Trace-Penalty Minimization for Large-scale Eigenspace Computation

Zaiwen Wen* Chao Yang† Xin Liu‡ Yin Zhang§

Updated August 30, 2013

Abstract

The Rayleigh-Ritz (RR) procedure, including orthogonalization, constitutes a major bottleneck in computing relatively high-dimensional eigenspaces of large sparse matrices. Although operations involved 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.

Keywords. eigenvalue computation, exact quadratic penalty approach, gradient methods

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

*Department of Mathematics, MOE-LSC and Institute of Natural Sciences, Shanghai Jiaotong University, CHINA (zw2109@sjtu.edu.cn). Research supported in part by NSFC grant 11101274, and Humboldt Research Fellowship for Experienced Researchers.

†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, 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, 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.

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, which can often be performed efficiently on both sequential and modern parallel 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.

There are two types of operations that can potentially become bottlenecks. 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 (including Arnoldi or Lanczos) procedure or a QR factorization at the complexity of at least $O(nk^2)$ where n is the dimension of the target matrix and k is the number of desired eigenpairs. Another potentially high-cost procedure is the Rayleigh-Ritz (RR) calculation [14] used to extract eigenvalue and eigenvector approximations from a subspace of dimension $p \geq k$. The RR 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, the RR procedure can sometimes be viewed as a way to construct an orthonormal basis also. The complexity for the RR procedure is at least $O(nk^2 + k^3)$. When the number k is small, the costs of these two types of operations are minor or even negligible. However, when k increases to a moderate portion of the matrix dimension n, these costs can represent a significant, even dominant, portion of the overall cost.

The use of parallel computers can greatly reduce the solution time. However, to make efficient use of these computers, we must ensure that our algorithm is scalable with respect to the number of processors or cores. Although the standard Krylov subspace iterative algorithms 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. Block methods, such as the locally optimal block preconditioned conjugate gradient (LOBPCG) algorithm [10], the block Krylov-Schur algorithm [26] and the Chebyshev-Davidson algorithm [24, 25], are more scalable because more concurrency can be exploited in multiplying a sparse matrix with a block of vectors.

However, block methods have so far not addressed the relatively high cost of performing an RR calculation at each iteration. Although parallel algorithms for solving the dense projected eigenvalue problem are available in multi-thread LAPACK [2] libraries for shared-memory paralle computers and in the ScaLAPACK [4] 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. One way to address this issue is to

use a "spectrum slicing" algorithm [1, 5, 8] that divides the part of the spectrum of interest into a number of intervals and compute eigenvalues within each interval in parallel. However, this approach would require computing interior eigenvalues in each interval which is generally a more difficult task. Moreover, a good initial guess of the eigenvalues of interest is needed so that the spectrum can be divided in an efficient manner [1].

In the Chebyshev-Davidson algorithm [24, 25], the number of RR steps is amortized over a large number of SpMVs because a Chebyshev matrix polynomial filter is applied to a block of vectors before an RR calculation is performed to update the approximate eigenvectors. However, an apparent drawback of this algorithm is the difficulty to take advantage of a good pre-conditioner when it is available. In addition, it is difficult to apply a Chebyshev polynomial filter to generalized eigenvalue problems.

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 starts from the trace minimization formulation 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

$$\min_{X\in\mathbb{R}^{n\times k}} \ \mathrm{tr}(X^{\mathrm{T}}AX), \ \text{s.t.}\ X^{\mathrm{T}}X = I. \tag{1}$$

A major theoretical result of this paper is to establish an equivalence relationship between problem (1) and the following unconstrained optimization problem

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

when the penalty parameter $\mu > 0$ takes suitable finite values. As is well recognized, the objective function in (2) is the classic quadratic (or Courant) penalty function [6, 13, 18] for the constrained problem (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 (2) is essentially equivalent to (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 (2) *trace-penalty minimization*. A key difference between trace minimization 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 pre-conditioning into trace-penalty minimization in a straightforward manner.

In this paper, we consider applying gradient-type methods to the trace-penalty minimization problem

(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 [15]). In this paper we evaluate the efficiency of our algorithm not by measuring the time it takes to compute eigenpairs to a high accuracy close to machine precision, but rather the time it takes to achieve a moderate accuracy in computed eigenpairs. Once good estimates are at hand, there exist a number of techniques that can perform further refinements to obtain a higher accuracy. For example, moderately accurate estimates can be further improved by using a "spectral slicing" type of algorithm [1, 5, 8]. Another possibility is to apply polynomial filtering to do refinements. For the proposed trace-penalty minimization problem (2), we have experimented with various algorithmic options in Matlab, developed a Fortran implementation and parallelize 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. Numerical results are reported in Section 4. Finally, we conclude the paper in Section 5.

2 Trace-Penalty Minimization: Model Analysis

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

$$A = Q_n \Lambda_n Q_n^{\mathrm{T}}, \tag{3}$$

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

$$Q_i = [q_1,\ q_2,\ \ldots, q_i] \in \mathbb{R}^{n\times i}, \quad \Lambda_i = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_i) \in \mathbb{R}^{i\times i}, \tag{4}$$

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 (3). 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 (3) as a generic form of decomposition that represents all possible alternatives.

Given a positive integer $k \le n$, it is well known that the eigenvector matrix Q_k is a solution to the trace minimization problem (1). As is stated in the introduction, instead of solving (1) directly, we propose to solve the trace-penalty minimization problem (2). We first analyze the relationship between the two

problems (1) and (2), and then derive some useful properties for (2).

2.1 Equivalence and other Properties

We start with the following definition of equivalence.

Definition 2.1. *Problem (2) is said to be* equivalent *to (1) if each of its global minimizers spans a k-dimensional eigenspace associated with k smallest eigenvalues of A.*

The first-order necessary conditions for trace minimization (1) can be written as

$$AX - X(X^{\mathrm{T}}AX) = 0, \;\; X^{\mathrm{T}}X = I.$$

On the other hand, the first-order necessary condition for trace-penalty minimization (2) is simply

$$\nabla f_\mu(X) = AX + \mu X(X^{\mathrm{T}}X - I) = 0. \tag{5}$$

Let $L(\mathbb{R}^{n\times k}, \mathbb{R}^{n\times k})$ be the space of linear operators that map $\mathbb{R}^{n\times k}$ to $\mathbb{R}^{n\times k}$. The Fréchet derivative of ∇f_μ at X, or the Hessian of $f_\mu(X)$, has the operator form

$$\nabla^2 f_\mu(X)(S) = AS + \mu S(X^{\mathrm{T}}X - I) + \mu X(S^{\mathrm{T}}X + X^{\mathrm{T}}S), \tag{6}$$

from which one can also derive the matrix representation of $\nabla^2 f_\mu(X)$ in terms of Kronecker products.

The first-order necessary condition (5) implies that each stationary point X of (2) spans an invariant subspace of A, since $\nabla f_\mu(X) = 0$ is obviously equivalent to

$$AX = X(I - X^{\mathrm{T}}X)\mu.$$

Trivially, $X = \mathbf{0} \in \mathbb{R}^{n\times k}$ is always a stationary point of (2). We first study this trivial stationary point and show that it can be eliminated as a minimizer if the penalty parameter μ is sufficiently large.

Lemma 2.2. *Let $\mu > 0$. If $\mu \le \lambda_1$, the zero matrix $X = \mathbf{0} \in \mathbb{R}^{n\times k}$ is the only stationary point of problem (2); otherwise, it is not a minimizer. Moreover, $X = \mathbf{0}$ is a maximizer when $\mu > \lambda_n$.*

Proof. Rearranging (5), we have

$$(\mu I - A)X = \mu X(X^{\mathrm{T}}X). \tag{7}$$

Multiplying X^{T} on both side of (7) yields

$$X^{\mathrm{T}}(\mu I - A)X = \mu(X^{\mathrm{T}}X)^2. \tag{8}$$

If $\mu \le \lambda_1$, the matrix on the left is negative semidefinite while the one on the right is positive semidefinite, forcing the only solution $X = \mathbf{0}$. When $\mu > \lambda_1$, it suffices to note that the Hessian of f_μ at $X = \mathbf{0}$ is

$\nabla^2 f_\mu(\mathbf{0}) = I \otimes (A - \mu I)$ which is not positive semidefinite. Finally, we note that $\nabla^2 f_\mu(\mathbf{0})$ is negative definite when $\mu > \lambda_n$. □

The next lemma shows that any stationary point of (2) can be expressed in terms of eigenpairs of A.

Lemma 2.3. *Let $\mu > 0$ and $(U, D) \in \mathbb{R}^{n\times k} \times \mathbb{R}^{k\times k}$ denote k eigenpairs of A so that $AU = UD$, $U^{\mathrm{T}}U = I$ and D is diagonal. A matrix $X \in \mathbb{R}^{n\times k}$ is a stationary point of* (2) *if and only if*

$$X = U[P(I - D/\mu)]^{1/2}V^{\mathrm{T}}, \tag{9}$$

where $V \in \mathbb{R}^{k\times k}$ is an arbitrary orthogonal matrix, and $P \in \mathbb{R}^{k\times k}$ is a diagonal, projection matrix with diagonal entries

$$P_{ii} = \begin{cases} 0, & \text{if } \mu \le D_{ii}, \\ 0 \text{ or } 1, & \text{otherwise}. \end{cases} \tag{10}$$

In Particular, X is a rank-k stationary point only if $P = I$ and $\mu I - D \succ 0$ (being positive definite).

Proof. We will provide a proof for the case where X is of full-rank. The rank-deficient cases can be proved along the similar line, though more notationally involved and tedious.

Suppose that X is a full rank stationary point, which spans an invariant subspace of A. Since every k-dimensional invariant subspace of A can be spanned by a set of k eigenvectors, we can write $X = UW$ where U consists of k unit eigenvectors of A and $W \in \mathbb{R}^{k\times k}$ is nonsingular. Upon substituting $X = UW$ into (7), we derive

$$U(\mu I - D)W = \mu UW(W^{\mathrm{T}}W) \;\Leftrightarrow\; I - D/\mu = WW^{\mathrm{T}} \;\Leftrightarrow\; W = (I - D/\mu)^{1/2}V^{\mathrm{T}}$$

for some orthogonal $V \in \mathbb{R}^{k\times k}$ (which can possibly hold only if $\mu > D_{ii}$ for $i = 1, 2\cdots, k$). □

Now we establish the equivalence between the trace-penalty minimization model (2) and the trace minimization model (1) for proper μ values.

Theorem 2.4. *Problem (2) is equivalent to (1) if and only if*

$$\mu > \max(0, \lambda_k). \tag{11}$$

Specifically, any global minimizer $\hat{X}$ of (2) *has a singular-value decomposition of the form:*

$$\hat{X} = Q_k(I - \Lambda_k/\mu)^{1/2}V^{\mathrm{T}} \tag{12}$$

where Q_k and Λ_k are defined as in (4)*, and $V \in \mathbb{R}^{k\times k}$ is any orthogonal matrix.*

Proof. It can be easily seen from (8) that condition (11) is necessary for the existence of a rank-k stationary point. On the other hand, suppose that μ satisfies (11). Using Lemma 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 (12) 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. □

The next theorem indicates that our trace-penalty minimization model (2) can have far fewer undesirable, full-rank stationary points than the trace minimization model (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 2.5. *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 (12). 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.*

Proof. To prove the first statement, we show 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 (9) with $V = I$, that is

$$\hat{X} = U[P(I - D/\mu)]^{1/2} = U[(I - D/\mu)P]^{1/2}, \tag{13}$$

where $AU = UD$, $U^{\mathrm{T}}U = I$, and D is diagonal. The proof still holds for an arbitrary orthogonal matrix V since the function value $f_\mu(\hat{X})$ is invariant with respect to V. Substituting (13) into the Hessian formula (6), we obtain

$$\nabla^2 f_\mu(\hat{X})(S) = AS - S(\mu(I-P) + DP) + \mu\hat{X}(S^{\mathrm{T}}\hat{X} + \hat{X}^{\mathrm{T}}S). \tag{14}$$

We next show that there exists two different matrices $S \in \mathbb{R}^{n\times k}$ such that $\mathrm{tr}(S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S)) < 0$ and $\mathrm{tr}(S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S)) > 0$, respectively, 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 (13). Letting $P = I$ in (14) 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

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

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 (14) 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 follows from Lemma 2.3 that $P_{ll} = 0$ and $q_n^{\mathrm{T}}\hat{X}_l = 0$. Otherwise, the column $U_l \neq q_n$, 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 complete the proof of the first statement.

The second part of this theorem is a direct consequence of the full-rank requirement and the stationary-point expression (9) which, together, demands $\mu I \succ D$. Hence, for $\mu \in (\max(0, \lambda_k), \lambda_{k+p})$, D can only have a set of k smallest eigenvalues of A on its diagonal. □

2.2 Error Bounds between Optimality Conditions

After establishing the equivalence between our trace-penalty minimization model (2) and the original trace minimization model (1), we investigate the relationship between the first-order optimality conditions of the two models, which would play an important role in setting the stopping tolerance for an iterative algorithm to solve (2).

Given any approximate solution X of (2), an orthonormal basis for the range space of X, say $Y(X)$, is a feasible solution of (1). Specifically, let X be of full rank and $X = U\Sigma V^{\mathrm{T}}$ denote the partial (or economy-form) singular value decomposition (SVD) of X, where $U \in \mathbb{R}^{n\times k}$ and $V \in \mathbb{R}^{k\times k}$ have orthonormal columns, and $\Sigma \in \mathbb{R}^{k\times k}$ is a diagonal matrix with the singular values of X on its diagonal. Then a particular choice for $Y(X)$ is

$$Y(X) \triangleq U. \tag{15}$$

Consequently, the violation of the first-order necessary conditions of the trace minimization (1) can be measured by the Frobenious norm of the residual

$$R(X) \triangleq AY(X) - Y(X)\left(Y(X)^{\mathrm{T}}AY(X)\right). \tag{16}$$

Lemma 2.6. *Let $\mu > \max(0, \lambda_k)$, and $\nabla f_\mu(X)$ and $R(X)$ be defined as in* (5) *and* (16)*, respectively. Then*

$$\|R(X)\|_F \le \sigma_{\min}^{-1}(X)\|\nabla f_\mu(X)\|_F, \tag{17}$$

where $\sigma_{\min}(X)$ is the smallest singular value of X. Moreover, for any global minimizer $\hat{X}$ and any $\epsilon > 0$, there exists $\delta > 0$ such that whenever $\|X - \hat{X}\|_F \le \delta$,

$$\|R(X)\|_F \le \frac{1+\epsilon}{\sqrt{1-\lambda_k/\mu}}\|\nabla f_\mu(X)\|_F. \tag{18}$$

Proof. Recall that $X = U\Sigma V^{\mathrm{T}}$ where the columns of U form an orthonormal basis for the range space of X. Projecting $\nabla f_\mu(X)$ onto the null space of X^{T} and using the definition of $R(X)$ in (16), we obtain

$$\begin{aligned}(I - UU^{\mathrm{T}})\nabla f_\mu(X) &= (I - UU^{\mathrm{T}})(AX + \mu X(X^{\mathrm{T}}X - I)) \\ &= (I - UU^{\mathrm{T}})AX = (I - UU^{\mathrm{T}})AU\Sigma V^{\mathrm{T}} \\ &= R(X)\Sigma V^{\mathrm{T}}.\end{aligned}$$

A rearrangement of the above equality gives $R(X) = (I-UU^{\mathrm{T}})\nabla f_\mu(X)V\Sigma^{-1}$,which leads to (17) through

the following steps,

$$\begin{aligned}\|R(X)\|_F &= \|(I-UU^{\mathrm{T}})\nabla f_\mu(X)V\Sigma^{-1}\|_F \\ &\leq \|(I-UU^{\mathrm{T}})\nabla f_\mu(X)V\|_F\,\|\Sigma^{-1}\|_2 \\ &= \|(I-UU^{\mathrm{T}})\nabla f_\mu(X)\|_F\,\sigma_{\min}^{-1}(X) \\ &\leq \sigma_{\min}^{-1}(X)\,\|\nabla f_\mu(X)\|_F,\end{aligned}$$

where the last inequality is due to the fact that $I-UU^{\mathrm{T}}$ is a projection.

To see the second part of this lemma, we recall Theorem 2.4 that gives $\sigma_{\min}(\hat{X}) = \sqrt{1-\lambda_k/\mu}$ at any global minimizer $\hat{X}$. This completes the proof. □

2.3 The Hessian at Solution

An important quantity for smooth unconstrained optimization is the condition number of the Hessian at a solution, which is defined (in Euclidean norm) as

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

where $\lambda_{\max}(\cdot)$ (or $\lambda_{\min}(\cdot)$) stands for the largest (or the smallest) eigenvalue of the referred matrix, and $\hat{X}$ is a global minimizer of (2). Obviously, κ is infinity when the involved matrix is singular.

If $k=1$ and $\lambda_1 < \lambda_2$, according to Theorem 2.4 there exists exactly two isolated global minimizers for (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 2.4 that there is no isolated global minimizer. Hence, the Hessian at any solution is singular. If fact, let $P = -P^T \in \mathbb{R}^{k\times k}$ be any nonzero skew-symmetric matrix. Then we can easily verify from (5) and (6) that for $S = \hat{X}P \neq 0$, $\nabla^2 f_\mu(\hat{X})(S) = \nabla f_\mu(\hat{X})P = 0$, implying Hessian singularity. Clearly, this singularity manifold, defined by $S = \hat{X}P$, lies within the optimal eigenspace spanned by the columns of $\hat{X}$, which directly reflects the fact that the optimal solution set is a connected non-singleton set. However, from the viewpoint of numerical computation, such type of singularity is benign since we would be equally satisfied with any point in the solution set.

The following result, Lemma 2.7, examines the restricted condition number of the Hessian outside of the optimal eigenspace. Specifically, the definition is

$$\kappa\left(\nabla^2 f_\mu(\hat{X})\,\Big|_{Q_k^\perp}\right) \triangleq \frac{\max\limits_{S\in\mathbb{R}^{n\times k}}\left\{\mathrm{tr}(S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S)) : \mathrm{tr}(S^{\mathrm{T}}S)=1, S^{\mathrm{T}}Q_k=0\right\}}{\min\limits_{S\in\mathbb{R}^{n\times k}}\left\{\mathrm{tr}(S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S)) : \mathrm{tr}(S^{\mathrm{T}}S)=1, S^{\mathrm{T}}Q_k=0\right\}}. \tag{19}$$

Lemma 2.7. *Let $k > 1$, $\mu > \max(0, \lambda_k)$ and $\hat{X}$ be any global minimizer of f_μ in* (2)*. Then*

$$\kappa\left(\nabla^2 f_\mu(\hat{X})\Big|_{Q_k^\perp}\right) = \frac{\lambda_n - \lambda_1}{\lambda_{k+1} - \lambda_k}. \tag{20}$$

Proof. Consider any $S \in \mathbb{R}^{n\times k}$ such that $S^{\mathrm{T}}Q_k = 0$ and $\mathrm{tr}(S^{\mathrm{T}}S) = 1$. In view of (6) and (12),

$$S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S) = S^{\mathrm{T}}AS + \mu S^{\mathrm{T}}S(\hat{X}^{\mathrm{T}}\hat{X} - I) = S^{\mathrm{T}}AS - S^{\mathrm{T}}SV\Lambda_k V^{\mathrm{T}}, \tag{21}$$

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

$$\lambda_{k+1} \le \mathrm{tr}(S^{\mathrm{T}}AS) \le \lambda_n. \tag{22}$$

On the other hand, we note that $\mathrm{tr}(S^{\mathrm{T}}S(V\Lambda_k V^{\mathrm{T}} - \lambda_1 I)) \ge 0$ and $\mathrm{tr}(S^{\mathrm{T}}S(\lambda_k I - V\Lambda_k V^{\mathrm{T}})) \ge 0$, since both are traces for products of symmetric positive semidefinite matrices. These two inequalities imply that

$$\lambda_1 \le \mathrm{tr}(S^{\mathrm{T}}SV\Lambda_k V^{\mathrm{T}}) \le \lambda_k, \tag{23}$$

given the fact that $\mathrm{tr}(S^{\mathrm{T}}S) = 1$. From (21), (22) and (23) we deduce

$$\lambda_{k+1} - \lambda_k \le \mathrm{tr}\left(S^{\mathrm{T}}\nabla^2 f_\mu(\hat{X})(S)\right) \le \lambda_n - \lambda_1, \tag{24}$$

which proves that the left-hand side of (20) is no greater than the right hand side of (20). Furthermore, the lower and upper bounds in (24) 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 (20) must hold, which completes the proof. □

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, the worst-case conditioning in (20) is unlikely to happen in practice as converging iterates remain full rank near a solution.

2.4 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:

$$\min_{X\in\mathbb{R}^{n\times k}} \mathrm{tr}(X^{\mathrm{T}}AX), \text{ s.t. } X^{\mathrm{T}}BX = I, \tag{25}$$

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

$$\min_{X\in\mathbb{R}^{n\times k}} \; f_\mu(X) := \frac{1}{2}\mathrm{tr}(X^{\mathrm{T}}AX) + \frac{\mu}{4}\|X^{\mathrm{T}}BX - I\|_F^2. \tag{26}$$

In fact, by change of variable $Z = B^{\frac{1}{2}}X$ (where the symmetric matrix $B^{\frac{1}{2}}$ satisfies $B^{\frac{1}{2}}B^{\frac{1}{2}} = B$), the generalized eigenvalue problem (25) can be converted to a standard eigenvalue problem

$$\min_{Z\in\mathbb{R}^{n\times k}} \; \mathrm{tr}(Z^{\mathrm{T}}\bar{A}Z), \text{ s.t. } Z^{\mathrm{T}}Z = I, \tag{27}$$

where $\bar{A} = B^{-\frac{1}{2}}AB^{-\frac{1}{2}}$. As a result, our model analysis, directly applicable to (27), can be translated to (26) in a straightforward manner. For example, Theorem 2.4 gives the global minimizers of (27) as

$$Z = \bar{Q}_k(I - \Lambda_k/\mu)V^{\mathrm{T}},$$

where Λ_k is diagonal with k smallest eigenvalues of $\bar{A}$ on its diagonal that also happen to be the generalized eigenvalues of the matrix pair (A, B), and $\bar{Q}_k$ consists of corresponding eigenvector of $\bar{A}$. By the change of variables $Z = B^{\frac{1}{2}}X$, then

$$X = Q_k(I - \Lambda_k/\mu)V^{\mathrm{T}}, \tag{28}$$

where $Q_k = B^{-\frac{1}{2}}\bar{Q}_k$ consists of the generalized eigenvectors associated with the k smallest generalized eigenvalues in Λ_k. Naturally, the equivalence of the trace-penalty minimization model (26) to the trace minimization model (25) requires that $\mu > \max(0, \lambda_k)$ where λ_k is a k-th smallest generalized eigenvalue of the matrix pair (A, B).

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:

$$\min_{X\in\mathbb{R}^{n\times k}} \; \mathrm{tr}(X^{\mathrm{T}}AX), \text{ s.t. } X^{\mathrm{T}}BX = I, \;\; U^{\mathrm{T}}X = 0. \tag{29}$$

The variation (29) can arise from a deflation procedure where U is constructed from already converged eigenvectors. The trace-penalty minimization model corresponding to (29) is

$$\min_{X\in\mathbb{R}^{n\times k}} \; f_\mu(X) := \frac{1}{2}\mathrm{tr}(X^{\mathrm{T}}AX) + \frac{\mu}{4}\|X^{\mathrm{T}}BX - I\|_F^2, \text{ s.t. } U^{\mathrm{T}}X = 0. \tag{30}$$

Starting from X^0 such that $U^{\mathrm{T}}X^0 = 0$, a projected-gradient method for solving (30) has the form $X^{j+1} = X^j - \alpha^j(I - UU^{\mathrm{T}})\nabla f_\mu(X^j)$ which is just a slight modification of regular gradient methods to be discussed in details in the next section.

The principle of trace-penalty minimization can in fact be applied to other types of eigenvalue problems, but for the sake of space we will leave further extensions to future work.

3 Algorithmic Framework

3.1 Gradient Methods for Trace-Penalty Minimization

The trace-penalty minimization model 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:

$$X^{j+1} = X^j - \alpha^j \nabla f_\mu(X^j), \tag{31}$$

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 2.5 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 lemma suggests that iterates generated by (31) will most likely remain full rank.

Proposition 3.1. *Let X^{j+1} be generated by* (31) *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:*

$$[(X^j)^{\mathrm{T}} \nabla f_\mu(X^j)]u = \lambda[(X^j)^{\mathrm{T}}(X^j)]u. \tag{32}$$

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.

Proof. Suppose that X^{j+1} is rank deficient. Then there exists a~~n~~ nonzero vector u such that $X^{j+1}u = 0$. In view of (31), we have

$$X^j u - \alpha^j \nabla f_\mu(X^j)u = 0. \tag{33}$$

Hence, (32) holds under $\lambda = 1/\alpha^j$ after multiplying both sides of (33) 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 (5) implies that $(X^j)^{\mathrm{T}}\nabla f_\mu(X^j)$ is symmetric. Therefore, (32) is a generalized symmetric eigenvalue problem. The second part of the proposition follows directly from (33). □

We next present a few strategies for choosing the step size α^j. Given an arbitrary direction $D \in \mathbb{R}^{n\times k}$, the objective function $f_\mu(X + \alpha D)$ is a quartic function of α; precisely,

$$\begin{aligned} f_\mu(X + \alpha D) \;=\; & \frac{1}{2}\mathrm{tr}(X^{\mathrm{T}}AX) + \frac{\mu}{4}\mathrm{tr}(B^{\mathrm{T}}B) + \left(\mathrm{tr}(D^{\mathrm{T}}AX) + \frac{\mu}{2}\mathrm{tr}(B^{\mathrm{T}}W)\right)\alpha \\ & + \left(\mathrm{tr}(D^{\mathrm{T}}AD) + \frac{\mu}{2}\mathrm{tr}(B^{\mathrm{T}}H) + \frac{\mu}{4}\mathrm{tr}(W^{\mathrm{T}}W)\right)\alpha^2 \\ & + \left(\frac{\mu}{2}\mathrm{tr}(W^{\mathrm{T}}H)\right)\alpha^3 + \left(\frac{\mu}{4}\mathrm{tr}(H^{\mathrm{T}}H)\right)\alpha^4, \end{aligned} \tag{34}$$

where $B = X^{\mathrm{T}}X - I$, $W = D^{\mathrm{T}}X + X^{\mathrm{T}}D$ and $H = D^{\mathrm{T}}D$. The steepest descent gradient method computes the step size by using a~~n~~ one-dimensional exact minimization, i.e., $\alpha^j = \mathrm{argmin}\, f_\mu(X^j - \alpha\nabla f(X^j))$,

which is determined by a root of the cubic equation $df_\mu(X^j - \alpha\nabla f(X^j)/d\alpha = 0$. Note that $\mu > 0$ and $\mathrm{tr}(H^{\mathrm{T}}H) > 0$ for $\nabla f_\mu(X) \neq 0$, a positive root always exits. Although executing exact line searches along each steepest descent direction often converges slowly, it has been demonstrated in [21, 22] that mixing it with some other step sizes in an alternative fashion can accelerate convergence significantly.

Another successful approach is to use line search with a Barzilai-Borwein (BB) size [3]. Let

$$S^j := X^j - X^{j-1} \quad \text{and} \quad Y^j = \nabla f_\mu(X^j) - \nabla f_\mu(X^{j-1}). \tag{35}$$

The BB step size is

$$\alpha^j_{\mathrm{BB1}} = \frac{\mathrm{tr}((S^j)^{\mathrm{T}}Y^j)}{\|Y^j\|_{\mathrm{F}}^2} \quad \text{or} \quad \alpha^j_{\mathrm{BB2}} = \frac{\|S^j\|_{\mathrm{F}}^2}{\mathrm{tr}((S^j)^{\mathrm{T}}Y^j).} \tag{36}$$

Since $S^j = \alpha^{j-1}\nabla f_\mu(X^{j-1})$, the computation of the BB step sizes only requires to store one intermediate matrix Y^j in (35). 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 (36) 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 (36), $\delta \in (0, 1)$ and h is the smallest positive integer satisfying the condition

$$f_\mu(X^j - \alpha\delta^h\nabla f^j_\mu) \leq 2f_\mu(X^j). \tag{37}$$

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 [7, 9, 23]. We have found, however, that on our trace-penalty function $f_\mu(X)$ condition (37) 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 (36), the simple line search condition (37), and a termination rule

$$\|\nabla f_\mu(X^j)\|_{\mathrm{F}} \leq \epsilon, \tag{38}$$

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 ϵ, $\delta \in (0,1)$ 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 $\alpha\delta^h$ satisfying (37);
  update $X^{j+1} = X^j - \alpha^j \nabla f_\mu(X^j)$;
  compute α using the first formula in (36);
  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 (35), 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

$$Y = \operatorname*{argmin}_{X\in\mathbb{R}^{n\times k}} \{f_\mu(X) : X \in \mathcal{S}\}, \tag{39}$$

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

$$f_\mu(Y) \le f_\mu(X^J). \tag{40}$$

On the other hand, consider the subspace trace minimization problem

$$U = \operatorname*{argmin}_{X\in\mathbb{R}^{n\times d}} \left\{\mathrm{tr}(X^{\mathrm{T}}AX) : X^{\mathrm{T}}X = I,\ X \in \mathcal{S}\right\}, \tag{41}$$

where d is the dimension of the subspace $\mathcal{S}$. Clearly, the RR step $(U, \Sigma) = \mathrm{RR}(X^J)$ is equivalent to solving (41) for $\mathcal{S} = \mathbf{span}\{X^J\}$ (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 (39) and (40) can be explicitly

constructed from the RR step output $(U, \Sigma) = \mathrm{RR}(X^J)$. We first consider the simple case $\mathcal{S} = \mathbf{span}\{X^J\}$.

Lemma 3.2. *Let $\mathcal{S} = \mathbf{span}\{X^J\}$ where $X^J \in \mathbb{R}^{n\times k}$ has full rank, and let U be defined in* (41) *so that $U^{\mathrm{T}}AU = \Sigma$ is diagonal. Then a Y in* (39) *has the form $Y = U(I - \Sigma/\mu)^{1/2}$, provided that $\mu I \succ \Sigma$.*

Now we prove a more general result that contains the above as a special case.

Lemma 3.3. *Let $\mathcal{S} \supseteq \mathbf{span}\{X^J\}$ have dimension $d \geq k$, and U be defined in* (41) *so that $U^{\mathrm{T}}AU = \Sigma$ is diagonal whose diagonal elements are arranged in an ascending order. Then a matrix Y in* (39) *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*

$$D_{ii} = \max\left(0, 1 - \frac{1}{\mu}\Sigma_{ii}\right)^{1/2}, \ i = 1, 2, \cdots, k. \tag{42}$$

Proof. Since $U \in \mathbb{R}^{n\times d}$ is a basis of $\mathcal{S}$, the solution of (39) can be expressed as $X = UW$ for some $W \in \mathbb{R}^{d\times k}$. Substituting $X = UW$ into (39) and noting that $U^{\mathrm{T}}AU = \Sigma$ and $U^{\mathrm{T}}U = I$, we reduce (39) to

$$\min_{W\in\mathbb{R}^{d\times k}} f_\mu(UW) = \frac{1}{2}\mathrm{tr}(W^{\mathrm{T}}\Sigma W) + \frac{\mu}{4}\|W^{\mathrm{T}}W - I\|_F^2. \tag{43}$$

Using the fact that Σ is a diagonal matrix, it can be verified (see Theorem 2.4) that $W = \begin{pmatrix} D & 0 \end{pmatrix}^{\mathrm{T}}$, with the diagonal matrix D defined as in (42), is indeed a solution of (43). Therefore, $Y = UW = U_k D$. □

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

$$\|\nabla f_\mu(X^j)\|_{\mathrm{F}} \leq \epsilon_i \ \max(1, \|AX^j\|_{\mathrm{F}}) \tag{44}$$

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.

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 (41) to be $\mathcal{S} = \mathbf{span}\left\{X^{j-1}, X^j, AX^j\right\}$, then the RR step would generate a step similar to those in the LOBPCG algorithm [10]. 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 penalty parameter μ. Theorem 2.4 states that $\mu > \max(0, \lambda_k)$ is necessary and sufficient for the equivalence between (1) and (2). A more restrictive range for μ is given in

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**
    compute the smallest integer h so that $\alpha^j = \alpha\delta^h$ satisfying (37);
    update $X^{j+1} = X^j - \alpha^j \nabla f_\mu(X^j)$;
    compute α using the first formula in (36);
    increment j and continue.
  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 $\epsilon_{i+1} = \delta_\epsilon \epsilon_i$ and increment i.

Theorem 2.5 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, Lemma 2.6 suggests that 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 $(X^0)^{\mathrm{T}} A X^0$, which provides an initial choice

$$\mu = \max(c_1,\, c_2 \max(\mathrm{diag}((X^0)^{\mathrm{T}} A X^0))), \tag{45}$$

where $c_1 > 0$ and $c_2 > 1$ are two constants for safeguarding. Another estimation of μ comes from the structure of the minimizer $\hat{X}$ given in (12), which yields the eigenvalue decomposition

$$\mu(I - \hat{X}^{\mathrm{T}}\hat{X}) = V \Lambda_k V^{\mathrm{T}}, \tag{46}$$

where V and Λ_k are defined in Theorem 2.4. Once a "good" iterate X^j is at hand after some iterations during trace-penalty minimization, the relationship (46) implies that λ_k can be estimated from the maximum eigenvalues of $I - (X^j)^{\mathrm{T}} X^j$, i.e., $\bar{\lambda}_k^j = \mu\lambda_{\max}(I - (X^j)^{\mathrm{T}} X^j)$. Since the computational cost of approximating the largest eigenvalue of a $k \times k$ matrix is relatively low, the penalty parameter μ can be updated during trace-penalty minimization by the formula

$$\mu = \max(c_1,\, c_2 \bar{\lambda}_k^j). \tag{47}$$

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

$$\mu = \max(c_1,\, c_2 \theta_k). \tag{48}$$

In favorable cases where the gap between λ_k and λ_{k+1} is relatively large, our strategy of choosing μ slightly larger than λ_k would have a good chance to satisfy both $\mu > 0$ and $\mu \in (\lambda_k, \lambda_{k+1})$, provided that $\lambda_k > 0$. In order to ensure $\lambda_k > 0$, our current strategy is to first scale the matrix A by $\sigma \approx |\lambda_1|$, assuming that $\lambda_1 < 0$, and then add a positive shift $\omega > 1$, obtaining

$$\hat{A} = \frac{1}{\sigma} A + \omega I \tag{49}$$

that is at least close to being positive semidefinite. After performing trace-penalty minimization to $\hat{A}$, the above scale and shift can be easily reversed to recover the eigenvalues of A.

To estimate λ_1, we note that the well-known Gershgorin circle theorem implies that

$$\lambda_1 \geq u_1 := \min_{i=1,\ldots,n} \left\{ A_{ii} - \sum_{j \neq i} |A_{ij}| \right\}.$$

In addition, the relationship between matrix norms implies that

$$u_2 := \frac{\max(\|A\|_\infty, \|A\|_F)}{\sqrt{n}} \leq \|A\|_2 = \max(|\lambda_1|, |\lambda_n|).$$

Hence, without too much computation a reasonable value of σ in (49) is taken as

$$\sigma = \max(\min(|u_1|, u_2), 1). \tag{50}$$

As long as σ is not much smaller than $|\lambda_1|$, setting ω to a moderate number between 1 to 10 usually works well in our tests. In our numerical experiments, we always take the safe value of $\omega = 10$.

4 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.

4.1 Solvers, Test Matrices and Platform

We choose to compare EigPen with two state-of-the-art eigensolvers: the locally optimal preconditioned conjugate gradient (LOBPCG) algorithm and the preconditioned iterative multi-method eigensolver [16, 17] (PRIMME, version 1.1)[1]. Both EigPen and LOBPCG are implemented in Fortran and parallelized by using OpenMP. Instead of using the official version of LOBPCG — BLOPEX[2], we adopt an implementation previously developed by the second author of this paper, since this version supports operations in blocks of vectors while none of current BLOPEX implementations does (up to the writing of the present paper in

[1]Downloadable from `http://http://www.cs.wm.edu/~andreas/software`

[2]Downloadable from `http://code.google.com/p/blopex`

2013). The algorithmic framework of our LOBPCG version is the same as BLOPEX. PRIMME is written in the C language and it is compiled using the OpenMP flags and linked to the same BLAS and LAPACK libraries as EigPen and LOBPCG. We also compared EigPen with an OpenMP version of ARPACK [12]. Because the performance of ARPACK is much worse than that of EigPen except for a few problems, we choose not to report its performance here. The poor performance of ARPACK is mainly due to the lack of effective parallelization for SpMVs since they are performed one vector at a time. Since ARPACK is built on top of BLAS2, the relative poor performance of dense matrix operations also contributed to the poor performance.

In order to exhibit the fundamental algorithmic difference between EigPen and LOBPCG, we do not add any well-known acceleration or stabilization techniques such as deflation to their implementations. The block sizes of EigPen and LOBPCG are set to the same number so that both solvers consume almost the same amount of memory. On the other hand, PRIMME is a sophisticated software package based on Davidson/Jacobi-Davidson iteration. Instead of computing all eigenvectors in a single large block as is done in LOBPCG and EigPen, PRIMME calculates a few eigenpairs at a time using a windowed approach with locking. In addition, it is enhanced by a few special techniques, such as restarting and stagnation-proof locking, etc. 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 [11], 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 $\sigma = 5.7769 \times 10^9$ as is defined by (50) and denote the scaled matrix by "shallow_water1s".

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%

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)

[3]Downloadable from `http://www.cise.ufl.edu/research/sparse/matrices`

[4]Detailed information is available at `http://www.nersc.gov/users/computational-systems/hopper/`

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 "MagnyCours" processor.

We use the multi-threaded version of the Cray Scientific Libraries package, LibSci, which includes multi-threaded versions BLAS and LAPACK subroutines optimized for Cray XE6. While each individual sparse matrix-vector multiplication (SpMV) is not parallelized in all solvers, a loop-level parallelization is applied so that the columns of AX are 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). 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 [26] and the (block) Chebyshev-Davidson algorithm [24, 25] are not included in our comparison, partly because suitable Fortran/OpenMP implementations of these algorithms were unavailable for our tests.

4.2 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,

$$\text{res}_i(U) = \frac{\|Au_i - \theta_i u_i\|_2}{\max(1, |\theta_i|)} \leq tol, \quad i = 1, \cdots, nev, \tag{51}$$

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 the Ritz-pair). 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 $nev \times 1.1$ (round to the nearest integer).

In EigPen, the initial penalty parameter μ is computed by (45) and it is updated by (47) at most three times in the first outer iteration of EigPen. After an RR step is executed, μ is set according to (48) and is fixed throughout the next round of inner iterations. The constants $c_1 = 0.1$ and $c_2 = 1.1$ are used in (45), (47) and (48). 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 $\mathrm{res}_i \leq \mathrm{tol}$, for $i = 1, \ldots, nev$, after each RR step:

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

As is already mentioned, in order to facilitate the selection of our penalty parameter μ in EigPen we perform scaling and shifting as in (49) where σ is given by (50) and $\omega = 10$ is always used.

The basic termination rule of PRIMME is

$$\|Au_i - \theta_i u_i\|_2 \leq tol, \quad i = 1, \cdots, nev. \tag{52}$$

To prevent stagnation, PRIMME also employs certain heuristics to declare near-convergence of a Ritz pair even when (52) is not satisfied. Due to the existence of such heuristics, we keep the stopping rules of PRIMME intact, which of course differ from (51). Since the magnitude of many eigenvalues is of order $O(1)$ for the tested matrices, the two stopping rules (51) and (52) are nevertheless comparable to a large extent. Throughout our experiments, we set the PRIMME parameters

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

Our experiments show that, at least on the particular computer node and the particular set of test problems, the above parameter setting provides overall better performance than the default PRIMME settings (though it might not be the best possible since our tuning was not exhaustive).

4.3 Overall Performance

We first report the overall performance of ARPACK, LOBPCG and EigPen 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 "– –".

We observe from Table 2 that LOBPCG did not succeed on three matrices: c_65, Ga10As10H30 and Ga3As3H12 for $tol = 10^{-4}$. In general, EigPen is faster than LOBPCG, especially on larger problems, with only one significant exception on the matrix c_65 for $tol = 10^{-3}$. 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

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

		$tol = 10^{-3}$			$tol = 10^{-4}$		
Matrix	nev	LOBPCG	PRIMME	EigPen	LOBPCG	PRIMME	EigPen
Andrews	600	646	525	248	1129	581	601
C60	200	38	23	35	53	27	47
c_65	500	572	757	5944	– –	820	6612
cfd1	700	1983	4250	703	2890	3103	1327
finance	700	1668	1359	639	4679	1108	901
Ga10As10H30	1000	8439	3691	2538	– –	3997	4125
Ga3As3H12	600	9639	1086	760	– –	1309	1183
OPF3754	200	22	23	13	43	28	22
shallow_water1s	800	1799	2734	420	3782	1842	1247
Si10H16	200	89	40	33	100	47	62
Si5H12	200	100	49	32	115	57	38
SiO	400	307	214	134	1866	249	231
wathen100	300	344	154	137	1109	164	289

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 LOBPCG (denoted by "LOBPCG-iter"), 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 EigPen makes only a few RR calls, whereas LOBPCG calls RR at every iteration.

Table 3: The number of iterations performed by LOBPCG and EigPen

	$tol = 10^{-3}$		$tol = 10^{-4}$	
Matrix	LOBPCG-iter	EigPen-(GD,RR)	LOBPCG-iter	EigPen-(GD,RR)
Andrews	71	(113, 5)	86	(299, 5)
C60	50	(178, 9)	65	(266, 6)
c_65	167	(5368,12)	2180	(6011,11)
cfd1	135	(253, 4)	205	(523, 4)
finance	119	(235, 5)	533	(348, 4)
Ga10As10H30	317	(334, 5)	130	(590, 5)
Ga3As3H12	499	(310, 3)	465	(550, 3)
OPF3754	34	(86, 1)	79	(152, 1)
shallow_water1s	96	(108, 4)	242	(236, 6)
Si10H16	87	(150, 8)	100	(316, 7)
Si5H12	90	(134, 5)	103	(176, 5)
SiO	83	(175, 5)	415	(332, 7)
wathen100	165	(320, 5)	808	(746, 6)

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 (51) is satisfied for all $i \leq nev$. To measure the accuracy of the computation, we compute the relative difference between θ_i and the true eigenvalue λ_i computed in advance by ScaLAPACK [4]. The maximum relative

errors among all eigenvalues, which is measured by

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

are reported in Table 4, and the relative errors between the sum of the nev eigenvalues, defined by

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

are presented in Table 5. From these tables, we see that LOBPCG 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 4: A comparison of err_θ among different solvers

	$tol = 10^{-3}$			$tol = 10^{-4}$		
Matrix	LOBPCG	PRIMME	EigPen	LOBPCG	PRIMME	EigPen
Andrews	4.58e-05	1.61e-07	4.58e-05	4.58e-05	2.93e-09	4.58e-05
C60	4.34e-05	2.45e-06	4.34e-05	4.34e-05	2.74e-08	4.34e-05
c_65	4.79e-05	1.54e-08	4.79e-05	– –	2.15e-10	4.79e-05
cfd1	1.19e-05	1.60e-04	2.44e-04	5.37e-07	6.92e-07	6.90e-06
finance	4.80e-05	2.44e-06	8.02e-05	4.80e-05	3.60e-08	4.80e-05
Ga10As10H30	4.97e-05	2.82e-07	4.97e-05	– –	3.65e-09	4.97e-05
Ga3As3H12	4.69e-05	9.43e-08	4.69e-05	– –	7.45e-10	4.69e-05
OPF3754	3.46e-05	2.17e-15	3.46e-05	3.46e-05	1.47e-15	3.46e-05
shallow_water1s	1.29e-04	1.25e-04	3.22e-04	4.88e-05	2.07e-07	4.88e-05
Si10H16	4.33e-05	3.44e-07	4.33e-05	4.33e-05	5.50e-09	4.33e-05
Si5H12	3.86e-05	2.02e-07	3.86e-05	3.86e-05	1.87e-09	3.86e-05
SiO	4.81e-05	2.01e-07	4.81e-05	4.81e-05	8.67e-10	4.81e-05
wathen100	3.17e-05	6.14e-08	3.17e-05	3.17e-05	5.24e-10	3.17e-05

Table 5: A comparison of err_{trace} among different solvers

	$tol = 10^{-3}$			$tol = 10^{-4}$		
Matrix	LOBPCG	PRIMME	EigPen	LOBPCG	PRIMME	EigPen
Andrews	1.94e-07	2.98e-08	4.56e-07	1.07e-07	4.65e-10	1.07e-07
C60	4.91e-08	6.03e-08	7.84e-09	9.01e-08	8.87e-10	9.01e-08
c_65	8.20e-07	1.19e-09	8.20e-07	– –	1.68e-11	8.20e-07
cfd1	3.50e-04	9.51e-04	3.57e-03	1.43e-06	7.62e-06	2.00e-05
finance	5.02e-07	2.82e-07	1.39e-06	3.91e-07	3.86e-09	3.91e-07
Ga10As10H30	3.10e-07	5.42e-08	3.07e-07	– –	7.91e-10	3.10e-07
Ga3As3H12	1.01e-06	1.74e-08	1.01e-06	– –	2.12e-10	1.01e-06
OPF3754	8.09e-07	1.97e-16	8.09e-07	8.09e-07	1.97e-16	8.09e-07
shallow_water1s	6.85e-06	3.44e-06	1.65e-05	1.62e-06	3.32e-08	1.62e-06
Si10H16	3.16e-06	1.76e-07	3.16e-06	3.16e-06	2.41e-09	3.16e-06
Si5H12	1.20e-07	4.28e-08	1.20e-07	5.50e-07	5.36e-10	5.50e-07
SiO	1.29e-06	2.02e-08	1.17e-06	1.29e-06	2.14e-10	1.29e-06
wathen100	3.05e-07	1.09e-08	7.67e-08	3.05e-07	1.46e-10	3.05e-07

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

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

in Table 6. We observe that EigPen ususally returns a slightly smaller residual error than LOBPCG 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, since heuristics are occasionally invoked by PRIMME to declare near-convergence of some Ritz pairs, the final accuracy can sometimes be larger than the given tolerance, for example, on the matrices cfd1 and shallow_water1s for $tol = 10^{-3}$.

Table 6: A comparison of err_{res} among different solvers

	$tol = 10^{-3}$			$tol = 10^{-4}$		
Matrix	LOBPCG	PRIMME	EigPen	LOBPCG	PRIMME	EigPen
Andrews	9.36e-04	7.18e-04	6.34e-04	9.20e-05	6.31e-05	4.37e-05
C60	8.82e-04	7.99e-04	6.98e-04	9.31e-05	8.29e-05	7.22e-05
c_65	9.78e-04	1.12e-04	1.57e-04	– –	1.10e-05	4.62e-05
cfd1	9.84e-04	1.09e-03	5.57e-04	9.80e-05	9.92e-05	8.70e-05
finance	9.83e-04	7.89e-04	7.34e-04	9.96e-05	7.50e-05	8.34e-05
Ga10As10H30	9.95e-04	9.53e-04	6.47e-04	– –	8.89e-05	4.80e-05
Ga3As3H12	9.08e-04	9.61e-04	9.91e-04	– –	7.96e-05	5.85e-05
OPF3754	3.03e-04	2.45e-08	2.61e-04	9.09e-05	2.55e-09	9.14e-08
shallow_water1s	8.70e-04	1.28e-03	9.21e-04	9.61e-05	9.59e-05	5.88e-05
Si10H16	9.72e-04	8.69e-04	8.31e-04	9.01e-05	7.81e-05	9.27e-06
Si5H12	8.88e-04	7.98e-04	7.74e-04	9.78e-05	7.78e-05	8.92e-05
SiO	9.85e-04	7.18e-04	9.89e-04	9.68e-05	8.02e-05	4.59e-05
wathen100	9.71e-04	6.30e-04	6.99e-04	7.72e-05	7.07e-05	6.18e-05

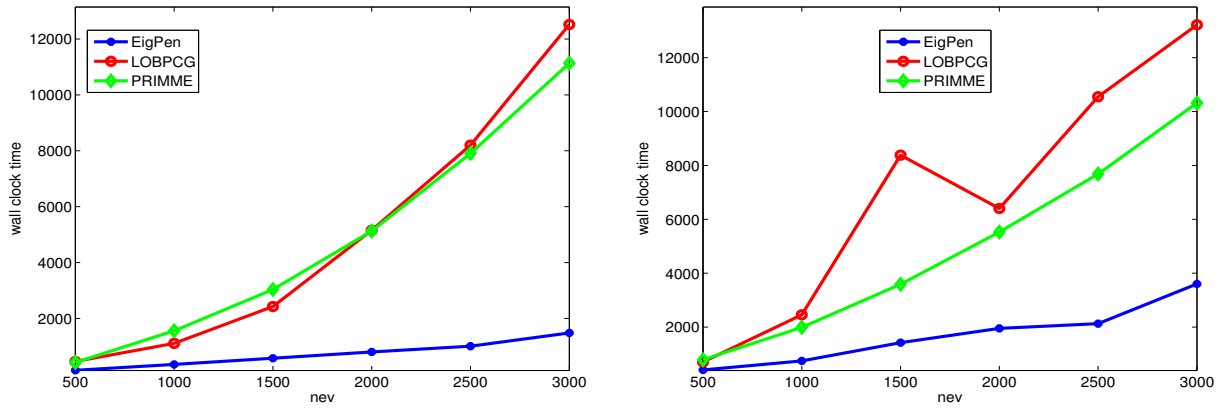
We understand that, under fixed test conditions, the performance of most solvers can in principle be optimized by extensively tuning parameters which we obviously did not attempt to do. The purpose of our numerical experiments in this section is not to pick winners for solving a particular problem set 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.

4.4 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 1. We observe that EigPen always takes less amount of time to run than LOBPCG 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 LOBPCG 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). On the other hand, the current version of PRIMME is not implemented in a way to take full advantage of high performance BLAS3/BLAS2 operations because it operates sequentially on matrices of small block sizes.

The tolerance $tol = 10^{-2}$ used in Figure 1 may be considered too loose for some circumstances. We



(a) Andrews (b) Ga3As3H12

Figure 1: A comparison of wall-clock times by LOBPCG, PRIMME and EigPen to compute nev eigenpairs of the matrices Andrews and Ga3As3H12 as nev increases.

did try the same experiment with $tol = 10^{-3}$ and obtained similar results. However, since LOBPCG terminated abnormally on both $nev = 1000$ and $nev = 1500$ with $tol = 10^{-3}$, we decide not to present the graphs for $tol = 10^{-3}$ with incomplete data. 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 2, 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 (BLAS2, including subroutin `DGEMV` in BLAS), 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 LOBPCG, 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 (35) for the BB step size computation, the cost associated with such data movement is notable. In LOBPCG, the relative cost of SpMV also decreases as nev increase. However, the run time of RR increases more rapidly as nev increases. When $nev \geq 1500$, the run time of RR is higher than that of BLAS3. Note

that the RR time for LOBPCG seems out of proportion when nev is equal to 1000 and 1500 for the matrix Ga3As3H12. The reason is that we have to perform a few singular value decompositions (SVD) to repair a rank deficient basis in LOBPCG and the cost of SVD is counted in RR. The rank deficiency is caused by eigenvalues clusters near the 1000th and the 1500th eigenvalues. 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 LOBPCG or EigPen.

Figures 1 and 2 clearly demonstrate that the advantages of the EigPen 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 calcuations small in EigPen and making use of more BLAS3 operations, we can make it more efficient than LOBPCG and PRIMME for large nev values.

4.5 Parallel scalability

In this subsection, we examine parallel scalability of LOBPCG, PRIMME and EigPen. For brevity, we again only show results for the Andrews and Ga3As3H12 matrices 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 LOBPCG and EigPen and set `primme.maxMatvecs` $= 5000$ for PRIMME since the speedup factor should remain essentially unchanged as more iterations are performed.

Figure 3 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 LOBPCG and 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 LOBPCG. 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 `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 `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.

4.6 Preconditioning for EigPen

One can also introduce a preconditioner in EigPen similar to LOBPCG. 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 (2) yields a problem with a better conditioned Hessian. It is straightforward to show that a preconditioned gradient method can be described by

$$X^{j+1} = X^j - \alpha^j M^{-1} \nabla f_\mu(X^j), \tag{53}$$

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 [20] — a MATLAB toolbox for solving the Kohn-Sham equations in electronic structure calculation. The Kohn-Sham problem is discretized by a planewave expansion of the eigenfunctions. We choose to use KSSOLV because it is well known that good preconditioners are available for planewave discretized Kohn-Sham Hamiltonian [19], and these preconditioners are diagonal in Fourier space, making their applications extremely cheap. Our experiments in this subsection are performed in MATLAB 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 2011b. For this experiment, we choose to use our MATLAB code for the convenience of interfacing with the MATLAB toolbox KSSOLV and utilizing MATLAB's built-in complex arithmetics.

In order to clearly see the effect of preconditioning, we compare Algorithm 1 using the unpreconditioned scheme (31) and the preconditioned scheme (53), respectively, with the tolerance $\epsilon = 10^{-10}$. The dimension n of the matrices are reported in Table 7. The number of eigenvalues to be computed is set to the number of occupied states of the corresponding molecular system. The number of iterations and the CPU time in seconds required by the two schemes, without and with preconditioning, are presented in Table 7. The corresponding iteration history of the gradient norm $\{\|\nabla f_\mu(X^j)\|_F\}$ are depicted in Figure 4. It is clear that preconditioning can improve the performance of EigPen significantly.

We should point out that it is generally not easy to identify an effective and efficient preconditioner for an

Table 7: A comparison: iteration number and CPU time without and with precondition

problem			without precondition		with precondition	
matrix	n	nev	iter	time	iter	time
SiH4	2103	4	126	2.1	62	1.1
Benzene	8407	15	220	53.5	105	26.1

eigenvalue problem. An ideal preconditioner should be close to A^{-1} so that the first term of the Hessian (6) associated with the transformed problem becomes well conditioned. Yet, the pre-conditioner itself should not be too ill-conditioned; otherwise, the subsequent terms in (6) may become ill-conditioned, possibly making the entire Hessian ill-conditioned. 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.

5 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. 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, cut down the use of the Rayleigh-Ritz (RR) procedure.

In most existing eigensolvers, basis orthogonalization and dense eigenvalue decomposition together in the RR procedure constitute the dominant computational tasks and become a formidable bottleneck in reaching high parallel scalability. Our approach greatly reduces the number of RR calls in exchange for BLAS3-rich matrix operations, mainly dense matrix-matrix multiplications, that are highly scalable. 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 by the RR procedure (including orthogonalization) as in other solvers.

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

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 Vecharynski 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.

References

[1] H. M. AKTULGA, L. LIN, C. HAINE, E. G. NG, AND C. YANG, *Parallel Eigenvalue Calculation based on Multiple Shift-invert Lanczos and Contour Integral based Spectral Projection Method*, LBNL Tech Report, Nov 2012.

[2] 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.

[3] J. BARZILAI AND J. M. BORWEIN, *Two-point step size gradient methods*, IMA J. Numer. Anal., 8 (1988), pp. 141–148.

[4] 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.

[5] R. CHELIKOWSKY AND Y. SAAD, *A spectrum slicing method for the Kohn-Sham problem*, Comp. Phys. Comm., 183 (2012), p. 497.

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

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

[8] H. R. FANG AND Y. SAAD, *A filtered Lanczos procedure for extreme and interior eigenvalue problems*, SIAM J. Sci. Comp., 34 (2013), pp. A2220–A2246.

[9] 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.

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

[11] 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.

[12] 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.

[13] J. NOCEDAL AND S. J. WRIGHT, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering, Springer, New York, second ed., 2006.

[14] B. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, 1980.

[15] Y. SAAD, J. R. CHELIKOWSKY, AND S. M. SHONTZ, *Numerical methods for electronic structure calculations of materials*, SIAM Rev., 52 (2010), pp. 3–54.

[16] A. STATHOPOULOS AND J. 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.

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

[18] W. SUN AND Y. YUAN, *Optimization Theory and Methods: Nonlinear Programming*, Springer, New York, 2006.

[19] 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.

[20] C. YANG, J. C. MEZA, B. LEE, AND L.-W. WANG, *KSSOLV—a MATLAB toolbox for solving the Kohn-Sham equations*, ACM Trans. Math. Softw., 36 (2009), pp. 1–35.

[21] Y.-X. YUAN, *A new stepsize for the steepest descent method*, J. Comput. Math., 24 (2006), pp. 149–156.

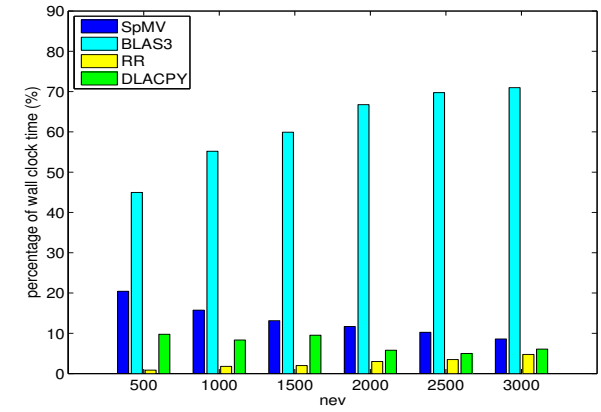
[22] ——, *Step-sizes for the gradient method*, in Third International Congress of Chinese Mathematicians. Part 1, 2, vol. 2 of AMS/IP Stud. Adv. Math., 42, pt. 1, Amer. Math. Soc., 2008, pp. 785–796.

[23] H. ZHANG AND W. W. HAGER, *A nonmonotone line search technique and its application to unconstrained optimization*, SIAM J. Optim., 14 (2004), pp. 1043–1056.

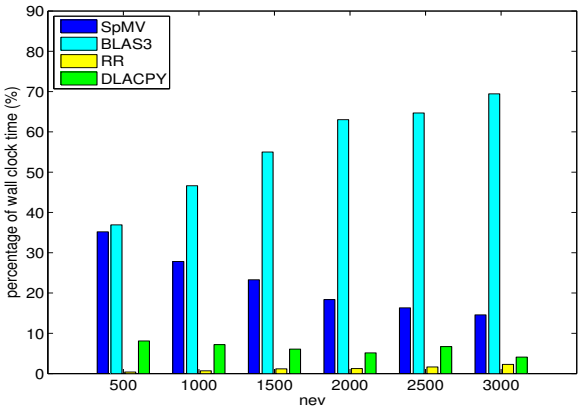
[24] Y. ZHOU, *A block Chebyshev-Davidson method with inner-outer restart for large eigenvalue problems*, J. Comput. Phys., 229 (2010), pp. 9188–9200.

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

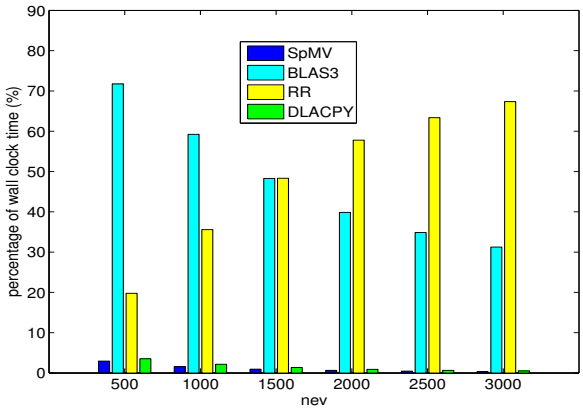
[26] ——, *Block krylovschur method for large symmetric eigenvalue problems*, Numerical Algorithms, 47 (2008), pp. 341–359.



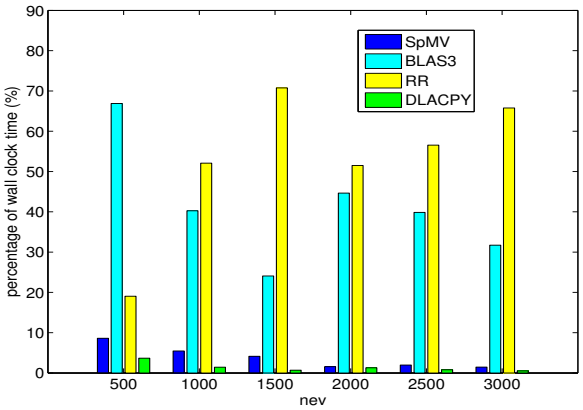
(a) Andrews: EigPen



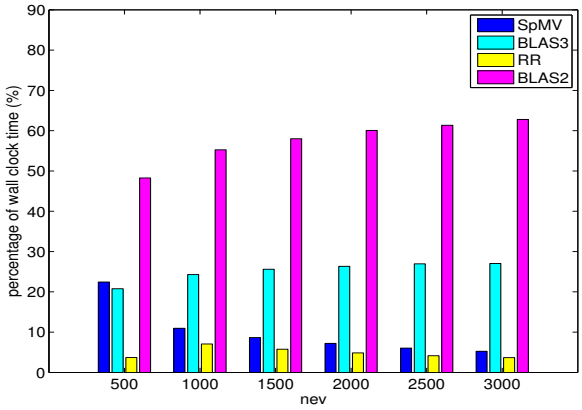
(b) Ga3As3H12: EigPen



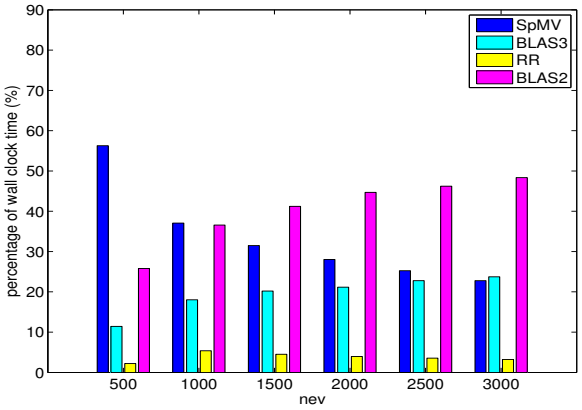
(c) Andrews: LOBPCG



(d) Ga3As3H12: LOBPCG

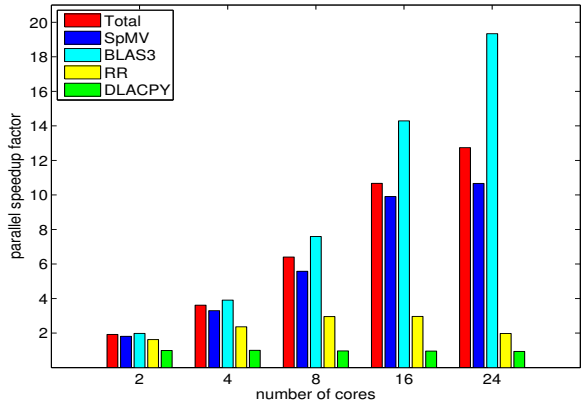


(e) Andrews: PRIMME

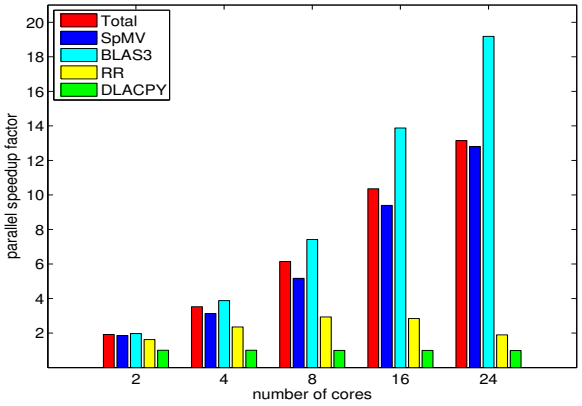


(f) Ga3As3H12: PRIMME

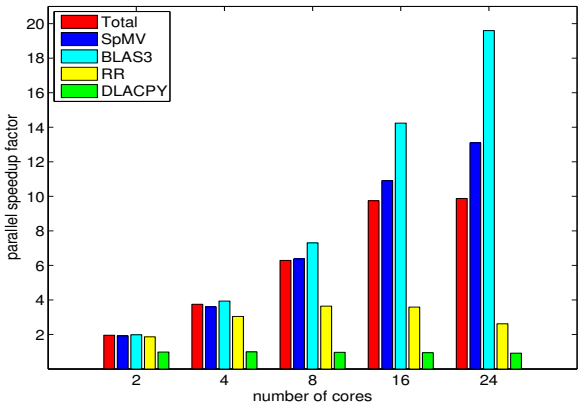
Figure 2: A comparison of timing profile among EigPen, LOBPCG and PRIMME.



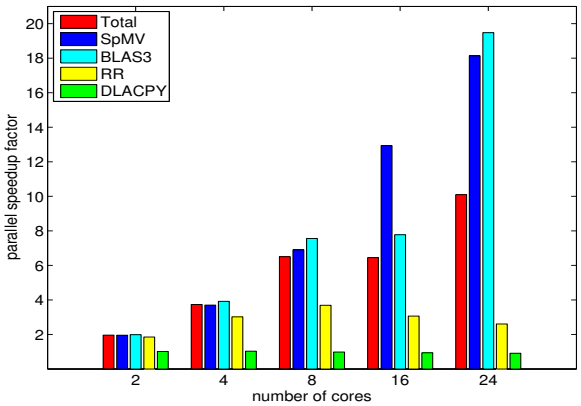
(a) Andrews: EigPen



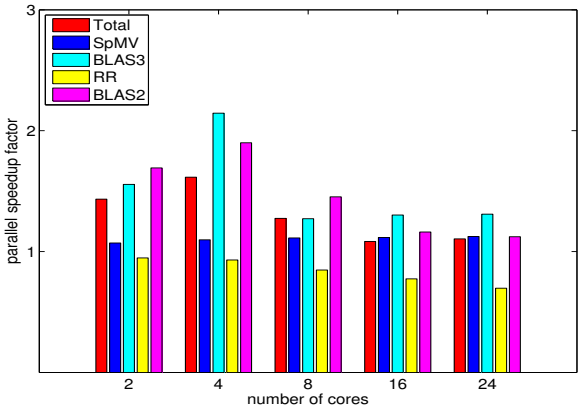
(b) Ga3As3H12: EigPen



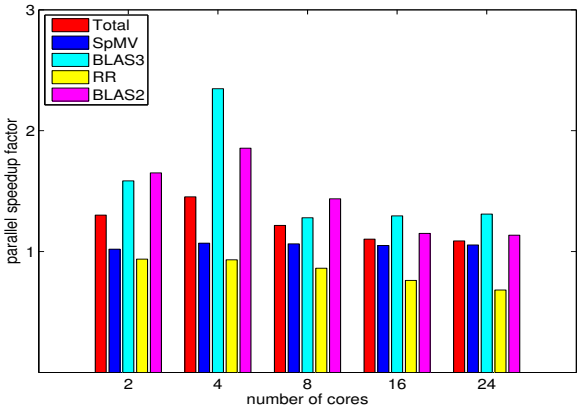
(c) Andrews: LOBPCG



(d) Ga3As3H12: LOBPCG

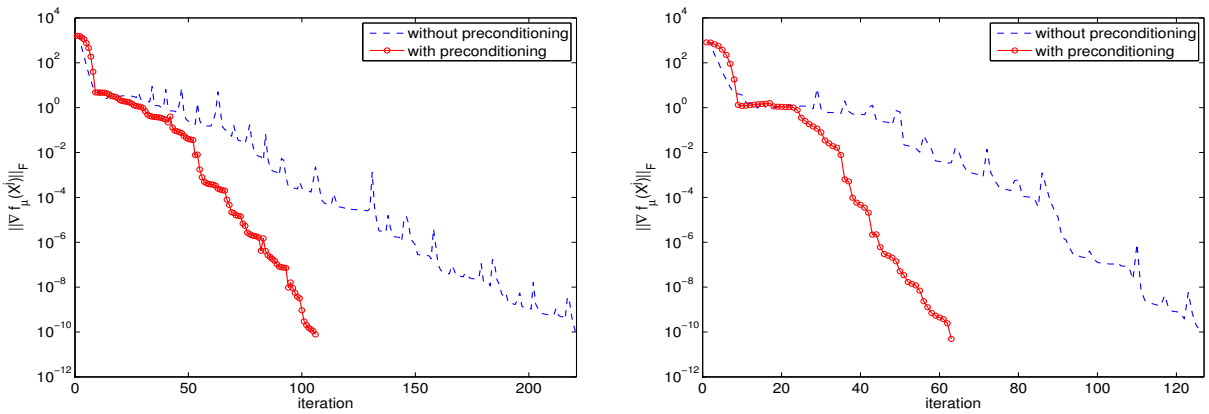


(e) Andrews: PRIMME



(f) Ga3As3H12: PRIMME

Figure 3: A comparison of speedup factor among EigPen, LOBPCG and PRIMME.



(a) Benzene (b) SiH4

Figure 4: The iteration history of gradient norm on benzene and SiH4 without/with preconditioning.