BLOCK ALGORITHMS WITH AUGMENTED RAYLEIGH-RITZ PROJECTIONS FOR LARGE-SCALE EIGENPAIR COMPUTATION

ZAIWEN WEN[‡] AND YIN ZHANG[§]

Abstract.
Most iterative algorithms for eigenpair computation consist of two main steps: a subspace update (SU) step that generates bases for approximate eigenspaces, followed by a Rayleigh-Ritz (RR) projection step that extracts approximate eigenpairs. So far the predominant methodology for the SU step is based on Krylov subspaces that builds orthonormal bases piece by piece in a sequential manner. In this work, we investigate block methods in the SU step that allow a higher level of concurrency than what is reachable by Krylov subspace methods. To achieve a competitive speed, we propose an augmented Rayleigh-Ritz (ARR) procedure and analyze its rate of convergence under realistic conditions. Combining this ARR procedure with a set of polynomial accelerators, as well as utilizing a few other techniques such as continuation and deflation, we construct a block algorithm designed to reduce the number of RR steps and elevate concurrency in the SU steps. Extensive computational experiments are conducted in Matlab on a representative set of test problems to evaluate the performance of two variants of our algorithm in comparison to two well-established, high-quality eigensolvers ARPACK and FEAST. Numerical results, obtained on a many-core computer without explicit code parallelization, show that when computing a relatively large number of eigenpairs, the performance of our algorithms is competitive with, and frequently superior to, that of the two state-of-the-art eigensolvers.

1. Introduction. For a given real symmetric matrix $A \in \mathbb{R}^{n\times n}$, let $\lambda_1, \lambda_2, \cdots, \lambda_n$ be the eigenvalues of A sorted in an descending order: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, and $q_1, \ldots, q_n \in \mathbb{R}^n$ be the corresponding eigenvectors such that $Aq_i = \lambda_i q_i$, $\|q_i\|_2 = 1$, $i = 1, \ldots, n$ and $q_i^T q_j = 0$ for $i \neq j$. The eigenvalue decomposition of A is defined as $A = Q_n \Lambda_n Q_n^T$, where, for any integer $i \in [1, n]$,

$$Q_i = [q_1,\ q_2,\ \ldots, q_i] \in \mathbb{R}^{n\times i}, \quad \Lambda_i = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_i) \in \mathbb{R}^{i\times i}, \tag{1.1}$$

where $\mathrm{diag}(\cdot)$ denotes a diagonal matrix with its arguments on the diagonal. For simplicity, we also write $A = Q\Lambda Q^T$ where $Q = Q_n$ and $\Lambda = \Lambda_n$. In this paper, we consider A to be large-scale, which usually implies that A is sparse. Since eigenvectors are generally dense, in practical applications, instead of computing all n eigenpairs of A, it is only realistic to compute $k \ll n$ eigenpairs corresponding to k largest or smallest eigenvalues of A. Fortunately, these so-called exterior (or extreme) eigenpairs of A often contain the most relevant or valuable information about the underlying system or dataset represented by the matrix A. As the problem size n becomes ever larger, the scalability of algorithms with respect to k has become a critical issue even though k remains a small portion of n.

Most algorithms for computing a subset of eigenpairs of large matrices are iterative in which each iteration consists of two main steps: a subspace update step and a projection step. The subspace update step varies from method to method but with a common goal in finding a matrix $X \in \mathbb{R}^{n\times k}$ so that its column space is a good approximation to the k-dimensional eigenspace spanned by k desired eigenvectors. Once X is obtained and orthonormalized, the projection step, often referred to as the Rayleigh-Ritz (RR) procedure, aims to extract from X a set of approximate eigenpairs (see more details in Section 2) that are optimal in a sense. More complete treatments of iterative algorithms for computing subsets of eigenpairs can be found, for example, in the books [1, 16, 21, 3, 26].

At present, the predominant methodology for subspace updating is still Krylov subspace methods, as represented by Lanczos type methods [9, 12] for real symmetric matrices. These methods generate an orthonormal matrix X one (or a few) column at a time in a sequential mode. Along the way, each column is multiplies by the matrix A and made orthogonal to all the previous columns. In contrast to Krylov subspace methods, block methods, as represented by the classic simultaneous subspace iteration method [18], carry out the multiplications of A to all columns of X at the same time in a batch mode. As such, block methods generally demand a lower level of communication intensity.

The operation of the sparse matrix A multiplying a vector, or SpMV, used to be the most relevant complexity measure for algorithm efficiency. As Krylov subspace methods generally tend to require considerably fewer SpMVs than block methods do, they had naturally become the methodology of choice for the past a few decades even up to date. However, the evolution of modern computer architectures, particularly the emergence of multi/many-core architectures, has seriously eroded the relevance of SpMV (and arithmetic operations in general) as a leading complexity measure, as communication costs have, gradually but surely, become more and more predominant.

The purpose of this work is to construct, analyze and test a framework for block algorithms that can efficiently, reliably and accurately compute a relatively large number of exterior eigenpairs of large-scale matrices. The algorithm

[‡] Beijing International Center for Mathematical Research, Peking University, Beijing, CHINA (wenzw@pku.edu.cn). Research supported in part by NSFC grants 11322109 and 11421101, and by the National Basic Research Project under the grant 2015CB856000.

[§] Department of Computational and Applied Mathematics, Rice University, Houston, UNITED STATES (yzhang@rice.edu). Research supported in part by NSF DMS-1115950 and NSF DMS-1418724.

framework is constructed to take advantages of multi/many-core or parallel computers, although a study of parallel scalability itself will be left as a future topic. It appears widely accepted that a key property hindering the competitiveness of block methods is that their convergence can become intolerably slow when decay rates in relevant eigenvalues are excessively flat. A central task of our algorithm construction is to rectify this issue of slow convergence.

Our framework starts with an outer iteration loop that features an enhanced RR step called the augmented Rayleigh-Ritz (ARR) projection which can provably accelerate convergence under mild conditions. For the SU step, we consider two block iteration schemes whose computational cost is dominated by block SpMVs: (i) the classic power method applied to multiple vectors without periodic orthogonalization, and (ii) a recently proposed Gauss-Newton method. For further acceleration, we apply our block SU schemes to a set of polynomial accelerators, say $\rho(A)$, aiming to suppress the magnitudes of $\rho(\lambda_j)$ where λ_j's are the unwanted eigenvalue of A for $j > k$. In addition, a deflation scheme is utilized to enhance the algorithm's efficiency. Some of these techniques have been studied in the literature over the years (e.g. [20, 29] on polynomial filters), and are relatively well understood. In practice, however, it is still a nontrivial task to integrate all the aforementioned components into an efficient and robust eigensolver. For example, an effective use of a set of polynomial filters involves the choice of polynomial types and degrees, and the estimations of intervals in which eigenvalues are to be promoted or suppressed. There are quite a number of choices to be made and parameters to be chosen that can significantly impact algorithm performance.

Specifically, our main contributions are summarized as follows.

1. An augmented Rayleigh-Ritz (ARR) procedure is proposed and analyzed that provably speeds up convergence without increasing the block size of the iterate matrix X in the SU step (thus without increasing the cost of SU steps). This ARR procedure can significantly reduce the number of RR projections needed, at the cost of increasing the size of a few RR calls.
2. A versatile and efficient algorithmic framework is constructed that can accommodate different block methods for subspace updating. In particular, we revitalize the power method as an exceptionally competitive choice for a high level of concurrency. Besides ARR, our framework features several important components, including
    - a set of low-degree, non-Chebyshev polynomial accelerators that seem less sensitive to erroneous intervals than the classic Chebyshev polynomials;
    - a bold stoping rule for SU steps that demands no periodic orthogonalizations and welcomes a (near) loss of numerical rank.

With regard to the issue of basis orthogonalization, we recall that in traditional block methods such as the classic subspace iteration, orthogonalization is performed either at every iteration or frequently enough to prevent the iterate matrix X from losing rank. On the contrary, our algorithms aim to make X numerically rank-deficient right before performing an RR projection.

The rest of this paper is organized as follows. An overview of relevant iterative algorithms for eigenpair computation is presented in Section 2. The ARR procedure and our algorithm framework are proposed in Section 3. We analyze the ARR procedure in Section 4. The polynomial accelerators used by us are given in Section 5. A detailed pseudocode for our algorithm is outlined in Section 6. Numerical results are presented in Section 7. Finally, we conclude the paper in Section 8.

2. Overview of Iterative Algorithms for Eigenpair Computation.

Algorithms for eigenvalue problem have been extensively studied for decades. We will only briefly review a small subset of them that are most closely related to the present work.

Without loss of generality, we assume for convenience that A is positive definite (after a shift if necessary). Our task is to compute k largest eigenpairs (Q_k, Λ_k) for some $k \ll n$ where by definition $AQ_k = Q_k\Lambda_k$ and $Q_k^TQ_k = I \in \mathbb{R}^{k\times k}$. Replacing A by a suitable function of A, say $\lambda_1 I - A$, one can also in principle apply the same algorithms to finding k smallest eigenpairs as well.

An RR step is to extract approximate eigenpairs, called Ritz-pairs, from a given matrix $Z \in \mathbb{R}^{n\times m}$ whose range space, $\mathcal{R}(Z)$, is supposedly an approximation to a desired m-dimensional eigenspace of A. Let $\mathbf{orth}(Z)$ be the set of orthonormal bases for the range space of Z. The RR procedure is described as Algorithm 1 below, which is also denoted by a map $(Y, \Sigma) = \mathrm{RR}(A, Z)$ where the output (Y, Σ) is a Ritz pair block.

It is known (see [16], for example) that Ritz pairs are, in a certain sense, optimal approximations to eigenpairs in $\mathcal{R}(Z)$, the column space of Z.

2.1. Krylov Subspace Methods.

Krylov subspaces are the foundation of several state-of-the-art solvers for large-scale eigenvalue calculations. By definition, for given matrix $A \in \mathbb{R}^{n\times n}$ and vector $v \in \mathbb{R}^n$, the Krylov

Algorithm 1: Rayleigh-Ritz procedure: $(Y, \Sigma) = \mathrm{RR}(A, Z)$

```
Given $Z \in \mathbb{R}^{n \times m}$, orthonormalize $Z$ to obtain $U \in \mathbf{orth}(Z)$.
Compute $H = U^T AU \in \mathbb{R}^{m \times m}$, the projection of $A$ onto $\mathbf{orth}(Z)$.
Compute the eigen-decomposition $H = V^T \Sigma V$, where $V^T V = I$ and $\Sigma$ is diagonal.
Assemble the Ritz pairs $(Y, \Sigma)$ where $Y = UV \in \mathbb{R}^{n \times m}$ satisfies $Y^T Y = I$.
```

subspace of order k is $\mathrm{span}\{v, Av, A^2 v, \ldots, A^{k-1} v\}$. Typical Krylov subspace methods include Arnoldi algorithm for general matrices (e.g., [12, 11]) and Lanczos algorithm for symmetric (or Hermitian) matrices (e.g., [23, 10]). In either algorithm, orthonormal bases for Krylov subspaces are generated through a Gram-Schmidt type process. Jacobi-Davidson methods (e.g., [2, 24]) are based on a different framework, but they too rely on Krylov subspace methodologies to solve linear systems at every iteration.

As is mentioned in the introduction, Krylov-subspace type methods are generally most efficient in terms of the number of SpMVs (sparse matrix-dense vector multiplications). Indeed, they remain the method of choice for computing a small number eigenpairs. However, due to the sequential process of generating orthonormal bases, Krylov-subspace type methods incur a low degree of concurrency, especially as the dimension k becomes relatively large. To improve concurrency, multiple-vector versions of these algorithms have been developed where each single vector in matrix-vector multiplication is replaced by a small number of multiple vectors. Nevertheless, such a remedy can only provide a limited relief in the face of the inherent scalability barrier as k grows. Another well-known limitation of Krylov subspace methods is the difficulty to warm-start them from a given subspace. Warm-starting is important in an iterative setting in order to take advantages of available information computed at previous iterations.

2.2. Classic Subspace Iteration.

The simple (or simultaneous) subspace iteration (SSI) method (see [18, 19, 25, 27], for example) extends the idea of the power method which computes a single eigenpair corresponding to the largest eigenvalue (in magnitude). Starting from an initial (random) matrix U, SSI performs repeated matrix multiplications AU, followed by periodic orthogonalizations and RR projections. The main purpose of orthogonalization is to prevent the iterate matrix U from losing rank numerically. In addition, since the rates of convergence for different eigenpairs are uneven, numerically converged eigenvectors can be deflated after each RR projection. A version of SSI algorithm is presented as Algorithm 2 below, following the description in [26].

Algorithm 2: Subspace Iteration

```
Initialize orthonormal matrix $U \in \mathbb{R}^{n \times m}$ with $m = k + q \geq k$.
while the number of converged eigenpairs is less than k, do
    while convergence is not expected, do
        while the columns of U are sufficiently independent, do
            Compute $U = AU$
        Orthogonalize the columns of $U$.
    Perform an RR step using $U$.
    Check convergence and deflate.
```

In the above SSI framework, q extra vectors, often called guard vectors, are added into iterations to help improve convergence at the price of increasing the iteration cost.

A main advantage of SSI is the use of simultaneous matrix-block multiplications instead of individual matrix-vector multiplications. It enables fast memory access and highly parallelizable computation on modern computer architectures. Furthermore, SSI method has a guaranteed convergence to the largest k eigenpairs from any generic starting point as long as there is a gap between the k-th and the $(k+1)$-th eigenvalues of A. As is points out in [26], “combined with shift-and-invert enhancement or Chebyshev acceleration, it sometimes wins the race”. However, a severe shortcoming of the SSI method is that its convergence speed depends critically on eigenvalue distributions that can, and often does, become intolerably slow in the face of unfavorable eigenvalue distributions. Thus far, this drawback has essentially prevented the SSI method from being used as a computational engine to build robust, reliable and efficient general-purpose eigensolvers.

2.3. Trace Maximization Methods. Computing a k-dimensional eigenspace associated with k largest eigenvalues of A is equivalent to solving an orthogonality constrained trace maximization problem:

$$\max_{X \in \mathbb{R}^{n \times k}} \operatorname{tr}(X^T AX), \ \ \mathbf{s.t.}\ X^T X = I. \tag{2.1}$$

This formulation can be easily extended to solving the *generalized eigenvalue problem* where $X^T X = I$ is replace by $X^T BX = I$ for a symmetric positive definite matrix $B \in \mathbb{R}^{n \times n}$. When maximization is changed to minimization, one computes an eigenspace associated with k smallest eigenvalues. The algorithm TraceMin [22] solves the trace minimization problem using a Newton type method.

Some block algorithms have been developed based on solving (2.1), include the locally optimal block preconditioned conjugate gradient method (LOBPCG) [7] and more recently the limited memory block Krylov subspace optimization method (LMSVD) [13]. At each iteration, these methods solve a subspace trace maximization problem of the form

$$Y = \operatorname*{arg\,max}_{X \in \mathbb{R}^{n \times k}} \left\{ \operatorname{tr}(X^T AX) : X^{\mathrm{T}} X = I, \ X \in \mathcal{S} \right\}, \tag{2.2}$$

where $X \in \mathcal{S}$ means that each column of X is in the given subspace $\mathcal{S}$ which varies from method to method. LOBPCG constructs $\mathcal{S}$ as the span of the two most recent iterates $X^{(i-1)}$ and $X^{(i)}$, and the residual at $X^{(i)}$, which is essentially equivalent to

$$\mathcal{S} = \operatorname{span}\left\{ X^{(i-1)}, X^{(i)}, AX^{(i)} \right\}, \tag{2.3}$$

where the term $AX^{(i)}$ may be pre-multiplied by a pre-conditioning matrix. In the LMSVD method, on the other hand, the subspace $\mathcal{S}$ is spanned by the current i-th iterate and the previous p iterates; i.e.,

$$\mathcal{S} = \operatorname{span}\left\{ X^{(i)}, X^{(i-1)}, ..., X^{(i-p)} \right\}, \tag{2.4}$$

In general, the subspace $\mathcal{S}$ should be constructed such that the cost of solving (2.2) can be kept relatively low. The parallel scalability of these algorithms, although improved from that of Krylov subspace methods, is now limited by the frequent use of basis orthogonalizations and RR projections involving $m \times m$ matrices where m is the dimension of the subspace $\mathcal{S}$ (for example, $m = 3k$ in LOBPCG).

2.4. Polynomial Acceleration. Polynomial filtering has been used in eigenvalue computation in various ways (see, for example, [20, 26, 29, 6]). For a polynomial function $\rho(t) : \mathbb{R} \to \mathbb{R}$ and a symmetric matrix with eigenvalue decomposition $A = Q\Lambda Q^T$, it holds that

$$\rho(A) = Q\rho(\Lambda)Q^T = \sum_{i=1}^{n} \rho(\lambda_i) q_i q_i^T, \tag{2.5}$$

where $\rho(\Lambda) = \operatorname{diag}(\rho(\lambda_1), \rho(\lambda_2), \dots, \rho(\lambda_n))$. By choosing a suitable polynomial function $\rho(t)$ and replacing A by $\rho(A)$, we can change the original eigenvalue distribution into a more favorable one at a cost. To illustrate the idea of polynomial filtering, suppose that $\rho(t)$ is a good approximation to the step function that is one on the interval $[\lambda_k, \lambda_1]$ and zero otherwise. For a generic initial matrix $X \in \mathbb{R}^{n \times k}$, it follows from (2.5) that $\rho(A)X \approx Q_k Q_k^T X$, which would be an approximate basis for the desired eigenspace. In practice, however, approximating a non-smooth step function by polynomials is an intricate and demanding task which does not always lead to efficient algorithms.

For the purpose of convergence acceleration, the most often used polynomials are the Chebyshev polynomials (of the first kind), defined by the three-term recursion:

$$\rho_{d+1}(t) = 2t\rho_d(t) - \rho_{d-1}(t), \ \ d \geq 1, \tag{2.6}$$

where $\rho_0(t) = 1$ and $\rho_1(t) = t$. Some recent works that use Chebyshev polynomials include [29, 6], for example.

2.5. FEAST. The FEAST algorithm [17, 28] is based on complex contour integrals for computing all eigenvalues in a given interval $[a, b] \subset \mathbb{R}$ and their corresponding eigenvectors. It is equivalent to using a rational function filter in subspace iteration.

Let $\mathcal{C}$ be the circle on the complex plane centered at $c = \frac{a+b}{2}$ with radius $r = \frac{b-a}{2}$, which can be parameterized by the function $\phi(t) = c + re^{\iota\frac{\pi}{2}(1+t)}$ for $t \in [-1, 3]$ where $\iota^2 = -1$ is the imaginary unit. By the Cauchy integral theorem, for any $\mu \notin \mathcal{C}$

$$\frac{1}{2\pi\iota}\oint_{\mathcal{C}} \frac{1}{z-\mu}dz = \frac{1}{2\pi\iota}\int_{-1}^{1}\left[\frac{\phi'(t)}{\phi(t)-\mu} - \frac{\overline{\phi'(t)}}{\overline{\phi(t)}-\mu}\right]dt = \begin{cases}1, & \text{if } |\mu - c| < r\\ 0, & \text{if } |\mu - c| > r\end{cases},$$

where the integral on $[1, 3]$ has been equivalently transformed into $[-1, 1]$. Applying a q-point Gauss-Legendre quadrature formula with weight-node pairs (w_l, t_l), $l = 1, 2, \ldots, q$, such that $w_l > 0$ and $t_l \in (-1, 1)$, the above integral can be approximated by the rational function

$$\rho(\mu) = \sum_{l=1}^{q}\left(\frac{\sigma_l}{\phi_l - \mu} - \frac{\overline{\sigma_l}}{\overline{\phi_l} - \mu}\right),$$

where $\phi_l = \phi(t_l)$ and $\sigma_l = w_l\phi'(t_l)/(2\pi\iota)$. Since none of ϕ_l's is real and A is symmetric, the matrices $\phi_l I - A$ and $\overline{\phi_l}I - A$ are all invertible for $l = 1, 2, \ldots, q$. Therefore,

$$\rho(A) = \sum_{l=1}^{q}\sigma_l(\phi_l I - A)^{-1} - \sum_{l=1}^{q}\overline{\sigma_l}(\overline{\phi_l}I - A)^{-1} \tag{2.7}$$

is a rational function filter approximating a desired step function on the real line. The application of this filter to $X \in \mathbb{R}^{n\times m}$, i.e., computing $\rho(A)X$, will require solving q (since all quantities involved are real) linear systems of equations with m right-hand sides each. It is notable that these linear systems could be solved independently in parallel.

In order to compute all eigenpairs in an interval $[a, b]$, FEAST need to estimate the number of eigenvalues in the interval $[a, b]$. It repeatedly applies the rational filter $X = \rho(A)X$, followed by an RR projection. A high-level summary of the FEAST algorithm is presented as Algorithm 3.

Algorithm 3: A abstract version of FEAST

```
Input $[a, b]$ and $m$ – estimated number of eigenvalues in $[a, b]$.
Choose a Gauss-Legendre quadrature formula with $q$ nodes.
Initialize a matrix $X \in \mathbb{R}^{n\times m}$.
while not "converged", do
    Compute $X = \rho(A)X$ with $\rho(\cdot)$ given in (2.7).
    Do RR projection using $X$ to extract Ritz pairs.
```

It should be clear that the performance of FEAST depends strongly on the efficiency of solving the linear systems of equations involved in applying the rational filter $\rho(A)$ to X. In addition, in order to compute the k largest eigenpairs, for example, one need to supply FEAST with an interval $[a, b] \supseteq [\lambda_k, \lambda_1]$. The quality of this interval $[a, b]$ could have a significant effect on the performance of FEAST.

2.6. A Gauss-Newton Algorithm. A Gauss-Newton (GN) algorithm is recently proposed in [14] to compute the eigenspace associated with k largest eigenvalues of A based on solving the nonlinear least squares problem: $\min \|XX^{\mathrm{T}} - A\|_{\mathrm{F}}^2$,where $X \in \mathbb{R}^{n\times k}$, $\|\cdot\|_{\mathrm{F}}^2$ is the Frobenius norm squared and A is assumed to have at least k positive eigenvalues. If the eigenpairs of A are required, then an RR projection must be performed afterwards.

It is shown in [14] that at any full-rank iterate $X \in \mathbb{R}^{n\times k}$, the GN method takes the simple closed form

$$X^{+} = X + \alpha\left(I - \frac{1}{2}X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}\right)\left(AX(X^{\mathrm{T}}X)^{-1} - X\right),$$

where the parameter $\alpha > 0$ is a step size. Notably, this method requires to solve a small $k \times k$ linear system at each iteration. It is also shown in [14] that the fixed step $\alpha \equiv 1$ is justifiable from either a theoretical or an empirical viewpoint, which leads to a parameter-free algorithm given as Algorithm 4, named simply as GN. For more theoretical and numerical results on this GN algorithm, we refer readers to [14].

Algorithm 4: A GN Algorithm: $X = \text{GN}(A, X)$

```
Initialize $X \in \mathbb{R}^{n\times k}$ to a rank-$k$ matrix.
while "the termination criterion" is not met, do
    Compute $Y = X\left(X^T X\right)^{-1}$ and $Z = AY$.
    Compute $X = Z - X(Y^{\mathrm{T}} Z - I)/2$.
Perform an RR step using $X$ if Ritz-pairs are needed.
```

3. Augmented Rayleigh-Ritz Projection and Our Algorithm Framework.

We first introduce the augmented Rayleigh-Ritz or ARR procedure. It is easy to see that the RR map $(Y, \Sigma) = \mathrm{RR}(A, Z)$ is equivalent to solving the trace-maximization subproblem (2.2) with the subspace $\mathcal{S} = \mathcal{R}(Z)$, while requiring $Y^{\mathrm{T}} AY$ to be a diagonal matrix Σ. For a fixed number k, the larger the subspace $\mathcal{R}(Z)$ is, the greater chance there is to extract better Ritz pairs. The classic SSI always sets Z to the current iterate $X^{(i)}$, while both LOBPCG [7] and LMSVD [13] augment $X^{(i)}$ by additional blocks (see (2.3) and (2.4), respectively). Not surprisingly, such augmentations are the main reason why algorithms like LOGPCG and LMSVD generally achieve faster convergence than that of the classic SSI.

In this work, we define our augmentation based on a block Krylov subspace structure. That is, for some integer $p \geq 0$ we define

$$\mathcal{S} = \mathrm{span}\{X, AX, A^2X, \ldots, A^pX\}. \tag{3.1}$$

This choice (3.1) of augmentation is made mainly because it enables us to conveniently analyze the acceleration rates induced by such an augmentation (see the next Section). It is more than likely that some other choices of $\mathcal{S}$ may be equally effective as well.

The optimal solution of the trace maximization problem (2.2), restricted in the subspace $\mathcal{S}$ in (3.1), can be computed via the RR procedure, i.e., Algorithm 1. We formalize our augmented RR procedure as Algorithm 5, which will often be referred to simply as ARR.

Algorithm 5: ARR: $(Y, \Sigma) = \text{ARR}(A, X, p)$

```
Input $X \in \mathbb{R}^{n\times k}$ and $p \geq 0$ so that $(p+1)k < n$.
Construct augmentation $\mathbf{X}_p = [X \;\; AX \;\; A^2X \;\; \cdots \;\; A^pX]$.
Perform an RR step using $(\hat{Y}, \hat{\Sigma}) = \mathrm{RR}(A, \mathbf{X}_p)$.
Extract $k$ leading Ritz pairs $(Y, \Sigma)$ from $(\hat{Y}, \hat{\Sigma})$.
```

We next introduce an abstract version of our algorithmic framework with ARR projections. It will be named ARRABIT (standing for ARR and block iteration). A set of polynomial functions $\{\rho_d(t)\}$, where d is the polynomial degree, and an integer $p \geq 0$ are chosen at the beginning of the algorithm. At each outer iteration, we perform the two main steps: subspace update (SU) step and augmented RR (ARR) step. There are two sets of stopping criteria: inner criteria for the SU step, and outer criteria for detecting the convergence of the whole process.

In principle, the SU step can be fulfilled by any reasonable updating scheme and it does not require orthogonalizations. In this paper, we consider the classic power iteration as our main updating scheme, i.e., for $X = [x_1 \; x_2 \; \cdots, x_m] \in \mathbb{R}^{n\times m}$, we do

$$x_i = \rho(A)x_i \quad \text{and} \quad x_i = \frac{x_i}{\|x_i\|_2}, \;\; j = 1, 2, \cdots, m.$$

Since the power iteration is applied individually to all columns of the iterate matrix X, we call this scheme *multi-power method* or MPM. Here we intensionally avoid to use the term *subspace iteration* because, unlike in the classic SSI, we do not perform any orthogonalization during the entire inner iteration process.

To examine the versatility of the ARRABIT framework, we also use the Gauss-Newton (GN) method, presented in Algorithm 4, as a second updating scheme. Since the GN variant requires solving $k \times k$ linear systems, its scalability with respect to k may be somewhat lower than that of the MPM variant. Together, we present our ARRABIT algorithmic framework in Algorithm 6. The two variants, corresponding to "inner solvers" MPM and GN, will be named ARRABIT-MPM and ARRABIT-GN, or simply MPM and GN.

Algorithm 6: Algorithm ARRABIT (abstract version)

```
Input $A \in \mathbb{R}^{n\times n}$, $k$, $p$ and $\rho(t)$. Initialize $X \in \mathbb{R}^{n\times k}$.
while not "converged", do
    while "inner criteria" are not met, do
        if MPM is the inner solver, then
            $X = \rho(A)X$, then normalize columns individually.
        if GN is the inner solver, then
            $X = \text{GN}(\rho(A), X)$, as is given by Algorithm 4.
    ARR projection: $(X, \Sigma) = \text{ARR}(A, X, p)$, as in Algorithm 5.
    Possibly adjust $p$, the degree of $\rho(t)$, and perform deflation.
```

It is worth mentioning that the "inner criteria" in the ARRABIT framework can have a significant impact on the efficiency of Algorithm 6. Against the conventional wisdom, we do not attempt to keep X numerically full rank by periodic orthogonalizations which can be quite costly. Instead, we keep iterating until we detect that X is about to lose, or has just lost, numerical rank. More details on this issue will be given in Algorithm 8 in Section 6.

4. Analysis of the Augmented Rayleigh-Ritz Procedure.

4.1. Notation. Recall that the eigen-decomposition of $A \in \mathbb{R}^{n\times n}$ is $A = Q\Lambda Q^T$. In anticipation of later usage, for integer $h \in [1, n)$ we introduce the partition $Q = [Q_h \; Q_{h+}]$ where, as previously defined, $Q_h = [q_1 \;\; q_2 \;\; \cdots q_h]$ and

$$Q_{h+} = [q_{h+1} \;\; q_{h+2} \;\; \cdots \;\; q_n]. \tag{4.1}$$

Let $X \in \mathbb{R}^{n\times k}$ be an approximate basis for $\mathcal{R}(Q_k)$, the range space of Q_k or the eigenspace spanned by the first k eigenvectors of A. It is desirable for X to have a large projection $Q_k Q_k^T X = \sum_{i=1}^k q_i q_i^T X$ onto $\mathcal{R}(Q_k)$ relative to that onto $\mathcal{R}(Q_{k+})$. Therefore, a good measure for the relative accuracy of X is the following ratio

$$\delta_k(X) \triangleq \frac{\max_{i>k} \|q_i^T X\|}{\min_{i\le k} \|q_i^T X\|}, \tag{4.2}$$

where $\|q_i^T X\| = \|(q_i q_i^T)X\|$ measures the size of the projection of X onto the span of the i-th eigenvector q_i. Clearly, the smaller $\delta_k(X)$ is, the better is X as an approximate basis for $\mathcal{R}(Q_k)$.

Let $Y \in \mathbb{R}^{n\times k}$ be another approximate basis for the eigenspace $\mathcal{R}(Q_k)$ which is constructed from X. To compare Y with X, we naturally compare $\delta_k(Y)$ with $\delta_k(X)$. More precisely, we will try to estimate the ratio $\delta_k(Y)/\delta_k(X)$ and show that under reasonable conditions, it can be made much less than the unity.

To facilitate presentation, we introduce the following Vandermonte matrix constructed from the spectrum of A:

$$V = \begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^p \\ 1 & \lambda_2 & \lambda_2^2 & \cdots & \lambda_2^p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \lambda_n & \lambda_n^2 & \cdots & \lambda_n^p \end{pmatrix} \in \mathbb{R}^{n\times(p+1)}, \tag{4.3}$$

where $\lambda_1, \cdots, \lambda_n$ are the eigenvalues of A.

4.2. Technical Results. Before calling the ARR procedure, we have an iterate matrix $X \in \mathbb{R}^{n\times k}$. From X, we construct the augmented matrix $[X \;\; AX \;\; \cdots \;\; A^pX] \in \mathbb{R}^{n\times(p+1)k}$ which we call $\mathbf{X}_p$ for a given $p \geq 0$. In view of the eigen-decomposition $A = Q\Lambda Q^T$, we have the expression $\mathbf{X}_p = Q\hat{G}$ where

$$\hat{G} = [Q^TX \;\; \Lambda Q^TX \;\; \cdots \;\; \Lambda^pQ^TX]. \tag{4.4}$$

We next normalize the rows of $\hat{G}$. Let D be the diagonal matrix whose diagonal consists of the row norms of $\hat{G}$. From the structure of $\hat{G}$ in (4.4), it is easy to see that

$$D_{ii} = \|e_i^T\hat{G}\| = \|q_i^TX\|\|e_i^TV\|, \;\; i = 1, 2, \cdots, n, \tag{4.5}$$

where e_i is the i-th column of the $n \times n$ identity matrix and V is defined in (4.3). Let $D^\dagger$ be the pseudo-inverse of D, that is, $D^\dagger$ is a diagonal matrix with

$$(D^\dagger)_{ii} = \begin{cases} 1/D_{ii}, & \text{if } D_{ii} \neq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4.6}$$

The normalization of the rows of $\hat{G}$ in (4.4) defines another matrix

$$G = D^\dagger\hat{G} = [C \;\; \Lambda C \;\; \cdots \;\; \Lambda^pC], \tag{4.7}$$

where $C = D^\dagger Q^TX$ and the nonzero rows of G all have unit norm. Now we rewrite

$$\mathbf{X}_p = QDD^\dagger\hat{G} = QDG. \tag{4.8}$$

Let m be a parameter varying in the following range: for $p \geq 0$ such that $k + pk < n$,

$$m \in [k, k + pk]. \tag{4.9}$$

We perform the partition

$$\mathbf{X}_p = [Q_m \; Q_{m+}] \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \end{bmatrix} = [Q_m \; Q_{m+}] \begin{bmatrix} D_1G_1 \\ D_2G_2 \end{bmatrix}, \tag{4.10}$$

where D and G are partitioned following that of Q. In particular, G_1 consists of the first m rows of G and G_2 the last $n - m$ rows of G.

In the sequel, we will make use of an important assumption on $G_1 \in \mathbb{R}^{m\times(p+1)k}$ which we formally name as the G_1-Assumption:

$$G_1\text{-Assumption: the first } m \text{ rows of } G \text{ (or } \hat{G}) \text{ are linearly independent.} \tag{4.11}$$

The G_1-Assumption implies that (i) $D_1 > 0$, and (ii) the pseudo-inverse $G_1^\dagger$ exists such that $G_1G_1^\dagger = I_{m\times m}$. Let

$$\mathbf{Y}_p = \mathbf{X}_pG_1^\dagger D_1^{-1} = [Q_m \; Q_{m+}] \begin{bmatrix} I \\ D_2G_2G_1^\dagger D^{-1} \end{bmatrix}. \tag{4.12}$$

In particular, we are interested in the first k columns of $\mathbf{Y}_p$, i.e., by Matlab notation,

$$Y = \mathbf{Y}_p(:, 1\!:\!k) \in \mathbb{R}^{n\times k}. \tag{4.13}$$

We summarize what we already have for Y into the following lemma.

LEMMA 4.1. *Let $A = Q\Lambda Q^T$ be the eigen-decomposition of $A = A^T \in \mathbb{R}^{n\times n}$. For integers $k > 0$ and $p \geq 0$ satisfying $(p + 1)k < n$, and $m \in [k, k + pk]$, let G, $\mathbf{X}_p$, $\mathbf{Y}_p$ and Y be defined as in* (4.7)*,* (4.8)*,* (4.12) *and* (4.13)*, respectively. Under the G_1-Assumption,*

$$Y = Q_mE_k + Q_{m+}SE_k, \tag{4.14}$$

where $S = D_2G_2G_1^\dagger D_1^{-1}$ and $E_k \in \mathbb{R}^{m\times k}$ consists of the first k columns of the $m \times m$ identity matrix.

Proof. The equality directly follows from (4.12) and (4.13). □

Since Y is extracted from the subspace $\mathcal{R}(\mathbf{X}_p)$ constructed from X, a central question is how much improvement Y can provide over X as an approximate basis for $\mathcal{R}(Q_k)$. We study this question by comparing the accuracy measure $\delta_k(Y)$ relative to $\delta_k(X)$. First, we estimate $\delta_k(Y)$.

LEMMA 4.2. *Under the conditions of Lemma 4.1,*

$$\delta_k(Y) \le \frac{\max_{i>m} d_i}{\min_{i\le k} d_i} \max_{1\le i\le n-m} \|e_i^T G_2 G_1^\dagger E_k\|. \tag{4.15}$$

where $d = \operatorname{diag}(D)$ *with* D_{ii} *defined in* (4.5).

Proof. It follows from (4.14) that

$$q_i^T Y = \begin{cases} e_i^T, & i \in [1,k] \\ \mathbf{0}^T, & i \in (k,m] \\ e_{i-m}^T S E_k, & i \in (m,n] \end{cases}$$

where $e_i \in \mathbb{R}^k$, $\mathbf{0} \in \mathbb{R}^k$ and $e_{i-m} \in \mathbb{R}^{n-m}$. These formulas imply that in the definition (4.2) the denominator term $\min_{i\le k} \|q_i^T Y\| = 1$; thus

$$\delta_k(Y) = \max_{i>k} \|q_i^T Y\| = \max_{i>m} \|q_i^T Y\|. \tag{4.16}$$

In view of the formula $S = D_2 G_2 G_1^\dagger D_1^{-1}$, and the definition of D in (4.5), we have

$$q_i^T Y = d_i e_{i-m}^T G_2 G_1^\dagger D_1^{-1} E_k, \quad i \in (m,n].$$

Therefore, for $i \in (m,n]$, $\|q_i^T Y\| \le \frac{d_i}{\min_{j\le k} d_j} \|e_{i-m}^T G_2 G_1^\dagger E_k\|$. It follows that

$$\max_{i>m} \|q_i^T Y\| \le \frac{\max_{i>m} d_i}{\min_{i\le k} d_i} \max_{1\le i\le n-m} \|e_i^T G_2 G_1^\dagger E_k\|,$$

which, together with (4.16), establishes (4.15). □

4.3. Main Results. We first extend the definition (4.2) for $\delta_k(X)$ into a more general form. For any matrix M of n rows, we define

$$\Gamma_{k,m}(M) \triangleq \frac{\max_{i>m} \|e_i^T M\|}{\min_{i\le k} \|e_i^T M\|}. \tag{4.17}$$

By this definition, $\delta_k(X) = \Gamma_{k,k}(Q^T X)$.

It is worth observing that (i) $\Gamma_{k,m}(M)$ is monotonically non-increasing with respect to m for fixed k and M; (ii) $\Gamma_{k,m}(M)$ is small if the first k rows of M are much larger in magnitude than the last $n-m$; (iii) if $\{\|e_i^T M\|\}$ is non-increasing, then $\Gamma_{k,m}(M) \le 1$.

Specifically, since the eigenvalues of A are ordered in a descending order, for the matrix V in (4.3) we have

$$\Gamma_{k,m}(V) = \frac{\|e_{m+1}^T V\|}{\|e_k^T V\|} = \left(\frac{1 + \lambda_{m+1}^2 + \cdots + \lambda_{m+1}^{2p}}{1 + \lambda_k^2 + \cdots + \lambda_k^{2p}} \right)^{\frac{1}{2}} \le 1, \tag{4.18}$$

Evidently, the faster the decay is between λ_k and λ_{m+1}, the smaller is $\Gamma_{k,m}(V)$.

Moreover, when $M = z \in \mathbb{R}^n$ is a vector which is in turn the element-wise multiplication of two other vectors, say $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$ so that $z_i = x_i y_i$ for $i = 1, \cdots, n$, then it holds that

$$\Gamma_{k,m}(z) \le \Gamma_{k,m}(x)\,\Gamma_{k,m}(y). \tag{4.19}$$

In our first main result, we refine the estimation of $\delta_k(Y)$ and compare it to $\delta_k(X)$.

THEOREM 4.3. *Under the conditions of Lemma 4.1,*

$$\delta_k(Y) \le \Gamma_{k,m}(Q^T X)\Gamma_{k,m}(V) \left\| G_1^\dagger E_k \right\|_2. \tag{4.20}$$

Furthermore,

$$\frac{\delta_k(Y)}{\delta_k(X)} \leq \frac{\max_{j>m} \|q_j^T X\|}{\max_{j>k} \|q_j^T X\|} \Gamma_{k,m}(V) \left\| G_1^\dagger E_k \right\|_2 . \tag{4.21}$$

Proof. Observe that the ratio in the right-hand side of (4.15) is none other than $\Gamma_{k,m}(d)$. Applying (4.19) to $M = d$ where $d = \operatorname{diag}(D)$ with D_{ii} defined in (4.5), $x_i = \|q_i^T X\|$ and $y_i = \|e_i^T V\|$, we derive $\Gamma_{k,m}(d) \leq \Gamma_{k,m}(Q^T X)\Gamma_{k,m}(V)$.We observe that $\|e_i^T G_2 G_1^\dagger E_k\| \leq \|G_1^\dagger E_k\|_2$ for all $i \in [1, n-m]$, since the row vectors $e_i^T G_2$ are all unit vectors. Substituting the above two inequalities into (4.15), we arrive at (4.20). To derive (4.21), we simply observe that

$$\Gamma_{k,m}(Q^T X) = \frac{\max_{j>m} \|q_j^T X\|}{\min_{j\leq k} \|q_j^T X\|} = \delta_k(X) \frac{\max_{j>m} \|q_j^T X\|}{\max_{j>k} \|q_j^T X\|}.$$

Substituting the above into (4.20) and dividing both sides by $\delta_k(X)$, we obtain (4.21). □

To put the above results into perspective, let us examine the right-hand side of (4.21). Clearly, the first term, the ratio involving $\|q_j^T X\|$'s, is always less than or equal to one since $k \leq m$, and it decreases as m increases. In particular, when $m = k + 1 + pk$ with $p > 0$ and a large k, then $m \gg k$ and the ratio can be tiny as long as there is a significant decay in $\{\|q_j^T X\|\}_{j=1}^n$ between indices k and m. In addition, from (4.18), we know that the second term $\Gamma_{k,m}(V) \leq 1$ and can be far less than one if there is a large decay between λ_k and λ_{m+1}. The third term $\|G_1^\dagger E_k\|_2$, however, presents a complicating factor. How this term behaves as p increases requires a scrutiny which will be the topic of Section 4.4.

Similarly, we can examine the right-hand side of (4.20) in which only the first term is different. Given a good approximate basis X for which the row norms of $Q^T X$ have a nontrivial decay, we can also have $\Gamma_{k,m}(Q^T X) \ll 1$; and the faster the decay is, the smaller is the term $\Gamma_{k,m}(Q^T X)$. Therefore, with the exception of the term $\|G_1^\dagger E_k\|_2$, all the terms in the right-hand sizes of (4.20) and (4.21) are small under reasonable conditions.

Next we consider the case where $X \in \mathbb{R}^{n\times k}$ is the result of applying a block power iteration q times to an initial random matrix $X_0 \in \mathbb{R}^{n\times k}$,

$$X = \rho(A)^q X_0 = Q\rho(\Lambda)^q Q^T X_0, \tag{4.22}$$

where $\rho(A)$ is a polynomial or rational matrix function accelerator (or filter) such that

$$\min_{1\leq j\leq k} |\rho(\lambda_j)| = |\rho(\lambda_k)| \geq |\rho(\lambda_{k+1})| \geq \cdots \geq |\rho(\lambda_{m+1})| = \max_{m<j\leq n} |\rho(\lambda_j)|. \tag{4.23}$$

THEOREM 4.4. *Let X be defined in (4.22) from an initial matrix $X_0 \in \mathbb{R}^{n\times k}$. Assume that the conditions of Lemma 4.1 hold. Then there exists a constant c_m such that*

$$\delta_k(Y) \leq c_m \left| \frac{\rho(\lambda_{m+1})}{\rho(\lambda_k)} \right|^q , \tag{4.24}$$

where

$$c_m = \Gamma_{k,m}(Q^T X_0)\Gamma_{k,m}(V) \left\| G_1^\dagger E_k \right\|_2 . \tag{4.25}$$

Moreover, there exists a constant c'_m such that

$$\frac{\delta_k(Y)}{\delta_k(X)} \leq c'_m \left| \frac{\rho(\lambda_{m+1})}{\rho(\lambda_{k+1})} \right|^q , \tag{4.26}$$

where

$$c'_m = \frac{\max_{j>m} \|q_j^T X_0\|}{\min_{j>k} \|q_j^T X_0\|} \Gamma_{k,m}(V) \left\| G_1^\dagger E_k \right\|_2 . \tag{4.27}$$

Proof. It follows from $Q^T X = \rho(\Lambda)^q Q^T X_0$ that

$$\|q_i^T X\| = |\rho(\lambda_i)|^q \|q_i^T X_0\|, \;\; i = 1, \cdots, n. \tag{4.28}$$

Applying (4.19) to (4.28), we obtain $\Gamma_{k,m}(Q^T X) \le \Gamma_{k,m}(\rho(\Lambda)^q)\Gamma_{k,m}(Q^T X_0)$ which establishes (4.24), upon substituting into (4.20).

To prove (4.26), we first use (4.28) to calculate

$$\frac{\max_{j>m} \|q_j^T X\|}{\max_{j>k} \|q_j^T X\|} = \frac{\max_{j>m} |\rho(\lambda_j)|^q \|q_j^T X_0\|}{\max_{j>k} |\rho(\lambda_j)|^q \|q_j^T X_0\|} \le \left|\frac{\rho(\lambda_{m+1})}{\rho(\lambda_{k+1})}\right|^q \frac{\max_{j>m} \|q_j^T X_0\|}{\min_{j>k} \|q_j^T X_0\|}.$$

Then substituting the above into (4.21) yields (4.26). □

Let us also state a couple of special cases of (4.24).

COROLLARY 4.5. *If the G_1-Assumption holds for $m = k + pk$, then there exist constants C_p and C_p' such that*

$$\delta_k(Y) \le C_p \left|\frac{\rho(\lambda_{k+1+pk})}{\rho(\lambda_k)}\right|^q \qquad \textit{and} \qquad \frac{\delta_k(Y)}{\delta_k(X)} \le C_p' \left|\frac{\rho(\lambda_{k+1+pk})}{\rho(\lambda_{k+1})}\right|^q.$$

In particular, when there is no augmentation ($p = 0$) and no acceleration ($\rho(t) = t$), the convergence rate reduces to $\delta_k(Y) \le C_0 \, |\lambda_{k+1}/\lambda_k|^q$.

Finally, we remark that all of our results point out that there exists a matrix $Y \in \mathbb{R}^{n\times k}$ in the augmented subspace $\mathcal{R}(\mathbf{X}_p)$ (which is constructed from the matrix X) that is a better approximate basis for $\mathcal{R}(Q_k)$ than X is, under reasonable conditions. It is known that the Ritz pairs produced by the RR procedure are optimal approximations to the eigenpairs of A from the input subspace (see [16] for example). Therefore, the derived bounds in this section should be attainable by the Ritz pairs generated by the ARR procedure.

4.4. Validity of G_1-Assumption. A key condition for our results is the G_1-Assumption, given in (4.11), that requires the first m rows of G in (4.7) to be linearly independent. Under this assumption, the larger m is, the better the convergence rate could be.

Let us examine the matrix G_1 consisting of the first m rows of G in (4.7). To simplify notation, we use H for G_1, redefine C as the first m row of C in (4.7), and consider the matrix

$$H = [C \;\; \Lambda_m C \;\; \cdots \;\; \Lambda_m^p C] \in \mathbb{R}^{m\times(p+1)k}, \tag{4.29}$$

where Λ_m is the $m \times m$ leading block of Λ whose disgonal is assumed to be positive.

We first give a necessary condition for the m rows of H to be linearly independent.

PROPOSITION 4.6. *Let $m \in (k, k + pk]$ for $p > 0$. The matrix $H \in \mathbb{R}^{m\times(p+1)k}$ defined in* (4.29) *has full rank m only if Λ_m has no more than k equal diagonal elements (i.e., Λ_m contains no eigenvalue of multiplicity greater than k).*

Proof. Without loss of generality, suppose that the first $k + 1$ diagonal elements of Λ_m are all equal, i.e., $\lambda_1 = \lambda_2 = \cdots = \lambda_{k+1} = \alpha$. Then the first $k + 1$ rows of H, say H', is of the form $H' = [C' \;\; \alpha C' \;\; \cdots \;\; \alpha^p C']$, where C' consists of the first $k + 1$ rows of C. Since all column blocks are scalar multiples of C' which has k columns, the rank of H is at most k. independent of m. □

The fact that H is built from C which has only k columns dictates that to have $\mathrm{rank}(H)$ greater than k, it is necessary that the maximum multiplicity of Λ_m must not exceed k.

On the other hand, the next result says that when $p = 1$ and m reaches its upper bound $2k$, a multiplicity equal to k is sufficient for H to attain the full rank $2k$ (i.e., to be nonsingular) in a generic case.

First, let us do the partitioning

$$C = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}, \quad \Lambda_m = \begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix}, \quad H = \begin{bmatrix} C_1 & \Lambda_1 C_1 \\ C_2 & \Lambda_2 C_2 \end{bmatrix}. \tag{4.30}$$

where $m = 2k$, and C_j, Λ_j, $j = 1, 2$, are all $k \times k$ submatrices. Recall that Λ_1 consists of the first k eigenvalues of A and Λ_2 the next k eigenvalues.

PROPOSITION 4.7. *Let $p = 1$, $m = 2k$, and C, Λ_m and H be defined as in* (4.30). *Let r be the maximum multiplicity of Λ_m. Assume that any $k \times k$ submatrix of C is nonsingular. Then H is nonsingular for $r = k$.*

Proof. We will show that when λ_1 or λ_{k+1} has multiplicity k, then H is nonsingular. All the other cases can be similarly proven with appropriate permutations before partitioning (4.30) is done.

First, the nonsingularity of H is equivalent to that of

$$\begin{bmatrix} C_1 & \Lambda_1 C_1 \\ C_2 & \Lambda_2 C_2 \end{bmatrix} \begin{bmatrix} C_1^{-1} & \\ & C_1^{-1} \end{bmatrix} = \begin{bmatrix} I & \Lambda_1 \\ C_2 C^{-1} & \Lambda_2 C_2 C^{-1} \end{bmatrix} = \begin{bmatrix} I & \Lambda_1 \\ F & \Lambda_2 F \end{bmatrix},$$

where $F \triangleq C_2 C_1^{-1}$ is nonsingular by our assumption. Eliminating the (2,1) block, we obtain

$$\begin{bmatrix} I & \Lambda_1 \\ F & \Lambda_2 F \end{bmatrix} \longrightarrow \begin{bmatrix} I & \Lambda_1 \\ 0 & \Lambda_2 F - F\Lambda_1 \end{bmatrix}$$

Hence, the nonsingularity of H is equivalent to that of $F\Lambda_1 - \Lambda_2 F$, or in turn equivalent to that of the following matrix:

$$K = \Lambda_1 - F^{-1}\Lambda_2 F. \tag{4.31}$$

If the multiplicity of λ_1 is k (implying that $\Lambda_1 = \lambda_k I$), (4.31) reduces to $K = F^{-1}(\lambda_k I - \Lambda_2)F$. On the other hand, if the multiplicity of λ_{k+1} is k (implying that $\Lambda_2 = \lambda_{k+1} I$), then $K = \Lambda_1 - \lambda_{k+1} I$. In either case, K is nonsingular since $\lambda_{k+1} < \lambda_k$; hence, so is H. (Also in either case, K becomes singular for multiplicity $r > k$ which implies $\lambda_{k+1} = \lambda_k$.) □

In Proposition 4.7, we assume that every $k \times k$ submatrix of C is nonsingular. It is well-known that for a generic random matrix C, this assumption holds with high probability. Therefore, in a generic setting Proposition 4.7 holds with high probability.

Now the unproven case is for maximum multiplicity $r < k$. Let us rewrite K in (4.31) into a sum of two matrices,

$$K = (\Lambda_1 - \lambda_k I) + F^{-1}(\lambda_k I - \Lambda_2)F. \tag{4.32}$$

The first is diagonal and positive semidefinite, and the second has positive eigenvalues when $\lambda_k > \lambda_{k+1}$, but is generally asymmetric. So far, we have not been able to find a result that guarantees nonsingularity for such a matrix K. However, in a generic setting where K comes from random matrices, nonsingularity should be expected with high probability (which has been empirically confirmed by our numerical experiments).

It should be noted that G_1 being nonsingular with $m = k + kp$ represents the best scenario where the acceleration potential of p-block augmentation is fully realized. However, $m < k + kp$ does not represent a failure, considering the fact that as long as $m > k$, an acceleration is still realized to some extent.

Once it is established for $p = 1$ and $m = 2k$ that in a generic setting H is nonsingular whenever the maximum multiplicity of Λ_m is less than or equal to k, the same result can in principle be extended to the case of $p = 3$ by considering

$$H = [C \;\; \Lambda C \;\; \Lambda^2 C \;\; \Lambda^3 C] = [[C \;\; \Lambda C] \;\; \Lambda^2[C \;\; \Lambda C]] = [\hat{C} \;\; \hat{\Lambda}\hat{C}],$$

where $\hat{C} = [C \;\; \Lambda C]$ and $\hat{\Lambda} = \Lambda^2$, which has the same form as for the case $p = 1$. It will also cover the case of $p = 2$ where the matrix involved is a submatrix of the one for $p = 3$.

It is worth noting that $m = (p+1)k$ could be kept constant if k is decreased while p is increased. Is it sensible to use fewer vectors in power iterations but to compensate it with an augmentation of more blocks? Although in some cases this strategy works well, in general it seems to be a risky approach for two reasons. First, the smaller k is, the more likely it is to encounter matrices that have eigenvalues of multiplicity greater than k. In this case, by Proposition 4.6, the benefit of augmentation could become limited. Secondly, we have observed in numerical experiments that the condition number of G_1 tends to increase as p increases, which would in turn increase the constants c_m and c'_m in (4.20)-(4.21). These facts suggest that using a small k and a large p to compute more than k eigenpairs could be numerically problematic. In our implementation, we choose to be conservative by using the default value of $p = 1$, while setting k to be slightly bigger than the number of eigenpairs to be computed.

5. Polynomial Accelerators. To construct polynomial accelerators (or filters) $\rho(t)$, we use Chebyshev interpolants on highly smooth functions. Chebyshev interpolants are polynomial interpolants on Chebyshev points of the second kind, defined by

$$t_j = -\cos(j\pi/N), 0 \le j \le N, \tag{5.1}$$

where $N \ge 1$ is an integer. Obviously, this set of $N+1$ points are in the interval $[-1,1]$ inclusive of the two endpoints. Through any given data values $f_j, j = 0, 1, \cdots, N$, at these $N+1$ Chebyshev points, the resulting unique polynomial interpolant of degree N or less is a Chebyshev interpolant. It is known that Chebyshev interpolants are "near-best" [5].

Our choices of functions to be interpolated are

$$f_d(t) = (f_1(t))^d \quad \text{where} \quad f_1(t) = \max(0,t)^{10}, \tag{5.2}$$

and d is a positive integer. Obviously, $f_d(t) \equiv 0$ for $t \le 0$ and $f_d(1) \equiv 1$. The power 10 is rather arbitrary and exchangeable with other numbers of similar magnitude without making notable differences.

The functions in (5.2) are many times differentiable so that their Chebyshev interpolants converge relatively fast, see [15]. Interpolating such smooth functions on Chebyshev points helps reducing the effect of the Gibbs phenomenon and allows us to use relatively low-degree polynomials.

There is a well-developed open-source Matlab package called `Chebfun` [4] for doing Chebyshev interpolations, among many other functionalities[1]. In this work, we have used `Chebfun` to construct Chebyshev interpolants as our polynomial accelerators. Specifically, we interpolate the function $f_d(t)$ by the d-th degree Chebyshev interpolant polynomial, say,

$$\psi_d(t) = \gamma_1 t^d + \gamma_2 t^{d-1} + \ldots + \gamma_d t + \gamma_{d+1}. \tag{5.3}$$

Suppose that we want to dampen the eigenvalues in an interval $[a,b]$, where $a \le \lambda_n$ and $b < \lambda_k$, while magnifying eigenvalues to the right of $[a,b]$. Then we map the interval $[a,b]$ onto $[-1,1]$ by an affine transformation and then apply $\psi_d(\cdot)$ to A. That is, we apply the following polynomial function to A,

$$\rho_d(t) = \psi_d\left(\frac{2t-a-b}{b-a}\right). \tag{5.4}$$

Let $\Gamma_d = (\gamma_1, \gamma_2, \cdots, \gamma_{d+1})$ denote the coefficients of the polynomial $\psi_d(t)$ in (5.3). The corresponding matrix operation $Y = \rho_d(A)X$ can be implemented by Algorithm 7 below.

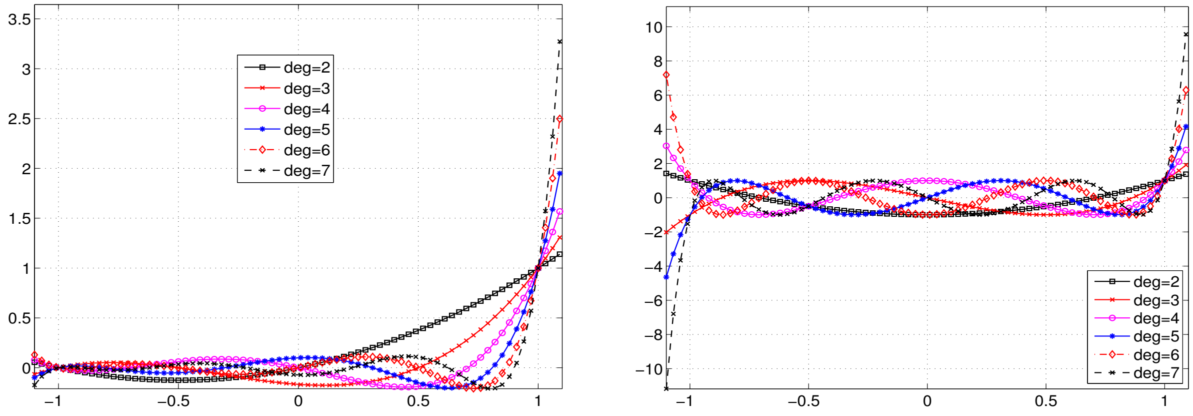
Algorithm 7: Polynomial function: $Y = \text{POLY}(A, X, a, b, \Gamma_d)$

```
Compute $c_0 = \frac{a+b}{a-b}$ and $c_1 = \frac{2}{b-a}$. Set $Y = \gamma_1 X$.
for $j = 1, 2, \ldots, d$ do  $Y = c_0 Y + c_1 AY + \gamma_{j+1} X$.
```

For a quick comparison, we plot our Chebyshev interpolates of degrees 2 to 7 and the Chebyshev polynomials of degrees 2 to 7 side by side in Figure 5.1. For both kinds of polynomials, the higher the degree is, the closer the curve is to the vertical line $t = 1$. We observe that inside the interval $[-1,1]$, our Chebyshev interpolates have lower profiles (with magnitude less than or around 0.2 except near 1) than the Chebyshev polynomials which oscillate between ±1, while outside $[-1,1]$ the Chebyshev polynomials grow faster.

The idea of polynomial acceleration is straightforward and old, but its success is far from foolproof, largely due to inevitable errors in estimating intervals within which eigenvalues are supposed to be suppressed or promoted. The main reason for us to prefer our Chebyshev interpolates over the classic Chebyshev polynomials is that their lower profiles tend to make them less sensitive to erroneous intervals, hence easier to control. Indeed, our numerical comparison, albeit limited, appears to justify our choice.

[1] Also see the website `http://www.chebfun.org/docs/guide/guide04.html`



(a) Chebyshev interpolants of degrees 2 to 7

(b) Chebyshev polynomials of degrees 2 to 7

FIG. 5.1. *illustration of polynomial functions*

6. Details of ARRABIT Algorithms.

In this section, we describe technical details and give parameter choices for our ARRABIT algorithm which computes k eigenpairs corresponding to k algebraically largest eigenvalues of a given symmetric matrix A.

Guard vectors. When computing k eigenpairs, it is a common practice to compute a few extra eigenpairs to help guard against possible slow convergence. For this purpose, a small number of "guard vectors" are added to the iterate matrix X. In general, the more guard vectors are used, the less iterations are needed for convergence, but at a higher cost per iteration on memory and computing time. In our implementation, we set the number of columns in iterate matrix X to $k+q$, where by default q is set to $0.1k$ (rounded to the nearest integer).

Estimation of λ_n and λ_{k+q}. To apply polynomial accelerators, we need to estimate the interval $[a,b] = [\lambda_n, \lambda_{k+q}]$ which contains unwanted eigenvalues. The smallest eigenvalue λ_n is computed by calling the the Matlab built-in solver EIGS (i.e., ARPACK [12]). Given an initial matrix $X \in \mathbb{R}^{n\times(k+q)}$ whose columns are orthogonalized, an under-estimation of λ_{k+q} can be taken as the smallest eigenvalue of the projected matrix X^TAX (which requires an RR projection). As the iterations progress, more accurate estimates of λ_{k+q} will becomes available after each later ARR projection.

Outer loop stop rule. Let (x_i, μ_i), $i = 1, 2, \cdots, k$, be computed Ritz pairs where $x_i^T x_j = \delta_{ij}$. We terminate the algorithm when the following maximum relative residual norm becomes smaller than a prescribed tolerance tol, i.e.,

$$\texttt{maxres} := \max_{i=1,\ldots,k} \{\texttt{res}_i\} \le \mathrm{tol}, \tag{6.1}$$

where

$$\texttt{res}_i := \frac{\|Ax_i - \mu_i x_i\|_2}{\max(1, |\mu_i|)}, \; i = 1, \cdots, k. \tag{6.2}$$

The algorithm is also stopped in the following three cases: (i) if a maximum number of iterations, denoted by "maxit", is reached (by default maxit = 30); or (ii) if the maximum relative residual norm has not been reduced after three consecutive outer iterations; or (iii) if most Ritz pairs have residuals considerably smaller than tol and the remaining have residuals slightly larger than tol; specifically, $\texttt{maxres} < (1 + 9h/k)\mathrm{tol}$ $(< 10 * \mathrm{tol})$, where h is the number of Ritz pairs with residuals less than $0.1 * \mathrm{tol}$. In our experiments we also monitor the computed partial trace $\sum_{i=1}^k \mu_i$ at the end for all solvers as a check for correctness.

Continuation. When a high accuracy (say, $\mathrm{tol} \le 10^{-8}$) is requested, we use a continuation procedure to compute Ritz-pairs satisfying a sequence of tolerances: $\mathrm{tol}_1 > \mathrm{tol}_2 > \cdots \ge \mathrm{tol}$, and use the computed Ritz-pairs for tol_t as the starting point to compute the next solution for tol_{t+1}. In our implementation, we use the update scheme

$$\mathrm{tol}_{t+1} = \max(10^{-2}\,\mathrm{tol}_t, \; \mathrm{tol}), \tag{6.3}$$

where tol_1 is chosen to be considerably larger than tol. A main reason for doing such a continuation is that our deflation procedure (see below) is tolerance-dependent. At the early stages of the algorithm, a stringent tolerance would delay the activation of deflation and likely cause missed opportunities in reducing computational costs.

Inner loop parameters and stop rule. Both MPM and GN are tested as inner solvers to update X. These inner solvers are applied to the shifted matrix $A - aI$ which is supposedly positive semidefinite since a is a good approximation to λ_n (computed by EIGS in our implementation). We check inner stopping criteria every maxit_2 iterations and check them at most maxit_1 times. In the present version, the default values for these two parameters are $\mathrm{maxit}_1 = 10$ and $\mathrm{maxit}_2 = 5$ Therefore, the maximum number of inner iterations allowed is $\mathrm{maxit}_1 \times \mathrm{maxit}_2 = 50$.

The inner loop stopping criteria are either

$$\texttt{rc} = \texttt{rcond}(X^T X) \le \mathrm{tol}_t \quad \text{or} \quad \texttt{rc}/\texttt{rcp} > 0.99, \tag{6.4}$$

where tol_t is the current tolerance (in a continuation sequence) and rcp is the previously computed $\mathrm{rcond}(X)$. In (6.4), we use the rcond subroutine in LAPACK (also used by Matlab) to estimate the reciprocal 1-norm condition number of $X^T X$, which we find to be relatively inexpensive. The first condition in (6.4) indicates that X is about to lose (or have just lost) rank numerically, which implies that we achieve the goal of eliminating the unwanted eigenspace numerically. However, it is probable that a part of the desired eigenspace is also sacrificed, especially when there are clusters among the desired eigenvalues. Fortunately, this problem can be corrected, at a cost, in later iterations after deflation. On the other hand, the second condition is used to deal with the situation where the conditioning of X does not deteriorate, which occurs from time to time in later iterations when there exists little or practically no decay in the relevant eigenvalues.

Deflation. Since Ritz pairs normally have uneven convergence rates, a procedure of detecting and setting aside Ritz pairs that have "converged" is called deflation or locking, which is regularly used in eigensolvers because it not only reduces the problem size but also facilitates the convergence of the remaining pairs. In our algorithm, a Ritz pair (x_i, μ_i) is considered to have "converged" with respect to a tolerance tol_t if its residual (see (6.2) for definition) satisfies

$$\texttt{res}_i \le \max(10^{-14}, \mathrm{tol}_t^2). \tag{6.5}$$

After each ARR projection, we collect the converged Ritz vectors into a matrix Q_c, and start the next iteration from those Ritz vectors "not yet converged", which we continue to call X. Obviously, whenever Q_c is nonempty X is orthogonal to Q_c. Each time we check the stopping rule in the inner loop, we also perform a projection $X = X - Q_c(Q_c^T X)$ to ensure that X stays orthogonal to Q_c. In addition, the next ARR projection will also be performed in the orthogonal complement of $\mathcal{R}(Q_c)$. That is, we apply an ARR projection to the matrix $Y - Q_c(Q_c^T Y)$ for $Y = [X \ AZ \ \cdots \ A^p X]$. At the end, we always collect and keep $k + q$ leading Ritz pairs from both the "converged" and the "not yet converged" sets.

Augmentation blocks. The default value for the number of augmentation blocks is $p = 1$, but this value may be adjusted after each ARR projection. We increase p by one when we find that the relevant Ritz values show a small decay and at the same time the latest decrease in residuals is not particularly impressive. Specifically, we set $p = p + 1$ if

$$\frac{\mu_{k+q}}{\mu_k} > 0.95 \quad \text{and} \quad \frac{\texttt{maxres}}{\texttt{maxresp}} > 0.1, \tag{6.6}$$

where maxresp is the maximum relative residual norm at the previous iteration. The values 0.95 and 0.1 are set after some limited experimentation and by no means optimal. For k relatively large, since the memory demand grows significantly as p increases, we also limit the maximum value of p to $p_{\max} = 3$.

Polynomial degree. Under normal conditions, the higher degree is used in a polynomial accelerator, the fewer number of iterations will be required for convergence, but at a higher cost per iteration. A good balance is needed. Let d and $d_{\max}$ be the initial and the largest polynomial degrees, respectively. We use the default values $d = 3$ and $d_{\max} = 15$. Let $\rho_d(t)$ be the polynomial function defined in (5.4). After each ARR step, we adjust the degree based on estimated spectral information of $\rho_d(A)$ computable using the current Ritz values. We know that the convergence rate of the inner solvers would be satisfactory if the eigenvalue ratio $\rho_d(\lambda_{k+q})/\rho_d(\lambda_k)$ is small. Based on this consideration, we calculate

$$\hat{d} = \min_{d \ge 3} \left\{ d \in \mathbb{Z} : \frac{\rho_d(\mu_{k+q}^*)}{\rho_d(\mu_k^*)} < 0.9 \right\}, \tag{6.7}$$

and then apply the cap $d_{\max}$ by setting

$$d = \min(\hat{d}, d_{\max}) \tag{6.8}$$

where μ_k^* and μ_{k+q}^* are a pair of Ritz values corresponding to the iteration with the smallest residual "`maxres`" defined in (6.1) (therefore the most accurate so far). The value of 0.9 is of course adjustable.

Finally, a pseudocode for our ARRABIT algorithm with all the above features is presented as Algorithm 8. This is the version used to produce the numerical results of this paper. As one can see, ARRABIT algorithm uses A only in matrix multiplications.

Algorithm 8: Algorithm ARRABIT (detailed version)

```
Input $A \in \mathbb{R}^{n\times n}$, integer $k \in (0, n)$ and tolerance $\mathrm{tol} > 0$.
Choose $d$ and $d_{\max}$, the initial and maximum polynomial degrees.                /* initialize */
Choose $p$ and $p_{\max}$, the initial and maximum number of augmentation blocks.
Choose $q \geq 0$, the number of guard vectors, so that $(p+1)(k+q) < n$.
Set tolerance parameters: $t = 1$, $\mathrm{tol}_t \geq \mathrm{tol}$ and $\mathrm{tol}_d = \max(10^{-14}, \mathrm{tol}_t^2)$.
Initialize converged Ritz pairs $(Q_c, \Sigma_c) = \emptyset$ for deflation purposes.
Initialize an i.i.d. Gaussian random matrix $X \in \mathbb{R}^{n\times(k+q)}$.
Estimate the interval $[\lambda_n, \lambda_{k+q}] \approx [a, b]$.
for $j = 1, \ldots, \mathrm{maxit}$ do                                                 /* outer loop */
    Initialize rc to infinity.
    for $i_1 = 1, 2, \cdots, \mathrm{maxit}_1$, do                                     /* inner loop */
        for $i_2 = 1, 2, \cdots, \mathrm{maxit}_2$, do                          /* call inner solvers */
            if MPM is the inner solver, then                                                 /* MPM */
                Call $X = \mathrm{POLY}(A - aI, X, 0, b - a, \Gamma_d)$.               /* accelerator */
                Normalize the columns of $X$ individually.
            if GN is the inner solver, then                                                   /* GN */
                Compute $Y = X\left(X^T X\right)^{-1}$.
                Call $Z = \mathrm{POLY}(A - aI, Y, 0, b - a, \Gamma_d)$.               /* accelerator */
                Compute $X = Z - X(Y^{\mathrm{T}} Z - I)/2$.
            Compute $X = X - Q_c(Q_c^T X)$ if $Q_c \neq \emptyset$.                     /* projection */
        Set $\mathrm{rcp} = \mathrm{rc}$ and compute $\mathrm{rc} = \mathrm{rcond}(X^T X)$.
        if the inner stop rule (6.4) is met, then break.                          /* end inner loop */
    Compute $Y = [X, AX, \ldots, A^p X]$.                                          /* augmentation */
    $Y = Y - Q_c(Q_c^T Y)$ if $Q_c \neq \emptyset$.                                    /* projection */
    Perform ARR step: $(X, \Sigma) = \mathrm{RR}(A, Y)$.                                      /* ARR */
    Extract $k + q$ leading Ritz pairs $(x_i, \mu_i)$ from $(Q_c, \Sigma_c)$ and $(X, \Sigma)$.
    Overwrite $(X, \Sigma)$ by the $k + q$ Ritz pairs. Compute residuals by (6.2).
    if the outer stop rule (6.1) is met for tol, then
        output the Ritz pairs $(X, \Sigma)$ and exit.                            /* output and exit */
    if the outer stop rule (6.1) is met for $\mathrm{tol}_t$ then                    /* continuation */
        Set $\mathrm{tol}_{t+1} = \max\left(10^{-2}\mathrm{tol}_t, \mathrm{tol}\right)$, $b = \mu_{k+q}$ and $t = t + 1$.
    Collect converged Ritz pairs in $(Q_c, \Sigma_c)$ that satisfy (6.5).              /* deflation */
    Overwrite $(X, \Sigma)$ by the remaining not yet converged Ritz pairs.
    if rules in (6.6) are met, then set $p = \min(p + 1, p_{\max})$.                     /* update p */
    Update the polynomial degree by rules (6.7)-(6.8).                              /* update degree */
```

7. Numerical Results. In this section, we evaluate the performance of ARRABIT on a set of sixteen sparse matrixes. Although we have constructed the algorithm with parallel scalability in mind as a major motivating factor, a study of scalability issues in a massively parallel environment is beyond the scope of the current paper.

As a first step, we test the algorithm in Matlab environment, on a single computing node (2 processors) and without explicit code parallelization, to determine how it performs in comparison to established solvers. We have implemented our ARRABIT algorithm, as is described by the pseudocode Algorithm 8, in MATLAB. For brevity, the two variants, corresponding to the two choices of inner solvers, will be called MPM and GN, respectively.

We test two levels of accuracy in our experiments: $\mathrm{tol} = 10^{-6}$ or $\mathrm{tol} = 10^{-12}$. By our stoping rule, upon successful termination the largest eigenpair residual will not exceed 10^{-5} or 10^{-11}, respectively. Since our algorithm checks the termination rule only after each ARR call, it often returns solutions of higher accuracies than what is prescribed by the tol value.

7.1. Solvers, Platform and Test Matrices. Since it is impractical to carry out numerical experiments with a large number of solvers, we have carefully chosen two high-quality packages to compare with our ARRABIT code. One package is ARPACK[2] [12], which is behind the Matlab built-in iterative eigensolver EIGS, and will naturally serve as the benchmark solver. Another is a more recent package called FEAST [28] which has been integrated into Intel's Math Kernel Library (MKL) under the name "Intel MKL Extended Eigensolver"[3]. Both ARPACK and FEAST are written in Fortran. While ARPACK can be directly accessed through EIGS in Matlab, we call FEAST from Intel's MKL Library via Matlab's MEX external interfaces. In our experiments, all parameters in EIGS and FEAST are set to their default values, and each solver terminates with its own stopping rules using either $\mathrm{tol} = 10^{-6}$ or $\mathrm{tol} = 10^{-12}$.

We have also examined a few other solvers as potential candidates but decided not to use them in this paper, including but not limited to the filtered Lanczos algorithm[4] [6] and the Chebyshev-Davidson algorithm[5] [29]. Our initial tests indicated that, for various reasons, these solvers' overall performance could not measure up with that of the commercial-grade software packages ARPACK and FEAST on a number of test problems. This fact may be more of a reflection on the current status of software development for these solvers than on the merits of the algorithms behind.

It is important to note that FEAST is designed to compute all eigenvalues (and their eigenvectors) in an interval, which is given as an input along with an estimated number of eigenvalues inside the interval. When computing k largest eigenpairs, we have observed that the performance of FEAST is affected greatly by the quality of the two estimations: the interval itself and the number of eigenvalues inside the interval. When calling FEAST, we set (i) the interval to be $[\lambda_k^*, \lambda_1^*]$ where λ_k^* and λ_1^* are computed eigenvalues by EIGS using the same tolerance tol; and (ii) the estimated number of eigenvalues in the interval to $1.2k$ rounded to the nearest integer. We consider this setting to be fair, if not overly favorable, to FEAST.

Our numerical experiments are preformed on a single computing node of Edison[6], a Cray XC30 supercomputer maintained at the National Energy Research Scientific Computer Center (NERSC) in Berkeley. The node consists of two twelve-core Intel "Ivy Bridge" processors at 2.4 GHz with a total of 64 GB shared memory. Each core has its own L1 and L2 caches of 64 KB and 256 KB, respectively; A 30-MB L3 cache shared between 12 cores on the "Ivy Bridge" processor. We generate Matlab standalone executable programs and submit them as batch jobs to Edison. The reported runtimes are wall-clock times.

On a multi/many-core computer, memory access patterns and communication overheads have a notable impact on computing time. In Matlab, dense linear algebra operations are generally well optimized by using BLAS and LAPACK tuned to the CPU processors in use. On the other hand, we have observed that some sparse linear algebra operations in Matlab seem to have not been as highly optimized (at least in version 2013b). In particular, when doing multiplications between a large sparse matrix and a dense matrix (like AX), Matlab is often slower than a routine in Intel's Math Kernel Library (MKL) named "mkl_dcscmm" when it is invoked through Matlab's MEX external interfaces in our experiments. For this reason, we use this MKL routine in our Matlab code to perform the operation AX.

Our test matrices are selected from the University of Florida Sparse Matrix Collection[7]. For each matrix, we compute both k eigenpairs corresponding to k largest eigenvalues and those corresponding to k smallest eigenvalues. Many of the selected matrices are produced by PARSEC [8], a real space density functional theory (DFT) based code for electronic structure calculation in which the Hamiltonian is discretized by a finite difference method. We do not take into account any background information for these matrices; instead, we simply treat them algebraically as matrices.

Table 7.1 lists, for each matrix A, the dimensionality n, the number of nonzeros $nnz(A)$ and the density of A, i.e., the ratio $(nnz(A)/n^2)100\%$. The number of eigenpairs to be computed is set either to 1% of n rounded to the nearest integer or to $k = 1000$ whichever is smaller. Table 7.1 also reports the number of the nonzeros in

[2] See http://www.caam.rice.edu/software/ARPACK/

[3] See http://software.intel.com/en-us/intel-mkl (version 11.0.2 on our workstation)

[4] See http://www-users.cs.umn.edu/~saad/software/filtlan

[5] See http://faculty.smu.edu/yzhou/code.htm

[6] See http://www.nersc.gov/users/computational-systems/edison/

[7] See http://www.cise.ufl.edu/research/sparse/matrices

the Cholesky factor L of matrix $A - \alpha I$ where $\alpha = \max(2\lambda_n(A), 0)$. The factorization is carried out after an "approximate minimum degree" permutation performed by the Matlab function "amd", as is done by the following MATLAB line: $\texttt{t} = \texttt{amd(B)}$; $\texttt{L} = \texttt{chol(B(t,t),'lower')}$. We have also tested the "symmetric approximate minimum degree" permutation ("symamd" in Matlab), but the corresponding density of L is slightly larger on most matrices. The density of factor L and the computing time in seconds used by Cholesky factorization are also given in Table 7.1. Although all matrices A are very sparse, the Cholesky factors of some matrices, such as Ga10As10H30, Ga3As3H12 and Ge87H76, are quite dense. As a result, the Cholesky factorization time varies greatly from matrix to matrix. We mention that the spectral distributions of the test matrices can behave quite differently from matrix to matrix. Even for the same matrix, the spectrum of a matrix can change behavior drastically from region to region. Most notably, computing k smallest eigenpairs of many matrices in this set turns out to be more difficult than computing k largest ones.

The largest matrix size in this set is more than a quarter of million. Relative to the computing resources in use, we consider these selected matrices to be fairly large scale. Overall, we consider this test set reasonably diverse and representative, fully aware that there always exist instances out there that are more challenging to one solver or another.

TABLE 7.1
Information of Test Matrices

matrix name	n	k	$nnz(A)$	density of A	nnz(L)	density of L	time
Andrews	60000	600	760154	0.021%	117039940	6.502%	7.18
C60	17576	176	407204	0.132%	34144169	22.105%	1.62
cfd1	70656	707	1825580	0.037%	35877440	1.437%	1.81
finance	74752	748	596992	0.011%	2837714	0.102%	0.28
Ga10As10H30	113081	1000	6115633	0.048%	1562547805	24.439%	127.12
Ga3As3H12	61349	613	5970947	0.159%	596645077	31.705%	42.00
shallow_water1s	81920	819	327680	0.005%	2357535	0.070%	0.21
Si10H16	17077	171	875923	0.300%	56103003	38.474%	2.60
Si5H12	19896	199	738598	0.187%	78918573	39.871%	3.80
SiO	33401	334	1317655	0.118%	186085449	33.359%	10.01
wathen100	30401	304	471601	0.051%	1490209	0.322%	0.32
Ge87H76	112985	1000	7892195	0.062%	1403571238	21.990%	109.64
Ge99H100	112985	1000	8451395	0.066%	1477089634	23.141%	120.08
Si41Ge41H72	185639	1000	15011265	0.044%	3457063398	20.063%	358.53
Si87H76	240369	1000	10661631	0.018%	5568995364	19.277%	1499.80
Ga41As41H72	268096	1000	18488476	0.026%	6998257446	19.473%	2498.43

7.2. Comparison between RR and ARR. We first evaluate the performance difference between ARR and RR for both MPM and GN. Table 7.2 gives results for computing both k largest and smallest eigenpairs on the first six matrices in Table 7.1 to the accuracy of $\mathrm{tol} = 10^{-12}$. We note that RR and ARR correspond to $p = 0$ and $p > 0$, respectively, in Algorithm 8. In order to differentiate the effect of changing p from that of changing the polynomial degree, we also test a variant of Algorithm 8 with a fixed polynomial degree at $d = 8$ (by skipping line 34). In Table 7.2, "maxres" denotes the maximum relative residual norm in (6.1), "time" is the runtime measured in seconds, "RR" is the total number of the outer iterations, i.e., the total number of the RR or ARR calls made (excluding the one called in preprocessing for estimating λ_{k+q}), and "p" and "d" are the number of augmentation blocks and the polynomial degree, respectively, used at the final outer iteration. In addition, on the matrices cfd1 and finance we plot the (outer) iteration history of maxres in Figures 7.1 and 7.2 for computing k largest and smallest eigenpairs, respectively.

The following observations can be drawn from the table and figures.

- The performances of MPM and GN are similar. For both of them, ARR can accelerate convergence, reduce the number of outer iterations needed, and improve the accuracy, often to a great extent.
- The scheme of adaptive polynomial degree generally works better than a fixed polynomial degree. A more detailed look at the effect of polynomial degrees is presented in Section 7.3.
- The default value $p = 1$ for the number of augmentation blocks in ARR is generally kept unchanged (recall that it can be increased by the algorithm).
- The total number of ARR called is mostly very small, especially in the cases where the adaptive polynomial degree scheme is used and the k largest eigenpairs are computed (which tend to be easier than the k smallest ones). We observe from Figure 7.1 that in several cases a single ARR is sufficient to reach the accuracy of tol=1e-6 (even of tol=1e-12 in one case).

TABLE 7.2
Comparison results between RR and ARR with tol=1e-12

	MPM with RR				MPM with ARR				GN with RR				GN with ARR			
matrix	maxres	time	RR	p/d	maxres	time	RR	p/d	maxres	time	RR	p/d	maxres	time	RR	p/d
computing k largest eigpair by fix deg = 8																
Andrew.	9.5e-13	191	4	1/ 8	1.9e-06	250	9	3/ 8	9.0e-12	174	6	1/ 8	9.9e-13	104	2	1/ 8
C60	4.0e-12	45	11	3/ 8	6.3e-12	12	3	1/ 8	7.5e-12	44	22	3/ 8	1.4e-12	16	5	1/ 8
cfd1	9.8e-13	381	4	1/ 8	1.0e-12	296	4	1/ 8	9.8e-13	294	4	1/ 8	9.9e-13	206	2	1/ 8
financ.	9.9e-13	157	3	1/ 8	8.9e-13	151	3	1/ 8	1.0e-12	196	4	1/ 8	1.0e-12	141	2	1/ 8
Ga10As.	3.5e-13	1218	22	3/ 8	9.9e-13	1483	8	2/ 8	6.1e-12	910	8	1/ 8	9.9e-13	448	3	1/ 8
Ga3As3.	9.7e-13	467	6	1/ 8	9.8e-13	270	5	1/ 8	1.9e-12	307	8	1/ 8	9.4e-13	179	3	1/ 8
computing k largest eigpair with adaptive polynomial degree																
Andrew.	2.0e-11	337	9	3/ 5	8.8e-13	148	5	2/ 5	5.3e-12	319	17	3/ 5	1.0e-12	125	4	1/ 5
C60	8.7e-12	41	10	3/ 9	2.0e-12	13	3	1/ 9	4.2e-12	42	20	3/ 9	5.5e-12	13	3	1/ 9
cfd1	1.3e-12	441	5	1/ 3	9.8e-13	190	4	1/ 3	4.1e-12	482	17	3/ 3	9.9e-13	188	3	1/ 3
financ.	9.9e-13	256	4	1/ 3	1.3e-12	97	3	2/ 3	2.7e-12	380	14	3/ 3	1.1e-12	69	1	1/ 3
Ga10As.	4.7e-12	1199	6	1/ 5	9.6e-13	442	4	1/ 5	7.1e-12	1442	19	3/ 5	9.7e-13	580	4	1/ 6
Ga3As3.	2.9e-12	473	7	2/ 5	1.7e-12	169	4	1/ 5	3.9e-12	494	17	3/ 5	1.7e-12	198	4	1/ 5
computing k smallest eigpair by fix deg = 8																
Andrew.	4.2e-12	465	7	2/ 8	1.5e-13	219	6	2/ 8	7.2e-12	475	19	3/ 8	1.0e-12	199	5	1/ 8
C60	1.7e-12	30	9	3/ 8	6.8e-13	17	6	1/ 8	5.5e-12	24	13	3/ 8	6.7e-12	13	4	1/ 8
cfd1	4.1e-05	2870	30	3/ 8	6.0e-12	1543	21	3/ 8	1.5e-04	2505	30	3/ 8	7.9e-12	1394	22	3/ 8
financ.	3.8e-08	1759	30	3/ 8	5.1e-13	700	9	3/ 8	3.5e-06	1651	30	3/ 8	7.2e-13	713	11	3/ 8
Ga10As.	8.6e-10	2642	10	3/ 8	3.7e-12	1372	5	1/ 8	2.1e-02	1436	6	1/ 8	2.6e-12	961	4	1/ 8
Ga3As3.	7.2e-12	964	11	3/ 8	2.7e-12	489	4	1/ 8	4.2e-12	994	24	3/ 8	9.9e-13	381	4	1/ 8
computing k smallest eigpair with adaptive polynomial degree																
Andrew.	7.3e-12	466	8	3/ 8	9.7e-13	200	4	1/ 8	8.9e-12	505	21	3/ 8	1.1e-12	185	5	1/ 8
C60	6.7e-12	38	9	3/ 7	2.8e-12	26	9	3/ 6	4.0e-12	31	23	3/ 6	9.2e-13	15	8	2/ 6
cfd1	3.7e-08	2869	30	3/15	8.9e-12	719	4	1/15	2.3e-06	2515	30	3/15	4.2e-12	1017	12	3/15
financ.	3.7e-12	1391	9	3/15	1.4e-12	600	6	1/15	5.3e-12	1416	24	3/15	3.4e-12	467	5	1/15
Ga10As.	4.5e-11	3261	12	3/ 8	1.1e-12	1558	6	1/ 8	2.9e-12	3681	24	3/ 8	4.0e-12	963	3	1/ 9
Ga3As3.	5.9e-12	1046	8	3/ 9	9.9e-13	420	4	1/ 9	7.7e-12	1238	24	3/ 9	9.5e-13	338	5	1/ 9

7.3. Comparison on Polynomials. We next examine the effect of polynomial degrees on the convergence behavior of MPM and GN, again on the first six matrices in Table 7.1. We compare two schemes: the first is to use a fix degree among $\{4, 8, 15\}$ and skip line 34 of Algorithm 8, and the second is the adaptive scheme in Algorithm 8. The computational results are summarized in Table 7.3. We also plot the iteration history of `maxres`, for computing both k largest and smallest eigenpairs on the matrices cfd1 and finance in Figures 7.3 and 7.4, respectively. The numerical results lead to the following observations:

- Again the performances of MPM and GN are similar, and the default value $p = 1$ for augmentation is mostly unchanged.
- In general, the number of outer iterations is decreased as the polynomial degree is increased, but the runtime time is not necessarily reduced because of the extra cost in using higher-degree polynomials. Overall, our adaptive strategy seems to have achieved a reasonable balance.
- With fixed polynomial degrees, in a small number of test case MPM and GN fail to reach the required accuracy.

Finally, we compare the performance of Algorithm 8 either using Chebyshev interpolates defined in (5.3) or the Chebyshev polynomials defined in (2.6) on the first six matrices in Table 7.1. The comparison results are given in Table 7.4. Even though both types of polynomials work well on these six problems, some performance differences are still observable in favor of our polynomials.

7.4. Comparison with ARPACK and FEAST. We now compare MPM and GN with EIGS and FEAST for computing both k largest and smallest eigenpairs for all sixteen test matrices presented in Tables 7.1 (which also lists the k values). Computational results are summarized in Tables 7.5 and 7.6, where "SpMV" denotes the total number of SpMVs, counting each operation $AX \in \mathbb{R}^{n\times k}$ as k SpMVs.

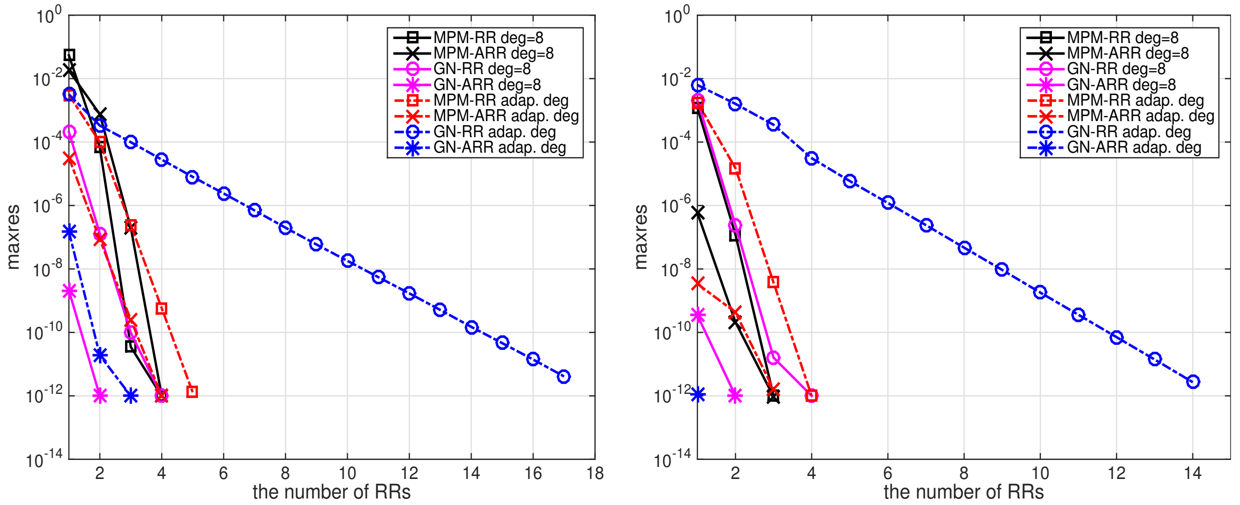
In addition, the speedup with respect to the benchmark time of EIGS is measured by the quantity $\log_2(\text{time}_{\text{EIGS}}/\text{time})$, as shown in Figures 7.5 and 7.6 where a positive bar represents a "speedup" and a negative one a "slowdown". In these two figures, matrices are ordered from left to right in ascending order of the solution time used by EIGS; that is, when moving from the left towards the right, problems become progressively more and more time-consuming for EIGS to solve. A quick glance at the figures tells us that MPM and GN provide clear speedups over EIGS on most problems,

TABLE 7.3
Comparison results of different polynomial degrees on tol=1e-12

	deg=4				deg=8				deg=15				adaptive deg			
matrix	maxres	time	RR	p/d	maxres	time	RR	p/d	maxres	time	RR	p/d	maxres	time	RR	p/d
MPM for k largest eigpair																
Andrew.	1.1e-12	127	5	2/ 4	1.9e-06	250	9	3/ 8	4.3e-12	165	4	1/15	8.8e-13	148	5	2/ 5
C60	1.6e-12	18	6	3/ 4	6.3e-12	12	3	1/ 8	9.7e-13	24	3	2/15	2.0e-12	13	3	1/ 9
cfd1	1.8e-12	206	3	1/ 4	1.0e-12	296	4	1/ 8	2.8e-12	411	5	2/15	9.8e-13	190	4	1/ 3
financ.	9.9e-13	102	3	1/ 4	8.9e-13	151	3	1/ 8	9.0e-13	175	4	1/15	1.3e-12	97	3	2/ 3
Ga10As.	1.3e-12	906	8	2/ 4	9.9e-13	1483	8	2/ 8	2.8e-01	5908	6	1/15	9.6e-13	442	4	1/ 5
Ga3As3.	7.6e-13	377	7	1/ 4	9.8e-13	270	5	1/ 8	2.8e-01	1483	6	1/15	1.7e-12	169	4	1/ 5
SLRP for k largest eigpair																
Andrew.	1.5e-12	116	4	1/ 4	9.9e-13	104	2	1/ 8	1.2e-13	187	2	1/15	1.0e-12	125	4	1/ 5
C60	1.5e-12	24	9	3/ 4	1.4e-12	16	5	1/ 8	7.1e-13	19	3	1/15	5.5e-12	13	3	1/ 9
cfd1	9.6e-13	185	2	1/ 4	9.9e-13	206	2	1/ 8	1.7e-13	324	2	1/15	9.9e-13	188	3	1/ 3
financ.	1.2e-12	77	1	1/ 4	1.0e-12	141	2	1/ 8	2.7e-13	327	2	1/15	1.1e-12	69	1	1/ 3
Ga10As.	5.9e-13	734	7	2/ 4	9.9e-13	448	3	1/ 8	2.9e-01	1122	6	1/15	9.7e-13	580	4	1/ 6
Ga3As3.	8.4e-12	205	4	1/ 4	9.4e-13	179	3	1/ 8	6.4e-02	442	6	1/15	1.7e-12	198	4	1/ 5
MPM for k smallest eigpair																
Andrew.	4.1e-13	247	9	3/ 4	1.5e-13	219	6	2/ 8	9.9e-13	448	5	1/15	9.7e-13	200	4	1/ 8
C60	1.6e-07	20	7	3/ 4	6.8e-13	17	6	1/ 8	7.9e-13	26	5	1/15	2.8e-12	26	9	3/ 6
cfd1	2.5e-07	1626	30	3/ 4	6.0e-12	1543	21	3/ 8	4.3e-12	1340	9	3/15	8.9e-12	719	4	1/15
financ.	6.9e-12	1002	21	3/ 4	5.1e-13	700	9	3/ 8	1.0e-12	586	5	1/15	1.4e-12	600	6	1/15
Ga10As.	9.4e-12	1893	15	3/ 4	3.7e-12	1372	5	1/ 8	1.8e-06	2198	6	2/15	1.1e-12	1558	6	1/ 8
Ga3As3.	4.9e-12	569	11	3/ 4	2.7e-12	489	4	1/ 8	9.7e-13	471	4	1/15	9.9e-13	420	4	1/ 9
SLRP for k smallest eigpair																
Andrew.	4.6e-12	315	10	3/ 4	1.0e-12	199	5	1/ 8	9.9e-13	208	3	1/15	1.1e-12	185	5	1/ 8
C60	1.2e-12	16	9	2/ 4	6.7e-12	13	4	1/ 8	4.1e-13	16	3	1/15	9.2e-13	15	8	2/ 6
cfd1	9.1e-07	1956	30	3/ 4	7.9e-12	1394	22	3/ 8	5.2e-12	1121	12	3/15	4.2e-12	1017	12	3/15
financ.	7.4e-12	1223	22	3/ 4	7.2e-13	713	11	3/ 8	1.6e-12	535	6	1/15	3.4e-12	467	5	1/15
Ga10As.	1.6e-12	1625	8	3/ 4	2.6e-12	961	4	1/ 8	1.0e-12	999	3	1/15	4.0e-12	963	3	1/ 9
Ga3As3.	4.8e-12	532	10	3/ 4	9.9e-13	381	4	1/ 8	9.8e-13	374	3	1/15	9.5e-13	338	5	1/ 9

TABLE 7.4
Comparison results on Chebyshev interpolates in (5.3) *and Chebyshev polynomials in* (2.6)

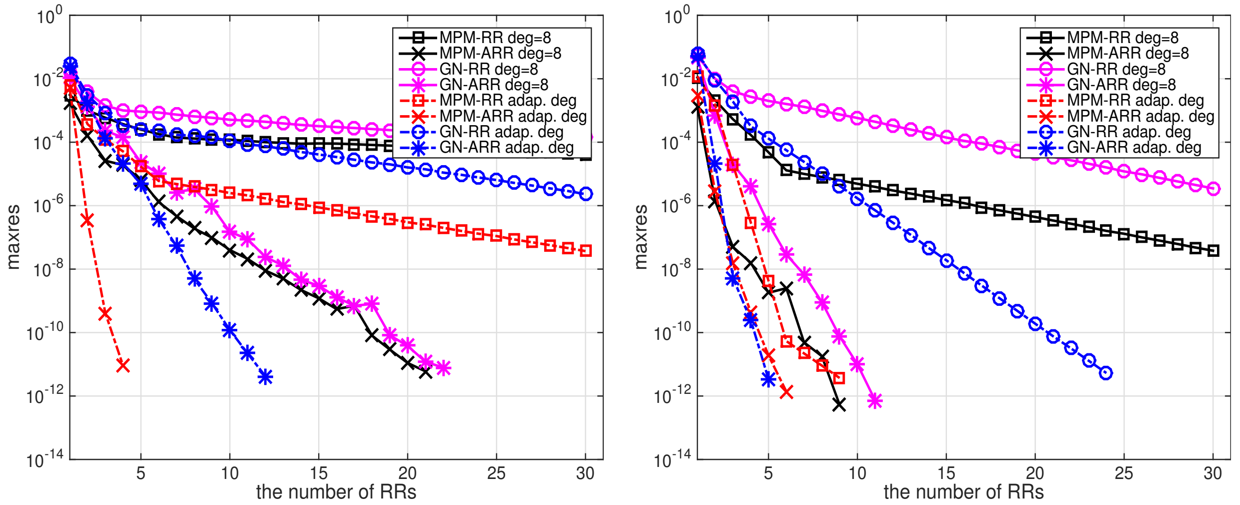
	MPM				MPM, Cheb. poly.				GN				GN, Cheb. poly.			
name	maxres	time	RR	p/d	maxres	time	RR	p/d	maxres	time	RR	p/d	maxres	time	RR	p/d
computing k largest eigpair, tol=1e-6																
Andrew.	2.6e-8	58	2	1/ 5	1.1e-6	60	2	1/ 3	3.0e-8	92	2	1/ 5	6.0e-7	89	2	1/ 3
C60	1.1e-9	13	2	1/ 9	3.9e-8	9	2	1/ 5	5.2e-7	9	2	1/ 8	8.8e-6	12	3	1/ 5
cfd1	5.6e-9	155	2	1/ 3	7.4e-7	144	2	1/ 2	1.5e-7	143	1	1/ 3	1.5e-7	146	1	1/ 3
financ.	1.6e-6	37	1	1/ 3	1.5e-10	51	1	1/ 3	1.1e-12	67	1	1/ 3	1.2e-10	68	1	1/ 3
Ga10As.	5.7e-8	264	2	1/ 5	4.6e-8	550	4	1/ 2	9.2e-7	380	2	1/ 5	2.0e-7	484	3	1/ 3
Ga3As3.	6.4e-8	101	2	1/ 5	1.6e-6	112	3	1/ 3	5.3e-7	136	2	1/ 5	6.2e-6	125	2	1/ 3
computing k largest eigpair, tol=1e-12																
Andrew.	8.8e-13	148	5	2/ 5	2.9e-12	199	7	2/10	1.0e-12	125	4	1/ 5	1.5e-12	160	7	3/ 3
C60	2.0e-12	13	3	1/ 9	4.6e-12	16	6	3/ 5	5.5e-12	13	3	1/ 9	4.5e-12	23	12	3/ 5
cfd1	9.8e-13	190	4	1/ 3	3.3e-13	230	6	2/ 2	9.9e-13	188	3	1/ 3	1.8e-12	215	4	1/ 2
financ.	1.3e-12	97	3	2/ 3	6.8e-12	87	3	1/ 2	1.1e-12	69	1	1/ 3	9.9e-13	93	2	1/ 2
Ga10As.	9.6e-13	442	4	1/ 5	9.0e-12	643	9	3/ 3	9.7e-13	580	4	1/ 6	1.3e-12	807	9	3/ 3
Ga3As3.	1.7e-12	169	4	1/ 5	2.2e-12	239	9	3/ 3	1.7e-12	198	4	1/ 5	4.7e-13	285	9	3/ 3
computing k smallest eigpair, tol=1e-6																
Andrew.	4.2e-7	113	2	1/ 8	6.1e-7	122	3	1/ 5	5.2e-9	168	3	1/ 8	2.6e-6	175	4	1/ 5
C60	9.6e-7	16	4	2/ 6	1.3e-6	11	3	1/ 4	2.4e-6	9	3	1/ 3	1.4e-6	10	4	1/ 4
cfd1	3.4e-7	601	2	1/15	5.0e-6	427	2	1/15	4.8e-6	614	5	2/15	2.7e-6	607	5	2/15
financ.	1.7e-6	338	2	1/15	3.2e-6	310	2	1/10	5.3e-9	379	3	1/15	9.3e-7	333	3	1/10
Ga10As.	6.2e-6	751	2	1/ 8	2.9e-6	744	3	1/ 5	1.8e-6	715	2	1/ 7	2.8e-6	907	3	1/ 5
Ga3As3.	6.9e-6	325	2	1/ 9	4.2e-7	269	2	1/ 5	1.7e-9	282	3	1/ 9	1.6e-6	369	5	2/ 5
computing k smallest eigpair, tol=1e-12																
Andrew.	9.7e-13	200	4	1/ 8	7.3e-12	243	8	3/ 5	1.1e-12	185	5	1/ 8	7.0e-12	293	11	3/ 5
C60	2.8e-12	26	9	3/ 6	3.4e-12	23	9	3/ 4	9.2e-13	15	8	2/ 6	2.1e-12	18	11	3/ 4
cfd1	8.9e-12	719	4	1/15	8.7e-12	1033	11	3/15	4.2e-12	1017	12	3/15	9.0e-12	1471	23	3/15
financ.	1.4e-12	600	6	1/15	8.6e-12	587	8	3/10	3.4e-12	467	5	1/15	9.0e-12	637	10	3/10
Ga10As.	1.1e-12	1558	6	1/ 8	7.6e-08	1629	9	3/15	4.0e-12	963	3	1/ 9	5.2e-12	1496	9	3/ 5
Ga3As3.	9.9e-13	420	4	1/ 9	2.0e-12	547	9	3/ 5	9.5e-13	338	5	1/ 9	3.6e-12	573	14	3/ 5



(a) matrix: cfd1

(b) matrix: finance

FIG. 7.1. *ARR vs RR: Iteration history of* `maxres` *for computing k largest eigenpairs*



(a) matrix: cfd1

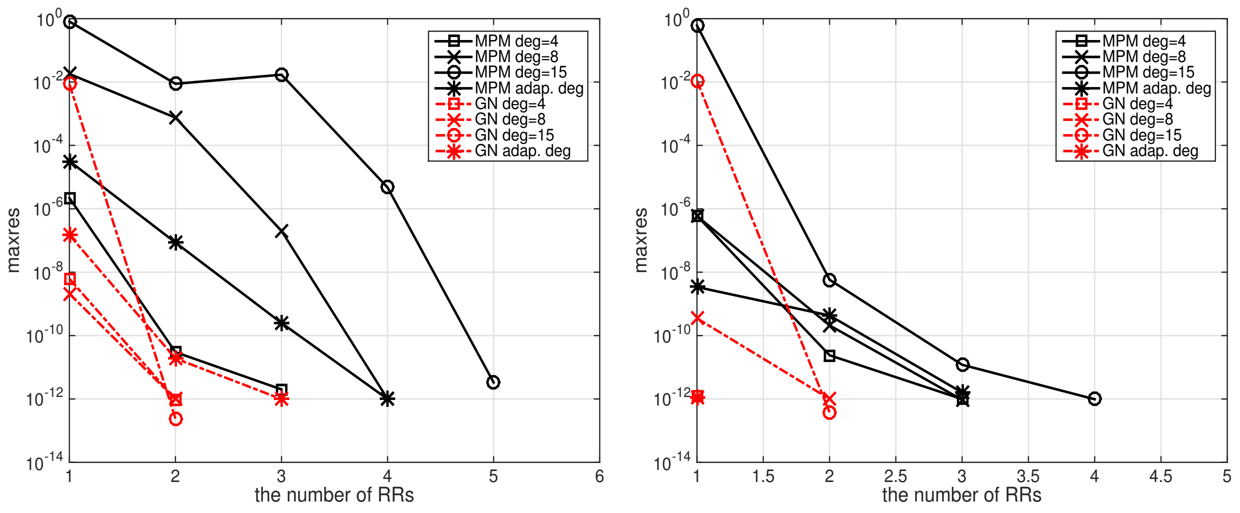
(b) matrix: finance

FIG. 7.2. *ARR vs RR: Iteration history of* `maxres` *for computing k smallest eigenpairs*

especially on the more time-consuming problems towards the right. For example, MPM and GN deliver a speedup of about 4 times on each of the seven most time-consuming problems in Figure 7.5(a), and a speedup of about 10 times on the most time-consuming problem Ga41As41H72 in Figure 7.6(a). On the other hand, compared to EIGS, FEAST's timing profile looks volatile with both big "speedups" and "slowdowns".

The benchmark solver EIGS usually, though not always, returns solutions more accurate than what is requested by the tolerance value. In particular, for $\mathrm{tol} = 10^{-6}$ the accuracy of EIGS solutions often reach the order of $O(10^{-12})$. This is due to the fact that EIGS need to maintain a high working accuracy to ensure proper convergence.

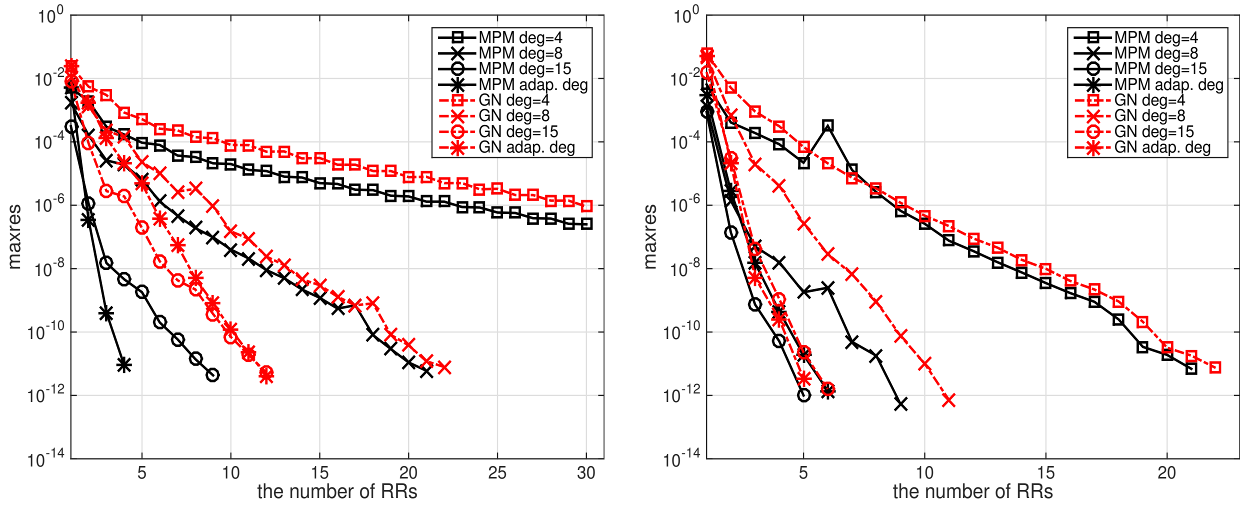
As is observed previously, it is often more time-consuming for EIGS, MPM and GN to compute k smallest eigenpairs than k largest ones on many test matrices. By examining the spectra of the matrices such as cfd1 and finance, we believe that this phenomenon is attributable to the property that these matrices tend to have a flatter end on the left end of their spectra. On the other hand, the behavior of FEAST appears less affected by this property but more by sparsity patterns (see below).



(a) matrix: cfd1

(b) matrix: finance

FIG. 7.3. *ARR: Iteration history of* `maxres` *for computing k largest eigenpairs using different polynomial degrees*



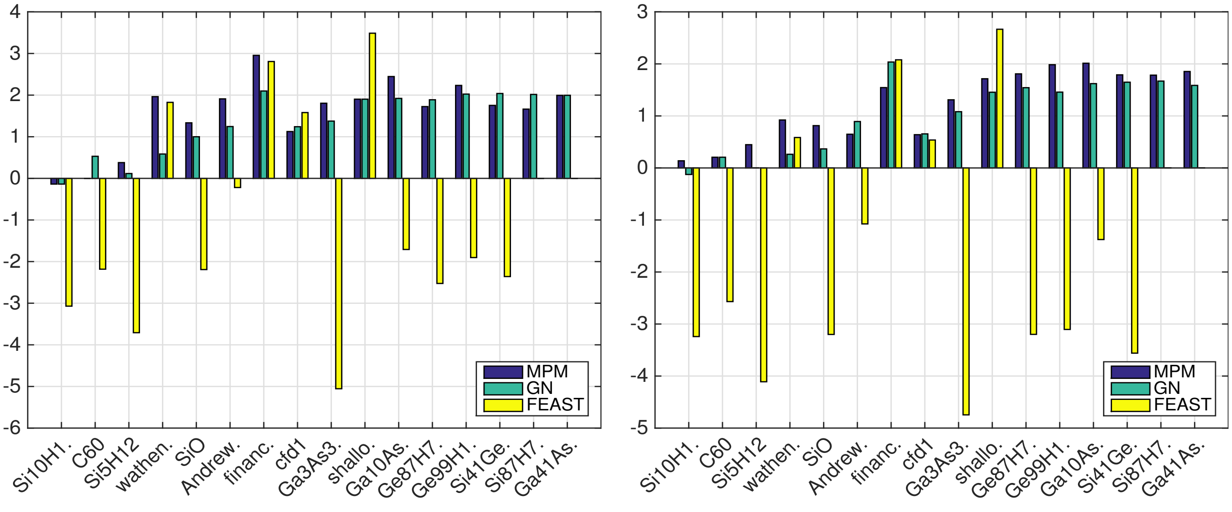
(a) matrix: cfd1

(b) matrix: finance

FIG. 7.4. *ARR: Iteration history of* `maxres` *for computing k smallest eigenpairs using different polynomial degrees*

Concerning the performance of FEAST, we make the following observations.

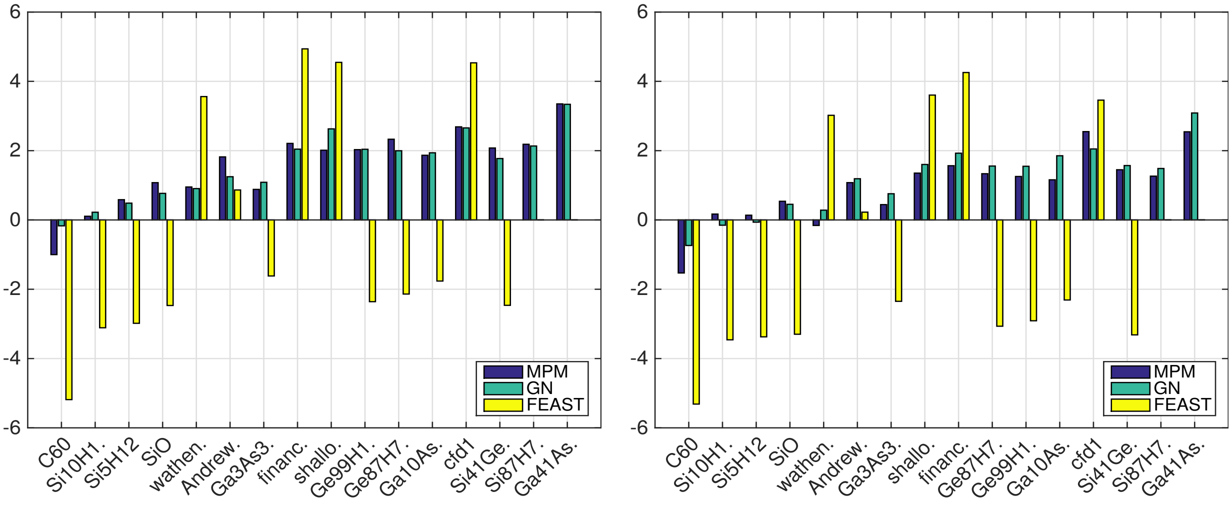
- FEAST solves most problems successfully but fails to correctly solve a few cases. When computing k largest eigenvalues for the matrix Ga10As10H30 FEAST returns the warning: “No eigenvalue has been found in the proposed search interval”. On matrix Ga3As3H12, it seems to exit normally with the output messages “Eigensolvers have successfully converged”, but the subsequently computed maximum relative residual norm in (6.1) is way too large at 0.29. On matrices Ga41As41H72 and Si87H76, when computing either k largest or smallest eigenpairs, FEAST terminates abnormally after spending a long computing time, with the message: “Eigensolvers ERROR: Problem from Inner Linear System Solver”. By examining the density of Cholesky factors for Ga41As41H72 and Si87H76 in Table 7.1, we speculate that the abnormal termination most likely has to do with excessive memory demands encountered by the inner linear system solver in Intel Math Kernel Library.
- For $\text{tol} = 10^{-12}$, FEAST is the fastest in solving finance and shallow_water1s for k largest eigenpairs, and in solving cfd1, finance, shallow_water1s and wathen100 for k smallest eigenpairs. On the other hand, FEAST can be significantly slower than others on matrices such as Ga10As10H30, Ga3As3H12, Ge87H76,



(a) $\text{tol} = 10^{-6}$

(b) $\text{tol} = 10^{-12}$

FIG. 7.5. *Speedup to* EIGS*:* $\log_2(\text{time}_{\text{EIGS}}/\text{time})$ *on computing* k *largest eigenpairs*



(a) $\text{tol} = 10^{-6}$

(b) $\text{tol} = 10^{-12}$

FIG. 7.6. *Speedup to* EIGS*:* $\log_2(\text{time}_{\text{EIGS}}/\text{time})$ *on computing* k *smallest eigenpairs*

Ge99H100, Si41Ge41H72, Si87H76 and Ga41As41H72. The performance of FEAST can be at least partly explained from the density of Cholesky factors L shown in Table 7.1, since FEAST uses a direct linear solver in Intel Math Kernel Library to compute factorizations of matrices of the form $(\phi_l I - A)$ in (2.7). We can clearly see the correlation that FEAST is fast when the density of the Cholesky factor is low and Cholesky factorization is fast.

With regard to the performance of MPM and GN, we make the following observations.

- MPM and GN both attain the required accuracy on all test problems, and they often return smaller residual errors than what is required by tol. Generally speaking, the two variants perform quite similarly in terms of both accuracy and timing.
- MPM and GN maintain a clear speed advantage over FEAST in most tested cases. They are faster than FEAST when either factorizations of shifted A are expensive, or when spectral distributions have a favorable decay (for example, on cfd1 for computing k largest eigenpairs).
- MPM and GN also maintain an overall speed advantage over EIGS, especially on those problems more time-consuming for EIGS (towards the right end of Figures 7.5 and 7.6). They are faster in spite of taking consid-

TABLE 7.5

Comparison results on computing k largest eigenpairs

	EIGS			FEAST			MPM				GN			
name	maxres	time	SpMV	maxres	time	RR	maxres	time	SpMV	RR/p/d	maxres	time	SpMV	RR/p/d
							tol=1e-6							
Andrew.	1.0e-7	218	3e+3	1.0e-8	254	5	2.6e-8	58	6e+4	2/ 1/ 5	3.0e-8	92	6e+4	2/ 1/ 5
C60	4.9e-8	13	2e+3	7.9e-9	59	3	1.1e-9	13	5e+4	2/ 1/ 9	5.2e-7	9	3e+4	2/ 1/ 8
cfd1	2.5e-14	338	3e+3	4.2e-8	113	4	5.6e-9	155	6e+4	2/ 1/ 3	1.5e-7	143	4e+4	1/ 1/ 3
financ.	3.1e-14	287	3e+3	6.1e-10	41	3	1.6e-6	37	2e+4	1/ 1/ 3	1.1e-12	67	3e+4	1/ 1/ 3
Ga10As.	4.2e-14	1439	8e+3	1.6e+0	4704	2	5.7e-8	264	1e+5	2/ 1/ 5	9.2e-7	380	1e+5	2/ 1/ 5
Ga3As3.	1.9e-8	353	5e+3	2.9e-1	11738	21	6.4e-8	101	7e+4	2/ 1/ 5	5.3e-7	136	6e+4	2/ 1/ 5
shallo.	1.5e-10	774	8e+3	5.2e-9	69	4	4.9e-9	207	2e+5	2/ 1/ 7	9.2e-8	207	1e+5	2/ 1/ 7
Si10H1.	5.6e-7	10	2e+3	2.6e-10	84	3	5.2e-9	11	4e+4	2/ 1/ 9	1.2e-10	11	3e+4	2/ 1/ 9
Si5H12	1.5e-12	13	2e+3	1.2e-8	170	3	1.0e-10	10	3e+4	2/ 1/ 6	4.6e-8	12	3e+4	2/ 1/ 6
SiO	1.4e-13	58	3e+3	4.1e-7	265	2	1.4e-8	23	4e+4	2/ 1/ 5	4.1e-7	29	4e+4	2/ 1/ 5
wathen.	5.5e-14	39	2e+3	6.0e-8	11	4	1.1e-6	10	2e+4	1/ 1/ 3	6.9e-11	26	4e+4	2/ 1/ 5
Ge87H7.	1.7e-8	1451	8e+3	5.3e-9	8352	3	6.5e-10	439	2e+5	2/ 1/ 6	1.2e-7	392	1e+5	2/ 1/ 6
Ge99H1.	2.5e-14	1636	8e+3	5.6e-7	6119	2	2.3e-9	348	1e+5	2/ 1/ 6	7.4e-8	402	1e+5	2/ 1/ 6
Si41Ge.	1.1e-8	2909	9e+3	3.9e-7	14929	2	1.6e-9	863	2e+5	2/ 1/ 7	5.8e-8	708	1e+5	2/ 1/ 7
Si87H7.	3.5e-14	3568	1e+4	2.8e-1	1702	1	4.0e-9	1126	3e+5	2/ 1/ 7	1.1e-7	882	1e+5	2/ 1/ 7
Ga41As.	7.4e-14	4100	1e+4	8.6e-1	1066	1	1.2e-10	1029	2e+5	3/ 1/ 5	2.1e-7	1028	1e+5	2/ 1/ 7
name	maxres	time	SpMV	maxres	time	RR	maxres	time	SpMV	RR/p/d	maxres	time	SpMV	RR/p/d
							tol=1e-12							
Andrew.	5.6e-14	232	4e+3	4.7e-14	489	9	8.8e-13	148	1e+5	5/ 2/ 5	1.0e-12	125	8e+4	4/ 1/ 5
C60	6.3e-13	15	2e+3	2.8e-13	89	5	2.0e-12	13	5e+4	3/ 1/ 9	5.5e-12	13	4e+4	3/ 1/ 9
cfd1	2.5e-14	296	3e+3	7.1e-14	204	8	9.8e-13	190	8e+4	4/ 1/ 3	9.9e-13	188	6e+4	3/ 1/ 3
financ.	2.1e-14	283	3e+3	2.1e-14	67	5	1.3e-12	97	5e+4	3/ 2/ 3	1.1e-12	69	3e+4	1/ 1/ 3
Ga10As.	4.8e-14	1784	8e+3	1.6e+0	4631	2	9.6e-13	442	2e+5	4/ 1/ 5	9.7e-13	580	2e+5	4/ 1/ 6
Ga3As3.	2.1e-14	419	5e+3	2.9e-1	11245	21	1.7e-12	169	1e+5	4/ 1/ 5	1.7e-12	198	1e+5	4/ 1/ 5
shallo.	4.6e-13	768	8e+3	1.9e-13	121	7	1.0e-12	234	2e+5	4/ 1/ 7	9.9e-13	280	2e+5	4/ 1/ 7
Si10H1.	5.3e-14	11	2e+3	4.0e-13	104	4	6.2e-13	10	3e+4	2/ 1/ 9	3.7e-14	12	3e+4	3/ 1/ 9
Si5H12	1.1e-14	15	2e+3	2.6e-13	259	5	9.5e-13	11	3e+4	2/ 1/ 6	5.3e-12	15	3e+4	3/ 1/ 6
SiO	1.4e-14	58	3e+3	4.7e-13	533	4	9.8e-13	33	5e+4	3/ 1/ 5	1.4e-12	45	6e+4	4/ 1/ 5
wathen.	4.3e-14	36	2e+3	5.1e-14	24	8	1.1e-12	19	4e+4	2/ 1/ 5	9.8e-13	30	4e+4	3/ 1/ 5
Ge87H7.	2.8e-14	1524	8e+3	1.3e-13	13993	5	4.8e-12	435	2e+5	3/ 1/ 6	1.0e-12	523	2e+5	4/ 1/ 6
Ge99H1.	8.4e-14	1563	8e+3	2.1e-14	13438	5	3.7e-12	395	2e+5	2/ 1/ 6	9.6e-13	569	2e+5	4/ 1/ 6
Si41Ge.	2.6e-14	2991	9e+3	2.5e-14	35270	5	9.9e-13	865	2e+5	3/ 1/ 7	1.1e-12	954	2e+5	3/ 1/ 7
Si87H7.	2.8e-14	3506	1e+4	2.8e-1	1924	1	1.0e-12	1018	2e+5	3/ 1/ 7	1.4e-12	1102	2e+5	3/ 1/ 7
Ga41As.	7.5e-14	4103	1e+4	8.6e-1	1242	1	7.9e-13	1135	2e+5	3/ 1/ 7	3.7e-12	1366	2e+5	3/ 1/ 7

erably more matrix-vector multiplications than EIGS, as can be seen from Tables 7.5 and 7.6, thanks to the benefits of relying on high-concurrency operations on many-core computers.

- MPM and GN generally require a smaller number ARR calls, often only two or three when computing k largest eigenpairs. In quite a number of cases (for example, on finance and wathen100 for MPM and so on), only a single ARR projection is taken which is absolutely optimal in order to extract approximate eigenpairs.
- The number of augmentation blocks used by MPM and GN is usually 1, and the final polynomial degree never reaches the maximum degree 15 except on cfd1, finance and wathen100 when computing k smallest eigenpairs.

In Figure 7.7, we plot runtimes of three categories: SpMV (i.e., AX), SU (lines 10 to 22 of Algorithm 8) and ARR (lines 23 to 27 of Algorithm 8). In particular, SpMVs are called in both SU and ARR, but overwhelmingly in the former. These are the major computational components of MPM and GN. The runtime of each category is measured in the percentage of wall-clock time spent in that category over the total wall-clock time. We can see, especially from the time-consuming problems on the right, that (i) the time of SU dominates that of RR, and (ii) the time of SpMVs, always done in batch of $k + q$, dominates the entire computation in almost all cases. These trends are much more pronounced (a) for MPM than for GN (recall that GN requires to solve $k \times k$ linear systems); and (b) for computing k smallest eigenpairs than for computing k largest ones (recall that the former is generally more difficult). These runtime profiles are favorable to parallel scalability since AX operations possess high concurrency for relatively large k.

In the final set of experiments, we examine the solvers' scalability with respect to k. We apply the solvers to matrices cfd1 and Ge87H76, with $\mathrm{tol} = 10^{-12}$, and vary k from $100, 200$ up to 1200 with increment 200 (there are exceptions for FEAST). The resulting solution times are plotted in Figures 7.8 and 7.9. In both figures, the slopes of

TABLE 7.6
Comparison results on computing k smallest eigenpairs

	EIGS			FEAST			MPM				GN			
name	maxres	time	SpMV	maxres	time	RR	maxres	time	SpMV	RR/p/d	maxres	time	SpMV	RR/p/d
							tol=1e-6							
Andrew.	4.9e-7	399	7e+3	8.5e-8	219	4	4.2e-7	113	1e+5	2/ 1/ 8	5.2e-9	168	1e+5	3/ 1/ 8
C60	2.2e-13	8	2e+3	6.4e-5	291	16	9.6e-7	16	5e+4	4/ 2/ 6	2.4e-6	9	2e+4	3/ 1/ 3
cfd1	4.7e-9	3871	6e+4	4.2e-8	167	7	3.4e-7	601	7e+5	2/ 1/ 15	4.8e-6	614	6e+5	5/ 2/ 15
financ.	1.2e-9	1563	2e+4	4.5e-8	51	4	1.7e-6	338	4e+5	2/ 1/ 15	5.3e-9	379	3e+5	3/ 1/ 15
Ga10As.	2.9e-12	2740	2e+4	8.9e-9	9302	4	6.2e-6	751	3e+5	2/ 1/ 8	1.8e-6	715	2e+5	2/ 1/ 7
Ga3As3.	1.7e-12	599	8e+3	7.3e-8	1837	3	6.9e-6	325	2e+5	2/ 1/ 9	1.7e-9	282	2e+5	3/ 1/ 9
shallo.	3.8e-14	1614	2e+4	6.1e-8	69	4	2.0e-8	400	4e+5	2/ 1/ 14	4.1e-6	261	2e+5	2/ 1/ 9
Si10H1.	1.5e-7	14	2e+3	1.2e-7	121	4	2.8e-7	13	5e+4	2/ 1/ 8	7.1e-6	12	3e+4	2/ 1/ 8
Si5H12	5.8e-12	21	3e+3	1.5e-8	166	3	3.3e-7	14	4e+4	2/ 1/ 8	6.5e-6	15	3e+4	2/ 1/ 8
SiO	2.7e-13	97	5e+3	5.6e-8	537	4	4.1e-7	46	9e+4	2/ 1/ 8	8.8e-10	57	9e+4	3/ 1/ 8
wathen.	1.4e-9	118	8e+3	8.4e-8	10	4	8.2e-6	61	2e+5	2/ 1/ 15	2.4e-7	63	1e+5	3/ 1/ 15
Ge87H7.	2.0e-13	2559	1e+4	2.7e-8	11268	4	4.8e-7	509	3e+5	2/ 1/ 9	8.1e-10	641	2e+5	3/ 1/ 9
Ge99H1.	2.1e-11	2319	1e+4	1.0e-8	11892	4	4.8e-7	568	3e+5	2/ 1/ 9	2.0e-6	564	2e+5	2/ 1/ 8
Si41Ge.	4.1e-9	4650	1e+4	1.2e-8	25658	4	6.3e-7	1102	3e+5	2/ 1/ 11	4.1e-10	1361	3e+5	3/ 1/ 11
Si87H7.	3.0e-13	5458	2e+4	3.3e+0	1842	1	3.2e-6	1201	3e+5	2/ 1/ 11	7.4e-6	1243	2e+5	2/ 1/ 10
Ga41As.	3.6e-7	32279	8e+4	8.6e-1	1095	1	2.1e-8	3166	5e+5	3/ 1/ 11	1.3e-6	3193	4e+5	3/ 2/ 11
name	maxres	time	SpMV	maxres	time	RR	maxres	time	SpMV	RR/p/d	maxres	time	SpMV	RR/p/d
							tol=1e-12							
Andrew.	1.2e-13	422	7e+3	4.1e-13	361	7	9.7e-13	200	2e+5	4/ 1/ 8	1.1e-12	185	2e+5	5/ 1/ 8
C60	2.6e-14	9	2e+3	6.4e-6	358	21	2.8e-12	26	7e+4	9/ 3/ 6	9.2e-13	15	4e+4	8/ 2/ 6
cfd1	2.9e-14	4209	6e+4	5.5e-14	383	16	8.9e-12	719	9e+5	4/ 1/ 15	4.2e-12	1017	1e+6	12/ 3/ 15
financ.	9.7e-13	1776	2e+4	5.5e-14	93	8	1.4e-12	600	7e+5	6/ 1/ 15	3.4e-12	467	4e+5	5/ 1/ 15
Ga10As.	2.8e-12	3479	2e+4	9.4e-14	17251	7	1.1e-12	1558	7e+5	6/ 1/ 8	4.0e-12	963	3e+5	3/ 1/ 9
Ga3As3.	1.2e-12	571	8e+3	3.8e-13	2908	5	9.9e-13	420	3e+5	4/ 1/ 9	9.5e-13	338	2e+5	5/ 1/ 9
shallo.	3.9e-14	1532	2e+4	2.7e-13	126	8	3.2e-12	600	6e+5	5/ 1/ 12	4.0e-13	505	4e+5	5/ 1/ 14
Si10H1.	7.9e-14	18	2e+3	2.1e-12	198	7	2.0e-12	16	5e+4	4/ 1/ 8	3.9e-13	20	5e+4	5/ 1/ 8
Si5H12	1.5e-13	22	3e+3	3.6e-14	228	5	2.1e-12	20	6e+4	4/ 1/ 8	9.6e-12	23	6e+4	4/ 1/ 8
SiO	2.7e-13	93	5e+3	2.7e-13	915	7	6.0e-13	64	1e+5	5/ 1/ 8	9.4e-13	68	1e+5	5/ 1/ 8
wathen.	8.2e-13	146	8e+3	1.0e-13	18	7	3.1e-12	163	5e+5	6/ 2/ 15	1.5e-12	120	3e+5	7/ 2/ 15
Ge87H7.	1.8e-13	2250	1e+4	1.5e-13	18852	7	2.6e-13	892	4e+5	5/ 1/ 9	9.9e-13	765	3e+5	5/ 1/ 9
Ge99H1.	1.8e-13	2353	1e+4	6.7e-14	17683	7	9.7e-13	986	5e+5	4/ 1/ 9	9.9e-13	804	3e+5	4/ 1/ 9
Si41Ge.	3.3e-13	4656	2e+4	1.3e-13	46386	7	9.9e-12	1705	5e+5	4/ 1/ 11	9.8e-13	1568	3e+5	5/ 1/ 11
Si87H7.	3.0e-13	5487	2e+4	3.3e+0	1854	1	1.1e-12	2284	6e+5	6/ 1/ 11	1.1e-12	1960	4e+5	5/ 1/ 11
Ga41As.	5.3e-12	33254	8e+4	8.6e-1	998	1	8.8e-13	5700	1e+6	7/ 2/ 11	1.7e-12	3913	5e+5	5/ 2/ 12

the time curves confirm that the three block algorithms, FEAST, MPM and GN, clearly scale better with respect to k than the Krylov subspace algorithm EIGS. Although EIGS can be the fastest for k small, its solution time increases at a faster pace than the block methods as k increases.

Among the block algorithms themselves, all three provide comparable performances on cfd1 when computing the k largest eigenpairs, while FEAST is the fastest when computing k smallest eigenpairs. On Ge87H76, which has a rather dense Cholesky factor, FEAST is much slower in all runs up to $k = 1000$ (runs for $k > 1000$ are skipped to save time).

8. Concluding Remarks. The goal of this paper is to construct a block algorithm of high scalability suitable for computing relatively large numbers of exterior eigenpairs for really large-scale matrices on modern computers. Our strategy is simple: to reduce as much as possible the number of RR calls (Rayleigh-Ritz projections) or, in other words, to shift as much as possible computation burdens to SU (subspace update) steps. This strategy is based on the following considerations. RR steps perform small dense eigenvalue decompositions, as well as basis orthogonalizations, thus possessing limited concurrency. On the other hand, SU steps can be accomplished by block operations like A times X, thus more scalable.

To reach for maximal concurrency, we choose the power iteration for subspace updating (and also include a Gauss-Newton method to test the versatility of our construction). It is well known that the convergence of the power method can be intolerably slow, preventing it from being used to drive general-purpose eigensolvers. Therefore, the key to success reduces to whether we could accelerate the power method sufficiently and reliably to an extent that it can compete in speed with Krylov subspace methods in general. In this work, such an acceleration is accomplished mainly through the use of three techniques: (1) an augmented Rayleigh-Ritz (ARR) procedure that can provably accelerate convergence under mild conditions; (2) a set of easy-to-control, low-degree polynomial accelerators; and (3) a bold stoping rule for SU steps that essentially allows an iterate matrix to become numerically rank-deficient. Of course, the success of our construction also depends greatly on a set of carefully integrated algorithmic details. The resulting algorithm is named ARRABIT, which uses A only in matrix multiplications.

Numerical experiments in Matlab on sixteen test matrices from the UF Sparse Matrix Collection show, convincingly in our view, that the accuracy and efficiency of ARRABIT is indeed competitive to start-of-the-art eigensolvers. Exceeding our expectations, ARRABIT can already provide multi-fold speedups over the benchmark solver EIGS, without explicit code parallelization and without running on massively parallel machines, on difficult problems. In particular, it often only needs two or three, sometimes just one, ARR projections to reach a good solution accuracy.

There are a number of future directions worth pursuing from this point on. For one thing, the robustness and efficiency of ARRABIT can be further enhanced by refining its construction and and tuning its parameters. Software development and an evaluation of its parallel scalability are certainly important. The prospective of extending the algorithm to non-Hermitian matrices and the generalized eigenvalue problem looks promising. Overall, we feel that the present work has laid a solid foundation for these and other future activities.

Acknowledgements. Most of 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.

REFERENCES

[1] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[2] M. Bollhöfer and Y. Notay, *JADAMILU: a software code for computing selected eigenvalues of large sparse symmetric matrices*, Comput. Phys. Comm., 177 (2007), pp. 951–964.

[3] J. W. Demmel, *Applied Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[4] T. A. Driscoll, N. Hale, and L. N. Trefethen, *Chebfun Guide*, Pafnuty Publications, 2014.

[5] H. Ehlich and K. Zeller, *Auswertung der normen von interpolationsoperatoren*, Mathematische Annalen, 164 (1966), pp. 105–112.

[6] H.-R. Fang and Y. Saad, *A filtered Lanczos procedure for extreme and interior eigenvalue problems*, SIAM J. Sci. Comput., 34 (2012), pp. A2220–A2246.

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

[8] 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.

[9] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat'l Bur. Std., 45 (1950), pp. 225–282.

[10] R. M. Larsen, *Lanczos bidiagonalization with partial reorthogonalization*, Aarhus University, Technical report, DAIMI PB-357, September 1998.

[11] R. B. Lehoucq, *Implicitly restarted Arnoldi methods and subspace iteration*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 551–562.

[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] X. Liu, Z. Wen, and Y. Zhang, *Limited memory block krylov subspace optimization for computing dominant singular value decompositions*, SIAM Journal on Scientific Computing, 35-3 (2013), pp. A1641–A1668.

[14] X. Liu, Z. Wen, and Y. Zhang, *An efficient Gauss-Newton algorithm for symmetric low-rank product matrix approximations*, SIAM Journal on Optimization, (To appear 2015).

[15] G. Mastroianni and J. Szabados, *Jackson order of approximation by lagrange interpolation. ii*, Acta Mathematica Hungarica, 69 (1995), pp. 73–82.

[16] B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, 1980.

[17] E. Polizzi, *Density-matrix-based algorithm for solving eigenvalue problems*, Phys. Rev. B, 79 (2009), p. 115112.

[18] H. Rutishauser, *Computational aspects of F. L. Bauer's simultaneous iteration method*, Numer. Math., 13 (1969), pp. 4–13.

[19] H. Rutishauser, *Simultaneous iteration method for symmetric matrices*, Numer. Math., 16 (1970), pp. 205–223.

[20] Y. Saad, *Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems*, Mathematics of Computation, 42 (1984), pp. 567–588.

[21] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press, 1992.

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

[23] D. C. Sorensen, *Implicitly restarted Arnoldi/Lanczos methods for large scale eigenvalue calculations*, in Parallel numerical algorithms (Hampton, VA, 1994), vol. 4 of ICASE/LaRC Interdiscip. Ser. Sci. Eng., Kluwer Acad. Publ., 1996, pp. 119–165.

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

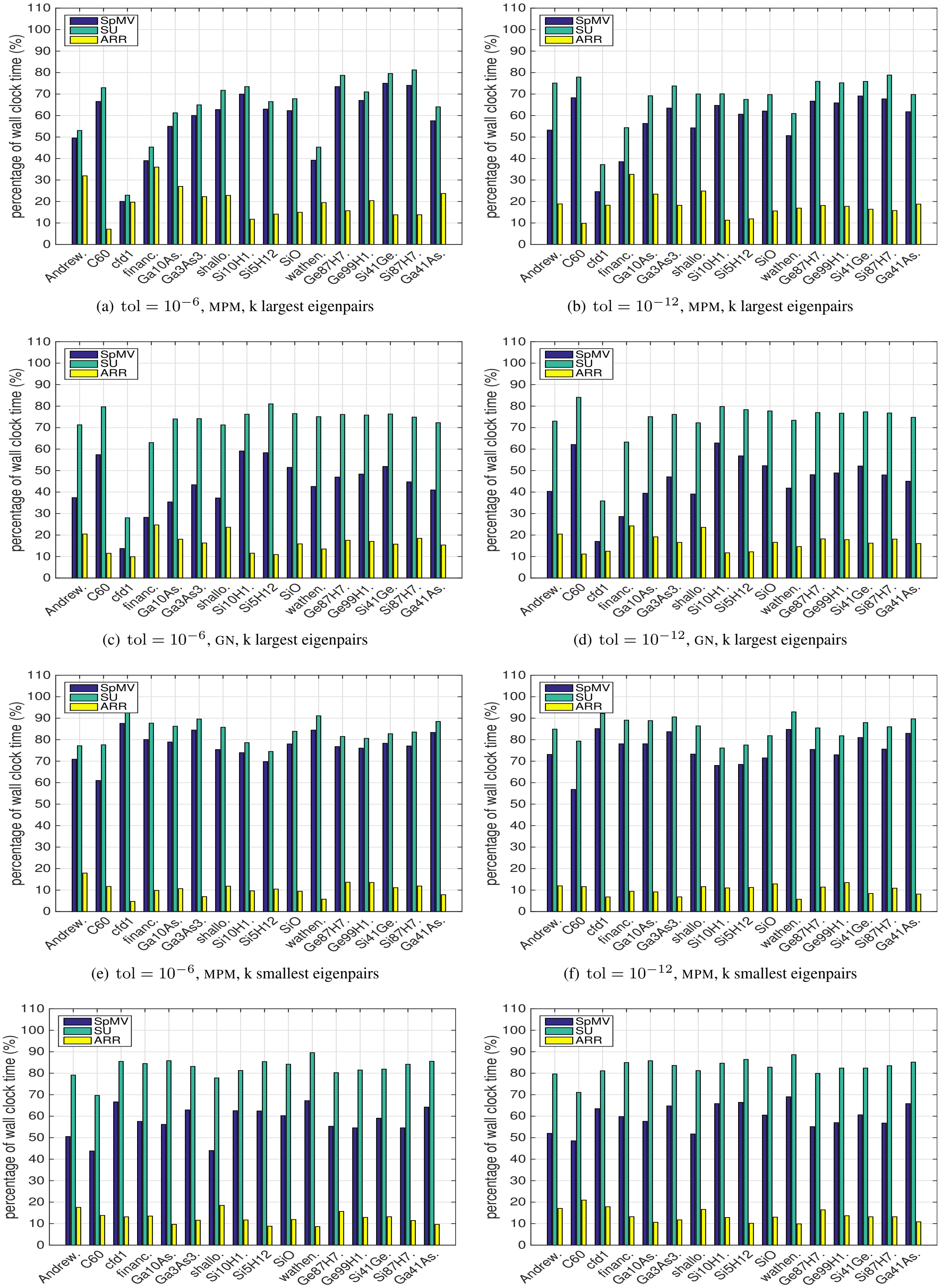
[25] G. W. STEWART, *Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices*, Numer. Math., 25 (1975/76), pp. 123–136.

[26] ———, *Matrix algorithms Vol. II: Eigensystems*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001.

[27] W. J. STEWART AND A. JENNINGS, *A simultaneous iteration algorithm for real matrices*, ACM Trans. Math. Software, 7 (1981), pp. 184–198.

[28] P. T. P. TANG AND E. POLIZZI, *FEAST as a subspace iteration eigensolver accelerated by approximate spectral projection*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 354–390.

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



(a) tol $= 10^{-6}$, MPM, k largest eigenpairs

(b) tol $= 10^{-12}$, MPM, k largest eigenpairs

(c) tol $= 10^{-6}$, GN, k largest eigenpairs

(d) tol $= 10^{-12}$, GN, k largest eigenpairs

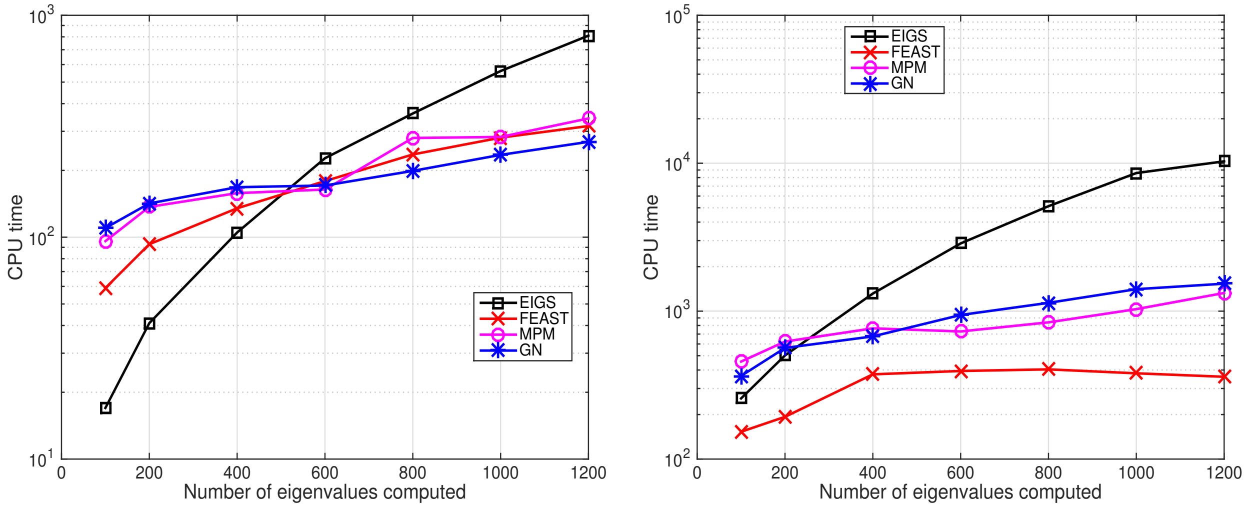
(e) tol $= 10^{-6}$, MPM, k smallest eigenpairs

(f) tol $= 10^{-12}$, MPM, k smallest eigenpairs

(g) tol $= 10^{-6}$, GN, k smallest eigenpairs

(h) tol $= 10^{-12}$, GN, k smallest eigenpairs

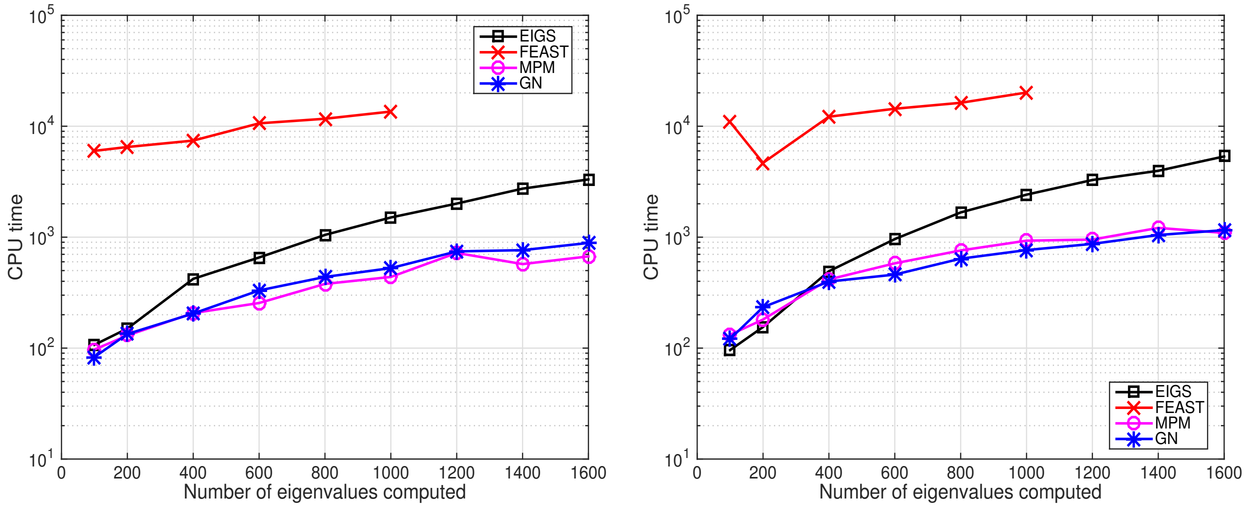
FIG. 7.7. *A comparison of timing profile among SpMV, SU and ARR*



(a) k largest eigenpairs, $tol = 10^{-12}$

(b) k smallest eigenpairs, $tol = 10^{-12}$

FIG. 7.8. *Comparison results of solution time for computing k eigenpairs of the matrix cfd1*



(a) k largest eigenpairs, $tol = 10^{-12}$

(b) k smallest eigenpairs, $tol = 10^{-12}$

FIG. 7.9. *Comparison results on solution time for computing k eigenpairs of the matrix Ge87H76.*