steps can help speed up trace-penalty minimization. Let

$$Y = \underset{X\in\mathbb{R}^{n\times k}}{\operatorname{argmin}} \{f_\mu(X) : X \in \mathcal{S}\}, \tag{3.7}$$

where $X \in \mathcal{S}$ means that every column of X is in the subspace $\mathcal{S}$. Let X^J be the iterate generated by the EigPen-B algorithm after J iterations. Clearly, as long as $X^J \in \mathcal{S}$ there holds

$$f_\mu(Y) \le f_\mu(X^J). \tag{3.8}$$

On the other hand, consider the subspace trace minimization problem

$$U = \underset{X\in\mathbb{R}^{n\times d}}{\operatorname{argmin}} \left\{ \mathrm{tr}(X^{\mathrm{T}}AX) : X^{\mathrm{T}}X = I,\ X \in \mathcal{S} \right\}, \tag{3.9}$$

where d is the dimension of the subspace $\mathcal{S}$. Clearly, the RR step $(U, \Sigma) = \mathrm{RR}(X^J)$ is equivalent to solving (3.9) for $\mathcal{S} =$ **span**$\{X^J\}$ (assuming that $X^J \in \mathbb{R}^{n\times k}$ has full rank) so that $U^{\mathrm{T}}AU = \Sigma$ is diagonal (otherwise, replace U by UV where $U^{\mathrm{T}}AU = V\Sigma V^{\mathrm{T}}$).

We now show that a “better point” Y for trace-penalty minimization in (3.7) and (3.8) can be explicitly constructed from the RR step output $(U, \Sigma) = \mathrm{RR}(X^J)$.

LEMMA 3.2. *Let $\mathcal{S} \supseteq$ **span**$\{X^J\}$ have dimension $d \geq k$, and U be defined in* (3.9) *so that $U^{\mathrm{T}}AU = \Sigma$ is diagonal whose diagonal elements are arranged in an ascending order. Then a matrix Y in* (3.7) *has the form $Y = U_k D$ where U_k consists of the first k columns of U, and $D \in \mathbb{R}^{k\times k}$ is a diagonal matrix whose i-th diagonal element is*

$$D_{ii} = \max\left(0, 1 - \Sigma_{ii}/\mu\right)^{1/2}, \; i = 1, 2, \cdots, k. \tag{3.10}$$

The proof of Lemma 3.2 is given in the Appendix A.5.

In Algorithm 2 below, we present our trace-penalty minimization algorithm with restarting, which is used to perform numerical experiments presented in the next section. The algorithm, called EigPen, contains two loops. The inner loop is stopped once the condition

$$\|\nabla f_\mu(X^j)\|_{\mathrm{F}} \leq \epsilon_i \;\max(1, \|AX^j\|_{\mathrm{F}}) \tag{3.11}$$

is met, where $\epsilon_i \in (0, 1)$ is a prescribed tolerance. Then an RR step is executed to construct Ritz-pairs and termination criteria are checked for the outer loop. If the algorithm does not stop, then a smaller tolerance $\epsilon_{i+1} = \delta_\epsilon \epsilon_i$ is set where $\delta_\epsilon \in (0, 1)$, and a better iterate is constructed from which the algorithm restarts the next round of inner iterations by calling EigPen-B.

Algorithm 2: Eigenspace by Penalty – enhanced version (EigPen)

```
Initialize $\bar{X}^0 \in \mathbb{R}^{n\times k}$ and estimate $\mu \in (\lambda_k, \lambda_n)$. Set $\epsilon_0, \delta, \delta_\epsilon \in (0,1)$ and $i = j = 0$.
while "not converged" do
    Set $X^j = \bar{X}^i$ and compute $\alpha = \|X^j\|_F / \|\nabla f_\mu(X^j)\|_F$.
    while $\|\nabla f_\mu(X^j)\|_F > \epsilon_i \cdot \max(1, \|AX^j\|_F)$ do
        the same steps in the loop of Algorithm 1.
    Execute the RR procedure $(X, \Sigma) = \mathrm{RR}(X^j)$ and let $\bar{X}^{i+1} = X(I - \Sigma/\mu)^{\frac{1}{2}}$.
    Update the tolerance $\epsilon_{i+1} = \delta_\epsilon \epsilon_i$, the penalty parameter $\mu$, and increment $i$.
```

The RR restart approach allows flexibility to integrate other techniques into EigPen. For example, at the j-th iteration, if one chooses the subspace $\mathcal{S}$ in problem (3.9) to be $\mathcal{S} =$ **span** $\left\{X^{j-1}, X^j, AX^j\right\}$, then the RR step would generate a step similar to those in the LOBPCG algorithm [8]. A key difference between LOBPCG and EigPen is that RR steps constitute the main workhorse of the former, but are utilized only a few times in the latter.

3.3. Penalty Parameter Adjustment. We now describe our approach to choosing the penalty parameter μ. Theorem 2.1 states that $\mu > \max(0, \lambda_k)$ is necessary and sufficient for the equivalence between (1.1) and (1.2). A more restrictive range for μ is given in Theorem 2.2 that eliminates all full-rank stationary points but the global minimizers. However, it requires the extra work of estimating, at the least, λ_{k+1}. On the other hand, μ should not be too close to λ_k, otherwise ill-conditioning could arise in trace-penalty minimization. On balance, we adopt a tractable strategy of choosing $\mu > \lambda_k$ (which is positive after a shifting if necessary) and keeping it reasonably close to λ_k, without attempting to make μ smaller than the next smallest eigenvalue.

Given an initial matrices $X^0 \in \mathbb{R}^{n\times k}$ whose columns are normalized, the kth smallest eigenvalue λ_k can be estimated by the maximal value of the diagonal entries of $(X^0)^{\mathrm{T}}AX^0$, which provides an initial choice

$$\mu = \max(c_1,\, c_2 \max(\mathrm{diag}((X^0)^{\mathrm{T}} A X^0))), \tag{3.12}$$

where $c_1 > 0$ and $c_2 > 1$ are two constants for safeguarding. A more accurate estimate of λ_k becomes available after an RR step is executed and the k-th Ritz-value θ_k is at hand. Then we use the formula

$$\mu = \max(c_1,\, c_2\theta_k). \tag{3.13}$$

4. Cost Analysis and Comparison. In this section, we analyze the cost of EigPen, and show why it can be much faster than some of the existing algorithms such as the LOBPCG algorithm used to solve the trace minimization problem and a Jacobi-Davidson type of algorithm in which one eigenvalue is computed at a time using an inexact Newton's method. Our cost model is not based solely on flop count or the asymptotic complexity of the algorithm, which is the same among all algorithms we consider here. It is based on an estimation of the time to solution (TTS), and its change with respect to changes in the relative costs of different computational components of the algorithm. Such relative cost varies with respect to the number of eigenvalues to be computed and the scalabilities of each computational component.

The advantage of using TTS estimation is that it takes into account flop count, memory access efficiency and parallel scalability. The drawback is that a precise cost model is difficult to establish in the most general setting because it is difficult to quantify different contributing factors to TTS in a single model. We use a simplified model here and give some typical scenarios and possibilities to illustrate the potential benefit of EigPen. We give more concrete examples in the next section to compare the performance of EigPen to that of two different solvers. The actual performance of EigPen relative to other algorithms depends on how some of the computational components are implemented, and architecture specific variables that are difficult to quantify with simple parameters in a cost model.

To simplify our analysis, we assume that the TTS (in an arbitrary time unit) for an algorithm M, where M is either EigPen, LOBPCG or Jacobi-Davidsion (JD), takes the form

$$T_{\mathrm{M}} = \alpha t_{\mathrm{spmv}} + \beta t_{\mathrm{blas}} + \gamma t_{\mathrm{rr}}, \tag{4.1}$$

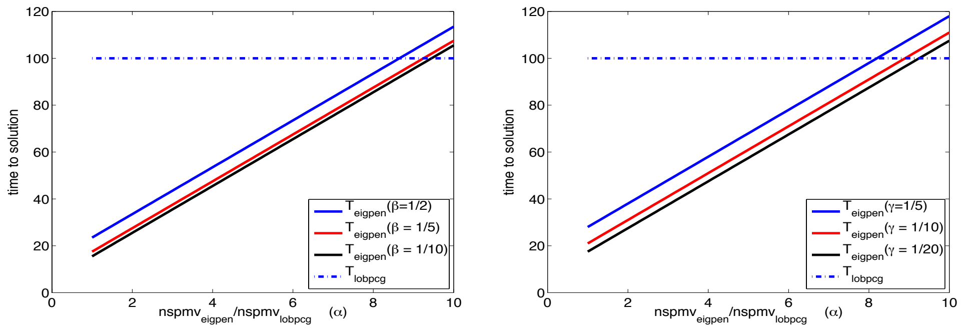
where αt_{spmv} is the total time used to perform SpMV, βt_{blas} is the time used to perform various types of BLAS operations, and γt_{rr} is the time used to solve the projected dense eigenvalue problem required in the Rayleigh-Ritz procedure. The parameters α, β, γ account for the prefactors in flop counts, the efficiency of memory access and parallel scalability. They allow us to see how the change of each individual timing component impacts the change in T_{M}. When we compare EigPen with another algorithm (either LOBPCG or JD), we first determine the percentage time spent in each computational component and set $\alpha = \beta = \gamma = 1$ for the algorithm we compare with. We then try to estimate the values of α, β and γ for EigPen, based on an estimation of the EigPen time spent in each computational component, relative to that spent in the algorithm we compare with.

We now examine a scenario in which $t_{\mathrm{spmv}} = 10$, $t_{\mathrm{blas}} = 20$ and $t_{\mathrm{rr}} = 70$ for LOBPCG. This type of timing breakdown is observed for test problems presented in the next section. Because EigPen uses a steepest descent type of method, its convergence rate can be lower than that of LOBPCG. More iterations (hence more SpMVs) may be needed to reach convergence. In this case, the parameter α corresponds to the ratio of the number of SpMVs performed in EigPen ($\mathrm{nspmv}_{\mathrm{eigpen}}$) over that performed in LOBPCG ($\mathrm{nspmv}_{\mathrm{logpcg}}$). We observe that with a few exceptions, this number is between 1 and 5 in the performance study we present in the next section.

Because the BLAS operations in EigPen typically involves matrices of dimension $n \times k$, whereas those in LOBPCG involves matrices of dimension $n \times 3k$, the BLAS operations performed in each LOBPCG iteration can be more costly than those in EigPen. On the other hand, since more EigPen iterations are needed to reach convergence, more BLAS calls are made. In the tests we performed, we observe the values of β, which measures the ratio of EigPen BLAS time over that of LOBPCG's, to be typically around $1/5$. In some cases, it is less than $1/10$. The real benefit of EigPen compared with LOBPCG is in the reduced RR cost, i.e., γ, which measure the ratio of EigPen RR cost over that of LOBPCG's, is typically much smaller than 1. In our test, the largest γ value we observed is $1/16$. In some cases, the value is smaller than $1/500$.

Figure 1 shows the how the TTS of EigPen changes with respect to the changes in the ratio $\alpha = \mathrm{nspmv}_{\mathrm{eigpen}}/\mathrm{nspmv}_{\mathrm{lobpcg}}$ for different choices of β and γ. The reference LOBPCG TTS is plotted as a horizontal blue dash-dotted line. When the plotted EigPen TTS is below this line, EigPen is faster than LOBPCG even if it performs more SpMVs.

We can see from Figure 1(b) that if we take $\beta = 1/5$ and $\gamma = 1/10$, which is a typical scenario observed in the next section, T_{eigpen} can be less than half of T_{lobpcg} even when α is around 3. The crossover point at which the TTS for two methods become almost the same occurs at roughly $\alpha = 9$, i.e., EigPen would take roughly the same time to reach convergence as LOBPCG even if it performs nine times as many SpMVs. For a fixed γ value as in Figure 1(a), a smaller β shifts the TTS curve to the right, which allows more EigPen iterations to be taken before it becomes less competitive to LOBPCG. The amount of shift depends on the percentage of LOBPCG time used to perform BLAS operations. In this particular scenario, the amount of shift becomes negligibly small when β is less than 1/5. Similarly, for a fixed β as in Figure 1(b), a smaller γ also shifts the TTS curve to the right. The amount of shift becomes negligibly small when γ is less than 1/20.



(a) $\gamma = 1/20$ (b) $\beta = 1/5$

FIG. 1. *A comparison of the* TTS *of* EigPen *(*T_{eigpen}*) with that of* LOBPCG *(*T_{lobpcg}*) for a specific scenario in which* $t_{\mathrm{spmv}} = 10$*,* $t_{\mathrm{blas}} = 20$*, and* $t_{\mathrm{rr}} = 70$*, and different choices of* α*,* β*,* γ *for* EigPen.

We now examine another scenario in which $t_{\mathrm{spmv}} = 45$, $t_{\mathrm{blas}} = 50$ and $t_{\mathrm{rr}} = 5$. This corresponds to the TTS for a JD type of algorithm in which the desired eigenvalues are computed in groups, and within each group one eigenpair is computed at a time. In this case t_{rr} should constitute a small percentage of the TTS because the RR procedure is performed within subspaces of relatively small dimensions, whereas t_{spmv} and t_{blas} take a larger share of the TTS.

When we compare the TTS of EigPen with that of JD, we can again assume that the ratio of the number of SpMVs performed in EigPen ($\mathrm{nspmv}_{\mathrm{eigpen}}$) over that performed in JD

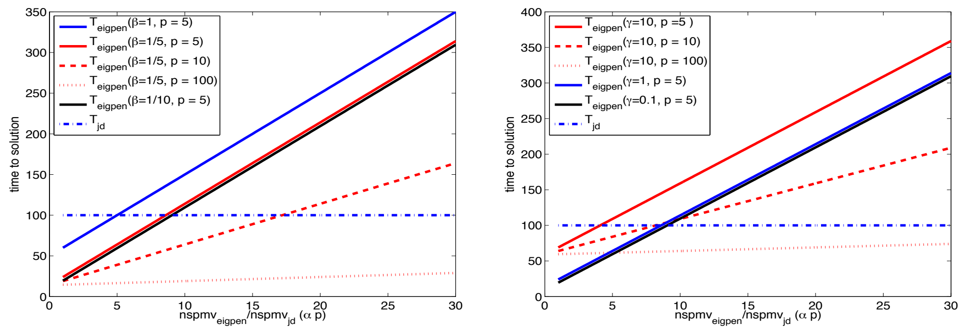
$(\mathrm{nspmv}_{\mathrm{jd}})$ is larger than 1. However, since a few eigenpairs are computed at a time in a JD type of algorithm, the concurrency that can be achieved in SpMV is limited. Therefore, the parameter α for EigPen should be adjusted to account for the lack of full concurrency. We set it to $\alpha = \mathrm{nspmv}_{jd}/\mathrm{nspmv}_{\mathrm{eigpen}}/p$, where $p > 1$ is chosen to represent the impact of higher SpMV concurrency in EigPen.

The BLAS operations in JD are used to perform basis orthogonalization as well as constructing Ritz vectors. The former consists of mostly level 2 BLAS operations, and its performance typically lags behind BLAS3 operations. The latter consists of dense matrix-matrix multiplications of a $n \times k$ matrix with a $k \times k$ matrix for $k \ll n$. The performance of this type of BLAS3 operations (in terms of flops per second ratio) is typically lower than that with a larger k. A larger k also allows more concurrency to be exploited. Thus the BLAS operations in a JD algorithm can be more costly than those in EigPen. In our experiments, the observed values of β are between 0.1 and 1 in a majority of test cases and, with two exceptions, are almost always less than 1.5.

Although the RR time in JD constitutes a small percentage of the TTS when eigenvalues are computed in groups, it is not necessarily less than the RR time used by EigPen because the RR procedure is called many times in JD. In our tests to be presented in section 5, the observed values of γ are between 0.01 and 0.5.

Figure 2 shows the how the TTS of EigPen changes with respect to the changes in the ratio $\mathrm{nspmv}_{\mathrm{eigpen}}/\mathrm{nspmv}_{\mathrm{jd}}$ for different choices of p, β and γ. The reference JD TTS is plotted as a horizontal dash-dotted line. In Figure 2(a), we set γ to 1, which is larger what we actually observed in our tests. We can see that even in this case, T_{eigpen} can be less than half of T_{jd} when $\beta = 1/5$, $p = 5$, α is around 3. The crossover point at which the TTS for two methods become almost the same occurs at roughly $\alpha = 9$. Changing p, which measures the relative parallel efficiency of SpMV in EigPen compared to that in JD, changes the slope of the TTS curve. When $p = 10$, which we can actually observe in our tests, the crossover point is beyond $\alpha = 15$. For $p = 100$, which is possible on large-scale machines, the crossover point cannot be seen in this figure.

In Figure 2(b), we set β to 1/5, which is within the range we observe in our tests, and see how the TTS curve change as we change both γ and p. Increasing γ shifts the curve to the right. But the amount of shift become negligible when $\gamma < 1/10$. Changing p has a more significant effect. Even when $\gamma = 10$, which is much larger than what we observe in our expriment, setting $p = 10$ moves the crossover point from $\alpha = 4$ to $\alpha = 9$.



(a) $\gamma = 1$ (b) $\beta = 1/5$

FIG. 2. *A comparison of the* TTS *of* EigPen *(T_{eigpen}) with that of* JD *(T_{jd}) for a specific scenario in which $t_{\mathrm{spmv}} = 50$, $t_{\mathrm{blas}} = 45$, and $t_{\mathrm{rr}} = 5$, and different choices of p, $\mathrm{nspmv}_{\mathrm{eigpen}}/\mathrm{nspmv}_{\mathrm{jd}}$, β, γ are made for* EigPen.

We should emphasize here that the scenarios described above are based on specific choices of timing breakdown for LOBPCG and JD algorithms. These timing breakdowns are certainly problem and machine specific. We simply use these to illustrate why EigPen can be more efficient in some cases. As k and the number of processors used in the computation increase, the LOBPCG timing breakdown will skew towards a larger percentage of the RR cost. On machines with many processors, the relative parallel efficiency of SpMV in EigPen will be much higher than that in a JD type of algorithm because multiple levels of concurrency can be exploited in the former. Both trends favor the performance of EigPen.

5. Numerical Experiments.

In this section, we test the performance of EigPen as a general solver for computing a set of smallest eigenvalues and their corresponding eigenvectors of sparse matrices.

5.1. Solvers, Test Matrices and Platform.

We choose to compare EigPen with two state-of-the-art eigensolvers: the Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX [1]) [7] which implements the LOBPCG algorithm [8], and the preconditioned iterative multi-method eigensolver [14, 15] (PRIMME, version 1.1)[2], which implements multiple methods that include generalized Davidson and Jacobi-Davidson algorithms. Both packages are implemented in the C language. EigPen is implemented in Fortran and parallelized by using OpenMP. We modified both BLOPEX and PRIMME by adding OpenMP directives wherever appropriate so that they can be executed on shared memory parallel machines. In addition, we replaced all native multivector dense linear algebra operations in BLOPEX by appropriate BLAS calls. We also compared EigPen with an OpenMP version of ARPACK [10]. Because the performance of ARPACK is not competitive except for a few problems, we choose not to report its performance here. The non-competitiveness of ARPACK is mainly due to the lack of effective parallelization for sequential SpMVs and BLAS2 dense matrix operations in ARPACK.

PRIMME allows a subset of eigenpairs to be computed using a windowed approach. This is the version that we use. In addition, PRIMME uses a few special techniques, such as restarting and stagnation-proof locking, etc. to enhance performance. These techniques can potentially be used to improve the performance of EigPen and LOBPCG also, but is currently not implemented in these codes.

We select a set of thirteen sparse test matrices[3] whose dimension n, the number of nonzero components nnz and sparsity are listed in Table 1. Many of these matrices are produced by PARSEC [9], a real space density functional theory (DFT) based code for electronic structure calculation in which the Hamiltonian is discretized by using finite difference. Since the smallest absolute value of the matrix "shallow_water1" is as large as $O(10^8)$ that causes trouble for PRIMME to converge, we divide the matrix by the number $\sigma = 5.7769 \times 10^9$ (which is an estimation of $|\lambda_1|$) and denote the scaled matrix by "shallow_water1s".

We perform most of our numerical experiments on a single node of Hopper[4], a Cray XE6 supercomputer maintained at the National Energy Research Scientific Computer Center (NERSC) in Berkeley. The node consists of two twelve-core AMD "MagnyCours" 2.1-GHz processors with a total of 32 gigabyte (GB) shared memory. However, memory access bandwidth and latency are nonuniform across all cores. Each core has its own 64 kilobytes (KB) L1 and 512 KB L2 caches. One 6-MB L3 cache shared among 6 cores on the Magny-Cours processor. There are four DDR3 1333-MHz memory channels per twelve-core "Mag-

[1]Downloadable from `http://code.google.com/p/blopex`
[2]Downloadable from `http://http://www.cs.wm.edu/~andreas/software`
[3]Downloadable from `http://www.cise.ufl.edu/research/sparse/matrices`
[4]More information at `http://www.nersc.gov/users/computational-systems/hopper/`

TABLE 1
Problem characteristics

Name	n	nnz	$100 \times \frac{nnz}{n^2}\%$
Andrews	60000	410077	0.01%
C60	17576	212390	0.07%
c_65	48066	204247	0.01%
cfd1	70656	948118	0.02%
finance	74752	335872	0.01%
Ga10As10H30	113081	3114357	0.02%
Ga3As3H12	61349	3016148	0.08%
OPF3754	15435	82231	0.03%
shallow_water1	81920	204800	<0.01%
Si10H16	17077	446500	0.15%
Si5H12	19896	379247	0.10%
SiO	33401	675528	0.06%
wathen100	30401	251001	0.03%

nyCours" processor.

All executables used here are linked with the multi-threaded version of the Cray Scientific Libraries package, LibSci, which includes multi-threaded versions BLAS and LAPACK subroutines optimized for Cray XE6. We do not parallelize each individual SpMV calculation, but add OpenMP directives to the loop that allows multiple columns of AX to be computed in parallel whenever X contains multiple columns. We also implement a block version of the Davidson algorithm. Without using a preconditioner, the algorithm is essentially a steepest descent algorithm directly applied to (1.1). Because its performance in our tests has been found to be clearly poorer compared to other algorithms discussed in this section, we do not include its timing measurements in the numerical results presented in this section. The block Krylov-Schur algorithm [22] and the (block) Chebyshev-Davidson algorithm [20, 21] are not included in our comparison, partly because suitable Fortran/OpenMP implementations of these algorithms were unavailable for our tests, and partly because these algorithms cannot easily take advantage of a preconditioner when it is available.

5.2. Termination Rules and Parameters. All tests on the aforementioned machine Hopper are run as batch jobs with a maximum wall-clock time limit of 6 hours. We terminate both EigPen and LOBPCG when the relative residual norm (defined below) for every Ritz-pair (u_i, θ_i) is smaller than a prescribed tolerance tol, that is,

$$\text{res}_i(U) = \frac{\|Au_i - \theta_i u_i\|_2}{\max(1, |\theta_i|)} \leq tol, \quad i = 1, \cdots, nev, \tag{5.1}$$

where nev is the number of smallest eigenvalues to be computed, u_i is the i-th column of U that satisfies $U^{\mathrm{T}}U = I$, and $\theta_i = u_i^T A u_i$ (recall that for EigPen we need to perform an RR step to obtain Ritz-pairs). We also terminate an algorithm when the number of iterations reaches a maximum of 10,000, but this limit was never reached in our experiments. For LOBPCG and EigPen, we set the dimension of X (denoted by k) to be slightly larger than nev to improve the convergence. Specifically, k is set to $nev \times 1.1$ (rounded to the nearest integer).

In EigPen (Algorithm 2), the initial penalty parameter μ is computed by (3.12). After an RR step is executed, μ is set according to (3.13) and is fixed throughout the next round of inner iterations. The constants $c_1 = 0.1$ and $c_2 = 1.1$ are used in (3.12) and (3.13). The initial tolerance ϵ^0 is set to tol and the backtracking constant δ is set to 0.25. The parameter δ_ϵ is adjusted dynamically according to the number of the converged eigenvectors (denoted by k_1) that satisfy the condition $\text{res}_i \le \text{tol}$, for $i = 1, \dots, nev$, after each RR step:

$$\delta_\epsilon = \begin{cases} 0.1, & \text{if } k_1 = 0, \\ 0.5, & \text{if } k_1 \le 0.9\, nev, \\ 0.6, & \text{if } k_1 \le 0.95\, nev, \\ 0.7, & \text{otherwise.} \end{cases}$$

The basic termination rule of PRIMME is

$$\|Au_i - \theta_i u_i\|_2 \le tol, \quad i = 1, \cdots, nev. \tag{5.2}$$

To prevent stagnation, PRIMME also employs certain heuristics to declare near-convergence of a Ritz pair even when (5.2) is not satisfied. Due to the existence of such heuristics, we keep the stopping rules of PRIMME intact, which of course differ from (5.1). Since the magnitude of many eigenvalues is of order $O(1)$ for the tested matrices, the two stopping rules (5.1) and (5.2) are nevertheless comparable to a large extent. Throughout our experiments, we set the PRIMME method parameter to `DYNAMIC`, which allows it to choose the most appropriate method to solve the eigenvalue problem automatically. We also set

$$\texttt{primme.maxBasisSize} = \max(100, \min(500, 0.5 nev)),$$
$$\texttt{primme.maxBlockSize} = 10.$$

Our experiments show that, at least on the particular computer node and the particular set of test problems, the last two parameters lead to better overall performance than that produced by the default PRIMME settings (though they might not be the best possible since our tuning was not exhaustive).

5.3. Overall Performance. We first report the overall performance of EigPen, BLOPEX and PRIMME on the test matrices listed in Table 1. The number of smallest eigenvalues (nev) to be computed is roughly 1% of the dimension of A. All algorithms are run in parallel with 24 cores. No preconditioner is used in these tests.

Our experiments are performed using two different tolerance values $tol = 10^{-3}$ and $tol = 10^{-4}$. The total wall-clock times taken by the three solvers are presented in Table 2.

Whenever a code terminates abnormally (either a run is stopped prematurely or the maximum wall-clock time limit of 6 hours is reached), the corresponding entry in the table is marked by "– –".

TABLE 2
A comparison of total wall-clock time ("– –" are abnormal terminations)

		$tol = 10^{-3}$			$tol = 10^{-4}$		
Matrix	nev	BLOPEX	PRIMME	EigPen	BLOPEX	PRIMME	EigPen
Andrews	600	1015	525	248	1273	581	601
C60	200	52	23	35	67	27	47
c_65	500	– –	757	5944	– –	820	6612
cfd1	700	4681	4250	703	6902	3103	1327
finance	700	3475	1359	639	– –	1108	901
Ga10As10H30	1000	– –	3691	2538	– –	3997	4125
Ga3As3H12	600	– –	1086	760	– –	1309	1183
OPF3754	200	– –	23	13	– –	28	22
shallow_water1s	800	3238	2734	420	4497	1842	1247
Si10H16	200	78	40	33	97	47	62
Si5H12	200	94	49	32	118	57	38
SiO	400	426	214	134	532	249	231
wathen100	300	543	154	137	702	164	289

We observe from Table 2 that for $tol = 10^{-3}$ BLOPEX did not succeed on four matrices: c_65, Ga10As10H30, Ga3As3H12 and OPF3754, and on an additional matrix, finance, for $tol = 10^{-4}$. In general, EigPen is faster than BLOPEX, especially on larger problems. EigPen is also mostly faster than PRIMME except on the matrix C60 and c_65 for $tol = 10^{-3}$. For $tol = 10^{-4}$, the comparison results between EigPen and PRIMME are mixed, with no clear speed advantage for either. We should point out that our current implementation of EigPen is still rather preliminary and its relative performance deteriorates as tol decreases, which is a limitation of EigPen at this point. Its performance can be improved by using better restarting and locking strategies similar to those used in PRIMME. For example, we have found by additional experiments that the performance of EigPen on matrix c_65, which has created most difficulties for EigPen, can indeed be significantly improved by a proper locking strategy.

In Table 3, we report the number of iterations performed by BLOPEX (denoted by "BLOPEX-RR"), the total number of gradient descent steps and the number of RR steps performed by EigPen (denoted by "EigPen-(GD,RR)" together in parentheses). The minimal, average and maximal number of the RR steps is, respectively, 1, 5.5 and 12. It is worth emphasizing that i) EigPen makes only a few RR calls, whereas BLOPEX calls RR at every iteration; ii) the cost of each gradient descent step in EigPen is much cheaper than that of the RR steps in BLOPEX.

TABLE 3
The number of iterations performed by BLOPEX and EigPen

	$tol = 10^{-3}$		$tol = 10^{-4}$	
Matrix	BLOPEX-RR	EigPen-(GD,RR)	BLOPEX-RR	EigPen-(GD,RR)
Andrews	48	(113, 5)	63	(299, 5)
C60	46	(178, 9)	62	(266, 6)
c_65	– –	(5368, 12)	– –	(6011, 11)
cfd1	138	(253, 4)	220	(523, 4)
finance	111	(235, 5)	– –	(348, 4)
Ga10As10H30	– –	(334, 5)	– –	(590, 5)
Ga3As3H12	– –	(310, 3)	– –	(550, 3)
OPF3754	– –	(86, 1)	– –	(152, 1)
shallow_water1s	65	(108, 4)	97	(236, 6)
Si10H16	56	(150, 8)	72	(316, 7)
Si5H12	61	(134, 5)	79	(176, 5)
SiO	57	(175, 7)	73	(332, 7)
wathen100	147	(320, 5)	194	(746, 6)

To support the cost analysis given in Section 4, and interpret the timing results reported in Table 2, we give ratios of SpMV counts, time spent in BLAS operations, and time spent in RR calculations between EigPen and BLOPEX, and those ratios between EigPen and PRIMME respectively in Table 4. Notice that the SpMV count ratios between EigPen and BLOPEX (column 2) and between EigPen and PRIMME (column 4) are largely between 1 and 5 with a few exceptions. The SpMV count ratios for C60 and c_65 are much larger. This observation partially explains the small TTS differences between EigPen and BLOPEX and the better performance of PRIMME compared to EigPen for these problems. We also notice that the BLAS time ratio between EigPen and PRIMME is much larger for C60 and c_65. This also partially explains why PRIMME performs better on these two matrices.

5.4. Accuracy. We next show the accuracy of the computed eigenpairs, as well as that of the computed minimum trace values. We should point out that when tol is relatively large, the i-th Ritz value θ_i may be closer to λ_j for $j > i$ than to λ_i. In this case, we may miss some eigenvalues even though the convergence criterion (5.1) is satisfied for all $i \le nev$. To measure the accuracy of the computation, we compute the relative difference between θ_i

TABLE 4
The ratio of SpMV counts, time spent in BLAS operations, and time spent in RR between EigPen and BLOPEX, and between EigPen and PRIMME respectively for the $tol = 10^{-3}$ case.

	BLOPEX			RRIMME		
Matrix	#SpMV	BLAS	RR	#SpMV	BLAS	RR
Andrews	3.8	0.35	0.004	3.1	0.50	0.092
C60	7.2	0.62	0.033	5.7	1.85	0.400
c_65	−−	−−	−−	10.8	12.86	0.142
cfd1	2.6	0.26	0.001	2.4	0.14	0.010
finance	2.7	0.31	0.001	2.7	0.45	0.042
Ga10As10H30	−−	−−	−−	3.7	0.87	0.041
Ga3As3H12	−−	−−	−−	3.6	1.24	0.048
OPF3754	−−	−−	−−	2.2	0.61	0.034
shallow_water1s	2.4	0.24	0.001	2.6	0.13	0.024
Si10H16	3.5	0.37	0.017	3.0	1.48	0.274
Si5H12	2.8	0.29	0.011	2.1	1.12	0.202
SiO	3.9	0.42	0.006	2.8	0.88	0.151
wathen100	2.4	0.29	0.003	2.2	1.45	0.091

and the true eigenvalue λ_i computed in advance by ScaLAPACK [3]. The maximum relative errors among all eigenvalues, which is measured by

$$\text{err}_\theta = \max_{i=1,\ldots,nev} \frac{|\theta_i - \lambda_i|}{\max(1, |\lambda_i|)},$$

are reported in Table 5, and the relative errors between the sum of the nev eigenvalues, defined by

$$\text{err}_{\text{trace}} = \frac{|\sum_{i=1}^{nev} \theta_i - \sum_{i=1}^{nev} \lambda_i|}{\max(1, |\sum_{i=1}^{nev} \lambda_i|)},$$

are presented in Table 6. From these tables, we see that BLOPEX and EigPen achieve the same level of accuracy on most problems. Compared with the other two solvers, PRIMME generally obtains better accuracy in computed eigenvalues except on the matrices cfd1 and shallow_water1s (though it does not produce more accurate eigenvectors, as is shown below).

TABLE 5
A comparison of err_θ *among different solvers*

	$tol = 10^{-3}$			$tol = 10^{-4}$		
Matrix	BLOPEX	PRIMME	EigPen	BLOPEX	PRIMME	EigPen
Andrews	1.04e-03	1.61e-07	4.58e-05	2.21e-03	2.93e-09	4.58e-05
C60	1.15e-06	2.45e-06	4.34e-05	8.32e-09	2.74e-08	4.34e-05
c_65	−−	1.54e-08	4.79e-05	−−	2.15e-10	4.79e-05
cfd1	8.32e-05	1.60e-04	2.44e-04	9.03e-05	6.92e-07	6.90e-06
finance	1.50e-03	2.44e-06	8.02e-05	−−	3.60e-08	4.80e-05
Ga10As10H30	−−	2.82e-07	4.97e-05	−−	3.65e-09	4.97e-05
Ga3As3H12	−−	9.43e-08	4.69e-05	−−	7.45e-10	4.69e-05
OPF3754	−−	2.17e-15	3.46e-05	−−	1.47e-15	3.46e-05
shallow_water1s	5.49e-05	1.25e-04	3.22e-04	1.19e-04	2.07e-07	4.88e-05
Si10H16	2.01e-02	3.44e-07	4.33e-05	6.42e-08	5.50e-09	4.33e-05
Si5H12	5.86e-07	2.02e-07	3.86e-05	3.35e-08	1.87e-09	3.86e-05
SiO	2.79e-03	2.01e-07	4.81e-05	1.23e-08	8.67e-10	4.81e-05
wathen100	3.34e-03	6.14e-08	3.17e-05	8.12e-04	5.24e-10	3.17e-05

To measure the accuracy of the approximate eigenvectors, we also report the maximum residual error defined by

$$\text{err}_{\text{res}} = \max_{i=1,\ldots,nev} \text{res}_i$$

in Table 7. We observe that EigPen ususally returns a slightly smaller residual error than BLOPEX in this set of tests. On most matrices, the accuracy of PRIMME-computed eigenvectors is overall comparable to that of those computed by the other two solvers. However,

TABLE 6
A comparison of err_{trace} *among different solvers*

	$tol = 10^{-3}$			$tol = 10^{-4}$		
Matrix	BLOPEX	PRIMME	EigPen	BLOPEX	PRIMME	EigPen
Andrews	4.33e-06	2.98e-08	4.56e-07	9.04e-06	4.65e-10	1.07e-07
C60	4.28e-08	6.03e-08	7.84e-09	2.81e-10	8.87e-10	9.01e-08
c_65	– –	1.19e-09	8.20e-07	– –	1.68e-11	8.20e-07
cfd1	5.60e-04	9.51e-04	3.57e-03	4.20e-05	7.62e-06	2.00e-05
finance	2.64e-05	2.82e-07	1.39e-06	– –	3.86e-09	3.91e-07
Ga10As10H30	– –	5.42e-08	3.07e-07	– –	7.91e-10	3.10e-07
Ga3As3H12	– –	1.74e-08	1.01e-06	– –	2.12e-10	1.01e-06
OPF3754	– –	1.97e-16	8.09e-07	– –	1.97e-16	8.09e-07
shallow_water1s	2.14e-06	3.44e-06	1.65e-05	4.44e-07	3.32e-08	1.62e-06
Si10H16	4.66e-04	1.76e-07	3.16e-06	2.65e-09	2.41e-09	3.16e-06
Si5H12	7.54e-08	4.28e-08	1.20e-07	2.37e-09	5.36e-10	5.50e-07
SiO	4.16e-05	2.02e-08	1.17e-06	7.40e-10	2.14e-10	1.29e-06
wathen100	2.00e-05	1.09e-08	7.67e-08	3.30e-06	1.46e-10	3.05e-07

since heuristics are occasionally invoked by PRIMME to declare near-convergence of some Ritz pairs, the final accuracy can sometimes be worse than the given tolerance, for example, on the matrices cfd1 and shallow_water1s for $tol = 10^{-3}$.

TABLE 7
A comparison of err_{res} *among different solvers*

	$tol = 10^{-3}$			$tol = 10^{-4}$		
Matrix	BLOPEX	PRIMME	EigPen	BLOPEX	PRIMME	EigPen
Andrews	9.97e-04	7.18e-04	6.34e-04	9.99e-05	6.31e-05	4.37e-05
C60	9.95e-04	7.99e-04	6.98e-04	9.95e-05	8.29e-05	7.22e-05
c_65	– –	1.12e-04	1.57e-04	– –	1.10e-05	4.62e-05
cfd1	1.00e-03	1.09e-03	5.57e-04	1.00e-04	9.92e-05	8.70e-05
finance	1.00e-03	7.89e-04	7.34e-04	– –	7.50e-05	8.34e-05
Ga10As10H30	– –	9.53e-04	6.47e-04	– –	8.89e-05	4.80e-05
Ga3As3H12	– –	9.61e-04	9.91e-04	– –	7.96e-05	5.85e-05
OPF3754	– –	2.45e-08	2.61e-04	– –	2.55e-09	9.14e-08
shallow_water1s	1.00e-03	1.28e-03	9.21e-04	1.00e-04	9.59e-05	5.88e-05
Si10H16	9.99e-04	8.69e-04	8.31e-04	9.98e-05	7.81e-05	9.27e-06
Si5H12	1.00e-03	7.98e-04	7.74e-04	1.00e-04	7.78e-05	8.92e-05
SiO	9.98e-04	7.18e-04	9.89e-04	1.00e-04	8.02e-05	4.59e-05
wathen100	1.00e-03	6.30e-04	6.99e-04	1.00e-04	7.07e-05	6.18e-05

We understand that, under fixed test conditions, the performance of most solvers can in principle be optimized by extensively tuning parameters which we obviously did not attempt to do. The purpose of our numerical experiments in this section is not to pick winners for solving a particular problem set on a particular machine. Instead, we wish to establish that a preliminary code based on the proposed trace-penalty formulation could perform competitively under a rather generic setting, and even favorably under suitable conditions as we will show next.

5.5. Performance profile and dependency on eigenspace dimension. In this subsection, we examine how the three solvers perform when the number of desired eigenpairs (nev) increases. For brevity, we only show results for two matrices Andrews and Ga3As3H12, but similar profiles can be observed for other matrices as well.

In the following experiments, we set the convergence tolerance to $tol = 10^{-2}$ and nev to a set of increasing values $\{500, 1000, 1500, 2000, 2500, 3000\}$. We run all solvers on all 24 cores of the one computer node of Hopper. The wall-clock time measurements are plotted against nev for all solvers in Figure 3. BLOPEX are terminated after reaching the maximum wall-clock time limit of 6 hours when $nev = 2500$ and 3000. We observe that EigPen always takes less amount of time to run than BLOPEX or PRIMME. The difference in wall-clock time increases quickly as nev increases. This observation suggests that the benefit of using

EigPen become increasingly greater as nev becomes larger at least when the convergence tolerance is relatively large. The key reason that EigPen performs much better than BLOPEX for large nev is that, by performing far fewer RR steps, it is able to leverage BLAS3 operations that are highly optimized for Hopper (and other high performance computers). As nev increases, EigPen can achieve a higher level concurrency in SpMV compared to PRIMME. This is the main reason that makes EigPen perform better than PRIMME for large nev's.

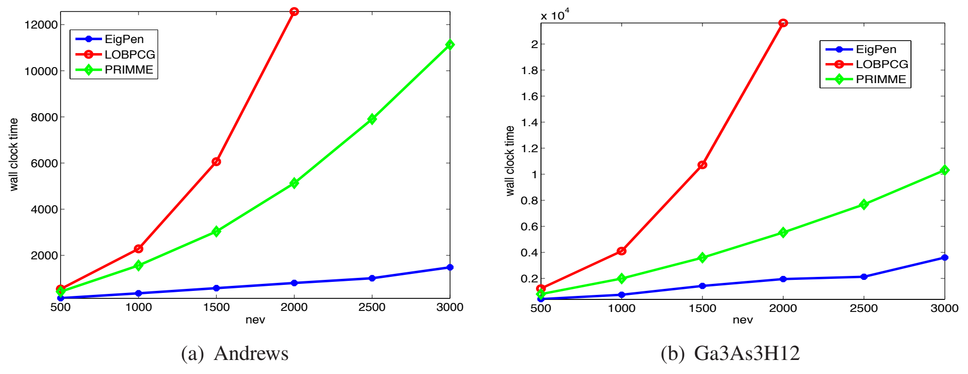
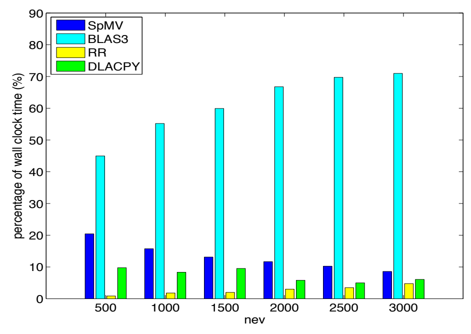


FIG. 3. *A comparison of wall-clock times by BLOPEX, PRIMME and EigPen to compute nev eigenpairs of the matrices Andrews and Ga3As3H12 as nev increases.*

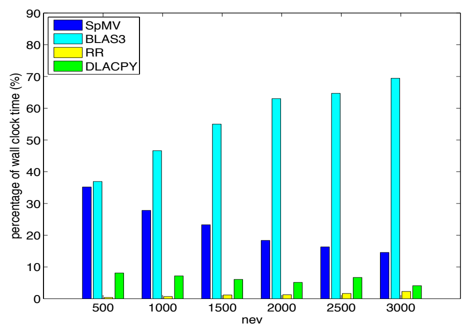
The tolerance $tol = 10^{-2}$ used in Figure 3 may be considered too loose for some circumstances. We did try the same experiment with $tol = 10^{-3}$ and obtained similar results. With all other conditions equal, as the value of tol decreases, the observed performance gap between EigPen (as it is implemented now) and others seems to gradually narrow and diminish.

In Figure 4, we show run times of four categories: sparse matrix vector multiplications (SpMV), dense matrix-matrix operations (BLAS3, including subroutines `DGEMM`, `DSYMM`, `DSYRK`, `DSYTRS`, `DPOTRF`, and `DTRSM` in BLAS and LAPACK), dense matrix-vector operations (mainly the subroutine `DGEMV` in BLAS2), Rayleigh-Ritz (RR, subroutines `DSYGVD, DSYEVD, DGESVD` in LAPACK) calculations, and matrix copying (the `DLACPY` subroutine in LAPACK). These are the major computational components of both EigPen and BLOPEX, albeit in different proportions. Two clarifications are in order here. Firstly, we categorize these subroutines only at the highest solver level. As such, any call to `DGEMM` inside the subroutine `DSYEVD`, for example, is not counted as in the BLAS3 category. Secondly, although the "correctness" of such a classification scheme may be debatable, it does not alter the overall fact, as is clearly shown by our computational results, that the category BLAS3 is much more scalable than the category RR on our test platform.

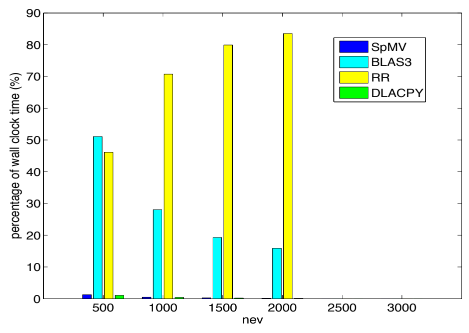
The run time of each category is measured in terms of the percentage of wall-clock time spent in that category over the total wall-clock time. We can clearly see that for EigPen the run time of BLAS3 dominates the entire computation in almost all cases. The BLAS3 time increases steadily as nev increases from 500 to 3000, while the SpMV time decreases steadily. The run time of RR is negligible. However, since our implementation of EigPen performs extra matrix copying when computing the gradient difference Y^j defined in (3.3) for the BB step size computation, the cost associated with such data movement is notable. In BLOPEX, the relative cost of SpMV is negligible as nev increase. However, the run time of RR increases more rapidly as nev increases. When $nev \geq 1000$, the run time of RR is higher than that of BLAS3. In PRIMME, the run time of BLAS2 increases steadily as nev increases. For most cases, the run time of BLAS2 accounts for more than 40% of the total wall-clock time and the time percentage of the BLAS3 (of all sizes, large or small) is around 20%. On the other hand, the BLAS2 routine `DGEMV` is rarely called in either BLOPEX or EigPen.



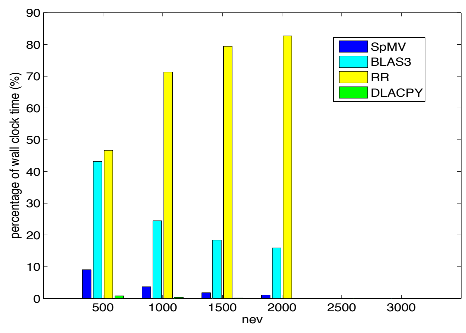
(a) Andrews: EigPen



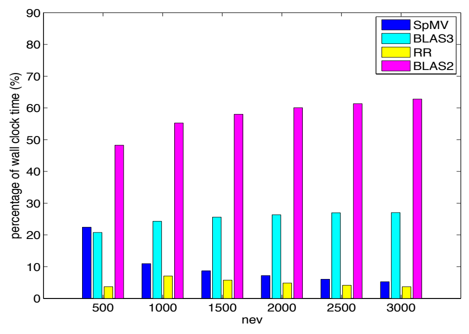
(b) Ga3As3H12: EigPen



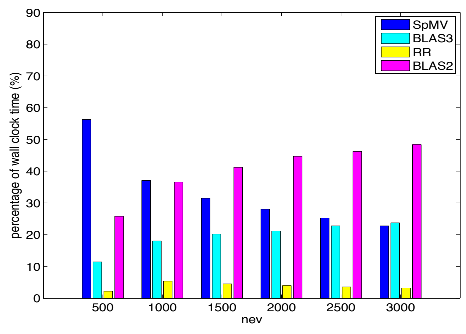
(c) Andrews: BLOPEX



(d) Ga3As3H12: BLOPEX



(e) Andrews: PRIMME



(f) Ga3As3H12: PRIMME

FIG. 4. *A comparison of timing profile among EigPen, BLOPEX and PRIMME.*

Figures 3 and 4 clearly demonstrate that the advantages of the EigPen algorithm over the LOBPCG algorithm are due to fewer Rayleigh-Ritz calculations. This advantage is more pronounced when the number of eigenpairs to be computed (nev) is large because the cost of Rayleigh-Ritz calculation grows rapidly with respect to nev (and k). Although the complexity of the Rayleigh-Ritz calculation is the same as that associated with the dense matrix-matrix operations required for updating the approximate solution in EigPen, dense matrix-matrix operations can be implemented efficiently on modern high performance parallel computers whereas it is more difficult to achieve the same level of efficiency for RR calculations. As a result, by keeping the number of Rayleigh-Ritz calculations small in EigPen and making use of more BLAS3 operations, we can make it more efficient than BLOPEX for large nev values. When compared to PRIMME, the advantage of EigPen can be seen from a much lower percentage of the SpMV cost as well as the absence of BLAS2 operation.

5.6. Parallel scalability. In this subsection, we examine parallel scalability of BLOPEX, PRIMME and EigPen. For brevity, we again only show results for the two matrices, Andrews and Ga3As3H12, using $nev = 1500$, although similar results can be seen for other test problems as well. We define the speedup factor for running a code on p cores as

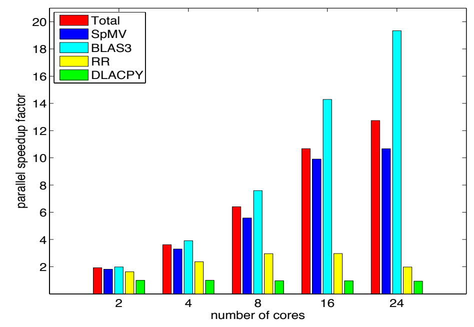
$$\text{speedup-factor}(p) = \frac{\text{wall-clock time for a single core run}}{\text{wall-clock time for a } p\text{-core run}}.$$

We only run 5 iterations for BLOPEX and EigPen and set $\texttt{primme.maxMatvecs} = 5000$ for PRIMME since the speedup factor should remain essentially unchanged as more iterations are performed.

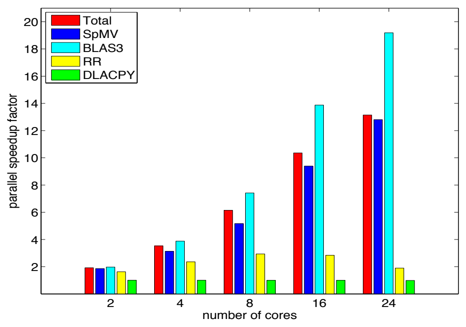
Figure 5 shows the speedup factors associated with SpMV, BLAS3, RR and DLACPY, as well as the overall computation, when the parallelized Fortran codes are run with 2, 4, 8, 16 and 24 cores. As we can clearly see from the figure that the speedup factors for BLAS3 are nearly perfect when EigPen are run on as many as 24 cores. The scalability of SpMV is almost as good for the Ga3As3H12 problem. But it is slightly worse beyond 8 cores for the Andrews matrix, which we believe is due to a higher sparsity of the Andrews matrix that makes the effect of thread overhead more prominent in parallel SpMV calculations. However, the speedup factor for RR increases slowly with respect to the number of cores up to 8 cores, then it starts to decrease. Because computation in EigPen is heavily dominated by BLAS3 (and to a lesser extent by SpMV) that scales much better than RR, the overall scalability of EigPen is better than that of BLOPEX. In PRIMME, the operations SpMV and RR show little or no speedup throughout, while BLAS2 and BLAS3 operations provide a degree of initial speedup from 2 to 4 cores but no speedup afterwards from 8 to 24 cores. The best observed speedup factor for BLAS2 and BLAS3 is around 2 when 4 cores are used. Our experiments indicates that as the parameter $\texttt{primme.maxBlockSize}$ increases to 50 or 100, the speedup factor of BLAS3 can reach a healthy level, but the total wall-clock time does not necessarily improve accordingly. On this particular computer node and this set of test problems, our experiments suggest that $\texttt{primme.maxBlockSize} = 10$ appear to be a near optimal value, providing a good balance between scalability and running time, which was the reason that this value has been selected in our tests. The existing implementation of PRIMME has limited scalability because a small number of eigenpairs is computed at a time, making it a sequential process. Conceptually, it is possible to divide the spectrum into subintervals in advance and simultaneously run PRIMME all intervals. But this type of approach has its own difficulties in terms of appropriate strategies for dividing the spectrum, maintaining orthogonality, and making sure no eigenvalue is missed or double counted, to name a few.

5.7. Preconditioning for EigPen. One can also introduce a preconditioner in EigPen. The use of a preconditioner essentially amounts to a change of variable in the form of $Y = LX$. If an appropriate nonsingular L is chosen, substituting $X = L^{-1}Y$ into (1.2) yields a problem with a better conditioned Hessian. It is straightforward to show that a preconditioned gradient method can be described by

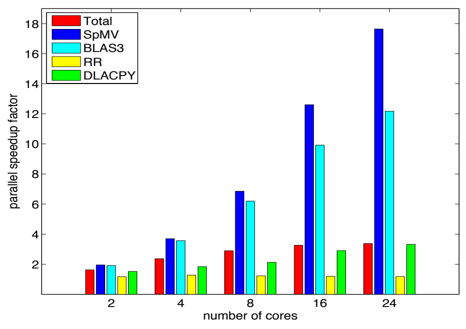
$$X^{j+1} = X^j - \alpha^j M^{-1} \nabla f_\mu(X^j), \tag{5.3}$$



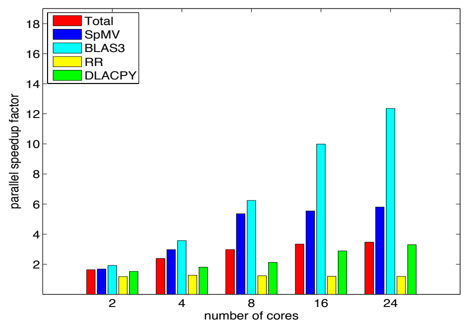
(a) Andrews: EigPen



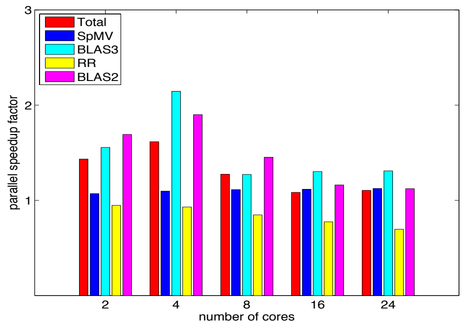
(b) Ga3As3H12: EigPen



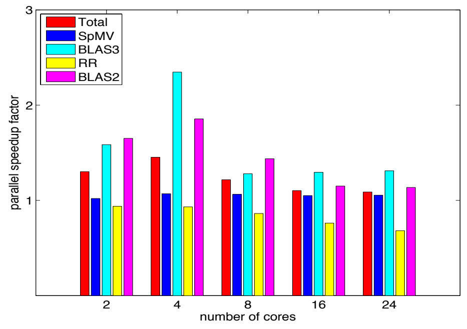
(c) Andrews: BLOPEX



(d) Ga3As3H12: BLOPEX



(e) Andrews: PRIMME



(f) Ga3As3H12: PRIMME

FIG. 5. *A comparison of speedup factor among EigPen, BLOPEX and PRIMME.*

where $M = L^T L$ is the preconditioner.

We now demonstrate that the performance of EigPen can be improved by preconditioning using two matrices "Benzene" and "SiH4" generated from KSSOLV [18] — a MATLAB toolbox for solving the Kohn-Sham equations arising from electronic structure calculation. The matrix dimension and the number of eigenvalues to be computed are $(n, k) = (8407, 15)$ for "Benzene", and $(n, k) = (2103, 4)$ for "SiH4". These are small matrices and the number of eigenvalues to be computed are relatively small as well. These matrices are not explicitly constructed, but are accessed through a matrix-vector multiplication function provided by KSSOLV. We choose to use KSSOLV, which uses a planewave expansion to discretize the Kohn-Sham problem, partly because it is well known that good preconditioners are available for planewave discretized Kohn-Sham Hamiltonian [17], and partly because these preconditioners, which are diagonal in the Fourier space, are easy to extract from KSSOLV. Because KSSOLV is written in MATLAB, we choose to use the MATLAB version of EigPen , and compare its performance with the publicly released MATLAB version of LOBPCG [5]. Our experiments in this subsection are performed on a Dell Precision M4700 workstation with Intel i7-3720QM CPU at 2.60GHz ($\times 8$) and 16GB of memory running Ubuntu 12.04 and MATLAB 2013a.

We ran both EigPen and LOBPCG with and without a preconditioner. The convergence history of the preconditioned and unpreconditioned versions of EigPen are shown in Figure 6 for both matrices. Convergence is declared when the Frobenius norm of the gradient of the penalized trace is less than a tolerance of 10^{-10}. We can clearly see that the use of a preconditioner reduces the number of EigPen iterations by a factor of two for these problems. In Table 8, we also list the number of gradient descent steps executed by EigPen (denoted by "GD"), the number of RR steps used by BLOPEX (denoted by "RR"), the total number of SpMVs used by both methods to reach convergence, the Frobenius norm of the residual of the computed Ritz pairs (denoted by "res") and the total wallclock time taken.

We can see that the number of SpMVs used by a preconditioned EigPen is two or three times the number of SpMVs used by LOBPCG, just like the unpreconditioned cases we reported earlier. We showed in both the cost analysis presented in section 4 and numerical experiments presented earlier in this section that the TTS used by EigPen can still be much lower than that used by LOBPCG even when EigPen performs three times as many SpMVs as those performed by LOBPCG. We do not see such an advantage in measured wallclock time here because the number of computed eigenvalues is relatively small for these problems. Hence the cost of RR is relatively low. We believe that for larger problems in which hundreds or thousands of eigenvalues are needed, the performance difference between the preconditioned versions of EigPen and LOBPCG will be similar to that between the unpreconditioned versions of these two algorithms.

TABLE 8
A comparison without and with preconditioning

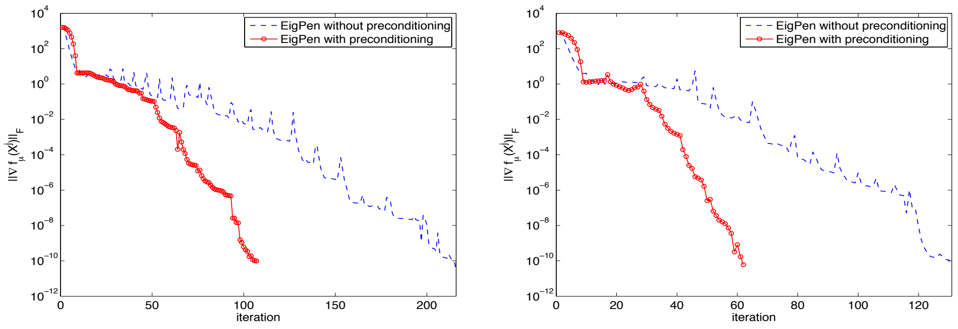
	EigPen without / with preconditioning				BLOPEX without / with preconditioning			
Problem	res	time	SpMV	GD	res	time	SpMV	RR
SiH4	7.2e-12 / 7.6e-12	0.9 / 0.5	520 / 244	130 / 61	9.9e-11 / 9.1e-11	2.7 / 0.9	402 / 112	111 / 28
Benzene	4.1e-12 / 2.6e-11	16.1 / 8.3	3225 / 1590	215 / 106	1.0e-10 / 9.3e-11	20.0 / 6.1	1453 / 527	181 / 40

We should point out that it is generally not easy to identify an effective and efficient preconditioner for an eigenvalue problem. The purpose of presenting the above example is not to promote a particular pre-conditioner, but rather to demonstrate the fact that EigPen can indeed take advantage of a good pre-conditioner whenever it is available. The issue of preconditioning for our approach certainly remains a topic of further and more careful study.

6. Conclusion.

The objective of this paper is to develop an algorithmic approach with elevated parallel scalability in order to effectively utilize massively parallel computers for large-scale eigenspace computation. We propose and study an unconstrained optimization model, called trace-penalty minimization, that requires no orthogonality by itself. Theoretically, with properly chosen penalty parameter values the proposed formulation yields the optimal eigenspace with fewer undesirable saddle points than the classic trace minimization model. Computationally, it enables unconstrained optimization techniques to compute approximate eigenspaces and, consequently, reduces the use of the Rayleigh-Ritz procedure.

[5]`http://www.mathworks.com/matlabcentral/fileexchange/48-lobpcg-m`



(a) Benzene (b) SiH4

FIG. 6. *The iteration history of gradient norm on benzene and SiH4 without/with preconditioning of EigPen.*

The bottleneck preventing many existing eigensolvers from reaching high parallel scalability normally comes from one or more of the following three sources: sequential SpMVs, sequential basis orthogonalization or dense eigenvalue decomposition in the RR procedure. Being a block algorithm that utilizes block SpMVs, our approach also reduces the number of RR calls in exchange for highly scalable, BLAS3-rich dense matrix operations. Given that dense matrix-matrix multiplications have typically been optimized in high-performance mathematical libraries, our approach has the potential to take advantages of tens or hundreds of thousands-way concurrency on modern multi/many-core systems.

As a first step of numerical evaluation, we test our new eigensolver EigPen, based on solving the trace-penalty model by a gradient method, coded in Fortran and parallelized using OpenMP, in comparison to two state-of-the-art solvers. On a 24-core computer node, EigPen already provides competitive or even favorable performance when high-precision solutions are not required. Most importantly, our numerical results do confirm that EigPen clearly demonstrates a higher degree of parallel scalability, precisely due to the property that its computations are dominated by highly optimized BLAS3 operations, instead of either by the RR computation (including orthogonalization) or by sequential SpMVs.

The performance of EigPen can be further improved in several aspects, including speeding up convergence and improving accuracy, with the help of a number of techniques such as deflation and locking, polynomial filtering, and Newton-type methods. Two particularly important topics of investigation are (i) a comprehensive study of preconditioning issues, and (ii) a careful evaluation of the parallel efficiency of our approach using the Message Passing Interface (MPI).

Acknowledgements. The computational results were obtained at the National Energy Research Scientific Computing Center (NERSC), which is supported by the Director, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under contract number DE-AC02-05CH11232. Z. Wen would like to thank Prof. Michael Ulbrich for hosting his visit at Technische Universität München. X. Liu would like to thank Prof. Yuhong Dai for discussing nonlinear programming techniques for eigenvalue computation. C. Yang would like to thank Dr. Eugene Vencharynski for helping test EigPen, especially the preconditioned version. The authors are grateful to Prof. Andreas Stathopoulos and two anonymous referees for their detailed and valuable comments and suggestions.

Appendix A. Proofs of technical results.

A.1. Proof of Theorem 2.1. It can be easily seen that condition (2.3) is necessary for the existence of a rank-k stationary point. On the other hand, suppose that μ satisfies (2.3). It is suffice to consider the representation $X = UW$, where U consists of any k eigenvectors of A and $W \in \mathbb{R}^{k\times k}$. Hence, we obtain

$$2f_\mu(X) = \mathrm{tr}(DWW^{\mathrm{T}}) + \frac{\mu}{2}\|W^{\mathrm{T}}W - I\|_F^2,$$

where $D = \mathrm{Diag}(d) \in \mathbb{R}^{k\times k}$ is a diagonal matrix with k eigenvalues of A on the diagonal corresponding to eigenvectors in U. A short calculation shows that

$$\begin{aligned}2f_\mu(X) &= \frac{\mu}{2}\|WW^{\mathrm{T}} + (D/\mu - I)\|_F^2 + \mathrm{tr}(D) - \frac{1}{2\mu}\mathrm{tr}(D^2)\\ &\geq \frac{\mu}{2}\|(D/\mu - I)_+\|_F^2 + \mathrm{tr}(D) - \frac{1}{2\mu}\mathrm{tr}(D^2)\\ &= \sum_{i=1}^k \left(\frac{\mu}{2}\left(\frac{d_i}{\mu} - 1\right)_+^2 + d_i - \frac{d_i^2}{2\mu}\right) \equiv \sum_{i=1}^k \theta(d_i),\end{aligned}$$

where $(t)_+ = \max(0,t)$ and

$$\theta(t) = \frac{\mu}{2}\left(\frac{t}{\mu} - 1\right)_+^2 + t - \frac{t^2}{2\mu} = \begin{cases} t - t^2/(2\mu), & t < \mu,\\ \mu/2, & t \geq \mu.\end{cases}$$

Note that $\theta(t)$ is monotonically nondecreasing since $\theta'(t) = 1 - t/\mu > 0$ in $(-\infty,\mu)$. Substituting the formulation of $\hat{X}$ defined in (2.4) into $f_\mu(\hat{X})$, we obtain

$$2f_\mu(\hat{X}) = \mathrm{tr}(\Lambda_k) - \frac{1}{2\mu}\mathrm{tr}(\Lambda_k^2) = \sum_{i=1}^k \theta(\lambda_i) \leq 2f_\mu(X),$$

which verifies that $\hat{X}$ is a global minimizer. This completes the proof. ☐

A.2. Proof of Theorem 2.2. We only prove the first statement by showing that for $\mu \in (\max(0,\lambda_k),\lambda_n)$ any stationary point other than the global minimizers can only be saddle points. Without loss of generality, consider stationary points in the form of

$$\hat{X} = U[P(I - D/\mu)]^{1/2} = U[(I - D/\mu)P]^{1/2}, \tag{A.1}$$

where $AU = UD$, $U^{\mathrm{T}}U = I$, D is diagonal, and $P \in \mathbb{R}^{k\times k}$ is a diagonal, projection matrix with diagonal entries

$$P_{ii} = \begin{cases} 0, & \text{if } \mu \leq D_{ii},\\ 0 \text{ or } 1, & \text{otherwise.}\end{cases} \tag{A.2}$$

Substituting (A.1) into the Hessian formula (2.7), we obtain

$$\nabla^2 f_\mu(\hat{X})(S) = AS - S(\mu(I - P) + DP) + \mu\hat{X}(S^{\mathrm{T}}\hat{X} + \hat{X}^{\mathrm{T}}S). \tag{A.3}$$

We next show that there exist different matrices $S \in \mathbb{R}^{n\times k}$ at which $\mathrm{tr}(S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S))$ takes opposite signs, unless the stationary point $\hat{X}$ is constructed from eigenvectors associated with a set of k smallest eigenvalues which corresponds to the global minimum.

First assume that $\hat{X}$ has full rank. Then $\mu I \succ D$ and $P = I$ in (A.1). Letting $P = I$ in (A.3) yields

$$\nabla^2 f_\mu(\hat{X})(S) = AS - SD + \mu\hat{X}(S^{\mathrm{T}}\hat{X} + \hat{X}^{\mathrm{T}}S).$$

For $S = U$, we have $S^{\mathrm{T}}\hat{X} = \hat{X}^{\mathrm{T}}S = (I - D/\mu)^{1/2}$ and

$$\mathrm{tr}(S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S)) = 0 + 2\,\mathrm{tr}(\mu I - D) > 0.$$

On the other hand, if $\hat{X}$ is not a global minimizer, without loss of generality we can assume that U contains q_j but not q_i where $\lambda_i < \lambda_j$. Let S contain all zero columns except a single nonzero column that is q_i at the position so that the only nonzero column of SD is $q_i\lambda_j$. For such an S, we have $S^{\mathrm{T}}\hat{X} = 0$ and

$$\mathrm{tr}(S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S)) = q_i^{\mathrm{T}}(Aq_i - q_i\lambda_j) + \mu\,\mathrm{tr}(S^{\mathrm{T}}\hat{X}(S^{\mathrm{T}}\hat{X} + \hat{X}^{\mathrm{T}}S)) = (\lambda_i - \lambda_j) < 0.$$

Hence, all full-rank stationary points are saddle points except the global minimizers.

We now consider the rank-deficient case, namely, there exists at least one zero entry in the diagonal of P, say $P_{ii} = 0$ for some $i \in [1, k]$. Let $\bar{U}$ be the remaining matrix after deleting the i-th column from U. Since $\mathrm{rank}(\bar{U}) = k-1$, there must exist at least one column, denoted by q_j, of Q_k that is not contained in $\bar{U}$. Then it holds $q_j^{\mathrm{T}}\bar{U} = 0$ and $q_j^{\mathrm{T}}Aq_j \le \lambda_k$. Let S contain all zero columns except one nonzero column that is q_j at the i-th position so that both $SP = 0$ and $S^{\mathrm{T}}\hat{X} = 0$. Consequently, in view of (A.3) we have

$$\mathrm{tr}(S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S)) = q_j^{\mathrm{T}}Aq_j - \mu + \mu\,\mathrm{tr}(S^{\mathrm{T}}\hat{X}(S^{\mathrm{T}}\hat{X} + \hat{X}^{\mathrm{T}}S)) \le (\lambda_k - \mu) + 0 < 0.$$

On the other side, let S contain all zero columns except that the i-th column is q_n. For any integer $l \in [1, k]$, if the column $U_l = q_n$, then it can shown that $P_{ll} = 0$ and $q_n^{\mathrm{T}}\hat{X}_l = 0$. Otherwise, the column $U_l \ne q_n$, thus $q_n^{\mathrm{T}}U_l = 0$ which implies $q_n^{\mathrm{T}}\hat{X} = 0$. By our assumption, $\mu < q_n^{\mathrm{T}}Aq_n = \lambda_n$. Hence, $\mathrm{tr}(S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S)) = \lambda_n - \mu > 0$. This completes the proof. □

A.3. Proof of Lemma 2.3. Consider any $S \in Q_k^{\perp}$. In view of (2.7) and (2.4),

$$S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S) = S^{\mathrm{T}}AS + \mu S^{\mathrm{T}}S(\hat{X}^{\mathrm{T}}\hat{X} - I) = S^{\mathrm{T}}AS - S^{\mathrm{T}}SV\Lambda_k V^{\mathrm{T}}, \tag{A.4}$$

where $V \in \mathbb{R}^{k\times k}$ is orthogonal. Since the columns of S are contained in the eigenspace associated with $\{\lambda_{k+1}, \cdots, \lambda_n\}$ and $\mathrm{tr}(S^{\mathrm{T}}S) = 1$, we obtain

$$\lambda_{k+1} \le \mathrm{tr}(S^{\mathrm{T}}AS) \le \lambda_n. \tag{A.5}$$

On the other hand, we note that both $\mathrm{tr}(S^{\mathrm{T}}S(V\Lambda_k V^{\mathrm{T}} - \lambda_1 I))$ and $\mathrm{tr}(S^{\mathrm{T}}S(\lambda_k I - V\Lambda_k V^{\mathrm{T}}))$ are nonnegative, since both are traces for products of symmetric positive semidefinite matrices. These two inequalities imply that

$$\lambda_1 \le \mathrm{tr}(S^{\mathrm{T}}SV\Lambda_k V^{\mathrm{T}}) \le \lambda_k, \tag{A.6}$$

given the fact that $\mathrm{tr}(S^{\mathrm{T}}S) = 1$. From (A.4), (A.5) and (A.6) we deduce

$$\lambda_{k+1} - \lambda_k \le \mathrm{tr}\left(S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S)\right) \le \lambda_n - \lambda_1, \tag{A.7}$$

which proves that the left-hand side of (2.8) is no greater than the right hand side of (2.8). Furthermore, the lower and upper bounds in (A.7) are attained at the $n \times k$ rank-one matrices $S = [0 \cdots 0\, q_{k+1}]$ and $S = [q_n\, 0 \cdots 0]$, respectively. Therefore, the equality in (2.8) must hold, which completes the proof. □

A.4. Proof of Proposition 3.1. Suppose that X^{j+1} is rank deficient. Then there exists a nonzero vector u such that $X^{j+1}u = 0$. In view of (3.1), we have

$$X^j u - \alpha^j \nabla f_\mu(X^j) u = 0. \tag{A.8}$$

Hence, (3.2) holds under $\lambda = 1/\alpha^j$ after multiplying both sides of (A.8) by $(X^j)^{\mathrm{T}}/\alpha^j$. Due to the full rank of X^j, $(X^j)^{\mathrm{T}}(X^j)$ is positive definite. The expression of the gradient in (2.5) implies that $(X^j)^{\mathrm{T}}\nabla f_\mu(X^j)$ is symmetric. Therefore, (3.2) is a generalized symmetric eigenvalue problem. The second part of the proposition follows directly from (A.8). □

A.5. Proof of Lemma 3.2. Since $U \in \mathbb{R}^{n\times d}$ is a basis of $\mathcal{S}$, the solution of (3.7) can be expressed as $X = UW$ for some $W \in \mathbb{R}^{d\times k}$. Substituting $X = UW$ into (3.7) and noting that $U^{\mathrm{T}}AU = \Sigma$ and $U^{\mathrm{T}}U = I$, we reduce (3.7) to

$$\min_{W\in\mathbb{R}^{d\times k}} f_\mu(UW) = \frac{1}{2}\mathrm{tr}(W^{\mathrm{T}}\Sigma W) + \frac{\mu}{4}\|W^{\mathrm{T}}W - I\|_F^2. \tag{A.9}$$

Using the fact that Σ is a diagonal matrix, it can be verified (see Theorem 2.1) that $W = \begin{pmatrix} D & 0\end{pmatrix}^{\mathrm{T}}$, with the diagonal matrix D defined as in (3.10), is indeed a solution of (A.9). Therefore, $Y = UW = U_k D$. □

REFERENCES

[1] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen, *Lapack: a portable linear algebra library for high-performance computers*, in Proceedings of the 1990 ACM/IEEE conference on Supercomputing, Supercomputing '90, IEEE Computer Society Press, 1990, pp. 2–11.

[2] Jonathan Barzilai and Jonathan M. Borwein, *Two-point step size gradient methods*, IMA J. Numer. Anal., 8 (1988), pp. 141–148.

[3] L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK user's guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[4] R. Courant, *Variational methods for the solution of problems of equilibrium and vibrations*, Bull. Amer. Math. Soc., 49 (1943), pp. 1–23.

[5] Y. H. Dai, *On the nonmonotone line search*, J. Optim. Theory Appl., 112 (2002), pp. 315–330.

[6] L. Grippo, F. Lampariello, and S. Lucidi, *A nonmonotone line search technique for Newton's method*, SIAM J. Numer. Anal., 23 (1986), pp. 707–716.

[7] A. Knyazev, M. Argentati, I. Lashuk, and E. Ovtchinnikov, *Block locally optimal preconditioned eigenvalue xolvers (blopex) in hypre and petsc*, SIAM Journal on Scientific Computing, 29 (2007), pp. 2224–2239.

[8] Andrew V. Knyazev, *Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23 (2001), pp. 517–541.

[9] L. Kronik, A. Makmal, M. Tiago, M. M. G. Alemany, X. Huang, Y. Saad, and J. R. Chelikowsky, *PARSEC – the pseudopotential algorithm for real-space electronic structure calculations: recent advances and novel applications to nanostructures*, Phys. Stat. Solidi. (b), 243 (2006), pp. 1063–1079.

[10] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK users' guide: Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*, vol. 6 of Software, Environments, and Tools, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998.

[11] Jorge Nocedal and Stephen J. Wright, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering, Springer, New York, second ed., 2006.

[12] Yousef Saad, James R. Chelikowsky, and Suzanne M. Shontz, *Numerical methods for electronic structure calculations of materials*, SIAM Rev., 52 (2010), pp. 3–54.

[13] Ahmed H. Sameh and John A. Wisniewski, *A trace minimization algorithm for the generalized eigenvalue problem*, SIAM Journal on Numerical Analysis, 19 (1982), pp. pp. 1243–1259.

[14] Andreas Stathopoulos and James R. McCombs, *Nearly optimal preconditioned methods for hermitian eigenproblems under limited memory. Part II: Seeking many eigenvalues*, SIAM Journal on Scientific Computing, 29 (2007), pp. 2162–2188.

[15] ———, *PRIMME: preconditioned iterative multimethod eigensolver–methods and software description*, ACM Trans. Math. Softw., 37 (2010), pp. 21:1–21:30.

[16] Wenyu Sun and Yaxiang Yuan, *Optimization Theory and Methods: Nonlinear Programming*, Springer, New York, 2006.

[17] M. P. Teter, M. C. Payne, and D. C. Allan, *Solution of Schrödinger's equation for large systems*, Physical Review B, 40 (1989), pp. 12255–12263.

[18] Chao Yang, Juan C. Meza, Byounghak Lee, and Lin-Wang Wang, *KSSOLV—a MATLAB toolbox for solving the Kohn-Sham equations*, ACM Trans. Math. Softw., 36 (2009), pp. 1–35.

[19] Hongchao Zhang and William W. Hager, *A nonmonotone line search technique and its application to unconstrained optimization*, SIAM J. Optim., 14 (2004), pp. 1043–1056.

[20] Y. Zhou, *A block Chebyshev-Davidson method with inner-outer restart for large eigenvalue problems*, J. Comput. Phys., 229 (2010), pp. 9188–9200.

[21] Y. Zhou and Y. Saad, *A Chebyshev–Davidson algorithm for large symmetric eigenproblems*, SIAM J. Matrix Anal. and Appl., 29 (2007), pp. 954–971.

[22] ———, *Block krylovschur method for large symmetric eigenvalue problems*, Numerical Algorithms, 47 (2008), pp. 341–359.